Global Development via Local Observational Construction Steps[⋆]

Michel Bidoit[1], Donald Sannella[2], and Andrzej Tarlecki[3]

[1] Laboratoire Spécification et Vérification, CNRS & ENS de Cachan, France
[2] Laboratory for Foundations of Computer Science, University of Edinburgh, UK
[3] Institute of Informatics, Warsaw University and Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract. The way that refinement of individual "local" components of a specification relates to development of a "global" system from a specification of requirements is explored. Observational interpretation of specifications and refinements add expressive power and flexibility while bringing in some subtle problems. The results are instantiated in the context of CASL architectural specifications.

1 Introduction

There has been a great deal of work in the algebraic specification tradition on formalizing the rather intuitive and appealing idea of program development by stepwise refinement, including [EKMP82,Gan83,GM82,Sch87,ST88b]; for a recent survey, see [EK99]). There are many issues that make this a difficult problem, and some of them are rather subtle, one example being the relationship between specification structure and program structure. There are difficult interactions and tradeoffs, an obvious one being between the expressive power of a specification formalism and the ease of reasoning about specifications. Different approaches give more or less prominence to different issues. An overview that covers most of our own contributions is [ST97], with some more recent work addressing the problem of how to prove correctness of refinement steps [BH98], the design of a convenient formalism for writing specifications [ABK+03,BST02], and applications to data refinement in typed λ-calculus [HLST00].

A new angle that we explore here is the "global" effect of refining individual "local" components of a specification. This involves a well-known technique from algebraic specification, namely the use of pushouts of signatures and amalgamation of models to build large systems by composition of separate interrelated components. The situation becomes considerably more subtle when observational interpretation of specifications and refinements is taken into account.

Part of the answer has already been provided, the main references being Schoett's thesis [Sch87,Sch90] and our work on formal development in the EXTENDED ML framework [ST89]; the general ideas go back at least to [Hoa72].

[⋆] This work has been partially supported by KBN grant 7T11C 002 21 and European AGILE project IST-2001-32747 (AT), CNRS–PAS Research Cooperation Programme (MB, AT), and British–Polish Research Partnership Programme (DS, AT).

We have another look at these issues here, in the context of the Casl specification formalism [ABK$^+$03] and in particular, its *architectural specifications* [BST02]. Architectural specifications, for describing the modular structure of software systems, are probably the most novel feature of Casl. We view them here as a means of making complex refinement steps, by defining well-structured constructions to be used to build the overall system from implementations of individual units (these also include parametrized units, acting as constructions providing some local construction steps to be used in a more global context).

We begin by introducing in Sect. 2 some details of the underlying logical system we will be working with, and our assumptions concerning specifications built using this system. Our basic view of program development by means of consecutive local refinement steps is presented in Sect. 3. Then, an observational view of specifications is motivated and recalled in Sect. 4. The principal core of the work is in Sect. 5, where we combine the ideas of the previous two sections and discuss program development by local refinement steps with respect to an observational interpretation of the specifications involved. Section 6 introduces a simplified version of Casl architectural specifications, while Sect. 7 sketches their observational semantics and shows how the ideas of Sect. 5 are instantiated in this context. Further work and possible generalizations are discussed in Sect. 8. Due to lack of space we have been unable to include concrete examples that illustrate the definitions and results, but we plan to provide such material in a future extended version.

2 Signatures, Models and Specifications

A basic assumption underpinning algebraic specification and derived approaches to software specification and development is that software systems are modeled as algebras (of some kind) and their static properties are captured by algebraic signatures (again, adapted as appropriate). This leads to quite a flexible framework, which can be tuned as desired to cope with various programming features of interest by selecting the appropriate variation of algebra and signature. This flexibility has been formalized via the notion of *institution* [GB92] and related work on the theory of specifications and formal program development [ST88a,ST97,BH93]. However, rather than exploiting the full generality of institutions, to keep things simple and illustrative we will in this paper base our considerations on a very basic logical framework, leaving to a more extensive presentation elsewhere the required generalization and adaptation to a fully-fledged formalism such as Casl.

So, we will deal here with the usual notions of many-sorted algebraic signatures and signature morphisms; we will assume that all signatures contain a distinguished Boolean part: a sort *bool* with two constants *true* and *false* preserved by all signature morphisms. This yields the category **AlgSig** — it is cocomplete, and we will assume that it comes with some standard construction of pushouts.

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 a fixed, standard interpretation of the Boolean part of the signature. 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 $\mathbf{Alg}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$. We refer to [ST99] for a more detailed presentation of the technicalities and for the standard notations we will use in the following.

It can easily be checked that $\mathbf{Alg}$ is continuous, i.e., maps colimits of algebraic signatures to limits of (algebra) categories (the initial signature, containing the Boolean part only, is mapped to the category having as its only object the algebra providing the fixed interpretation for the Boolean part). In particular, the following *amalgamation property* holds:

Lemma 2.1. *Given a pushout in the category of algebraic signatures* $\mathbf{AlgSig}$*:*

$$\begin{tikzcd}
\Sigma_1 \arrow[r, "\iota'"] & \Sigma_1' \\
\Sigma \arrow[u, "\gamma"] \arrow[r, "\iota"'] & \Sigma' \arrow[u, "\gamma'"']
\end{tikzcd}$$

for any algebras $A_1 \in |\mathbf{Alg}(\Sigma_1)|$ and $A' \in |\mathbf{Alg}(\Sigma')|$ such that $A_1|_\gamma = A'|_\iota$ there exists a unique algebra $A_1' \in |\mathbf{Alg}(\Sigma_1')|$ such that $A_1'|_{\iota'} = A_1$ and $A_1'|_{\gamma'} = A'$; and similarly for algebra homomorphisms.

Given a signature Σ, terms and first-order formulae with equality are defined as usual. Σ-sentences are closed first-order formulae. Given a Σ-algebra A, a set of variables X and a valuation of variables $v\colon X \to |A|$, the *value* $t_{A[v]}$ of a term t with variables X in A under v and the *satisfaction* $A[v] \models \phi$ of a formula ϕ with variables X in A under v are defined as usual.

We will also employ a generalized notion of terms, modeling a pretty general idea of how a value may be determined in an algebra. Given a signature Σ, a *conditional term* of sort s with variables X is of the form $p = ((\phi_i, t_i)_{i\geq 0}, t)$, where for $i \geq 0$, ϕ_i are formulae with variables X, and t_i and t are terms of sort s with variables X. Given a Σ-algebra A and a valuation $v\colon X \to |A|$, the value $p_{A[v]}$ of such a conditional term p is $(t_k)_{A[v]}$ for the least $k \geq 0$ such that $A[v] \models \phi_k$, or $t_{A[v]}$ if no such $k \geq 0$ exists.

This allows for a further generalization of *derived signature morphisms* [SB83], where we allow such a morphism $\delta\colon \Sigma \to \Sigma'$ to map function symbols $f\colon s_1 \times \ldots \times s_n \to s$ to conditional terms of sort s with variables $\{x_1\colon s_1, \ldots, x_n\colon s_n\}$. Evidently, such a derived signature morphisms $\delta\colon \Sigma \to \Sigma'$ still determines a reduct function $\_|_\delta\colon |\mathbf{Alg}(\Sigma')| \to |\mathbf{Alg}(\Sigma)|$ on algebra classes (which in general does *not* extend to a reduct functor between algebra categories).

We will not need to know much about the formalism used for writing specifications. We just assume that some class of specifications is defined, equipped with a semantics that for any specification *SP* determines its signature $\mathit{Sig}(SP) \in |\mathbf{AlgSig}|$ and its class of *models* $\mathit{Mod}(SP) \subseteq |\mathbf{Alg}(\mathit{Sig}(SP))|$. We also assume that the class specifications is closed under *translation* along signature morphisms, i.e., for any specification SP and signature morphism $\sigma\colon Sig(SP) \to \Sigma'$, we have a specification $\sigma(SP)$ with $Sig(\sigma(SP)) = \Sigma'$ and $Mod(\sigma(SP)) = \{A' \in |\mathbf{Alg}(\Sigma')| \mid A'|_\sigma \in Mod(SP)\}$, and under *unions*, i.e., for any specifications SP_1 and SP_2 with common signature, we have a specification SP_1 **and** SP_2 with $Sig(SP_1\ \mathbf{and}\ SP_2) = Sig(SP_1) = Sig(SP_2)$ and $Mod(SP_1\ \mathbf{and}\ SP_2) = Mod(SP_1) \cap Mod(SP_2)$. So, specifications can for instance be basic specifications, given by a signature and a set of axioms (sentences) over this signature; or structured specifications built over the institution we have implicitly introduced above as defined in [ST88a]; or structured specifications built using more advanced structuring mechanisms such as those of Casl [ABK+03].

3 Program Development and Refinements

In this section we briefly recapitulate our view of the process by means of which software can be formally developed from an algebraic specification of requirements, see [ST88b,ST97]. This is followed by an explanation of the way that development steps can arise from "local" constructions.

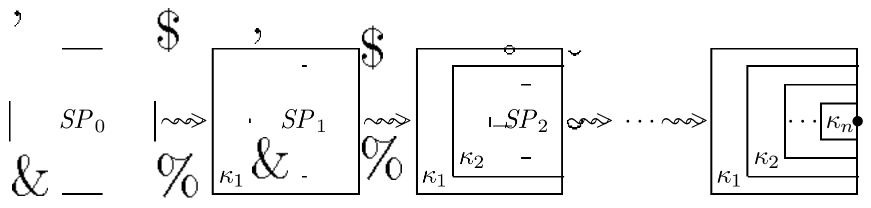
Given a requirements specification SP, the programmer's task is to provide a program that correctly implements it. In semantic terms, this amounts to building an algebra $A \in |\mathbf{Alg}(Sig(SP))|$ such that $A \in Mod(SP)$. At this level of generality and abstraction, we will not offer programming techniques for achieving this. We will instead concentrate on the methodological idea that one may proceed in a stepwise fashion by means of successive *refinements*, gradually enriching the original requirements specification with more and more implementation details until a directly implementable specification is obtained:

$$SP_0 \rightsquigarrow SP_1 \rightsquigarrow \cdots \rightsquigarrow SP_n$$

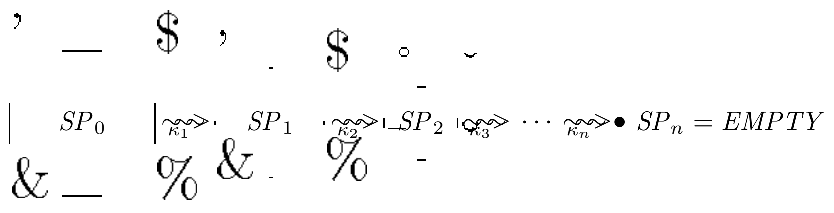
SP_0 is the original requirements specification and $SP_{i-1} \rightsquigarrow SP_i$ for $i = 1, \ldots, n$ are individual refinement steps. Joined together, these lead from SP_0 to a specification SP_n which is so detailed that it can be implemented directly — that is, such that an algebra $A_n \in Mod(SP_n)$ can be easily programmed. This "program" A_n correctly implements SP_0 provided we require that refinement steps preserve the specification signature and define:

$$SP \rightsquigarrow SP' \iff Mod(SP') \subseteq Mod(SP)$$

Although mathematically simple and quite powerful (in the context of a sufficiently rich specification formalism), this view of the development process may be made more practical by taking into account the fact that successive specifications in the above chain will tend to incorporate more and more details arising from successive design decisions. Some parts thereby become fully determined, and remain fixed until the development process is complete:



It seems only natural to separate the finished parts from the specification of what remains to be done. This gives the following picture:



where for $i = 1, \ldots, n$, the specifications SP_i now describe the part of the system that remains to be implemented, while each κ_i is a *parametrized program* [Gog84] which semantically amounts to a (possibly partial) function on algebras $\kappa_i \colon |\mathbf{Alg}(\mathit{Sig}(SP_i))| \rightharpoonup |\mathbf{Alg}(\mathit{Sig}(SP_{i-1}))|$ which we will call a *construction*. Now, given specifications SP and SP' and a construction $\kappa \colon |\mathbf{Alg}(\mathit{Sig}(SP'))| \rightharpoonup |\mathbf{Alg}(\mathit{Sig}(SP))|$, we define:

$$SP \underset{\kappa}{\rightsquigarrow} SP' \iff \mathit{Mod}(SP') \subseteq \mathit{dom}(\kappa) \text{ and } \kappa(\mathit{Mod}(SP')) \subseteq \mathit{Mod}(SP)$$

This definition captures the correctness requirements we impose on the individual refinement steps, which guarantee that given a successful development sequence:

$$SP_0 \underset{\kappa_1}{\rightsquigarrow} SP_1 \underset{\kappa_2}{\rightsquigarrow} \cdots \underset{\kappa_n}{\rightsquigarrow} SP_n = \mathit{EMPTY}$$

we obtain the algebra:

$$\kappa_1(\kappa_2(\ldots \kappa_n(\mathit{empty}) \ldots)) \in \mathit{Mod}(SP_0)$$

where *EMPTY* is the empty specification over the "empty" signature (i.e. the initial object in **AlgSig**, containing the Boolean part only) and *empty* is its unique standard realization.

Even though our presentation suggests a "top-down" development process, starting from the requirements specification and proceeding towards a situation where nothing is left to be implemented, this need not be the case in general. We can instead proceed "bottom-up", starting with *EMPTY* and successively providing constructions which add in bits and pieces in an incremental fashion until an implementation of the original specification is obtained. Or we can combine the two techniques, and proceed in a "middle-out" fashion. What matters is that at the end a chain of correct refinement steps emerges which links the requirements specification with *EMPTY*.

Another point about the above presentation is that it relies on a global view of specifications and their refinement: constructions are required to work on the whole system (represented as a model of the refining specification) and produce a whole system (represented as a model of the refined specification). Good practice suggests that there should be a way to develop such complex constructions in a well-structured way. In Sect. 6 we will present a specific view of how constructions may be built from smaller pieces, and how to decompose a development task into a number of subtasks via multi-argument constructions. For now, let us concentrate on one aspect of this, and discuss how to make refinement steps "local" — that is, how to use only *part* of the system built so far to implement some remaining parts of the requirements specification, and then incorporate the result in the system as a whole.

Technically, this means that we need to look at constructions that map Σ-algebras to Σ'-algebras, but apply them to parts cut out of "larger" Σ_G-algebras, where this "cutting out" is given as the reduct with respect to a signature morphism $\gamma\colon \Sigma \to \Sigma_G$ that fits the local argument signature into its global context. W.l.o.g. we can assume that constructions are *persistent*: the argument of a construction is always fully included in its result, without modification[4] — note that this assumption holds for all constructions that can be declared and specified in CASL, see Sect. 6. In fact, we generalize this somewhat by considering arbitrary signature morphisms rather than just inclusions.

Throughout the rest of the paper, we will repeatedly refer to the signatures and morphisms in the following pushout diagram:

$$\begin{tikzcd}
\Sigma_G \arrow[r, "\iota'"] & \Sigma'_G \\
\Sigma \arrow[u, "\gamma"] \arrow[r, "\iota"'] & \Sigma' \arrow[u, "\gamma'"']
\end{tikzcd}$$

where the local construction is along the bottom of the diagram, "cutting out" its argument from a larger algebra uses the signature morphism on the left, and the resulting global construction is along the top.

Definition 3.1. *Given a signature morphism $\iota\colon \Sigma \to \Sigma'$, a* local construction along ι *is a persistent partial function $F\colon |\mathbf{Alg}(\Sigma)| \rightharpoonup |\mathbf{Alg}(\Sigma')|$ (for each $A \in dom(F)$, $F(A)|_\iota = A$). We write $Mod(\Sigma \xrightarrow{\iota} \Sigma')$ for the class of all local constructions along ι.*

Given a local construction F along $\iota\colon \Sigma \to \Sigma'$, a morphism $\gamma\colon \Sigma \to \Sigma_G$ fitting Σ into a "global" signature Σ_G, and a Σ_G-algebra $\mathcal{G} \in |\mathbf{Alg}(\Sigma_G)|$, we define the global result *$F_G(\mathcal{G})$ of applying F to $\mathcal{G}$ by reference to the pushout diagram*

[4] Otherwise we would have to explicitly indicate "sharing" between the argument and result of each construction, and explain how such sharing is preserved by the various ways of putting together constructions, as was painfully spelled out in [ST89]. If necessary, superfluous components of algebras constructed using persistent constructions can be discarded at the end using the reduct along a signature inclusion.

above, using the amalgamation property: if $\mathcal{G}|_\gamma \in dom(F)$ *then* $F_G(\mathcal{G})$ *is the unique* Σ'_G*-algebra such that* $F_G(\mathcal{G})|_{\iota'} = \mathcal{G}$ *and* $F_G(\mathcal{G})|_{\gamma'} = F(\mathcal{G}|_\gamma)$*; otherwise* $F_G(\mathcal{G})$ *is undefined.*

This determines a global construction $F_G\colon |\mathbf{Alg}(\Sigma_G)| \rightharpoonup |\mathbf{Alg}(\Sigma'_G)|$*, which is persistent along* $\iota'\colon \Sigma_G \to \Sigma'_G$*.*

This way of "lifting" a persistent function to a larger context via a "*fitting morphism*" using signature pushout and amalgamation is well established in the algebraic specification tradition, going back at least to "parametrized specifications" with free functor semantics, see [EM85].

We will not dwell here on how particular (local) constructions are defined. Free functor semantics for parametrized specifications is one way to proceed, with the persistency requirement giving rise to additional proof obligations [EM85]. Perhaps closer to ordinary programming is to give explicitly a "definitional" derived signature morphism $\delta\colon \Sigma' \to \Sigma$ that defines Σ'-components in terms of Σ-components. The induced reduct function $\_|_\delta\colon |\mathbf{Alg}(\Sigma)| \to |\mathbf{Alg}(\Sigma')|$ is a local construction along a signature morphism $\iota\colon \Sigma \to \Sigma'$ whenever $\iota;\delta = id_\Sigma$.[5]

Suppose now that a local construction F along $\iota\colon \Sigma \to \Sigma'$ comes with a "semantic" specification of its input/output properties, given as a specification SP with $Sig(SP) = \Sigma$ of the requirements on its arguments together with a specification SP' with $Sig(SP') = \Sigma'$ of the guaranteed properties of its result. Again w.l.o.g. we require that $Mod(SP')|_\iota \subseteq Mod(SP)$, as is indeed ensured for instance in CASL.

Definition 3.2. *A local construction* F *along* $\iota\colon Sig(SP) \to Sig(SP')$ *is* strictly correct w.r.t. SP and SP' *if for all models* $A \in Mod(SP)$*,* $A \in dom(F)$ *and* $F(A) \in Mod(SP')$*. We write* $Mod(SP \xrightarrow{\iota} SP')$ *for the class of all local constructions along* ι *that are strictly correct w.r.t.* SP *and* SP'*.*

The following theorem shows how such locally correct constructions can be used for global refinement steps.

Theorem 3.3. *Given a local construction* $F \in Mod(SP \xrightarrow{\iota} SP')$*, specification* SP_G *with fitting morphism* $\gamma\colon Sig(SP) \to Sig(SP_G)$*, and specification* SP'_G *with* $Sig(SP'_G) = \Sigma'_G$*,* SP_G *correctly refines* SP'_G *via the global construction* F_G *(i.e.,* $SP'_G \underset{F_G}{\rightsquigarrow} SP_G$*) provided that*

- $Mod(SP_G) \subseteq Mod(\gamma(SP))$*, and*
- $Mod(\gamma'(SP')$ ***and*** $\iota'(SP_G)) \subseteq Mod(SP'_G)$*.*

Proof. Let $\mathcal{G} \in Mod(SP_G)$. Then $\mathcal{G}|_\gamma \in Mod(SP)$, and so $\mathcal{G}|_\gamma \in dom(F)$ and $F(\mathcal{G}|_\gamma) \in Mod(SP')$. Consequently $F_G(\mathcal{G}) \in Mod(\gamma'(SP')) \cap Mod(\iota'(SP_G))$. □

Informally, this captures directly a "bottom-up" process of building implementations, whereby we start with SP_G, find a local construction $F \in Mod(SP \xrightarrow{\iota} SP')$

[5] Composition of derived signature morphisms can be defined in the evident fashion, and equality of two derived signature morphisms is understood here semantically.

with a fitting morphism γ that satisfies the first condition, and define SP'_G such that the second condition is satisfied (e.g. take $SP'_G = \gamma'(SP')$ **and** $\iota'(SP_G)$). When proceeding "top-down", we start with the global requirements specification SP'_G. To use a local construction $F \in Mod(SP \xrightarrow{\iota} SP')$, we have to decide which part of the requirements it is going to implement by providing a signature morphism $\gamma'\colon Sig(SP') \to Sig(SP'_G)$, then construct the "pushout complement" $\gamma\colon Sig(SP) \to \Sigma_G$, $\iota'\colon \Sigma_G \to Sig(SP'_G)$ for ι and γ', and finally devise a specification SP_G with $Sig(SP_G) = \Sigma_G$ such that both conditions are satisfied. Then we can proceed with SP_G as the requirements specification for the components that remain to be implemented.

4 Observational Equivalence

So far, we have made few assumptions about the formalism used for writing specifications. Intuitively, it is clear that any such formalism should admit basic specifications given as sets of axioms over some fixed signature. The usual interpretation then is to take as models for such a basic specification all the algebras that satisfy the axioms. However, in many practical examples this turns out to be overly restrictive. The point is that only a subset of the sorts in the signature of a specification are typically intended to be directly observable — the others are treated as internal, with properties of their elements made visible only via *observations* leading to the observable sorts. This calls for a relaxation of the interpretation of specifications, as advocated in numerous "observational" or "behavioural" approaches, going back at least to [GGM76,Rei81]. The starting point is that given an algebraic signature, one has to fix a set of *observable sorts*. Then, roughly, two approaches are possible:

– introduce an internal *observational indistinguishability* relation between algebra elements, and re-interpret equality in the axioms as indistinguishability,
– introduce an external *observational equivalence* on algebras, and re-interpret specifications by closing their class of models under such equivalence.

It turns out that under some acceptable technical conditions, the two approaches are closely related and coincide for most basic specifications [BHW95,BT96]. However, the former approach seems more difficult to extend to structured specifications and parametrization. Hence, we follow here the latter possibility.

Definition 4.1. *Consider a signature Σ with* observable sorts $OBS \subseteq sorts(\Sigma)$. *We always assume that $bool \in OBS$. A* correspondence *between two algebras $A, B \in |\mathbf{Alg}(\Sigma)|$, written $\rho\colon A \bowtie B$, is a relation $\rho \subseteq |A| \times |B|$ that is closed under the operations*[6] *and is the identity on $|A|_{bool} = |B|_{bool}$. It is* observational *if it is bijective on observable sorts.*

Two algebras $A, B \in |\mathbf{Alg}(\Sigma)|$ are observationally equivalent*, written $A \equiv_{OBS} B$, if there exists an observational correspondence between them.*

[6] That is, for $f\colon s_1 \times \ldots \times s_n \to s$, $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$ and $b_1 \in |B|_{s_1}, \ldots, b_n \in |B|_{s_n}$, if $(a_1, b_1) \in \rho_{s_1}, \ldots, (a_n, b_n) \in \rho_{s_n}$ then $(f_A(a_1, \ldots, a_n), f_B(b_1, \ldots, b_n)) \in \rho_s$.

This formulation is due to [Sch87] (cf. "simulations" in [Mil71] and "weak homomorphisms" in [Gin68]) and is equivalent to other standard ways of defining observational equivalence between algebras, where a special role is played by *observable equalities*, i.e., equalities between terms of observable sorts.

It is easy to check that identities are correspondences and the class of correspondences is closed under composition and reducts w.r.t. signature morphisms.

Correspondences may in fact be identified with certain spans of homomorphisms: a correspondence $\rho{:}\, A \bowtie B$ is a span $(h_A{:}\, C \to A, h_B{:}\, C \to B)$ where, for each sort s distinct from *bool*, $|C|_s$ is a subset of the Cartesian product $|A|_s \times |B|_s$, $|C|_{bool} = |A|_{bool} = |B|_{bool}$, the homomorphisms are the projections for all sorts $s \neq bool$, and the identity on the carrier of the sort *bool*. Such a span is observational if the homomorphisms are bijective on observable sorts. This directly implies that the reduct of a correspondence along a signature morphism is a correspondence. More interestingly, for observational correspondences this extends to derived signature morphisms with observable conditions.

Consider a signature Σ with observable sorts $OBS \subseteq sorts(\Sigma)$. A conditional term $((\phi_i, t_i)_{i \geq 0}, t)$ is *OBS-admissible* if for all $i \geq 0$, ϕ_i are quantifier-free formulae with observable equalities only. A derived signature morphism $\delta{:}\, \Sigma' \to \Sigma$ is *OBS*-admissible if it maps Σ'-operations to *OBS*-admissible terms.

Lemma 4.2. *Let $\delta{:}\, \Sigma' \to \Sigma$ be an OBS-admissible derived signature morphism, A and B be two Σ-algebras, and $\rho{:}\, A \bowtie B$ be an observational correspondence. Then $\rho|_\delta{:}\, A|_\delta \bowtie B|_\delta$ is a correspondence as well. Moreover, it is observational for any set $OBS' \subseteq sorts(\Sigma')$ of observable sorts such that $\delta(OBS') \subseteq OBS$.* □

The view of correspondences as spans of homomorphisms also leads to an easy extension to correspondences of the amalgamation property given in Lemma 2.1 for algebras and homomorphisms.

Observational equivalence between algebras can be characterized in terms of the alternative approach based on internal indistinguishability. Consider a signature Σ with observable sorts $OBS \subseteq sorts(\Sigma)$ (with $bool \in OBS$) and an algebra $A \in |\mathbf{Alg}(\Sigma)|$. Let $\langle A \rangle_{OBS}$ be the subalgebra of A generated by the carriers of observable sorts. *Observational indistinguishability* on A, denoted by $\approx_{OBS}$, is the largest congruence on $\langle A \rangle_{OBS}$ that is the identity on observable sorts. The *observational quotient* of A, written $A/{\approx_{OBS}}$, is the quotient of $\langle A \rangle_{OBS}$ by $\approx_{OBS}$.

Theorem 4.3. *Consider a signature Σ with observable sorts $OBS \subseteq sorts(\Sigma)$. Two Σ-algebras are observationally equivalent if and only if their observational quotients are isomorphic.* □

So far we have considered observational equivalence w.r.t. a rather arbitrary set of observable sorts. In practice, however, for any development framework (and programming language), the set of types directly observable to the user is fixed and given in advance — for the framework at hand, the right choice seems to be to take the sort *bool* as the only observable sort. Note that choosing *bool* as the only observable sort is not a restriction, since one can always treat another sort as observable by introducing an "equality predicate" on it. Moreover, this choice will not prevent us from manipulating an explicit set of observable sorts (always keeping *bool* among them though) when considering "local" signatures for verification purposes.

We will consider observational equivalence of "global" models with respect to the single observable sort *bool* — we write $\equiv_{\{bool\}}$ simply as $\equiv$. For any "global" specification SP_G with $Sig(SP_G) = \Sigma_G$, we define its *observational interpretation* by abstracting from the standard interpretation as follows:

$$Abs_{\equiv}(SP_G) = \{\mathcal{G} \in |\mathbf{Alg}(\Sigma_G)| \mid \mathcal{G} \equiv \mathcal{H} \text{ for some } \mathcal{H} \in Mod(SP_G)\}.$$

5 Observational Refinement Steps

The most obvious way to re-interpret correctness of refinement steps $SP' \underset{\kappa}{\rightsquigarrow} SP$ to take advantage of the observational interpretation of specifications indicated in the previous section is to relax the earlier definition by requiring $Abs_{\equiv}(SP) \subseteq dom(\kappa)$ and $\kappa(Abs_{\equiv}(SP)) \subseteq Abs_{\equiv}(SP')$. This works, but misses a crucial point: when using a realization of a specification, we should be able to pretend that it satisfies the specification literally, even if when actually implementing it we are permitted to supply an algebra that is correct only up to observational equivalence. This leads to a new notion of *observational refinement*: given specifications SP and SP' and a construction $\kappa\colon |\mathbf{Alg}(Sig(SP'))| \rightharpoonup |\mathbf{Alg}(Sig(SP))|$, we define:

$$SP \overset{\equiv}{\underset{\kappa}{\rightsquigarrow}} SP' \iff Mod(SP') \subseteq dom(\kappa) \text{ and } \kappa(Mod(SP')) \subseteq Abs_{\equiv}(SP)$$

This relaxation has a price: observational refinements do not automatically compose! The crucial insight to resolve this problem comes from [Sch87], who noticed that well-behaved constructions satisfy the following *stability* property.

Definition 5.1. *A construction* $\kappa\colon |\mathbf{Alg}(\Sigma)| \rightharpoonup |\mathbf{Alg}(\Sigma')|$ *is* stable *if it preserves observational equivalence of algebras, i.e., for any algebras* $A, B \in |\mathbf{Alg}(\Sigma)|$ *such that* $A \equiv B$*, if* $A \in dom(\kappa)$ *then* $B \in dom(\kappa)$ *and* $\kappa(A) \equiv \kappa(B)$*.*

Now, if all the constructions involved are stable then from a successful chain of observational refinements

$$SP_0 \overset{\equiv}{\underset{\kappa_1}{\rightsquigarrow}} SP_1 \overset{\equiv}{\underset{\kappa_2}{\rightsquigarrow}} \ldots \overset{\equiv}{\underset{\kappa_n}{\rightsquigarrow}} SP_n = EMPTY$$

we obtain:

$$\kappa_1(\kappa_2(\ldots\kappa_n(empty)\ldots)) \in Abs_{\equiv}(SP_0).$$

The rest of this section is devoted to an analysis of conditions that ensure stability of constructions and observational correctness of refinement steps when the constructions arise via the use of local constructions, as in Sect. 3. The problem is that we want to restrict attention to conditions that are essentially local to the local constructions involved, rather than conditions that refer to all the possible global contexts in which such a construction can be used.

Let us start with the stability property.

Definition 5.2. *A local construction F along $\iota\colon \Sigma \to \Sigma'$ is* locally stable *if for any Σ-algebras $A, B \in |\mathbf{Alg}(\Sigma)|$ and correspondence $\rho\colon A \bowtie B$, $A \in dom(F)$ if and only if $B \in dom(F)$ and moreover, if this is the case then there exists a correspondence $\rho'\colon F(A) \bowtie F(B)$ that extends ρ (i.e., $\rho'\big|_{\iota} = \rho$).*

Proposition 5.3. *The composition of locally stable constructions is a locally stable construction.* □

Lemma 5.4. *If F is a locally stable construction along $\iota\colon \Sigma \to \Sigma'$ then for any signature Σ_G and fitting morphism $\gamma\colon \Sigma \to \Sigma_G$, the induced global construction $F_G\colon |\mathbf{Alg}(\Sigma_G)| \rightharpoonup |\mathbf{Alg}(\Sigma'_G)|$ along $\iota'\colon \Sigma_G \to \Sigma'_G$ is locally stable as well.*

Proof. Consider a correspondence $\rho_G\colon \mathcal{G} \bowtie \mathcal{H}$ between algebras $\mathcal{G}, \mathcal{H} \in |\mathbf{Alg}(\Sigma_G)|$. Its reduct is a correspondence $\rho_G\big|_{\gamma}\colon \mathcal{G}\big|_{\gamma} \bowtie \mathcal{H}\big|_{\gamma}$, so $\mathcal{G}\big|_{\gamma} \in dom(F)$ iff $\mathcal{H}\big|_{\gamma} \in dom(F)$, and consequently $\mathcal{G} \in dom(F_G)$ iff $\mathcal{H} \in dom(F_G)$. Suppose $\mathcal{G}\big|_{\gamma} \in dom(F)$. Then there exists a correspondence $\rho'\colon F(\mathcal{G}\big|_{\gamma}) \bowtie F(\mathcal{H}\big|_{\gamma})$ with $\rho'\big|_{\iota} = \rho_G\big|_{\gamma}$. Amalgamation of ρ_G and ρ' yields a correspondence $\rho'_G\colon F_G(\mathcal{G}) \bowtie F_G(\mathcal{H})$ such that $\rho'_G\big|_{\iota'} = \rho_G$. □

Corollary 5.5. *If F is a locally stable construction along $\iota\colon \Sigma \to \Sigma'$ then for any signature Σ_G and fitting morphism $\gamma\colon \Sigma \to \Sigma_G$, the induced global construction $F_G\colon |\mathbf{Alg}(\Sigma_G)| \rightharpoonup |\mathbf{Alg}(\Sigma'_G)|$ along $\iota'\colon \Sigma_G \to \Sigma'_G$ is stable.*

Proof. Let $\mathcal{G}, \mathcal{H} \in |\mathbf{Alg}(\Sigma_G)|$ be such that $\mathcal{G} \equiv \mathcal{H}$. Then there is a correspondence $\rho_G\colon \mathcal{G} \bowtie \mathcal{H}$. By Lemma 5.4, if $\mathcal{G} \in dom(F_G)$ then $\mathcal{H} \in dom(F_G)$ and there is a correspondence $\rho'_G\colon F_G(\mathcal{G}) \bowtie F_G(\mathcal{H})$, which proves $F_G(\mathcal{G}) \equiv F_G(\mathcal{H})$. □

This establishes a sufficient local condition which ensures that a local construction induces a stable global construction in every possible context of use.

The following is a corollary of Lemma 4.2.

Corollary 5.6. *Let $\delta\colon \Sigma' \to \Sigma$ be a $\{bool\}$-admissible derived signature morphism and $\iota\colon \Sigma \to \Sigma'$ be a signature morphism such that $\iota;\delta = id_{\Sigma}$. Then the reduct $F = \_\big|_{\delta}\colon Mod(\Sigma) \to Mod(\Sigma')$) is a local construction that is locally stable.* □

The above corollary supports the point put forward in [Sch87] that stable constructions are those that respect modularity in the software construction process. That is, such constructions can use the components provided by their imported parameters, but they cannot take advantage of their particular internal properties. This is the point of the requirement that δ be $\{bool\}$-admissible: any branching in the code must be governed by directly observable properties. This turns (local) stability into a directive for language design, rather than a condition to be checked on a case-by-case basis: in a language with good modularization facilities, all constructions that one can code should be locally stable.

Let us turn now to the issue of correctness w.r.t. given specifications.

Definition 5.7. *A local construction F along $\iota\colon Sig(SP) \to Sig(SP')$ is* observationally correct w.r.t. SP and SP' *if for every model $A \in Mod(SP)$, $A \in dom(F)$ and there exists a model $A' \in Mod(SP')$ and correspondence $\rho'\colon A' \bowtie F(A)$ such that $\rho'\big|_\iota$ is the identity.*

We write $Mod_{lc}(SP \xrightarrow{\iota} SP')$ for the class of all locally stable constructions along ι that are observationally correct w.r.t. SP and SP'.

The requirement above implies that $A'\big|_\iota = A$ and $A' \equiv_{\iota(sorts(\Sigma))} F(A)$, which in turn is in general stronger than $F(A) \in Abs_{\equiv_{\iota(sorts(\Sigma))}}(SP')$. It follows that if $F \in Mod_{lc}(SP \xrightarrow{\iota} SP')$ then there is some $F' \in Mod(SP \xrightarrow{\iota} SP')$ such that $dom(F') = dom(F)$ and for each $A \in Mod(SP)$, $F'(A) \equiv_{\iota(sorts(\Sigma))} F(A)$. However, in general $Mod(SP \xrightarrow{\iota} SP') \not\subseteq Mod_{lc}(SP \xrightarrow{\iota} SP')$, as strictly correct local constructions need not be stable. Moreover, it may happen that there are no stable observationally correct constructions, even if there are strictly correct ones: that is, we may have $Mod_{lc}(SP \xrightarrow{\iota} SP') = \emptyset$ even if $Mod(SP \xrightarrow{\iota} SP') \neq \emptyset$. This was perhaps first pointed out in [Ber87], in a different framework.

Counterexample 5.8. Let SP_1 include a non-observable sort s with two constants $a, b\colon s$, and let SP_2 enrich SP_1 by an observable sort o, two constants $c, d\colon o$ and axiom $c \neq d \iff a = b$. Then $Mod(SP_1 \to SP_2)$ is non-empty, with any construction in it mapping models satisfying $a = b$ to those that satisfy $c \neq d$, and models satisfying $a \neq b$ to those that satisfy $c = d$. But none of these constructions is stable!

Lemma 5.9. *Consider a local construction F along $\iota\colon Sig(SP) \to Sig(SP')$ that is observationally correct w.r.t. SP and SP'. Then, for every global signature Σ_G and fitting morphism $\gamma\colon Sig(SP) \to \Sigma_G$, for every $\mathcal{G} \in Mod(\gamma(SP))$ we have $\mathcal{G} \in dom(F_G)$ and there is some $\mathcal{G}' \in Mod(\gamma'(SP'))$ such that $\mathcal{G}'\big|_{\iota'} = \mathcal{G}$ and $\mathcal{G}' \equiv F_G(\mathcal{G})$.*

Proof. We have $\mathcal{G}\big|_\gamma \in Mod(SP)$, and so $\mathcal{G}\big|_\gamma \in dom(F)$ and there exist $A' \in Mod(SP')$ and a correspondence $\rho'\colon A' \bowtie F(\mathcal{G}\big|_\gamma)$ with identity reduct $\rho'\big|_\iota$. Consider the unique Σ'_G-algebra $\mathcal{G}'$ such that $\mathcal{G}'\big|_{\iota'} = \mathcal{G}$ and $\mathcal{G}'\big|_{\gamma'} = A'$. Then the identity $id_{\mathcal{G}}\colon \mathcal{G} \bowtie \mathcal{G}$ and $\rho'\colon A' \bowtie F(\mathcal{G}\big|_\gamma)$ amalgamate to a correspondence $\rho'_G\colon \mathcal{G}' \bowtie F_G(\mathcal{G})$, which proves that $F_G(\mathcal{G}) \equiv \mathcal{G}' \in Mod(\gamma'(SP'))$. □

If $F \in Mod_{lc}(SP \xrightarrow{\iota} SP')$ and $\gamma\colon Sig(SP) \to \Sigma_G$, then by Lemma 5.9 we obtain $\gamma'(SP') \overset{\equiv}{\underset{F_G}{\leadsto}} \gamma(SP)$, and since F_G is stable by Cor. 5.5, we can use this in the observational development process. Given two "global" specifications SP_G with $Sig(SP_G) = \Sigma_G$ and SP'_G with $Sig(SP'_G) = \Sigma'_G$, we have $SP'_G \overset{\equiv}{\underset{F_G}{\leadsto}} SP_G$ whenever $Mod(SP_G) \subseteq Abs_{\equiv}(\gamma(SP))$ and $Mod(\gamma'(SP')) \subseteq Abs_{\equiv}(SP'_G)$. But while the former requirement is quite acceptable, the latter is in fact impossible to achieve in practice since it implicitly requires that all the global requirements must follow (up to observational equivalence) from the result specification for the local construction. More practical requirements are obtained by generalizing Thm. 3.3 to the observational setting:

Theorem 5.10. *Given a local construction $F \in \mathit{Mod}_{lc}(SP \xrightarrow{\iota} SP')$, specification SP_G with fitting morphism $\gamma\colon \mathit{Sig}(SP) \to \mathit{Sig}(SP_G)$, and specification SP'_G with $\mathit{Sig}(SP'_G) = \Sigma'_G$, if*

(i) $\mathit{Mod}(SP_G) \subseteq \mathit{Abs}_{\equiv}(SP_G\ \textbf{\textit{and}}\ \gamma(SP))$*, and*
(ii) $\mathit{Mod}(\gamma'(SP')\ \textbf{\textit{and}}\ \iota'(SP_G)) \subseteq \mathit{Abs}_{\equiv}(SP'_G)$

then for every $\mathcal{G} \in \mathit{Mod}(SP_G)$, we have $\mathcal{G} \in \mathit{dom}(F_G)$ and $F_G(\mathcal{G}) \in \mathit{Abs}_{\equiv}(SP'_G)$. Consequently:

$$SP'_G \overset{\equiv}{\underset{F_G}{\rightsquigarrow}} SP_G.$$

Proof. Let $\mathcal{G} \in \mathit{Mod}(SP_G)$. Then $\mathcal{G} \equiv \mathcal{H}$ for some $\mathcal{H} \in \mathit{Mod}(SP_G) \cap \mathit{Mod}(\gamma(SP))$ by (i). By Lemma 5.9, $F_G(\mathcal{H}) \equiv \mathcal{H}'$ for some $\mathcal{H}' \in \mathit{Mod}(\gamma'(SP'))$ with $\mathcal{H}'|_{\iota'} = \mathcal{H} \in \mathit{Mod}(SP_G)$. Hence $\mathcal{H}' \in \mathit{Abs}_{\equiv}(SP'_G)$ by (ii). By stability of F_G (Cor. 5.5), $\mathcal{G} \in \mathit{dom}(F_G)$ and $F_G(\mathcal{G}) \equiv F_G(\mathcal{H}) \equiv \mathcal{H}'$, and so $F_G(\mathcal{G}) \in \mathit{Abs}_{\equiv}(SP'_G)$. □

Requirement (i) is perhaps the only surprising assumption in this theorem. Note though that it straightforwardly follows from the inclusion of strict model classes $\mathit{Mod}(SP_G) \subseteq \mathit{Mod}(\gamma(SP))$ (or equivalently, $\mathit{Mod}(SP_G)|_{\gamma} \subseteq \mathit{Mod}(SP)$), which is often easiest to verify. However, (i) is strictly stronger in general than the perhaps more expected $\mathit{Mod}(SP_G) \subseteq \mathit{Abs}_{\equiv}(\gamma(SP))$. This weaker condition turns out to be sufficient (and in fact, equivalent to (i)) if we additionally assume that the two specifications involved are *behaviourally consistent* [BHW95], that is, closed under observational quotients. When this is not the case, then the use of this weaker condition must be paid for by a stronger version of (ii):

$$\mathit{Abs}_{\equiv}(\gamma'(SP')) \cap \mathit{Mod}(\iota'(SP_G)) \subseteq \mathit{Abs}_{\equiv}(SP'_G),$$

which seems even less convenient to use than (i). Overall, we need a way to pass information on the global context from SP_G to SP'_G independently from the observational interpretation of the local construction and its correctness, and this must result in some inconvenience of verification on either the parameter or the result side.

6 Architectural Specifications

Using local constructions for global implementations of specifications, we have moved only one step away from the monolithic global view of specifications and constructions used to implement them. The notion of *architectural specification* [BST02] as introduced for CASL takes us much further. An architectural specification *prescribes* a decomposition of the task of implementing a requirements specification into a number of subtasks to implement specifications of "modular components" (called *units*) of the system under development. The units may be parametrized, and then we can identify them with local constructions; non-parametrized units are modeled as algebras. Another essential part of an architectural specification is a prescription of how the units, once developed, are to be put together using a few simple operators. One of these is an application of a parametrized unit which corresponds exactly to the lifting of a local construction to a larger context studied above. Thus, an architectural specification may be thought of as a definition of a complex construction to be used in a development process to implement a requirements specification by a number of specifications (of non-parametrized units), where the construction uses a number of specified local constructions to be developed as well.

For the sake of readability, we will discuss here only a very simple version of CASL architectural specifications, with a limited (but representative) number of constructs, shaped after a somewhat less simplified fragment used in [SMT$^+$01]; a generalization of the work presented here to full architectural specifications of CASL would be tedious but rather straightforward, except perhaps for the "unguarded import" mechanism, see [Hof01]. Our version of architectural specifications is defined as follows.

Architectural specifications: $ASP ::= \textbf{arch spec } Dcl^* \textbf{ result } T$
An architectural specification consists of a list of unit declarations followed by a unit result term.

Unit declarations: $Dcl ::= U\colon SP \mid U\colon SP_1 \xrightarrow{\iota} SP_2$
A unit declaration introduces a unit name with its type, which is either a specification or a specification of a parametrized unit, determined by a specification of its parameter and its result, which extends the parameter via a signature morphism ι.

Unit terms: $T ::= U \mid U[T \textbf{ fit } \sigma] \mid T_1 \textbf{ and } T_2$
A unit term is either a (non-parametrized) unit name, or a unit application with an argument that fits via a signature morphism σ, or an amalgamation of units.

Following the semantics of full CASL [CoFI02], see also [SMT$^+$01], we give the semantics of this CASL fragment in two stages: first we give its *extended static semantics* and then the *strict model semantics*.

An *extended static context* $\mathcal{C}_{st} = (P_{st}, \mathcal{B}_{st}, \Sigma_G)$ in which CASL phrases are elaborated, consists of a static context for parametrized units P_{st} mapping parametrized unit names to signature morphisms (from the parameter to the result signatures), a global context signature Σ_G, and an extended static context for non-parametrized units $\mathcal{B}_{st}$ mapping non-parametrized unit names to morphisms from the unit signature to Σ_G. From any such extended static context we can extract a *static context* $ctx(\mathcal{C}_{st}) = (P_{st}, B_{st})$ by forgetting the global context signature and restricting the information about non-parametrized units to their signatures only (sources of the morphisms given by $\mathcal{B}_{st}$).

Given a morphism $\theta\colon \Sigma_G \to \Sigma'_G$, we write $\mathcal{B}_{st};\theta$ for the extended static context $\mathcal{B}'_{st}$ with the same domain as $\mathcal{B}_{st}$ and such that for any name $U \in dom(\mathcal{B}_{st})$, $\mathcal{B}'_{st}(U) = \mathcal{B}_{st}(U);\theta$. Then the extended static context $\mathcal{C}_{st};\theta$ is $(P_{st}, (\mathcal{B}_{st};\theta), \Sigma'_G)$. $\mathcal{C}^{\emptyset}_{st}$ stands for the "empty" extended static context that consists of the empty parametrized and non-parametrized unit contexts and the initial signature.

Figure 1 gives rules to derive semantic judgments of the following forms:

$$\frac{\vdash Dcl^{*} \rhd \mathcal{C}_{st} \qquad \mathcal{C}_{st} \vdash T \rhd (\theta{:}\,\Sigma_G \to \Sigma'_G, i{:}\,\Sigma \to \Sigma'_G)}{\vdash \textbf{arch spec } Dcl^{*} \textbf{ result } T \rhd (ctx(\mathcal{C}_{st}), \Sigma)}$$

$$\frac{\begin{array}{c}\mathcal{C}^{\emptyset}_{st} \vdash Dcl_1 \rhd (\mathcal{C}_{st})_1 \\ \cdots \\ (\mathcal{C}_{st})_{n-1} \vdash Dcl_n \rhd (\mathcal{C}_{st})_n\end{array}}{\vdash Dcl_1 \ldots Dcl_n \rhd (\mathcal{C}_{st})_n}$$

$$\frac{\begin{array}{c}U \notin (dom(P_{st}) \cup dom(\mathcal{B}_{st})) \\ \Sigma'_G \text{ is the coproduct of } \Sigma_G \text{ and } Sig(SP) \\ \text{with injections } \theta{:}\,\Sigma_G \to \Sigma'_G, i{:}\,Sig(SP) \to \Sigma'_G\end{array}}{(P_{st}, \mathcal{B}_{st}, \Sigma_G) \vdash U{:}\,SP \rhd (P_{st}, (\mathcal{B}_{st};\theta) + \{U \mapsto i\}, \Sigma'_G)}$$

$$\frac{\begin{array}{c}\iota : Sig(SP_1) \to Sig(SP_2) \\ U \notin (dom(P_{st}) \cup dom(\mathcal{B}_{st}))\end{array}}{(P_{st}, \mathcal{B}_{st}, \Sigma_G) \vdash U{:}\,SP_1 \xrightarrow{\iota} SP_2 \rhd (P_{st} + \{U \mapsto \iota\}, \mathcal{B}_{st}, \Sigma_G)}$$

$$\frac{U \in dom(\mathcal{B}_{st})}{(P_{st}, \mathcal{B}_{st}, \Sigma_G) \vdash U \rhd (id_{\Sigma_G}, \mathcal{B}_{st}(U))}$$

$$\frac{\begin{array}{c}(P_{st}, \mathcal{B}_{st}, \Sigma_G) \vdash T \rhd (\theta{:}\,\Sigma_G \to \Sigma'_G, i{:}\,\Sigma_T \to \Sigma'_G) \\ P_{st}(U) = \iota{:}\,\Sigma \to \Sigma' \qquad \sigma{:}\,\Sigma \to \Sigma_T \\ (\iota'{:}\,\Sigma_T \to \Sigma'_T, \sigma'{:}\,\Sigma' \to \Sigma'_T) \text{ is the pushout of } (\sigma, \iota) \\ (\iota''{:}\,\Sigma'_G \to \Sigma''_G, i'{:}\,\Sigma'_T \to \Sigma''_G) \text{ is the pushout of } (i, \iota')\end{array}}{(P_{st}, \mathcal{B}_{st}, \Sigma_G) \vdash U[T \textbf{ fit } \sigma] \rhd (\theta;\iota'', i'{:}\,\Sigma'_T \to \Sigma''_G)}$$

$$\frac{\begin{array}{c}(P_{st}, \mathcal{B}_{st}, \Sigma_G) \vdash T_1 \rhd (\theta_1{:}\,\Sigma_G \to \Sigma^1_G, i_1{:}\,\Sigma_1 \to \Sigma^1_G) \\ (P_{st}, \mathcal{B}_{st}, \Sigma_G) \vdash T_2 \rhd (\theta_2{:}\,\Sigma_G \to \Sigma^2_G, i_2{:}\,\Sigma_2 \to \Sigma^2_G) \\ \Sigma = \Sigma_1 \cup \Sigma_2 \text{ with inclusions } \iota_1{:}\,\Sigma_1 \to \Sigma, \iota_2{:}\,\Sigma_2 \to \Sigma \\ (\theta'_2{:}\,\Sigma^1_G \to \Sigma'_G, \theta'_1{:}\,\Sigma^2_G \to \Sigma'_G) \text{ is the pushout of } (\theta_1, \theta_2) \\ \text{there is a (unique) morphism } j{:}\,\Sigma \to \Sigma'_G \text{ such that } \iota_1;j = i_1;\theta'_2 \text{ and } \iota_2;j = i_2;\theta'_1\end{array}}{(P_{st}, \mathcal{B}_{st}, \Sigma_G) \vdash T_1 \textbf{ and } T_2 \rhd (\theta_1;\theta'_2, j)}$$

Fig. 1. Extended static semantics

- $\vdash ASP \rhd (C_{st}, \Sigma)$: the architectural specification ASP yields a static context describing the units declared and the signature of the result unit;
- $\mathcal{C}_{st} \vdash Dcl \rhd \mathcal{C}'_{st}$: the unit declaration Dcl in the extended static context $\mathcal{C}_{st}$ yields a new extended static context $\mathcal{C}'_{st}$; similarly for a sequence of unit declarations;
- $(P_{st}, \mathcal{B}_{st}, \Sigma_G) \vdash T \rhd (\theta{:}\,\Sigma_G \to \Sigma'_G, i{:}\,\Sigma \to \Sigma'_G)$: the unit term T in the extended static context $(P_{st}, \mathcal{B}_{st}, \Sigma_G)$ extends the global context signature Σ_G to a new one Σ'_G along a signature morphism $\theta{:}\,\Sigma_G \to \Sigma'_G$ and yields the signature Σ of the unit built, indicating how the unit resides in the global context using the morphism $i{:}\,\Sigma \to \Sigma'_G$.

In the strict model semantics we work with *contexts* $\mathcal{C}$ that are sets of *unit environments* E. Environments map unit names to either local constructions (for parametrized units) or to individual algebras (for non-parametrized units). *Unit evaluators* UEv map unit environments to algebras.

Given an extended static unit context $\mathcal{C}_{st} = (P_{st}, \mathcal{B}_{st}, \Sigma_G)$, an environment E *fits* $\mathcal{C}_{st}$ if

- for each $U \in dom(P_{st})$, $E(U)$ is a local construction along $P_{st}(U)$, and
- there exists an algebra $\mathcal{G} \in |\mathbf{Alg}(\Sigma_G)|$ such that for each $U \in dom(\mathcal{B}_{st})$, $E(U) = \mathcal{G}|_{\mathcal{B}_{st}(U)}$; we say then that $\mathcal{G}$ *witnesses* E.

We write $ucx(\mathcal{C}_{st})$ for the class of all environments that fit $\mathcal{C}_{st}$. $\mathcal{C}^{\emptyset} = ucx(\mathcal{C}^{\emptyset}_{st})$ is the context which constrains no unit name. Given a unit context $\mathcal{C}$, a unit name U and a class of units $\mathcal{V}$, we write $\mathcal{C} \times \{U \mapsto \mathcal{V}\}$ for $\{E + \{U \mapsto V\} \mid E \in \mathcal{C}, V \in \mathcal{V}\}$, where $E + \{U \mapsto V\}$ maps U to V and otherwise behaves like E.

$$\frac{\vdash Dcl^* \Rightarrow \mathcal{C} \qquad \mathcal{C} \vdash T \Rightarrow UEv}{\vdash \mathbf{arch\ spec}\ Dcl^*\ \mathbf{result}\ T \Rightarrow (\mathcal{C}, UEv)}$$

$$\frac{\begin{array}{c}\mathcal{C}^{\emptyset} \vdash Dcl_1 \Rightarrow \mathcal{C}_1 \\ \cdots \\ \mathcal{C}_{n-1} \vdash Dcl_n \Rightarrow \mathcal{C}_n\end{array}}{\vdash Dcl_1 \ldots Dcl_n \Rightarrow \mathcal{C}_n}$$

$$\frac{}{\mathcal{C} \vdash U\colon SP \Rightarrow \mathcal{C} \times \{U \mapsto Mod(SP)\}}$$

$$\frac{}{\mathcal{C} \vdash U\colon SP_1 \xrightarrow{\iota} SP_2 \Rightarrow \mathcal{C} \times \{U \mapsto Mod(SP_1 \xrightarrow{\iota} SP_2)\}}$$

$$\frac{}{\mathcal{C} \vdash U \Rightarrow \lambda E \in \mathcal{C} \cdot E(U)}$$

$$\frac{\begin{array}{c}\mathcal{C} \vdash T \Rightarrow UEv; \quad \text{for each } E \in \mathcal{C}, UEv(E)|_{\sigma} \in dom(E(U)) \\ UEv' = \{E \mapsto A \mid E \in \mathcal{C}, A|_{\iota'} = UEv(E), A|_{\sigma'} = E(U)(UEv(E)|_{\sigma})\}\end{array}}{\mathcal{C} \vdash U[T\ \mathbf{fit}\ \sigma] \Rightarrow UEv'}$$

$$\frac{\begin{array}{c}\mathcal{C} \vdash T_1 \Rightarrow UEv_1 \qquad \mathcal{C} \vdash T_2 \Rightarrow UEv_2 \\ \cancel{\text{for each } E \in \mathcal{C}, \text{ there is a unique } A \in |\mathbf{Alg}(\Sigma)| \text{ such that}} \\ \cancel{A|_{\iota_1} = UEv_1(E), A|_{\iota_2} = UEv_2(E)} \\ UEv = \{E \mapsto A \mid E \in \mathcal{C}, A|_{\iota_1} = UEv_1(E), A|_{\iota_2} = UEv_2(E)\}\end{array}}{\mathcal{C} \vdash T_1\ \mathbf{and}\ T_2 \Rightarrow UEv}$$

Fig. 2. Strict model semantics

Figure 2 gives rules to derive semantic judgments of the following forms:

- $\vdash ASP \Rightarrow (\mathcal{C}, UEv)$: the architectural specification ASP yields a context $\mathcal{C}$ with environments providing interpretations for the units declared and the unit evaluator that for each such environment determines the result unit;
- $\mathcal{C} \vdash Dcl \Rightarrow \mathcal{C}'$: the unit declaration Dcl in the context $\mathcal{C}_{st}$ yields a new context $\mathcal{C}'_{st}$; similarly for a sequence of unit declarations;
- $\mathcal{C} \vdash T \Rightarrow UEv$: the unit term T in the context $\mathcal{C}$ yields a unit evaluator UEv that when given an environment (in $\mathcal{C}$) yields the unit resulting from the evaluation of T in this environment.

The rules rely on a successful run of the extended static semantics; this allows us to use the static concepts and notations introduced there. Moreover, the following invariants link the extended static semantics and model semantics and are maintained by the rules:

- $\vdash ASP \rhd\!\!\!\!\rhd (C_{st}, \Sigma)$ and $\vdash ASP \Rightarrow (\mathcal{C}, UEv)$: there is an extended static context $\mathcal{C}_{st}$ such that $ctx(\mathcal{C}_{st}) = C_{st}$ and $\mathcal{C} \subseteq ucx(\mathcal{C}_{st})$, $\mathcal{C} \subseteq dom(UEv)$, and for each $E \in \mathcal{C}$, $UEv(E) \in |\mathbf{Alg}(\Sigma)|$;
- $\mathcal{C}_{st} \vdash Dcl \rhd\!\!\!\!\rhd \mathcal{C}'_{st}$ and $\mathcal{C} \vdash Dcl \Rightarrow \mathcal{C}'$: if $\mathcal{C} \subseteq ucx(\mathcal{C}_{st})$ then $\mathcal{C}' \subseteq ucx(\mathcal{C}'_{st})$; similarly for a sequence of unit declarations;
- $\mathcal{C}_{st} \vdash T \rhd\!\!\!\!\rhd (\theta\colon \Sigma_G \to \Sigma'_G, i\colon \Sigma \to \Sigma'_G)$ and $\mathcal{C} \vdash T \Rightarrow UEv$: if $\mathcal{C} \subseteq ucx(\mathcal{C}_{st})$ then for each unit environment $E \in \mathcal{C}$ and each algebra $\mathcal{G}$ that witnesses E, there exists a model $\mathcal{G}' \in |\mathbf{Alg}(\Sigma'_G)|$ such that $\mathcal{G}'\big|_\theta = \mathcal{G}$ and $UEv(E) = \mathcal{G}'\big|_i$.

The invariants ensure that the crossed out premise of the unit amalgamation rule of the model semantics follows from the premises of the corresponding rule of the extended static semantics.

7 Observational Interpretation of Architectural Specifications

In this section we discuss an observational interpretation of the architectural specifications introduced in Sect. 6. The extended static semantics remains unchanged — observational interpretation of specifications does not affect their static properties. We provide, however, a new *observational model semantics*, with judgments written as $\_\_ \vdash \_\_ \overset{\equiv}{\Longrightarrow} \_\_$.

To begin with, the effect of unit declarations has to be modified, taking into account observational interpretation of the specifications involved, as discussed in Sects. 4 and 5. The new rules follow in Fig. 3.

No other modifications are necessary: all the remaining rules are the same for observational and strict model semantics. This should not be surprising: the interpretation of the constructs on unit terms remains the same, all we change is the interpretation of unit specifications.

Moreover, the observational model semantics can be linked to the extended static semantics in exactly the same way as in the case of the strict model semantics: the invariants stated in Sect. 6 carry over without change.

$$\frac{}{\mathcal{C} \vdash U{:}\, SP \overset{\equiv}{\Longrightarrow} \mathcal{C} \times \{U \mapsto Abs_{\equiv}(SP)\}}$$

$$\frac{}{\mathcal{C} \vdash U{:}\, SP_1 \xrightarrow{\iota} SP_2 \overset{\equiv}{\Longrightarrow} \mathcal{C} \times \{U \mapsto Mod_{lc}(SP_1 \xrightarrow{\iota} SP_2)\}}$$

Fig. 3. Observational model semantics — the modified rules

This does not mean that the two semantics quite coincide: there is one point in the model semantics where verification is performed, and the resulting verification conditions for strict and observational model semantics differ. Namely, in the rule for parametrized unit application, the premise

$$\text{for each } E \in \mathcal{C},\ UEv(E)\big|_{\sigma} \in dom(E(U))$$

checks whether what we can conclude about the argument ensures that it is indeed in the domain of the parametrized unit. Suppose the corresponding unit declaration was $U{:}\, SP \xrightarrow{\iota} SP'$. Then in the strict model semantics this requirement reduces to

$$\text{for each } E \in \mathcal{C},\ UEv(E)\big|_{\sigma} \in Mod(SP).$$

Now, in the observational model semantics, this is in fact replaced by a more permissive condition:

$$\text{for each } E \in \mathcal{C},\ UEv(E)\big|_{\sigma} \in Abs_{\equiv}(SP).$$

Of course, the situation is complicated by the fact that the contexts $\mathcal{C}$ from which environments are taken are different in the two semantics. In the simplest case, where the argument T is simply given as a unit name previously declared with a specification SP_T, for the strict model semantics the above verification condition is

$$Mod(SP_T) \subseteq Mod(SP)$$

while for the observational model semantics we get, as expected,

$$Mod(SP_T) \subseteq Abs_{\equiv}(SP).$$

In particular, it follows that there are statically correct architectural specifications ASP (i.e., $\vdash ASP \rhd\!\!\!\rhd (\mathcal{C}_{st}, \Sigma)$ for some extended static context $\mathcal{C}_{st}$ and signature Σ) that are observationally correct (i.e., $\vdash ASP \overset{\equiv}{\Longrightarrow} (\mathcal{C}_{obs}, UEv_{obs})$ for some unit context $\mathcal{C}_{obs}$ and evaluator UEv_{obs}) but *are not* strictly correct (i.e., for *no* unit context $\mathcal{C}$ and evaluator UEv can we derive $\vdash ASP \Rightarrow (\mathcal{C}, UEv)$).

A complete study of verification conditions for architectural specifications is beyond the scope of this paper; we refer to [Hof01] for work in this direction, which still has to be combined with the observational interpretation as given by the semantics here and presented in a simpler setting in Sect. 5. In the rest of this paper we will concentrate on some aspects of the relationship between the strict and observational model semantics and on stability of unit constructions as introduced in Sect. 6.

Our first aim is to show that the constructions that can be defined by architectural specifications are (locally) stable. To state this precisely, we need some more notation and terminology, as the constructions are captured here by unit evaluators operating on environments rather than on individual units.

Local constructions F_1, F_2 along $\iota\colon \Sigma \to \Sigma'$ are *observationally equivalent*, written $F_1 \equiv F_2$, if $dom(F_1) = dom(F_2)$ and for each $A \in dom(F_1)$ there exists a correspondence $\rho\colon F_1(A) \bowtie F_2(A)$ such that its reduct $\rho|_\iota$ is the identity on A.

Proposition 7.1. *Let F_1 and F_2 be observationally equivalent local constructions along $\iota\colon \Sigma \to \Sigma'$. Then if F_1 is locally stable then so is F_2.* □

Environments E_1, E_2 are *observationally equivalent*, written $E_1 \equiv E_2$, if $dom(E_1) = dom(E_2)$ and for each $U \in dom(E_1)$, $E_1(U) \equiv E_2(U)$.

A unit environment is *stable* if all the parametrized units it contains are locally stable. By Prop. 7.1, the class of stable environments is closed under observational equivalence. Given an extended static context $\mathcal{C}_{st}$, we write $ucx_{obs}(\mathcal{C}_{st})$ for the class of those unit environments in $ucx(\mathcal{C}_{st})$ that are stable. Then, given a unit context $\mathcal{C}$, we write $Abs_{\equiv}(\mathcal{C})$ for the class of all stable unit environments equivalent to a unit environment in $\mathcal{C}$; clearly, if $\mathcal{C} \subseteq ucx(\mathcal{C}_{st})$ for some static context $\mathcal{C}_{st}$ then $Abs_{\equiv}(\mathcal{C}) \subseteq ucx_{obs}(\mathcal{C}_{st})$.

Back to the stability of the constructions defined by architectural specifications: we want to show that if $\vdash ASP \rhd\!\!\!\rhd (\mathcal{C}_{st}, \Sigma)$ and $\vdash ASP \overset{\equiv}{\Longrightarrow} (\mathcal{C}_{obs}, UEv_{obs})$ then the unit evaluator UEv_{obs} is stable, i.e., maps observationally equivalent environments to observationally equivalent algebras. Unfortunately, this cannot be proved by a simple induction on the structure of the unit terms involved. The trouble is with amalgamation, since in general amalgamation is not stable — informally, joining the signatures of two algebras may introduce new observations for either or both of them.

Counterexample 7.2. Let Σ_1 and Σ_2 be signatures containing the Boolean part and a sort s (the same in both signatures). Moreover, let Σ_1 contain constants $a, b\colon s$; and let Σ_2 contain a function $f\colon s \to bool$. Since in either of the signatures there are no observations for the non-observable sort s, all algebras in $\mathbf{Alg}(\Sigma_1)$ are observationally equivalent, and similarly for algebras in $\mathbf{Alg}(\Sigma_2)$. However, observational equivalence between $(\Sigma_1 \cup \Sigma_2)$-algebras is non-trivial; for instance, algebras with $f(a) = f(b)$ are not equivalent to those where $f(a) \neq f(b)$. Consequently, given an algebra $A \in |\mathbf{Alg}(\Sigma_1)|$ with $a_A \neq b_A$, it is easy to indicate algebras $B, B' \in |\mathbf{Alg}(\Sigma_2)|$, with the same carrier of sort s as A and such that $B \equiv B'$, while the amalgamation of A with B and B', respectively, yields algebras in $\mathbf{Alg}(\Sigma_1 \cup \Sigma_2)$ that are not observationally equivalent.

However, the key point here is that amalgamation in unit terms in architectural specifications is not used as a construction on its own, but it just identifies a new part of the global context that has been constructed earlier. Since the "essential" constructions used to build new components of the global context are locally stable, such use of amalgamation can cause no harm.

To demonstrate this, we introduce a more detailed form of the semantics for unit terms, which carries more information about the construction of the global context performed on the way. Given $\mathcal{C}_{st} \vdash T \mathrel{\rhd\!\!\!>} (\theta{:}\, \Sigma_G \to \Sigma'_G, i{:}\, \Sigma \to \Sigma'_G)$, we derive judgments of the form $\mathcal{C}_{obs} \vdash T \overset{\equiv}{\Longrightarrow\!\!\!\!\Rightarrow} (\langle F_E \rangle_{E \in \mathcal{C}}, \mathit{UEv}_{obs})$, where for each $E \in \mathcal{C}_{obs}$, $F_E{:}\, |\mathbf{Alg}(\Sigma_G)| \rightharpoonup |\mathbf{Alg}(\Sigma'_G)|$ is a construction along $\theta{:}\, \Sigma_G \to \Sigma'_G$. The rules are given in Fig. 4 (*ID* in the first rule is the family of identities, appropriately indexed); as before, the rules rely on the notation introduced by the corresponding rules of the extended static semantics, see Fig. 1.

$$\frac{}{\mathcal{C} \vdash U \overset{\equiv}{\Longrightarrow\!\!\!\!\Rightarrow} (\mathit{ID}, \lambda E \in \mathcal{C} \cdot E(U))}$$

$$\frac{\begin{array}{c}
\mathcal{C} \vdash T \overset{\equiv}{\Longrightarrow\!\!\!\!\Rightarrow} (\langle F_E \rangle_{E \in \mathcal{C}}, \mathit{UEv}) \\
\text{for each } E \in \mathcal{C}, \mathit{UEv}(E)\big|_{\sigma} \in \mathit{dom}(E(U)) \\
\mathit{UEv}' = \{E \mapsto A \mid E \in \mathcal{C}, A\big|_{\iota'} = \mathit{UEv}(E), A\big|_{\sigma'} = E(U)(\mathit{UEv}(E)\big|_{\sigma})\} \\
\text{for } E \in \mathcal{C}, F'_E = \{\mathcal{G} \mapsto \mathcal{G}' \mid \mathcal{G} \in |\mathbf{Alg}(\Sigma_G)| \text{ witnesses } E, \\
\qquad\qquad \mathcal{G}'\big|_{\iota''} = \mathcal{G}, \mathcal{G}'\big|_{\sigma';i'} = E(U)(\mathit{UEv}(E)\big|_{\sigma})\}
\end{array}}{\mathcal{C} \vdash U[T \textbf{ fit } \sigma] \overset{\equiv}{\Longrightarrow\!\!\!\!\Rightarrow} (\langle F_E ; F'_E \rangle_{E \in \mathcal{C}}, \mathit{UEv}')}$$

$$\frac{\begin{array}{c}
\mathcal{C} \vdash T_1 \overset{\equiv}{\Longrightarrow\!\!\!\!\Rightarrow} (\langle F^1_E \rangle_{E \in \mathcal{C}}, \mathit{UEv}_1) \qquad \mathcal{C} \vdash T_2 \overset{\equiv}{\Longrightarrow\!\!\!\!\Rightarrow} (\langle F^2_E \rangle_{E \in \mathcal{C}}, \mathit{UEv}_2) \\
\mathit{UEv} = \{E \mapsto A \mid E \in \mathcal{C}, A\big|_{\iota_1} = \mathit{UEv}_1(E), A\big|_{\iota_2} = \mathit{UEv}_2(E)\} \\
\text{for } E \in \mathcal{C}, F_E = \{\mathcal{G} \mapsto \mathcal{G}' \mid \mathcal{G} \in |\mathbf{Alg}(\Sigma_G)| \text{ witnesses } E, \\
\qquad\qquad \mathcal{G}'\big|_{\theta_1} = F^1_E(\mathcal{G}), \mathcal{G}'\big|_{\theta_2} = F^2_E(\mathcal{G})\}
\end{array}}{\mathcal{C} \vdash T_1 \textbf{ and } T_2 \overset{\equiv}{\Longrightarrow\!\!\!\!\Rightarrow} (\langle F_E \rangle_{E \in \mathcal{C}}, \mathit{UEv})}$$

Fig. 4. Modified observational model semantics

Lemma 7.3. *If* $\mathcal{C}_{st} \vdash T \mathrel{\rhd\!\!\!>} (\theta{:}\, \Sigma_G \to \Sigma'_G, i{:}\, \Sigma \to \Sigma'_G)$ *and* $\mathcal{C}_{obs} \vdash T \overset{\equiv}{\Longrightarrow} \mathit{UEv}_{obs}$ *with* $\mathcal{C}_{obs} \subseteq \mathit{ucx}_{obs}(\mathcal{C}_{st})$*, then* $\mathcal{C}_{obs} \vdash T \overset{\equiv}{\Longrightarrow\!\!\!\!\Rightarrow} (\langle F_E \rangle_{E \in \mathcal{C}}, \mathit{UEv}_{obs})$ *for some family* $\langle F_E \rangle_{E \in \mathcal{C}_{obs}}$ *such that*

- *for* $E \in \mathcal{C}_{obs}$, $F_E{:}\, |\mathbf{Alg}(\Sigma_G)| \rightharpoonup |\mathbf{Alg}(\Sigma'_G)|$ *is persistent along* $\theta{:}\, \Sigma_G \to \Sigma'_G$*;*
- *for* $E \in \mathcal{C}_{obs}$*, if* $\mathcal{G} \in |\mathbf{Alg}(\Sigma_G)|$ *witnesses* E *then* $\mathcal{G} \in \mathit{dom}(F_E)$ *and* $\mathit{UEv}_{obs}(E) = F_E(\mathcal{G})\big|_i$*;*
- *the family* $\langle F_E \rangle_{E \in \mathcal{C}_{obs}}$ *is locally stable in the following sense: for* $E_1, E_2 \in \mathcal{C}_{obs}$ *such that* $E_1 \equiv E_2$*,* $\mathcal{G}_1, \mathcal{G}_2 \in |\mathbf{Alg}(\Sigma_G)|$ *that witness* E_1 *and* E_2*, respectively, and correspondence* $\rho{:}\, \mathcal{G}_1 \bowtie \mathcal{G}_2$*, if* $\mathcal{G}_1 \in \mathit{dom}(F_E)$ *then* $\mathcal{G}_2 \in \mathit{dom}(F_E)$ *as well and there exists a correspondence* $\rho'{:}\, F_{E_1}(\mathcal{G}_1) \bowtie F_{E_2}(\mathcal{G}_2)$ *with* $\rho'\big|_{\theta} = \rho$*.*

Proof. By induction on the structure of the unit term. In each case, the first two properties follow easily from the construction and Lemma 2.1. Prop. 5.3 and

Lemma 5.4 imply the last property for parametrized unit application and unit amalgamation. □

Since reducts preserve observational equivalence, Lemma 7.3 directly implies stability of unit constructions definable by architectural specifications:

Corollary 7.4. *If* $\vdash ASP \rhd\!\!\!\!\rhd (\mathcal{C}_{st}, \Sigma)$ *and* $\vdash ASP \overset{\equiv}{\Longrightarrow} (\mathcal{C}_{obs}, UEv_{obs})$ *then for any unit environments* $E_1, E_2 \in \mathcal{C}_{obs}$ *such that* $E_1 \equiv E_2$*, we have* $UEv_{obs}(E_1) \equiv UEv_{obs}(E_2)$. □

As already mentioned, the observational semantics is more permissive than the strict model semantics: there existence of a successful derivation of an observational meaning for an architectural specification does not in general imply that its strict model semantics is defined as well. Moreover, the observational semantics may "lose" some results permitted by the strict model semantics, which follows from Counterexample 5.8. However, if an architectural specification has a strict model semantics then its observational semantics is defined as well and up to observational equivalence, nothing new is added:

Theorem 7.5. *If* $\vdash ASP \rhd\!\!\!\!\rhd (\mathcal{C}_{st}, \Sigma)$ *and* $\vdash ASP \Rightarrow (\mathcal{C}, UEv)$ *then* $\vdash ASP \overset{\equiv}{\Longrightarrow} (\mathcal{C}_{obs}, UEv_{obs})$*, where for every* $E_{obs} \in \mathcal{C}_{obs}$ *there exists* $E \in \mathcal{C}$ *such that* $E_{obs} \equiv E$ *and* $UEv_{obs}(E_{obs}) \equiv UEv(E)$.

Proof. The following can be proved inductively:

1. $\vdash ASP \rhd\!\!\!\!\rhd (C_{st}, \Sigma)$ and $\vdash ASP \Rightarrow (\mathcal{C}, UEv)$: then $\vdash ASP \overset{\equiv}{\Longrightarrow} (\mathcal{C}_{obs}, UEv_{obs})$ with $\mathcal{C}_{obs} = Abs_{\equiv}(\mathcal{C})$ and for each stable $E \in \mathcal{C}$ (then necessarily $E \in \mathcal{C}_{obs}$) we have $UEv_{obs}(E) = UEv(E)$;
2. $\mathcal{C}_{st} \vdash Dcl \rhd\!\!\!\!\rhd \mathcal{C}'_{st}$ and $\mathcal{C} \vdash Dcl \Rightarrow \mathcal{C}'$, where $\mathcal{C} \subseteq ucx(\mathcal{C}_{st})$: then $Abs_{\equiv}(\mathcal{C}) \vdash Dcl \overset{\equiv}{\Longrightarrow} Abs_{\equiv}(\mathcal{C}')$, and similarly for sequences of unit declarations;
3. $\mathcal{C}_{st} \vdash T \rhd\!\!\!\!\rhd (\theta\colon \Sigma_G \to \Sigma'_G, i\colon \Sigma \to \Sigma'_G)$ and $\mathcal{C} \vdash T \Rightarrow UEv$ with $\mathcal{C} \subseteq ucx(\mathcal{C}_{st})$: then $Abs_{\equiv}(\mathcal{C}) \vdash T \overset{\equiv}{\Longrightarrow} UEv_{obs}$ where for each stable $E \in \mathcal{C}$ (then $E \in Abs_{\equiv}(\mathcal{C})$) we have $UEv_{obs}(E) = UEv(E)$.

The only potential difficulty is in the proof of item 3 for parametrized unit application, where to deduce the premise of the observational semantics rule that captures verification that the argument is in the domain of the parametrized unit, we need to rely on the corresponding premise of the strict model semantics, on the stability of the parametrized unit and on Lemma 7.3.

The theorem follows easily now: given the assumptions, item 1 implies that $\vdash ASP \overset{\equiv}{\Longrightarrow} (\mathcal{C}_{obs}, UEv_{obs})$ with $\mathcal{C}_{obs} = Abs_{\equiv}(\mathcal{C})$, and so for each $E_{obs} \in \mathcal{C}_{obs}$ there is a stable environment $E \in \mathcal{C}$ such that $E_{obs} \equiv E$. Thus, by item 1 and Cor. 7.4, $UEv(E) = UEv_{obs}(E) \equiv UEv_{obs}(E_{obs})$. □

8 Conclusions and Further Work

Apart from the preliminaries, this paper consists of two parts. Sects. 3, 4, and 5 recall a now rather standard and quite general view of the software development process, paying special attention to observational interpretation of the specifications involved and discussing in more detail than usual how "global" developments proceed using "local" constructions. We point out how observational interpretation of specifications leads to the crucial — and quite natural, Cor. 5.6 — stability requirement on the constructions, and how this in turn helps to establish correctness of development steps, Thm. 5.10. Then, Sects. 6 and 7 study how these general ideas may be instantiated in the context of architectural specifications as borrowed from CASL in a simplified version. We view here architectural specifications as means to build complex constructions to be used in the software development process. Observational interpretation of specifications brings out rather non-trivial issues. We study stability of the constructions involved, with the expected positive result in Cor. 7.4, and link the results under observational interpretation with those for the standard interpretation of architectural specifications, Thm. 7.5. Clearly, as mentioned in Sect. 7, this must be augmented with an analysis of the internal correctness of architectural specifications under observational interpretation.

Although formally we have worked in a specific — and simple — logical framework, it should be clear that much of the above applies to a wide range of institutions of interest. Rather than trying to embark on an exercise of formally spelling out the appropriate notion of "institution with extra structure", let us just remark that surprisingly little is required. A special notion of *observational model morphisms* that must be closed under composition and reducts, plus some extra categorical structure to identify "correspondences" as certain spans of such morphisms, seems necessary and sufficient to formulate most of the material presented. Notice that we have in effect not referred to the set of observable sorts in the technical development. The trick is, however, to study a sufficient number of special cases to demonstrate that observational intuitions in various institutions may well be captured by such a simple structure. Further justification may be provided via links with indistinguishability relations (via factorization properties, like Thm. 4.3, which in turn may require a richer context of *concrete institutions*, with model categories equipped with concretization structure subject to a number of technical requirements as in [BT96]).

On the other hand, to transfer the present work to the specific framework of CASL we need a precise and convincing definition of observational equivalence between CASL models (many-sorted algebras with predicates, partial operations and subsorting). In terms of the institutional structure hinted at above, our first attempts dictate to simply use *closed homomorphisms* as observational morphisms — but the resulting notion of equivalence needs a more detailed analysis, from both the methodological and technical point of view.

The semantics for our simplified architectural specifications made reference to the cocompleteness of the category of signatures and to the amalgamation property of the underlying institution. Many institutions enjoy these properties, including the many-sorted versions of various standard logics. However, the amalgamation property fails for full CASL with subsorts, as discussed in detail in [SMT$^+$01]. There are at least two ways to circumvent this problem. One is to present the global context as a diagram of signatures and a compatible family of models over this diagram, as in [SMT$^+$01]. The other possibility is to use an extension of the Casl institution to "enriched" signatures (where multiple embeddings between subsorts are allowed) and their corresponding models, where the amalgamation property holds, again presented in [SMT$^+$01].

References

[ABK$^+$03] E. Astesiano, M. Bidoit, H. Kirchner, B. Krieg-Brückner, P.D. Mosses, D. Sannella and A. Tarlecki. Casl: The Common Algebraic Specification Language. *Theoretical Computer Science*, to appear (2003). See also the Casl Summary at `http://www.brics.dk/Projects/CoFI/Documents/CASL/Summary/`.

[AKBK99] E. Astesiano, B. Krieg-Brückner and H.-J. Kreowski, eds. *Algebraic Foundations of Systems Specification*. Springer (1999).

[Ber87] G. Bernot. Good functors ... are those preserving philosophy! *Proc. 2nd Summer Conf. on Category Theory and Computer Science CTCS'87*, Springer LNCS 283, 182–195 (1987).

[BH93] M. Bidoit and R. Hennicker. A general framework for modular implementations of modular systems. *Proc. 4th Int. Conf. Theory and Practice of Software Development TAPSOFT'93*, Springer LNCS 668, 199–214 (1993).

[BH98] M. Bidoit and R. Hennicker. Modular correctness proofs of behavioural implementations. *Acta Informatica* 35(11):951–1005 (1998).

[BT96] M. Bidoit and A. Tarlecki. Behavioural satisfaction and equivalence in concrete model categories. *Proc. 20th Coll. on Trees in Algebra and Computing CAAP'96*, Linköping, Springer LNCS 1059, 241–256 (1996).

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

[BST02] M. Bidoit, D. Sannella and A. Tarlecki. Architectural specifications in Casl. *Formal Aspects of Computing*, to appear (2002). Available at `http://www.lsv.ens-cachan.fr/Publis/PAPERS/BST-FAC2002.ps`. Extended abstract: *Proc. 7th Intl. Conf. on Algebraic Methodology and Software Technology, AMAST'98.* Springer LNCS 1548, 341–357 (1999).

[CoFI02] The CoFI Task Group on Semantics. Semantics of the Common Algebraic Specification Language Casl. Available at `http://www.brics.dk/Projects/CoFI/Documents/CASL/Semantics/` (2002).

[EK99] H. Ehrig and H.-J. Kreowski. Refinement and implementation. In: [AKBK99], 201–242.

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

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

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

[Gin68] A. Ginzburg. *Algebraic Theory of Automata*. Academic Press (1968).

[Gog84] J. Goguen. Parameterized programming. *IEEE Trans. on Software Engineering* SE-10(5):528–543 (1984).

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

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

[GGM76] V. Giarratana, F. Gimona and U. Montanari. Observability concepts in abstract data type specifications. *Proc. 5th Intl. Symp. on Mathematical Foundations of Computer Science*, Springer LNCS 45, 576–587 (1976).

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

[Hof01] P. Hoffman. Verifying architectural specifications. *Recent Trends in Algebraic development Techniques, Selected Papers, WADT'01*, Springer LNCS 2267, 152-175 (2001).

[HLST00] F. Honsell, J. Longley, D. Sannella and A. Tarlecki. Constructive data refinement in typed lambda calculus. *Proc. 2nd Intl. Conf. on Foundations of Software Science and Computation Structures.* Springer LNCS 1784, 149–164 (2000).

[KHT$^+$01] B. Klin, P. Hoffman, A. Tarlecki, L. Schröder and T. Mossakowski. Checking amalgamability conditions for Casl architectural specifications. *Proc. 26th Intl. Symp. Mathematical Foundations of Computer Science MFCS'01*, Springer LNCS 2136, 451–463 (2001).

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

[Rei81] H. Reichel. Behavioural equivalence — a unifying concept for initial and final specification methods. *Proc. 3rd Hungarian Comp. Sci. Conference*, 27–39 (1981).

[Sch87] O. Schoett. *Data Abstraction and the Correctness of Modular Programming.* Ph.D. thesis, report CST-42-87, Dept. of Computer Science, Univ. of Edinburgh (1987).

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

[SB83] D. Sannella and R. Burstall. Structured theories in LCF. *Proc. Colloq. on Trees in Algebra and Programming.* Springer LNCS 159, 377–391 (1983).

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

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

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

[ST89] D. Sannella and A. Tarlecki. Toward formal development of ML programs: foundations and methodology. *Proc. Colloq. on Current Issues in Programming Languages, Intl. Joint Conf. on Theory and Practice of Software Development TAPSOFT'89*, Barcelona. Springer LNCS 352, 375–389 (1989).

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

[ST99] D. Sannella and A. Tarlecki. Algebraic preliminaries. In: [AKBK99], 13–30.

[SMT$^+$01] L. Schröder, T. Mossakowski, A. Tarlecki, P. Hoffman and B. Klin. Semantics of architectural specifications in Casl. *Proc. 4th Intl. Conf. Fundamental Approaches to Software Engineering FASE'01*, Genova. Springer LNCS 2029, 253–268 (2001).