Verifying Architectural Specifications

Piotr Hoffman

Warsaw University, Institute of Informatics,
Banacha 2, 02-097 Warszawa, Poland
piotrek@mimuw.edu.pl

Abstract. In this paper we develop methods for verifying the correctness of architectural specifications, a mechanism introduced in the CASL specification language. This mechanism offers a formal way to express implementation steps in program development. Each such step states that to implement the unit of interest, one may implement some other units and then assemble them in the prescribed manner. In this paper we define a formal institution-independent semantics of architectural specifications, as well as sound and complete methods for proving them correct, applicable in the case of many institutions, in particular first-order logic.

Introduction

The formal development of a program consists of three phases: writing a requirements specification, implementing it, and proving the correctness of the implementation with respect to the specification (see [ST97]). Architectural specifications are meant to be a tool aiding the developer in breaking down the implementation task into independent subtasks. These subtasks can be either further broken down, or directly coded in a chosen programming language. The benefits of dividing a programming task into independent subtasks are obvious: work on parts of the project can proceed in parallel, the project as a whole is easier to maintain and comprehend. Also, the correctness proof can now be constructed from the correctness proofs for the individual subtasks and a correctness proof of the subdivision (i.e., architectural specification) itself.

In this paper, we first recall the notion of *institution* in Section 1 and then define the syntax and formal semantics of architectural specifications in an institution-independent fashion in Section 2. Both are somewhat simplified versions of what has been first introduced in CASL, the specification language designed by the Common Framework Initative (CoFI). For more information on CASL, see [CASL00,CASL99,CASL]; for an in-depth discussion of architectural specifications, see [BST99].

In Section 3 we precisely define what we understand by verifying architectural specifications. Essentially, we will consider an architectural specification correct, if having completed the subtasks one can always assemble their implementations in the prescribed manner and indeed obtain an implementation of the original task. Our aim is to find an algorithm that, when fed an architectural specification, generates proof obligations equivalent to the correctness of the specification. Of course, we would like these obligations to be of a possibly simple form; we choose what we call *simple obligations*, i.e., statements of the form $\Phi \models_\Sigma \Psi$, where Φ and Ψ are finite sets of sentences of the underlying institution (these "sentences" could actually be structured specifications over some other institution — we would then end up with refinements of specifications). Note that this will *not* provide a decision procedure for the verification of architectural specifications in general — in fact, in typical institutions verification can easily be proven undecidable (since checking simple obligtions, as defined above, is typically undecidable).

In Section 4 we divide the procedure of generating obligations into two stages: first, we want to extract some *intermediate proof obligations* from an architectural specification; next, we want to transform these intermediate proof obligations into simple obligations. We then state our main theorem: under the assumption that the underlying institution has *logical amalgamation*, a notion closely connected to the amalgamation property (see [Tar99] and [SMTKH01]), one can perform stage one of our procedure; that is, one can reduce the problem of checking the correctness of an architectural specification to that of discharging some intermediate proof obligations. The proof of this result can be found in Section 5.

Sections 6 and 7 are devoted to stage two of the procedure of generating proof obligations, i.e., transforming intermediate proof obligations into simple obligations. In Section 6 we simplify the intermediate proof obligations, again under the assumption that the underlying institution has logical amalgamation. In Section 7 we show how to transform the simplified intermediate proof obligations into simple obligations in the case of first-order logic with equality. This section is the only part of the paper that is not institution-independent. Still, it seems that the techniques in use should be applicable to other institutions as well.

Finally, in Section 8 we propose to impose additional restrictions on architectural specifications, thus defining *strict* correctness. This allows us to divide the verification process into two parts: the first one is static and performed automatically, the second one involves non-trivial proof obligations discharged by means of a theorem-prover. The notions of static and strict correctness also form a link between this paper and the *extended static semantics* of [SMTKH01] and [KHTSM01].

1 Institutions

The definition of architectural specifications will be parametrized by an underlying logical system. To formalize this notion, we will use the concept of *institution* (see [GB92]), presented in this section. Before we dive into the details, we introduce some notation. For any function, morphism or functor f, by $dom(f)$ we denote its domain; if f is a morphism or functor, then by $cod(f)$ we denote its codomain. For any category C, by $|C|$ we denote the class of objects of C. Finally, composition is always written in diagrammatical order.

An institution $\mathbf{I}$ is a quadruple $(Sig, \mathbf{Mod}, Sen, \models)$, where:

- Sig is a category of *signatures*;
- $\mathbf{Mod} : Sig^{op} \to \mathbf{Class}$ is a functor into the quasi-category of all large sets;
- $Sen : Sig \to \mathbf{Set}$ is a functor into the category of all small sets;
- $\models$ is a family $\{\models_\Sigma\}_{\Sigma \in |Sig|}$, where $\models_\Sigma$ is a relation on $\mathbf{Mod}(\Sigma) \times Sen(\Sigma)$.

These data are subject to the so-called *satisfaction condition*, i.e., for any $\sigma : \Sigma \to \Delta$ in Sig, $\phi \in Sen(\Sigma)$ and $M \in \mathbf{Mod}(\Delta)$ the following equivalence holds:

$$M \models_\Delta Sen(\sigma)(\phi) \iff \mathbf{Mod}(\sigma)(M) \models_\Sigma \phi$$

Elements of $\mathbf{Mod}(\Sigma)$ are called *Σ-models* and elements of $Sen(\Sigma)$ are called *Σ-sentences*. We usually denote the (large) function $\mathbf{Mod}(\sigma)$ by $\cdot|_\sigma$ and call it the *(σ-)reduct*. For any $\sigma : \Sigma \to \Delta$ in Sig and $\phi \in Sen(\Sigma)$, by $\sigma(\phi)$ we denote $Sen(\sigma)(\phi)$ and we call it the *translation* of ϕ along σ.

The most prominent example of an institution is many-sorted first-order logic. A possible definition follows:

- a signature is a pair (S, p), where S is a finite set and p is a function with finite domain into $S^* \times S$; the elements of S are called *sorts* and the elements of $dom(p)$ are called *function symbols*;
- a signature morphism $\sigma : (S, p) \to (T, q)$ consists of two functions $\sigma^S : S \to T$ and $\sigma^F : dom(p) \to dom(q)$ such that for any $f \in dom(p)$, if $p(f) = (s_1 \ldots s_n, s)$, then $q(\sigma^F(f)) = (\sigma^S(s_1) \ldots \sigma^S(s_n), \sigma^S(s))$; composition is defined component-wise;
- a sentence over (S, p) is a first-order sentence involving variables of the sorts S, the function symbols $dom(p)$, and equality;
- the translation of a sentence along a signature morphism is defined naturally;
- a model over (S, p) assigns to each $s \in S$ a set called the *carrier* of s and to each $f \in dom(p)$ with $p(f) = (s_1 \ldots s_n, s)$ a function called the *interpretation* of f, whose domain is the product of the carriers of the sorts $s_1, \ldots, s_n$ and codomain is the carrier of the sort s;
- the reduct of a model M over (T, q) along a morphism $\sigma : (S, p) \to (T, q)$ assigns to each sort $s \in S$ the carrier of $\sigma^S(s)$ in M and to each function symbol $f \in dom(p)$ the interpretation of $\sigma^F(f)$ in M;
- the relations $\models_\Sigma$ are defined as usual.

With these definitions the satisfaction condition holds (cf. the equational case in [GB92]) and thus we have an institution; we denote it by FOL.

For any institution $\mathbf{I}$, by $|\mathbf{I}|$ we will denote the institution obtained from $\mathbf{I}$ by removing all the sentences, i.e., by setting $Sen(\Sigma) = \emptyset$ for all signatures Σ.

We will overload the symbols $\mathbf{Mod}$ and $\models$ as follows. For any set Φ of Σ-sentences, $\mathbf{Mod}_\Sigma(\Phi)$ denotes the class of all Σ-models satisfying all sentences from Φ. If additionally Ψ is a set of Σ-sentences, then we write $\Phi \models_\Sigma \Psi$ if $\mathbf{Mod}_\Sigma(\Phi) \subseteq \mathbf{Mod}_\Sigma(\Psi)$.

We parametrize the definition of architectural specifications by an underlying institution $\mathbf{I} = (Sig, \mathbf{Mod}, Sen, \models)$ and a subcategory $\mathcal{J}$ of Sig called the *inclusion subcategory.* We require that $\mathcal{J}$ be a partial order and that $|\mathcal{J}| = |Sig|$. It makes sense to impose further restrictions on the inclusions, but this will not be necessary in the following considerations. An inclusion of Σ into Δ will be denoted by $\iota_{\Sigma,\Delta}$. If this inclusion exists, we will write $\Sigma \leq \Delta$. In the case of FOL, we define a morphism σ to be an inclusion if both σ^S and σ^F are set-theoretic inclusions.

In this paper, we will always assume that signatures, signature morphisms and sentences are representable in a finite way, the translation of sentences is computable, the order $\leq$ is decidable and the binary least upper bound operation in $\leq$ is computable. These assumptions obviously hold in FOL.

2 Architectural Specifications

The language of architectural specifications is parametrized by an institution $\mathbf{I}$ and inclusion subcategory $\mathcal{J}$, and described by the grammar in Figure 1.

$$
\begin{array}{l}
\mathit{ASP} ::= \textbf{units}\ \mathit{UDS}\ \textbf{result}\ \mathit{UE}\ \textbf{implementing}\ \mathit{USP}\\
\mathit{UDS} ::= \epsilon\ |\\
\qquad\qquad \mathit{UDS}\ \mathit{ID} : \mathit{USP}\\
\mathit{USP} ::= \mathit{SP}\ |\\
\qquad\qquad \mathit{SP} \rightarrow \mathit{SP}\\
\mathit{UE} ::= \bullet\, T\ |\\
\qquad\qquad \lambda\ \mathit{ID} : \mathit{SP} \bullet T\\
T ::= \mathit{ID}\ |\\
\qquad \mathit{ID}(T)\ |\\
\qquad T\ \textbf{and}\ T\ |\\
\qquad \textbf{reduce}\ T\ \textbf{by}\ \sigma\ |\\
\qquad \textbf{let}\ \mathit{ID} = T\ \textbf{in}\ T
\end{array}
$$

Fig. 1. Language of architectural specifications.

Here, the *ID*s come from an infinite set of identifiers, the σs are signature morphisms in $\mathbf{I}$, and the *SP*s are finite presentations in $\mathbf{I}$, i.e. pairs (Σ, Φ), where Φ is a finite set of sentences over Σ.

An architectural specification states that, in order to implement a unit satisfying the specification in the **implementing** section, one can implement the units declared in the **units** section, and then assemble them as described in the **result** section. The denotation of the whole specification is a function taking any implementation of the declared units to a result unit as described by the unit expression *UE*.

Remark 1. The above language of architectural specifications differs somewhat from that used in CASL. One addition we have made is the **implementing** section, which we think makes the idea of architectural specifications more comprehensible; in CASL comments are used for the same purpose. Also, the reduct is more restrictive in CASL. On the other hand, we have left out a lot of constructs: multiparameter units, unit definitions, fitting morphisms in applications, declarations with imports, and translation. Also, we have chosen the specifications to be finite presentations. This last simplification does not mean that structured specifications cannot be dealt with in our framework; all one has to do is formalize them as sentences of an institution, which is indeed possible.

In order to introduce a formal semantics of architectural specifications, we need to provide some definitions. A *parametric signature* is a pair $P\Sigma = (\Sigma, \Delta)$ such that $\Sigma \leq \Delta$. A *unit signature* $U\Sigma$ is either a (regular) signature or a parametric signature. A *parametric unit* over a parametric signature (Σ, Δ) is a partial function U from $\mathbf{Mod}(\Sigma)$ to $\mathbf{Mod}(\Delta)$ which is *persistent*, i.e., for any $M \in dom(U)$ we have $U(M)|_{\iota_{\Sigma,\Delta}} = M$. A *unit* over a unit signature is a model over that signature, if it is a (regular) signature, or a parametric unit over that signature, if it is a parametric signature. By $\mathbf{PUnit}(P\Sigma)$ we denote the set of all parametric units over the parametric signature $P\Sigma$. A *parametric unit specification* over a parametric signature (Σ, Δ) is a pair of presentations $u = ((\Sigma, \Phi), (\Delta, \Psi))$. Then $\mathbf{PUnit}(u)$ is the set of units over (Σ, Δ) with domain $\mathbf{Mod}_\Sigma(\Phi)$ and into $\mathbf{Mod}_\Delta(\Psi)$. A parametric unit specification u is *inconsistent* if $\mathbf{PUnit}(u)$ is empty. A *unit specification* is either a presentation or a parametric unit specification. A *static environment* δ is a pair (B, P), where B maps identifiers to signatures and P maps identifiers to parametric signatures. The domains of B and P are required to be finite and disjoint. A *unit environment* fitting a static environment (B, P) is a map e, sending any identifier $ID \in dom(B)$ to a model in $\mathbf{Mod}(B(ID))$ and any identifier $ID \in dom(P)$ to a parametric unit in $\mathbf{PUnit}(P(ID))$. A *unit context* fitting a static environment δ is any set E of environments fitting δ. A *model function of type* (E, Σ) is a total function F from the context E into $\mathbf{Mod}(\Sigma)$. *Unit functions of type* $(E, U\Sigma)$, denoted UF, are defined analogically.

To deal with the **and**-construct, we will need to define two notions. First, we define the *sum* of signatures Σ, Δ to be their least upper bound $\Gamma = \Sigma \cup \Delta$ with respect to the order $\leq$, if it exists; if so, we call them *compatible*. Then for any models $M \in \mathbf{Mod}(\Sigma)$ and $N \in \mathbf{Mod}(\Delta)$ we define their *amalgamation*, denoted $M \oplus N$, to be a model $P \in \mathbf{Mod}(\Gamma)$ satisfying $P|_{\iota_{\Sigma,\Gamma}} = M$ and $P|_{\iota_{\Delta,\Gamma}} = N$, if it exists and is unique. Models are called *amalgamable* if their amalgamation exists.

The semantics will assign:

- to any unit specification USP, a unit signature $U\Sigma$ and a (large) set $\mathcal{U}$ of units over $U\Sigma$; i.e., either a signature Σ and a (large) set $\mathcal{M}$ of models over Σ or a parametric signature $P\Sigma$ and a (large) set $\mathcal{P}$ of parametric units over $P\Sigma$;
- to any list of unit declarations UDS, a static environment δ and a unit context E fitting δ;
- to any term T, given a static environment δ and a unit context E fitting δ, a signature Σ and a model function F of type (E, Σ);

– to any unit expression UE, given a static environment δ and a unit context E fitting δ, a unit signature $U\Sigma$ and a unit function UF of type $(E, U\Sigma)$;
– to any architectural specification ASP, a static environment δ, a unit signature $U\Sigma$ and a unit function UF of type $(dom(UF), U\Sigma)$, where $dom(UF)$ fits δ.

In the semantics rules below, we use the natural semantics style, with judgments written $\_\_ \vdash \_\_ \rhd \_\_$. We always implicitly add premises stating that objects denoted by δ are static environments, in any pair δ, E, E fits δ, etc. By $f\,[a/x]$ we denote the function which extends f with the mapping $x \mapsto a$.

$$\boxed{\vdash ASP \rhd \delta, U\Sigma, UF}$$

$$\frac{\begin{array}{c}\vdash UDS \rhd \delta, E \\ \delta, E \vdash UE \rhd U\Sigma, UF \\ \vdash USP \rhd U\Sigma, \mathcal{U} \\ \text{for all } e \in E, UF(e) \in \mathcal{U}\end{array}}{\vdash \textbf{units } UDS \textbf{ result } UE \textbf{ implementing } USP \rhd \delta, U\Sigma, UF}$$

$$\boxed{\vdash UDS \rhd \delta, E}$$

$$\frac{}{\vdash \epsilon \rhd \emptyset, \{\emptyset\}}$$

$$\frac{\begin{array}{c}\vdash UDS \rhd (B, P), E \\ \vdash USP \rhd \Sigma, \mathcal{M} \\ ID \notin dom(B) \cup dom(P)\end{array}}{\begin{array}{c}\vdash UDS\ ID : USP \rhd \\ (B\,[\Sigma/ID]\,, P)\,, \{\ e\,[M/ID]\ \mid e \in E,\ M \in \mathcal{M}\ \}\end{array}}$$

$$\frac{\begin{array}{c}\vdash UDS \rhd (B, P), E \\ \vdash USP \rhd P\Sigma, \mathcal{P} \\ ID \notin dom(B) \cup dom(P)\end{array}}{\begin{array}{c}\vdash UDS\ ID : USP \rhd \\ (B, P\,[P\Sigma/ID])\,, \{\ e\,[U/ID]\ \mid e \in E,\ U \in \mathcal{P}\ \}\end{array}}$$

$$\boxed{\vdash USP \rhd U\Sigma, \mathcal{U}}$$

$$\frac{}{\vdash (\Sigma, \Phi) \rhd \Sigma, \mathbf{Mod}_{\Sigma}(\Phi)}$$

$$\frac{\Sigma \leq \Delta}{\vdash (\Sigma, \Phi) \to (\Delta, \Psi) \rhd (\Sigma, \Delta), \mathbf{PUnit}((\Sigma, \Phi), (\Delta, \Psi))}$$

$$\boxed{\delta, E \vdash UE \rhd U\Sigma, UF}$$

$$\frac{(B,P), E \vdash T \rhd \Sigma, F}{(B,P), E \vdash \bullet T \rhd \Sigma, F}$$

$$\frac{\begin{array}{c}(B\,[\Sigma/ID]\,,P), \{\, e\,[M/ID] \mid e \in E,\ M \in \mathbf{Mod}_\Sigma(\Phi)\,\} \vdash T \rhd \Delta, F \\ \Sigma \leq \Delta \\ \text{for all } e \in E \text{ and } M \in \mathbf{Mod}_\Sigma(\Phi),\ F(e\,[M/ID])|_{\iota_{\Sigma,\Delta}} = M\end{array}}{\begin{array}{c}(B,P), E \vdash \lambda ID : (\Sigma, \Phi) \bullet T \rhd \\ (\Sigma, \Delta),\ \lambda e \in E \cdot \lambda M \in \mathbf{Mod}_\Sigma(\Phi) \cdot F(e\,[M/ID])\end{array}}$$

$$\boxed{\delta, E \vdash T \rhd \Sigma, F}$$

$$\frac{ID \in dom(B)}{(B,P), E \vdash ID \rhd B(ID), \lambda e \in E \cdot e(ID)}$$

$$\frac{\begin{array}{c}ID \in dom(P) \text{ and } P(ID) = (\Sigma, \Delta) \\ (B,P), E \vdash T \rhd \Sigma, F \\ \text{for all } e \in E,\ F(e) \in dom(e(ID))\end{array}}{(B,P), E \vdash ID(T) \rhd \Delta, \lambda e \in E \cdot e(ID)(F(e))}$$

$$\frac{\begin{array}{c}\delta, E \vdash T_1 \rhd \Sigma_1, F_1 \\ \delta, E \vdash T_2 \rhd \Sigma_2, F_2 \\ \Delta = \Sigma_1 \cup \Sigma_2 \\ \text{for all } e \in E, \text{ the models } F_1(e) \text{ and } F_2(e) \text{ are amalgamable}\end{array}}{\delta, E \vdash T_1 \textbf{ and } T_2 \rhd \Delta, \lambda e \in E \cdot F_1(e) \oplus F_2(e)}$$

$$\frac{\delta, E \vdash T \rhd cod(\sigma), F}{\delta, E \vdash \textbf{reduce } T \textbf{ by } \sigma \rhd dom(\sigma), \lambda e \in E \cdot F(e)|_\sigma}$$

$$\frac{\begin{array}{c}(B,P), E \vdash T \rhd \Sigma, F \\ ID \notin dom(B) \cup dom(P) \\ (B\,[\Sigma/ID]\,,P), \{\, e\,[F(e)/ID] \mid e \in E\,\} \vdash T' \rhd \Sigma', F'\end{array}}{(B,P), E \vdash \textbf{let } ID = T \textbf{ in } T' \rhd \Sigma', \lambda e \in E \cdot F'(e\,[F(e)/ID])}$$

It is easy to see that any given syntactic object has, in a given context, at most one denotation. If this denotation exists, we call the object *correct* in that context. In particular, an architectural specification *ASP* is *correct* if it has a denotation.

3 Correctness

In this paper we investigate ways of checking whether an architectural specification is correct. Obviously, the difficulty of this task depends on the underlying institution **I** and the inclusion category. If we choose **I** to be the institution of first-order logic, checking correctness will of course turn out undecidable, which is a straightforward consequence of first-order logic being undecidable. Generally, there are four conditions in the semantic rules which cannot always be checked statically. These are:

1. "for all $e \in E, UF(e) \in \mathcal{U}$" in the top-most rule;
2. "for all $e \in E, F(e) \in dom(e(ID))$" in the rule for application;
3. "for all $e \in E$, the models $F_1(e)$ and $F_2(e)$ are amalgamable" in the rule for amalgamation;
4. "for all $e \in E$ and $M \in \mathbf{Mod}_\Sigma(\Phi)$, $F(e\,[M/ID])|_{\iota_{\Sigma,\Delta}} = M$" in the rule for λ-definition.

In an institution as powerful as FOL, each of these conditions is sufficient to cause undecidability.

We call a statement of the form $\Phi \models_\Sigma \Psi$, where Φ, Ψ are finite sets of Σ-sentences, a *simple obligation* over Σ. Our aim in this paper is to produce an algorithm, for as general a case as possible, which will, given an architectural specification, generate a finite set of *proof obligations* which capture its correctness. More specifically, the algorithm will compute the following two finite sets:

- P, consisting of parametric unit specifications
- S, consisting of simple obligations

We will require that if the algorithm on ASP does not signal failure and produces sets P and S, then ASP is correct if and only if *either* one of the parametric unit specifications in P is inconsistent, *or* all obligations $\Phi \models_\Sigma \Psi$ from S are true. We will additionally require that if any parametric unit specification in P is inconsistent, then the denotation of ASP is the empty function. Of course, the algorithm may signal failure only if ASP is incorrect.

The above requirements effectively mean that if we want to prove the correctness of an architectural specification and are not interested in the case that it denotes the empty function, then we are left with checking simple obligations. This is a desirable situation, since proof methods and tools often exist for such obligations (see, e.g., [Pad99], [BCH99], [Borz98]).

4 Intermediate Proof Obligations and Logical Amalgamation

In this section we introduce an intermediate kind of proof obligation and we show that, under certain assumptions, one can reduce the problem of checking the correctness of an architectural specification to discharging such obligations.

Let Σ be a signature. We define *Σ-statements* to be:

– $\sigma \oplus \tau$, where $\sigma : \Delta \to \Sigma$, $\tau : \Gamma \to \Sigma$, Δ and Γ compatible
– $\sigma = \tau$, where $\sigma, \tau : \Delta \to \Sigma$
– $\sigma \neq \tau$, where $\sigma, \tau : \Delta \to \Sigma$

The statement $\sigma \oplus \tau$ is called the *amalgamation statement*, the statements $\sigma = \tau$ and $\sigma \neq \tau$ are called *positive* and *negative equality statements*, respectively. Let M be a model over Σ. We say that:

– $M \models_\Sigma \sigma \oplus \tau$ if $M|_\sigma$ and $M|_\tau$ are amalgamable
– $M \models_\Sigma \sigma = \tau$ if $M|_\sigma = M|_\tau$
– $M \models_\Sigma \sigma \neq \tau$ if $M|_\sigma \neq M|_\tau$

These statements are in general non-trivial, since the morphisms involved need not be inclusions.

The *intermediate proof obligations* are *satisfaction* and *amalgamability obligations*. A *satisfaction obligation* is a statement of the form $\Phi \cup \mathcal{E} \models_\Sigma \Psi$, an *amalgamability obligation* takes the form $\Phi \cup \mathcal{E} \models_\Sigma \sigma \oplus \tau$. Here, Φ and Ψ are finite sets of Σ-sentences, $\mathcal{E}$ is a finite set of equality Σ-statements, and $\sigma \oplus \tau$ is a Σ-statement, as introduced above.

Obviously, checking satisfaction obligations is computationally at least as hard as checking simple obligations. In particular, in FOL it is undecidable. Though amalgamability obligations seem less complex, this need not be the case — e.g., in FOL for any finite sets Φ and Ψ of sentences over Σ one can compute an amalgamability obligation which is true iff $\Phi \models_\Sigma \Psi$. To see this, let Δ be the signature Σ plus a new sort s and constants a and b of this sort. Now, consider the signature Δ_0 consisting of one sort s and one constant x of this sort, and two morphisms $\sigma, \tau : \Delta_0 \to \Delta$, one sending s to s and x to a, the other sending s to s and x to b. Then the obligation $(\Delta, \{(\bigwedge \Phi \Rightarrow \bigwedge \Psi) \Rightarrow a = b\}) \models_\Delta \sigma \oplus \tau$ is equivalent to $\Phi \models_\Sigma \Psi$. Another interesting example is the logic of CASL. In this logic problems very similar to the problem of checking whether $\models_\Sigma \sigma \oplus \tau$ holds have been shown undecidable, see Section 8 and [KHTSM01].

For any intermediate proof obligation without equality statements one can produce an architectural specification which is correct if and only if that obligation is true. This means that the intermediate obligations are certainly no harder than the original problem. As for the equality statements, they are needed to express the fact that a parametric unit applied to a given argument always yields the same result (actually, they are decompositions of implications between positive equality statements: if the arguments of two applications are equal, then so must be the results).

The category $Pres(\mathbf{I})$ of finite presentations in the institution $\mathbf{I}$ is defined as follows. Its objects are finite presentations, and a morphism between (Σ, Φ) and (Δ, Ψ) is simply a morphism $\sigma : \Sigma \to \Delta$ in Sig such that for any $M \in \mathbf{Mod}_\Delta(\Psi)$ we have $M|_\sigma \in \mathbf{Mod}_\Sigma(\Phi)$. This makes $\mathbf{Mod}$ extend to a (contravariant) functor on $Pres(\mathbf{I})$.

A *diagram* is a functor $D : I \to Pres(\mathbf{I})$ from a finite category I; the category I is called the *scheme* of the diagram. A family of models $\{M_i\}_{i \in |I|}$ is said to be *consistent* with D if:

– for any $i \in |I|$, $M_i \in \mathbf{Mod}(D(i))$
– for any $m : i \to j$ in I, $M_i = M_j|_{D(m)}$

We say that the institution $\mathbf{I}$ has *logical amalgamation* if for any diagram D with scheme I one can compute a sink $\{\alpha_i : D(i) \to (\Sigma, \Phi)\}_{i\in|I|}$ such that the set $\{\,\{M|_{\alpha_i}\}_{i\in|I|} \mid M \in \mathbf{Mod}_\Sigma(\Phi)\,\}$ is equal to the set of all families consistent with D. This sink is called an *amalgamating sink*.

The motivation behind the above notion is as follows. While verifying an architectural specification, we will build diagrams of presentations. Some nodes in these diagrams will correspond to declared non-parametric units, others will correspond to terms. Take a family consistent with the diagram. Then the models at the nodes corresponding to declarations define an environment, while the nodes corresponding to terms define the values of these terms in that environment (actually, to make this intuition precise we would have to add some information on the parametric units as well). In the verification process, two conditions need to be checked. First, whether the model at a given node always satisfies a given set of sentences; second, whether models at two given nodes are always amalagamable. These conditions will give rise to the satisfaction and amalgamability obligations, respectively.

Now, suppose we have a diagram D with scheme I and an amalgamating sink $\{\alpha_i : D(i) \to (\Sigma, \Phi)\}_{i\in|I|}$. Then instead of saying "for any family $\{M_i\}_{i\in|I|}$ consistent with D, $M_i \models \phi$", we may say "for any model $M \in \mathbf{Mod}_\Sigma(\Phi)$, $M|_{\alpha_i} \models \phi$"; this is easily equivalent to a satisfaction obligation. Of course, the details are much more complicated, e.g. when applications of parametric units come into play.

We now show that the logical amalgamation property is weaker than having finitely composable signatures (see [Tar99]) and computability of colimits:

Proposition 1. *If the signature category of* $\mathbf{I}$ *is finitely cocomplete, the colimits are computable, and the model functor preserves finite limits, then* $|\mathbf{I}|$ *has logical amalgamation.*

Proof. Let D be a diagram in $|\mathbf{I}|$ with scheme I and let $\{\alpha_i : D(i) \to \Sigma\}_{i\in|I|}$ be its colimit. We claim that α treated as a sink in $Pres\,(|\mathbf{I}|)$ is an amalgamating sink. Take any M over Σ. Then the family $\{M|_{\alpha_i}\}_{i\in|I|}$ is consistent with D, since α is a cocone. Now take any family $\{M_i\}_{i\in|I|}$ consistent with D. $\{\mathbf{Mod}(\alpha_i)\}_{i\in|I|}$ now forms a limiting cone over $D;\mathbf{Mod}$. Define a cone $\{f_i : A \to \mathbf{Mod}(D(i))\}_{i\in|I|}$ by taking $A = \mathbf{Mod}(\Sigma) \cup \{*\}$, where $*$ is a new element, and f_i to be $\mathbf{Mod}(\alpha_i)$ extended by $f_i(*) = M_i$. Then there exists a (large) function $h : A \to \mathbf{Mod}(\Sigma)$ factoring this cone through $\mathbf{Mod}(\alpha)$. Then we have $M_i = h(*)|_{\alpha_i}$, which proves the proposition. □

The following nicely generalizes the standard result of [GB92].

Proposition 2. *An institution* $\mathbf{I}$ *has logical amalgamation if and only if amalgamating sinks in* $\mathbf{I}$ *can be computed for any diagram* $D : I \to Pres(|\mathbf{I}|)$.

Proof. We will only prove the implication from right to left. Let D be a diagram with scheme I in $\mathbf{I}$. Denote by $|D|$ the diagram in $|\mathbf{I}|$ obtained from D by removing all the sentences. We can now compute an amalgamating sink $\{\alpha_i : |D|(i) \to (\Sigma, \Phi)\}_{i \in |I|}$ for $|D|$. Let $\Psi = \bigcup_{i \in |I|} \alpha_i(\pi_2(D(i)))$ and consider the sink obtained by treating the α_i as morphisms in $Pres(\mathbf{I})$ with domain $D(i)$ and codomain $(\Sigma, \Phi \cup \Psi)$. This sink is an amalgamating sink for D. $\square$

Corollary 1. *For any institution* $\mathbf{I}$*, if* $|\mathbf{I}|$ *has logical amalgamation, then so does* $\mathbf{I}$.

$\square$

Example 1. Take the institution FOL^{der} to be the same as FOL, but with *derived signature morphisms*. That is, a morphism σ from Σ to Δ works as usual on sorts, and may assign to a symbol $f : s_1 \times \cdots \times s_n \to s$ in Σ a term of sort $\sigma^S(s)$ in Δ involving the variables $x_1 : \sigma^S(s_1), \ldots, x_n : \sigma^S(s_n)$. Then the reduct of a model over Δ interprets f as the function which assigns to any tuple $v_1, \ldots, v_n$ of appropriate arguments the value of this term under the valuation $\{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$.

As can easily be seen, the signature category of FOL^{der} is not finitely cocomplete. Take Σ to be a signature with a single sort s and a function symbol $f : s \times s \to s$ and take Δ to contain only the sort s. We then may define two morphisms $\sigma, \tau : \Sigma \to \Delta$, which assign to f the terms x_1 and x_2, respectively. The diagram consisting of these two parallel morphisms does not have a colimit (i.e., coequalizer). In fact, no cocone over this diagram exists. However, an amalgamating sink exists: it consists of the signature Δ with the sentence "$\forall x : s \cdot \forall y : s \cdot x = y$" and the morphisms σ, τ, id_Δ.

Proposition 3. *The institutions* $|FOL|$ *and* FOL *have logical amalgamation.*

Proof. We will use the fact that the signature category of FOL is finitely cocomplete and the model functor preserves finite limits (cf. [GB84] for the equational case); it is obvious that the colimits are computable. That $|FOL|$ has logical amalgamation is now a consequence of Prop. 1. Then the fact that FOL has logical amalgamation follows by Corollary 1. $\square$

Theorem 1. *Assume* $\mathbf{I}$ *has logical amalgamation. Then there exists an algorithm which, given an architectural specification ASP, either signals failure, or computes a finite set of parametric unit specifications* P *and a finite set* S *of satisfaction and amalgamability obligations, and such that:*

$$\begin{aligned} &\textit{ASP is correct} \iff \textit{the algorithm does not signal failure, and} \\ &\qquad \textit{either some parametric unit specification in } P \textit{ is inconsistent,} \\ &\qquad\qquad \textit{or every obligation in } S \textit{ is true} \end{aligned}$$

Also, if an element of P *is inconsistent, then the denotation of ASP is the empty function.*

Let n be the number of declared parametric units in ASP and k_i, $i = 1, \ldots, n$, the number of times the parametric unit number i is applied in ASP, and let s be the number of amalgamations and applications in ASP. Then the cardinality of S is bounded by $4 + s \cdot 2^{|k_1^2-1|+\ldots+|k_n^2-1|}$ and the number of equality statements in every obligation is lower than $\frac{1}{2}\sum_{1\leq i\leq n} k_i(k_i-1)$.

The above theorem is a major step in verifying architectural specifications. However, the proof is quite technical and we delegate it to the next section. The bounds in the theorem are important, since they show that if all units are applied at most once, then the number of obligations is linear and the equality statements vanish altogether. This is also the reason for explicitly introducing the **let**-construct to the language of architectural specifications, since it allows us to remove common subexpressions and hence also the number of applications of a single parametric unit.

Remark 2. The semantics of architectural specifications considered in this paper is a so-called *non-generative* semantics, meaning that repeated application of a parametric unit to the same arguments always yields the same result. One could introduce a *generative* semantics (as that of functors in SML [Pau96]) in either of two equivalent ways. One method would be to define parametric units as multi-valued functions (i.e., relations) and rewrite the semantics appropriately. Another method would be to treat any declaration of a parametric unit F with some type as a sequence of declarations $F_1, \ldots, F_k$ with the same type and in the terms substitute F_i for the ith application of F. The above theorem implies that with generative semantics (defined either way), the cardinality of the set of obligations S will be bounded by $4 + s$ and the obligations themselves will not involve equality statements.

Example 2. The architectural specification in Fig. 2 is an example of a correct specification which is not correct with respect to a generative semantics. The diagram associated with this specification is depicted below (here, σ is the signature morphism used in the definition of the unit C''):

$$\begin{tikzcd}
A \arrow[r, "G"'] \arrow[d, "F"'] & C \arrow[dr, "H"'] & \\
B \arrow[dr, "H"'] & & C' \\
& B' \arrow[dr, "\iota_1"'] & C'' \arrow[u, "\sigma"'] \arrow[d, "\iota_2"'] \\
& & T
\end{tikzcd}$$

The correctness of this specification can be established as follows. First, all the applications are obviously correct. Second, the amalgamation defining T is correct because the carriers of the sort s in the units B' and C'' are identical, since they both come from the unit A. So, what remains to be checked is that the unit T always satisfies the sentence $a = a' \Rightarrow b = b'$. We will consider two cases. If the units B and C are not equal, then the interpretations of the constant a in B and the constant a in C must be different. Thus, in the unit T the constants a and a' have different interpretations and the sentence $a = a'$ does not hold in T. If, on the other hand, the units B and C are equal, then so are B' and C' (this is where the non-generativity comes into play). Thus the constants b and b' must have equal interpretations in T, and so $b = b'$ holds in T. In either case, the implication $a = a' \Rightarrow b = b'$ holds in T.

It is easy to see that the specification in Fig. 2 is not correct with respect to a generative semantics. What is more interesting is that there seems to be no obvious way of making it generatively correct, e.g. by using **let**-constructs for common subexpressions.

units
    A : { **sort** s; }
    F : { **sort** s; } $\rightarrow$ { **sort** s; **op** $a : s$; }
    G : { **sort** s; } $\rightarrow$ { **sort** s; **op** $a : s$; }
    H : { **sort** s; **op** $a : s$; } $\rightarrow$ { **sort** s; **ops** $a, b : s$; }
result
    • **let**
        $B = F(A)$
        $C = G(A)$
        $B' = H(B)$
        $C' = H(C)$
        C'' = **reduce** C' **by sort** $s \mapsto s$, **ops** $a' \mapsto a,\ b' \mapsto b$
        $T = B'$ **and** C''
    in
        T
implementing
    { **sort** s; **ops** $a, b, a', b' : s$; **axioms** $a = a' \Rightarrow b = b'$; }

Fig. 2. A correct architectural specification which is incorrect with respect to a generative semantics.

5 Proof of Theorem 1

An *extended environment* is a quadruple $e\delta = (D, b, p, lp)$, where:

- D is a presentation diagram with scheme I
- b is a partial injective function from identifiers to $|I|$
- p is a partial function from identifiers to parametric unit specifications
- the domains of b and p are disjoint
- lp labels some edges in I with elements of $dom(p)$ in such a way that, for any morphism m in its domain with $p(lp(m)) = ((\Sigma, \Phi), (\Delta, \Psi))$, $D(m)$ is an inclusion from Σ into Δ.

Intuitively, an extended environment consists of a diagram D whose meaning is as in the previous section, a map b which connects identifiers of non-parametric units to nodes in the diagram, a map p which gives the specifications of parametric units and a map lp such that $lp(m) = \mathit{ID}$ means that the edge m in the diagram D represents an application of the parametric unit ID.

Any extended environment $e\delta = (D, b, p, lp)$ defines a static environment (B, P), where B sends any identifier ID in $\mathit{dom}(b)$ to the signature of the presentation $D(b(\mathit{ID}))$, and P sends any identifier ID in $\mathit{dom}(p)$ to the parametric signature obtained by removing the sets of sentences from $p(\mathit{ID})$. This environment is denoted by $st(e\delta)$.

Let $e\delta = (D, b, p, lp)$ be an extended environment. We say that a family of models $\{N_i\}_{i \in |I|}$ consistent with D satisfies the *lp-condition*, if for any morphisms $m : i \to j$ and $n : k \to l$ with $lp(m) = lp(n)$, $N_i = N_k$ implies $N_j = N_l$. Thus, the lp-condition states that applying the same parametric unit to the same argument must yield the same result.

Let $\mathcal{B}$ send every ID in $\mathit{dom}(b)$ to a model in $\mathbf{Mod}(D(b(\mathit{ID})))$ and $\mathcal{P}$ send every ID in $\mathit{dom}(p)$ to a unit in $\mathbf{PUnit}(p(\mathit{ID}))$ (they *fit* $e\delta$). Now, let $\{N_i\}_{i \in |I|}$ be a model family consistent with D, such that $N_{b(\mathit{ID})} = \mathcal{B}(\mathit{ID})$ for $\mathit{ID} \in \mathit{dom}(b)$ and that for any m in $\mathit{dom}(lp)$ we have $\mathcal{P}(lp(m))(N_{dom(m)}) = N_{cod(m)}$. If such a family N exists and is unique, we denote it by $F_{e\delta}(\mathcal{B}, \mathcal{P})$. The interpretation here is that $F_{e\delta}(\mathcal{B}, \mathcal{P})$ extends the valuation of the declarations given by $\mathcal{B}$ and $\mathcal{P}$ to a valuation of all the terms represented in D (we note here that by definition it will always satisfy the lp-condition).

An extended environment $e\delta$ is *good* if $F_{e\delta}(\mathcal{B}, \mathcal{P})$ is defined for any $\mathcal{B}, \mathcal{P}$ fitting $e\delta$, and if, for any family $\{N_i\}_{i \in |I|}$ consistent with D and satisfying the lp-condition and any $m : i \to j$ in the domain of lp with $p(lp(m)) = ((\Sigma, \Phi), (\Delta, \Psi))$, we have $N_i \models_\Sigma \Phi$ and $N_j \models_\Delta \Psi$. Thus, an environment is good if all valuations on the declared units extend properly *and* if the diagram and lp-condition themselves ensure that all applications are correct.

Let $e\delta = (D, b, p, lp)$ be good. Then by $ctx(e\delta)$ we denote the set of environments of the form $\mathcal{B} \cup \mathcal{P}$, where $\mathcal{B}$ and $\mathcal{P}$ fit $e\delta$. This set is a context fitting $st(e\delta)$. Pick some node $i \in |\mathit{dom}(D)|$. Then by $mf_{e\delta,i}$ we denote the model function of type $(ctx(e\delta), \Sigma)$, where $D(i) = (\Sigma, \Phi)$, which sends any environment $e \in ctx(e\delta)$ to $(F_{e\delta}(\mathcal{B}, \mathcal{P}))_i$, where $\mathcal{B} = e|_{dom(b)}$ and $\mathcal{P} = e|_{dom(p)}$.

We say that an extended environment $e\delta' = (D', b, p, lp')$ *extends* $e\delta = (D, b, p, lp)$, if D' extends D and lp' extends lp.

Let $e\delta = (D, b, p, lp)$ and let $\alpha = \{\alpha_i : D(i) \to (\Sigma, \Phi)\}_{i \in |dom(D)|}$ be a sink. For any $\mathit{ID} \in \mathit{dom}(p)$, let V_{ID} be the set of vertices of the form $\mathit{dom}(m)$ for m with $lp(m) = \mathit{ID}$. Let $P = \{P_{\mathit{ID}}\}_{\mathit{ID} \in dom(p)}$ be a family where P_{ID} is a partition of V_{ID} (we denote the set of all such families by $part(lp)$). By $\mathcal{E}_{\alpha,P}$ we denote the following set of equality statements: for any pair (m, n) with $lp(m) = lp(n)$, if $\mathit{dom}(m)$ and $\mathit{dom}(n)$ are in one partitioning set of $P_{lp(m)}$ and $\mathit{cod}(m) \neq \mathit{cod}(n)$, then the statement $\alpha_{cod(m)} = \alpha_{cod(n)}$ is in $\mathcal{E}_{\alpha,P}$; if $\mathit{dom}(m)$ and $\mathit{dom}(n)$ are in different partitioning sets, then the statement $\alpha_{dom(m)} \neq \alpha_{dom(n)}$ is in $\mathcal{E}_{\alpha,P}$.

The following lemma is the reason for introducing the above sets of equality statements.

Lemma 1. *Let $e\delta = (D, b, p, lp)$ be good, $\alpha = \{\alpha_i : D(i) \to (\Sigma, \Phi)\}_{i \in |dom(D)|}$ be an amalgamating sink for D, i and j be nodes in the scheme of D such that $D(i)$ and $D(j)$ are presentations over compatible signatures and Ψ be a set of sentences over the signature Δ of $D(i)$. Then:*

$$\begin{aligned}&\text{for all } e \in ctx(e\delta),\ mf_{e\delta,i}(e) \models_\Delta \Psi \iff\\ &\qquad\text{either some parametric unit specification in } p \text{ is inconsistent,}\\ &\qquad\qquad\text{or for all } P \in part(lp) \text{ we have } \Phi \cup \mathcal{E}_{\alpha,P} \models_\Sigma \alpha_i(\Psi)\end{aligned}$$

and

$$\begin{aligned}&\text{for all } e \in ctx(e\delta),\ mf_{e\delta,i}(e) \text{ and } mf_{e\delta,j}(e) \text{ are amalgamable } \iff\\ &\qquad\text{either some parametric unit specification in } p \text{ is inconsistent,}\\ &\qquad\qquad\text{or for all } P \in part(lp) \text{ we have } \Phi \cup \mathcal{E}_{\alpha,P} \models_\Sigma \alpha_i \oplus \alpha_j\end{aligned}$$

□

Lemma 2. *Let $e\delta = (D, b, p, lp)$ be an extended environment. Let k_{ID} be the cardinality of the set $lp^{-1}(ID)$ for each identifier $ID \in dom(p)$. Then the cardinality of the set $part(lp)$ is bounded by $2^{\sum_{ID \in dom(p)} |k_{ID}^2 - 1|}$. Also, for any sink α over D and any $P \in part(lp)$, the cardinality of the set $\mathcal{E}_{\alpha,P}$ is bounded by $\frac{1}{2}\sum_{ID \in dom(p)} k_{ID}(k_{ID} - 1)$.*

□

We say that some parametric unit specification appears in the extended environment (D, b, p, lp) if it appears in p. We are now ready to state the main lemma needed for proving our theorem.

Lemma 3. *Assume that **I** has logical amalgamation. Let $e\delta$ be an extended environment such that if no unit specification in it is inconsistent, then it is good, and T a unit term. There exists an algorithm which either signals failure, or computes an extended environment $e\delta'$ extending $e\delta$, together with an object i of the scheme of the diagram of $e\delta'$, and a finite set A of intermediate proof obligations, and such that:*

$$\begin{aligned}&T \text{ is correct} \iff \text{the algorithm does not signal failure, and}\\ &\qquad\text{either some parametric unit specification in } e\delta \text{ is inconsistent,}\\ &\qquad\qquad\text{or all obligations in } A \text{ are true}\end{aligned}$$

Moreover, if T is correct and no parametric unit specification in $e\delta$ is inconsistent, then $e\delta'$ is good and the denotation of T in the context $st(e\delta)$, $ctx(e\delta)$ is $\pi_2(D(i))$, $mf_{e\delta',i}$, where D is the diagram of $e\delta'$.

Proof. The proof is by induction on the structure of T. We show the algorithm only, its correctness is easy but lengthy to check. We silently assume that failure propagates as if it were an exception. Let $e\delta = (D, b, p, lp)$.

The first case is $T =$ "ID". If $ID \notin dom(b)$, signal failure. Otherwise return $e\delta' = e\delta$, $i = b(ID)$, and $A = \emptyset$.

The second case is $T =$ "$ID(T_0)$". Let $e\delta_0 = (D_0, b, p, lp_0)$, i_0, A_0 be the result of the computation on $e\delta$ and T_0. If $ID \notin dom(p)$, signal failure. Otherwise, let $p(ID) = ((\Sigma, \Phi), (\Delta, \Psi))$ and $D_0(i_0) = (\Sigma', \Theta)$. If $\Sigma \neq \Sigma'$, signal failure. Otherwise add an edge labeled $\iota_{\Sigma,\Delta}$ with source i_0 and the target a new node i labelled by $(\Delta, \Psi \cup \iota_{\Sigma,\Delta}(\Theta))$ to D_0, obtaining D'. Let lp' be lp_0 with an additional label ID on this new edge. Let $\{\alpha_j : D_0(j) \to (\Gamma, \Omega)\}_{j \in |dom(D_0)|}$ be an amalgamating sink for D_0. Now return $e\delta' = (D', b, p, lp')$, i and $A = A_0 \cup \{\, \Omega, \mathcal{E}_{\alpha,P} \models_\Gamma \alpha_{i_0}(\Phi) \mid P \in part(lp_0) \,\}$.

The third case is $T =$ "T_1 **and** T_2". Let $e\delta_j = (D_j, b, p, lp_j)$, i_j, A_j be the result of the computation on $e\delta$ and T_j, $D_j(i_j) = (\Sigma_j, \Phi_j)$ for $j = 1, 2$. If Σ_1 and Σ_2 are incompatible, signal failure; otherwise, let $\Sigma = \Sigma_1 \cup \Sigma_2$. We may assume that the intersection of the schemes of D_1 and D_2 is the scheme of D. Let D_0 be the sum of D_1 and D_2. Let D' be D_0 with an additional node i labelled by $(\Sigma, \iota_{\Sigma_1,\Sigma}(\Phi_1) \cup \iota_{\Sigma_2,\Sigma}(\Phi_2))$ and edges from i_1 and i_2 to i, labelled by inclusions. Let $\{\alpha_j : D_0(j) \to (\Delta, \Psi)\}_{j \in |dom(D_0)|}$ be an amalgamating sink for D_0. Now, let $lp' = lp_1 \cup lp_2$, and return $e\delta' = (D', b, p, lp')$, i and $A = A_1 \cup A_2 \cup \{\, \Psi, \mathcal{E}_{\alpha,P} \models \alpha_{i_1} \oplus \alpha_{i_2} \mid P \in part(lp') \,\}$.

The fourth case is $T =$ "**reduce** T_0 **by** σ". Let $e\delta_0 = (D_0, b, p, lp_0)$, i_0, A_0 be the result of the computation on $e\delta$ and T_0. Let $D_0(i_0) = (\Sigma, \Phi)$. If $cod(\sigma) \neq \Sigma$, signal failure, else construct D' by adding to D_0 a node i labelled by $(dom(\Sigma), \emptyset)$ and an edge from i to i_0 labelled by σ. Return $e\delta' = (D', b, p, lp_0)$, i and A.

The last case is $T =$ "**let** $ID = T_1$ **in** T_2". If $ID \in dom(b)$ or $ID \in dom(p)$, signal failure. Let $e\delta_1 = (D_1, b, p, lp_1)$, i_1, A_1 be the result of the computation on $e\delta$ and T_1 and $e\delta_2 = (D_2, b\,[i_1/ID]\,, p, lp_2)$, i_2, A_2 its result on $(D_1, b\,[i_1/ID]\,, p, lp_1)$ and T_2. Now, return $e\delta' = (D_2, b, p, lp_2)$, $i = i_2$ and $A = A_1 \cup A_2$. □

Lemma 4. *For any correct sequence UDS of unit declarations, one can compute a good extended environment $e\delta$ such that the denotation of UDS is equal to $st(e\delta)$, $ctx(e\delta)$.*

□

Proof (Theorem 1). We will show the proof for the non-parametric case only, as the parametric case reduces to it easily. Let ASP be an architectural specification of the form "**units** UDS **result** $\bullet T$ **implementing** (Σ, Φ)". Define UDS' to be UDS extended by the declaration $F : (\Sigma, \Phi) \to (\Sigma, \Phi)$. Now compute an extended environment $e\delta$ for UDS' (Lemma 4). Next compute $e\delta'$, i, A for the term "$F(T)$" and $e\delta$, as in Lemma 3. Finally, as P return the set of parametric unit specifications appearing in $e\delta$ and as S the set A. That the bounds are as stated is a corollary of Lemma 2. □

6 Simplifying Intermediate Proof Obligations

In this section we show that positive equality statements in intermediate proof obligations are superfluous in an institution which has logical amalgamation. Essentially, the idea here is that any positive equality statement $\sigma = \tau$ can be substituted by a translation along a morphism which equalizes them; this idea can be generalized to work in any institution which has logical amalgamation, even if it has no coequalizers.

Let $\mathcal{E}$ be a finite set of positive Σ-statements. By $\eta_{\mathcal{E}} : (\Sigma, \emptyset) \to (\Gamma_{\mathcal{E}}, \Phi_{\mathcal{E}})$ we will denote a morphism in $Pres(\mathbf{I})$ such that the map $\cdot|_{\eta_{\mathcal{E}}}$ (with domain $\mathbf{Mod}_{\Gamma_{\mathcal{E}}}(\Phi_{\mathcal{E}})$) is in and onto $\mathbf{Mod}_{\Sigma}(\mathcal{E})$.

Proposition 4. *If the institution* $\mathbf{I}$ *has logical amalgamation, then the morphism* $\eta_{\mathcal{E}}$ *exists and can be computed for any finite set* $\mathcal{E}$ *of positive equality statements.*

Proof. Let $\mathcal{E} = \{\sigma_x = \tau_x\}_{x \in X}$. Take D to be a diagram with nodes v_x and a special node V, and morphisms $m_x, n_x : v_x \to V$ for any $x \in X$. The morphisms m_x are labeled with σ_x and n_x with τ_x, and the nodes are labelled accordingly. All sentence sets in the labels are empty. Now take the amalgamating sink of D and let $\eta_{\mathcal{E}} : D(V) \to (\Gamma_{\mathcal{E}}, \Phi_{\mathcal{E}})$ be the morphism in it associated with the node V. It is easy to see that this morphism induces a reduct which is in and onto $\mathbf{Mod}(\mathcal{E})$, since $\mathbf{Mod}(\mathcal{E})$ is in bijective correspondence with the set of all families consistent with D. □

For any Σ-statement γ and morphism $\eta : \Sigma \to \Gamma$ the *translation* of γ by η, denoted $\eta(\gamma)$, is defined by composing it with the elements of γ. Clearly, $M \models_{\Gamma} \eta(\gamma) \iff M|_{\eta} \models_{\Sigma} \gamma$, for any $M \in \mathbf{Mod}(\Gamma)$. This holds both for Σ-sentences and Σ-statements and we will call it the *extended satisfaction condition* (e.s.c.).

Proposition 5. *Let* $\mathcal{E}$ *and* $\mathcal{F}$ *be finite sets of positive and negative* Σ*-statements, respectively. Let* γ *be a* Σ*-sentence or* Σ*-statement and* Φ *a set of* Σ*-sentences. Then:*

$$\Phi \cup \mathcal{E} \cup \mathcal{F} \models_{\Sigma} \gamma \iff \Phi_{\mathcal{E}} \cup \eta_{\mathcal{E}}(\Phi \cup \mathcal{F}) \models_{\Gamma_{\mathcal{E}}} \eta_{\mathcal{E}}(\gamma)$$

Proof. For the implication from left to right, take any $M \models_{\Gamma_{\mathcal{E}}} \Phi_{\mathcal{E}} \cup \eta_{\mathcal{E}}(\Phi \cup \mathcal{F})$. By the e.s.c., $M|_{\eta_{\mathcal{E}}} \models_{\Sigma} \Phi \cup \mathcal{F}$ and by the definition of $\eta_{\mathcal{E}}$, $M|_{\eta_{\mathcal{E}}} \models_{\Sigma} \mathcal{E}$. Thus $M|_{\eta_{\mathcal{E}}} \models_{\Sigma} \gamma$ and, by the e.s.c., $M \models_{\Gamma_{\mathcal{E}}} \eta_{\mathcal{E}}(\gamma)$.

For the other implication, take any $M \models_{\Sigma} \Phi \cup \mathcal{E} \cup \mathcal{F}$. Then, by the definition of $\eta_{\mathcal{E}}$, there exists a model $N \models_{\Gamma_{\mathcal{E}}} \Phi_{\mathcal{E}}$ such that $M = N|_{\eta_{\mathcal{E}}}$. By the e.s.c., $N \models_{\Gamma_{\mathcal{E}}} \eta_{\mathcal{E}}(\Phi \cup \mathcal{F})$. Hence, $N \models_{\Gamma_{\mathcal{E}}} \eta_{\mathcal{E}}(\gamma)$, and, again by the e.s.c., $M \models_{\Sigma} \gamma$. □

Thus we have reduced the problem of checking intermediate proof obligations to checking obligations that do not involve any positive equality statements.

7 Transforming Simplified Intermediate Proof Obligations in FOL

In this section we aim at transforming the intermediate proof obligations generated in Theorem 1 and simplified in the previous section into simple obligations. We do this in the framework FOL of first-order logic, although we do conjecture that the approach is quite universal and would be successful in many similar logics as well. In the case of $|FOL|$ we obtain a decidability result.

Our transformations of intermediate proof obligations into simple obligations will be based on the following lemma. This lemma states that, in first-order logic with negative equalities, the only way to ensure that two different sorts have always equal carriers is to force these carriers to be empty.

Lemma 5. *Let $\mathcal{E}$ be a set of negative Σ-statements in FOL. Take a model $M \models_\Sigma \mathcal{E}$. There exists a model $N \models_\Sigma \mathcal{E}$ satisfying precisely the same Σ-sentences as M such that, for any pair of different sorts s, s' in Σ, either the carriers of both of them are empty in M or they are different in N.*

Proof. As N one can take a model where elements of the carrier of a sort s are pairs (s, x), with x an element of its carrier in M. The interpretations of function symbols are defined straightforwardly. Clearly, N is isomorphic to M, thus it satisfies the same Σ-sentences. Also, non-empty carriers are obviously different. It is clear that any negative statement true in M is true in N as well. $\square$

Theorem 2. *Any amalgamability obligation in FOL can be transformed into an equivalent satisfaction obligation.*

Proof. It suffices to consider an amalgamability obligation of the form $\Phi \cup \mathcal{E} \models_\Sigma \sigma \oplus \tau$ where $\mathcal{E}$ consists of negative statements only ($\sigma : \Delta \to \Sigma, \tau : \Gamma \to \Sigma$, Δ and Γ are compatible). Obviously, amalgamability in FOL means that for any sort s in the intersection of Δ and Γ, the carriers of $\sigma^S(s)$ and $\tau^S(s)$ in any model of $\Phi \cup \mathcal{E}$ are equal. The same applies for any function symbol $f : s_1 \times \cdots \times s_n \to s_{n+1}$. In the proof we formalize the observation that in the case of sorts, this can happen either if $\sigma^S(s) = \tau^S(s)$ or if both of these sorts have empty carriers; in the case of function symbols, we have two subcases. First, if $\sigma^S(s_i) \neq \tau^S(s_i)$ for some $1 \leq i \leq n+1$, then by the above observation, the carriers of both of these sorts are empty and, hence, both functions $\sigma^F(f)$ and $\tau^F(f)$ are empty and equal. If, on the other hand, the respective pairs of sorts are always equal, then we may write an axiom which states that $\sigma^F(f)$ and $\tau^F(f)$ are equal.

Now, let Ψ be the following set of Σ-sentences:

- for any sort s appearing in both Δ and Γ such that $\sigma^S(s) \neq \tau^S(s)$, the sentence ψ_s of the form "$(\forall x : \sigma^S(s) \cdot \neg(x = x)) \wedge (\forall x : \tau^S(s) \cdot \neg(x = x))$";
- for any function symbol f appearing in both Δ and Γ with profile $s_1 \times \ldots \times s_n \to s_{n+1}$ and such that $\sigma^S(s_i) = \tau^S(s_i)$ for $1 \leq i \leq n+1$, but $\sigma^F(f) \neq \tau^F(f)$, the sentence ψ_f of the form "$\forall x_1 : \sigma^S(s_1) \cdot \ldots \forall x_n : \sigma^S(s_n) \cdot \sigma^F(f)(x_1, \ldots, x_n) = \tau^F(f)(x_1, \ldots, x_n)$".

We claim that the following equivalence holds:

$$\Phi \cup \mathcal{E} \models_{\Sigma} \sigma \oplus \tau \iff \Phi \cup \mathcal{E} \models_{\Sigma} \Psi$$

It remains now to prove the above claim.

Assume the right side of the equivalence holds. Take $M \models_{\Sigma} \Phi \cup \mathcal{E}$. Then $M \models_{\Sigma} \Psi$. We will prove that $M|_{\sigma}$ and $M|_{\tau}$ are amalgamable. Toward this end we will check that common sorts and function symbols are interpreted identically in both models.

Let s be a sort appearing in both Δ and Γ. We are to prove that the interpretations of $\sigma^S(s)$ and $\tau^S(s)$ in M are identical. If $\sigma^S(s) = \tau^S(s)$, then this is trivial. Assume $\sigma^S(s) \neq \tau^S(s)$. Then $\psi_s \in \Psi$ and $M \models_{\Sigma} \psi_s$. Then the carriers of $\sigma^S(s)$ and $\tau^S(s)$ are indeed equal in M, since they are both empty.

Let f be a function symbol with profile $s_1 \times \ldots \times s_n \to s_{n+1}$ in both Δ and Γ. If for some $1 \leq i \leq n+1$ we have $\sigma^S(s_i) \neq \tau^S(s_i)$, then, as above, the carriers of sorts $\sigma^S(s_i)$ and $\tau^S(s_i)$ must be empty. Hence, the interpretations of $\sigma^F(f)$ and $\tau^F(f)$ must be empty functions and are thus equal. If, on the other hand, $\sigma^S(s_i) = \tau^S(s_i)$ for all $1 \leq i \leq n+1$, then we have $M \models_{\Sigma} \psi_f$, which ensures that the interpretations of $\sigma^F(f)$ and $\tau^F(f)$ are equal.

This proves the implication to the left. For the converse, assume $\Phi \cup \mathcal{E} \models_{\Sigma} \sigma \oplus \tau$ and take $M \models_{\Sigma} \Phi \cup \mathcal{E}$. We are to prove that $M \models_{\Sigma} \Psi$.

First take some $\psi_s \in \Psi$. According to Lemma 5, there exists a model $N \models_{\Sigma} \Phi \cup \mathcal{E}$ such that either the carriers of both $\sigma^S(s)$ and $\tau^S(s)$ are empty in M, or they are different in N. Since $N \models_{\Sigma} \Phi \cup \mathcal{E} \models_{\Sigma} \sigma \oplus \tau$, they cannot be different. Thus, they are empty. That is, $M \models_{\Sigma} \psi_s$.

Now take $\psi_f \in \Psi$. It is apparent that, since the interpretations of $\sigma^F(f)$ and $\tau^F(f)$ coincide in M, we must have $M \models_{\Sigma} \psi_f$. This finishes the proof. $\square$

Theorem 3. *Any satisfaction obligation in FOL can be transformed into an equivalent simple obligation.*

Proof. It is sufficient to consider an obligation of the form $\mathcal{E} \models_{\Sigma} \psi$, where $\mathcal{E}$ consists of negative statements only. This is because any obligation of the form $\Phi \cup \mathcal{E} \models_{\Sigma} \Psi$ can be transformed into $\mathcal{E} \models_{\Sigma} \bigwedge \Phi \to \bigwedge \Psi$.

The idea of this proof is that a negative equality obligation consisting of morphisms $\sigma, \tau : \Delta \to \Sigma$ can influence satisfaction of first-order sentences only as follows: if all sorts in Δ which are mapped to different sorts in Σ by σ and τ have empty carriers, then for some function symbol f the sorts in whose profile are mapped in the same way by σ and τ, the interpretations of $\sigma^F(f)$ and $\tau^F(f)$ are different. This intuition is formalized below.

For any statement e of the form $\sigma \neq \tau$, $\sigma, \tau : \Delta \to \Sigma$, define the following Σ-sentences:

- for any sort s in Δ such that $\sigma^S(s) \neq \tau^S(s)$, $\rho_{e,s} =$"$(\forall x : \sigma^S(s) \cdot \neg(x = x)) \wedge (\forall x : \tau^S(s) \cdot \neg(x = x))$"; the set of such sorts is denoted by S_e;
- for any function symbol f in Δ with profile $s_1 \times \ldots \times s_n \to s_{n+1}$, if for all $1 \leq i \leq n+1$ we have $\sigma^S(s_i) = \tau^S(s_i)$, then $\rho_{e,f} =$"$\exists x_1 : \sigma^S(s_1) \cdot \ldots \exists x_n :$

$\sigma^S(s_n) \cdot \neg(\sigma^F(f)(x_1, \ldots, x_n) = \tau^F(f)(x_1, \ldots, x_n))$"; the set of such function symbols is denoted by F_e;
– $\omega_e =$ "$\bigwedge_{s \in S_e} \rho_{e,s} \implies \bigvee_{f \in F_e} \rho_{e,f}$".

Let Ω be the set of all sentences ω_e for $e \in \mathcal{E}$. We claim that ψ the following equivalence holds:

$$\mathcal{E} \models_\Sigma \psi \iff \Omega \models_\Sigma \psi$$

The proof of the above claim proves the whole theorem.

To prove the implication from right to left, we will prove that $\mathcal{E} \models_\Sigma \Omega$. Assume $M \models_\Sigma \mathcal{E}$. Take any $e \in \mathcal{E}$ of the form $\sigma \neq \tau$ and $\omega_e \in \Omega$. If M satisifies the premise of ω_e then every pair of different sorts $\sigma^S(s)$, $\tau^S(s)$ has empty carriers in M. Thus, there must be some function symbol f such that $\sigma^F(f)$ and $\tau^F(f)$ have different interpretations in M. If a sort s is in the profile of f then it must be mapped identically by σ and τ; otherwise, both of the mapped sorts would be empty, hence the interpretations of $\sigma^F(f)$ and $\tau^F(f)$ would be empty and equal. But the existence of such a symbol f means precisely that $\rho_{e,f}$ is satisfied in M. So, $M \models_\Sigma \Omega$ and then $M \models_\Sigma \psi$.

Now we prove the implication from left to right. Assume $\mathcal{E} \models_\Sigma \psi$ and take a model $M \models_\Sigma \Omega$. According to Lemma 5, there exists a model N satisfying the same Σ-sentences as M and such that for any two sorts having non-empty carriers in M, their carriers in N differ. If we had $N \models_\Sigma \mathcal{E}$, then we would also have $M \models_\Sigma \psi$, which would end the proof. So suppose $N \not\models_\Sigma \mathcal{E}$. Thus there exists a statement $e \in \mathcal{E}$ of the from $\sigma \neq \tau$ such that $N \models_\Sigma \sigma = \tau$. To get a contradiction, it suffices to prove that then $N \not\models_\Sigma \omega_e$. Take any $s \in S_e$. The carriers of $\sigma^S(s)$ and $\tau^S(s)$ in N must be equal, hence they are empty. Thus the premise of ω_e is satisfied in N. If ω_e were satisfied in N, there would exist a function symbol $f \in F_e$ such that $N \models_\Sigma \rho_{e,f}$. But this would mean that the symbols $\sigma^F(f)$ and $\tau^F(f)$ have different interpretations in N, which contradicts $N \models_\Sigma \sigma = \tau$. Thus we conclude that the initial assumption that $N \not\models_\Sigma \mathcal{E}$ was false, which finishes the proof. □

It is perhaps worth pointing out that intermediate proof obligations in $|FOL|$ are decidable, making correctness in $|FOL|$ decidable. Because in $|FOL|$ we have no sentences at all, the satisfaction obligations are all trivially true. We also have:

Proposition 6. *Amalgamability obligations in $|FOL|$ are decidable.*

Proof. We may assume the amalgamability obligation to be of the form $\mathcal{E} \models_\Sigma \sigma \oplus \tau$, where $\mathcal{E}$ consists of negative statements only. This obligation is true if and only if:

– *either* there exists a statement of the form $\rho \neq \rho$ in $\mathcal{E}$,
– *or* the morphisms σ and τ coincide on the intersection of their domains.

It is obvious that the above conditions are sufficient for the obligation to be true. On the other hand, if σ and τ do not coincide on the intersection of their domains, then $\mathcal{E}$ must be inconsistent. This means that some statement of the form $\rho \neq \rho$ is an element of $\mathcal{E}$. □

8 Static and Strict Correctness

In Section 3 we listed the four conditions that have to be checked during the verification of an architectural specification. It is easy to see that in the case of FOL checking each of these conditions is undecidable. This is not at all unexpected in the case of conditions 1 (does the result unit satisfy the **implementing** specification?), and 2 (does a unit to which a parametric unit is applied satisfy the specification which defines the domain of this parametric unit?). However, conditions 3 and 4 have to do with amalgamability and do not involve any specifications or axioms. So, one would want to have these conditions checked automatically, during static analysis, as in the intuitively analogous case of checking sharing conditions in the module system of the SML programming language.

Toward this end, define, for any architectural specifiation ASP over $\mathbf{I}$, the architectural specification $|ASP|$ over $|\mathbf{I}|$, obtained by removing all sentences from ASP; thus, $|ASP|$ is the "static" version of ASP — meaning that it involves signatures and signature morphisms only. Let us say that ASP is *statically correct* if $|ASP|$ is correct in the usual sense (as a matter of fact, generativity problems aside, an architectural specification is statically correct if and only if it has a denotation with respect to the *extended static semantics* of [SMTKH01]).

Of course, checking static correctness is no harder than checking regular correctness. In particular, checking static correctness in FOL is decidable, thanks to Prop. 6. Similar results are true for many institutions of interest. Moreover, the following holds:

Proposition 7. *If ASP is statically correct, then in the elaboration of ASP conditions 3 and 4 will always be satisfied.*

□

units
 $A : \{$**sort** $s;$ **op** $a : s; \}$
 $F : \{$**sort** $s;$ **op** $a : s; \} \rightarrow \{$**sort** $s;$ **ops** $a, b : s;$ **axiom** $a = b; \}$
 $G : \{$**sort** $s;$ **op** $a : s; \} \rightarrow \{$**sort** $s;$ **ops** $a, b : s;$ **axiom** $a = b; \}$
result
 $\bullet\, F(A)$ **and** $G(A)$
implementing
 $\{$**sort** $s;$ **ops** $a, b : s; \}$

Fig. 3. A correct architectural specification which is not statically correct.

Call an architectural specification *strictly correct* if it is both correct and statically correct. We propose to require architectural specifications to be strictly correct. Then verifying a specification would consist of first checking (hopefully automatically) its static correctness, and then discharging only the *satisfaction* proof obligations generated by the algorithm presented in this paper. This procedure would be sound, since static correctness ensures that all the amalgamability obligations generated will be satisfied. However, it is not complete, in the sense that not all correct architectural specifications are statically correct. In Fig. 3 we have an example of an architectural specification which is correct, but is not statically correct (and hence not strictly correct). In the following discussion we use the notation $T.x$ for the component x of the model denoted by the term T in a given environment. In our example, we know that $A.s = F(A).s = G(A).s$ and $A.a = F(A).a = G(A).a$, and the axioms enforce $F(A).b = F(A).a$ and $G(A).b = G(A).a$. This causes the amalgamation $F(A)$ **and** $G(A)$ to be correct, since $F(A).a = G(A).a$ and $F(A).b = G(A).b$. If we remove the axioms, then $F(A).b$ and $G(A).b$ may be unrelated and hence the models $F(A)$ and $G(A)$ will not be amalgamable. In fact, we consider this incompletness valuable, since we feel that architectural specifications like the one in Fig. 3 are really flawed anyway.

As has already been said, the problem of checking static correctness is easily decidable for FOL and many other institutions. Unfortunately, the institution of CASL is an exception. The logic of CASL involves a form of subsorting where the inclusion of a subsort element into a supersort is performed by means of an (almost) arbitrary injective map between the respective carriers. These injective maps are required to satisfy only one simple compatability axiom: the injection of s into s' composed with the injection of s' into s'' must be equal to the injection of s into s''. These compatibility conditions are the reason of two related phenomena: lack of logical amalgamation (cf. [SMTKH01][1]), and undecidability of static correctness (see [KHTSM01], where this is proved and algorithms for checking static correctness in practically relevant cases are developed).

9 Conclusion and Future Work

In this paper we have shown — for a very general case — how to reduce the problem of verifying an architectural specification to the problem of checking some intermediate proof obligations. We have also shown — for the case of first-order logic — how to transform these intermediate proof obligations into simple obligations.

However, some work still needs to be done, if we aim at verifying architectural specifications in CASL. First, we need to generalize the above results to partial logic with sort generation constraints. Next, we need to check that the whole construction works for structured specifications as well. These steps should be fairly straightforward. Some problems even turn out simpler in CASL, since non-empty carriers are not admitted in its model theory.

The problems that arise with the introduction of subsorts require more work. The main obstacle here is the lack of logical amalgamation, which causes verification for CASL *without* any sentences (i.e., static correctness) to be undecidable already. What is needed here is an embedding of the CASL logic into a richer logic (*enriched* CASL) which does have the logical amalgamation property. This

[1] Actually, only lack of amalgamation has been discussed there, but the same argument can be used to show lack of logical amalgamation.

embedding will probably go along the lines of [SMT01]. A drawback of using such an embedding would be that as a result one would not be able to reason about CASL architectural specifications in pure CASL.

All this would allow us to generate obligations in enriched CASL for the sublanguage of architectural specifications presented in this paper and for the full CASL logic. In order to support full CASL architectural specifications, the additional architectural constructs, such as translation or fitting-morphisms in applications, will have to be taken care of.

Acknowledgments I would like to thank Andrzej Tarlecki for introducing me to the subject and for collaboration on architectural specifications in general. I am also grateful to Bartek Klin, as well as Till Mossakowski and Lutz Schröder, for many discussions and joint work on related topics. This research has been partially supported by Polish State Committee for Research (KBN) grant no. 7 T11C 002 21 and by ESPRIT working group 29432 (CoFI WG).

References

[BCH99] M. Bidoit, M. V. Cengarle, R. Hennicker: Proof Systems for Structured Specifications and Their Refinements. In E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner (eds.): Algebraic Foundations of Systems Specification, pp. 385–434. Springer, 1999.

[Borz98] T. Borzyszkowski: Completeness of a logical system for structured specifications. In F. Parisi Presicce (ed.): Recent Trends in Algebraic Development Techniques, Selected Papers, 12th Intl. Workshop, Proc. WADT'97, LNCS 1376, pp. 107–121. Springer, 1998.

[BST99] M. Bidoit, D. Sannella, A. Tarlecki. Architectural Specifications in CASL. In: Proc. 7th Intl. Conference on Algebraic Methodology and Software Technology (AMAST'98), LNCS 1548, pp. 341–357. Springer, 1999; final version to appear in Formal Aspects of Computing.

[CASL] E. Astesiano, M. Bidoit, H. Kirchner, B. Krieg-Brückner, P. Mosses, D. Sannella, A. Tarlecki. CASL: The Common Algebraic Specification Language. Theoretical Computer Science (to appear).

[CASL99] CoFI: CASL — Summary, version 1.0. In: Documents/CASL/Summary, [CoFI]. 1999.

[CASL00] CoFI: CASL — Semantics, version 1.0. In: Documents/CASL/Semantics, note S-9, [CoFI]. 2000.

[CoFI] CoFI, The Common Framework Initiative for Algebraic Specification and Development. Documents accessible at `http://www.brics.dk/Projects/CoFI` and `ftp://ftp.brics.dk/Projects/CoFI`.

[GB84] J. Goguen, R. Burstall: Some fundamental algebraic tools for the semantics of computation. Part 1: Comma categories, colimits, signatures, and theories. Theoretical Computer Science 31(2), pp. 175–209 (1984)

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

[KHTSM01] B. Klin, P. Hoffman, A. Tarlecki, L. Schröder, T. Mossakowski: Checking Amalgamability Conditions for CASL Architectural Specifications. In J. Sgall, A. Pultr, P. Kolman (eds.): Proc. 26th Intl. Symp. on Mathematical Foundations of Computer Science (MFCS'01), LNCS 2136, pp. 451–463. Springer, 2001.

[Pad99] P. Padawitz: Proof in Flat Specifications. In E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner (eds.): Algebraic Foundations of Systems Specification, pp. 321–384. Springer, 1999.

[Pau96] L. Paulson: ML for the Working Programmer. Cambridge University Press, 1991.

[SMT01] L. Schröder, T. Mossakowski, A. Tarlecki: Amalgamation in CASL via Enriched Signatures. In F. Orejas, P. Spirakis, J. van Leeuwen (eds.): Proc. 28th Intl. Coll. on Automata, Languages and Programming (ICALP'01), LNCS 2076, pp. 993–1004. Springer, 2001.

[SMTKH01] L. Schröder, T. Mossakowski, A. Tarlecki, B. Klin, P. Hoffman: Semantics of Architectural Specifications in CASL. In H. Hussmann (ed.): Proc. 4th Intl. Conf. on Fundamental Approaches to Software Engineering (FASE'01), LNCS 2029, pp. 253–268. Springer, 2001.

[ST88] D. Sannella, A. Tarlecki: Specifications in an Arbitrary Institution. Information and Computation vol. 76, 2/3, pp. 165–210 (1988).

[ST97] D. Sannella, A. Tarlecki: Essential Concepts of Algebraic Specification and Program Development. Formal Aspects of Computing 9, pp. 229–269 (1997).

[Tar99] A. Tarlecki: Institutions: An Abstract Framework for Formal Specifications. In E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner (eds.): Algebraic Foundations of Systems Specification, pp. 105–130. Springer, 1999.