Amalgamation in the Semantics of CASL

Lutz Schröder [a,1] Till Mossakowski [a,2] Andrzej Tarlecki [b,c,3]
Bartek Klin [d] Piotr Hoffman [b,3]

[a] *BISS, Department of Computer Science, Bremen University, P.O.Box 330 440, D-28334 Bremen, Germany*

[b] *Institute of Informatics, Warsaw University*

[c] *Institute of Computer Science, Polish Academy of Sciences*

[d] *BRICS, Århus University*

Abstract

We present a semantics for architectural specifications in CASL, including an extended static analysis compatible with model-theoretic requirements. The main obstacle here is the lack of amalgamation for CASL models. To circumvent this problem, we extend the CASL logic by introducing enriched signatures, where subsort embeddings form a category rather than just a preorder. The extended model functor satisfies the amalgamation property as well as its converse, which makes it possible to express the amalgamability conditions in the semantic rules in static terms. Using these concepts, we develop the semantics at various levels in an institution-independent fashion. Moreover, amalgamation for enriched CASL means that a variety of results for institutions with amalgamation, such as computation of normal forms and theorem proving for structured specifications, can now be used for CASL.

Key words: Algebraic specification, architectural specification, amalgamation, institutions, CASL.

Introduction

The use of formal methods within the development process of software systems is important especially for complex or safety-critical systems. Here, 'formal'

[1] Research supported by the DFG project HasCASL (KR 1191/7-1)
[2] Research supported by the DFG project MULTIPLE (KR 1191/5-1)
[3] Research partially supported by KBN grant 7 T11C 002 21

implies that the specification is based on a logical system, with a rigorously defined syntax and semantics. It has been recognized that structuring operations for the specification of software systems can be studied largely independently of the underlying logical system; the most prominent formalization of this concept is the notion of *institution* [20]. The recently developed language CASL [3,14,15] has an institution-independent semantics.

The principal motivation for developing the methods presented below is their application in an institution-independent semantics for CASL architectural specifications [43]. While many present-day algebraic specification languages (see e.g. [44,18,21,1,40]) provide operations for building large specifications in a structured fashion from smaller and simpler ones [11], necessarily different [38] mechanisms for describing the modular structure of software systems under development are a rather less common feature. Such a mechanism is supplied in CASL in the shape of *architectural specifications.*

The main idea is that architectural specifications describe branching points in system development by indicating units (modules) to be independently developed and showing how these units, once developed, are to be put together to produce the overall result. Semantically, units are viewed as given models of specifications, to be used as building blocks for models of more complex specifications, e.g. by amalgamating units or by applying parametrized units. Architectural specifications have been introduced and motivated in [6]. Here, we work with a simple subset of CASL architectural specifications, which is expressive enough to study the main mechanisms and features of the semantics.

A major problem with the semantics is the failure of the so-called amalgamation property in the CASL institution. Roughly speaking, this property states that models of given signatures can be combined to yield a uniquely determined model of a compound signature, provided that the original models coincide on common components.

The amalgamation property (called ‘exactness’ in [17]) is a major technical assumption in the study of specification semantics [39] and is important in many respects. To give a few examples: it allows the computation of normal forms for specifications [5,10], and it is a prerequisite for good behaviour w.r.t. parametrization [19] and conservative extensions [17,36]. The proof system for development graphs with hiding [32], which allow a management of change for structured specifications, is sound only for institutions with amalgamation. A Z-like state based language has been developed over an arbitrary institution with amalgamation [4].

Many standard logical systems (like multisorted equational [18] and first-order logic [29] with the respective standard notions of model) admit amalgamation, so quite often this property is taken for granted in work on specification formalisms. However, the expected amalgamation property fails in the setting of order-sorted algebra (when subsort relations are interpreted as arbitrary injections), in particular in the CASL institution. Generally, the amalgamation property may fail if there are components in the models that are not named in the signatures, e.g. the implicit universe in unsorted first-order logic (which destroys amalgamation for disjoint unions of signatures), the implicit set of kinds in LF [22], the implicit set of worlds in temporal or modal logics, or the implicit subsort injections in the CASL logic.

The lack of amalgamation makes it difficult to statically ensure that models can be indeed put together as prescribed by an architectural design.

The semantics of CASL architectural specifications is developed in three stages in order to circumvent this problem. The first step is a purely model-theoretic semantics. This semantics does not depend on the amalgamation property; rather, amalgamability is just *required* whenever it is needed. This makes the definition of the semantics as straightforward and permissive as possible, but leaves the user with the task of actually checking these model-theoretic requirements. Thus, the natural second step is to give a semantics of architectural specifications in terms of diagrams which express the sharing that is present in the unit declarations and definitions. This allows us to reformulate the model-theoretic amalgamability conditions in 'almost' static terms. In order to be able to make the static character of these conditions explicit in the third step, we need the amalgamation property. Since we cannot expect that this property holds in the given institution (as the case of the CASL institution shows), we assume that the latter is embedded, in a way that is compatible with the respective model theories, in an institution that does have amalgamation. Using this representation, we can restate the amalgamability conditions as entirely static factorization properties of signature morphisms.

In order for this institution-independent semantics of architectural specifications to be applied to the case of the CASL institution, one problem remains to be solved: the CASL institution has to be embedded into an institution that enjoys the amalgamation property.

The source of the failure of amalgamation for the CASL institution are the subsorts, or, more precisely, the implicit compatibility requirements for subsort embeddings. With this in mind, the main idea in the definition of the required extended institution is to generalize pre-orders of sorts to *categories* of sorts, i.e. to admit several different subsort embeddings between two given sorts; this gives rise to the notion of *enriched* CASL signature. With the amalgamation property available via enriched signatures, most of the results cited above can be applied to CASL by forming compound (colimit) signatures in the extended signature category. Checking the factorization properties arising in the third step of the semantics of architectural specifications requires (institution-specific) tool support; a calculus for this purpose is developed in [25].

The material is organized as follows: Sections 1 and 2 provide an introduction to CASL architectural sepcifications and the relevant institution-theoretic concepts, respectively. The basic and extended institution-independent semantics of architectural specifications is laid out in Sections 3 and 4. In Sections 5 and 6, the notion of (standard) CASL signature is recalled, and the notion of enriched CASL signature is introduced. Section 7 is devoted to the proof that the colimit property in the extended signature category is not only a sufficient, but also a necessary criterion for the amalgamation property. Finally, the results obtained are applied to the problem of statically analysing architectural specifications in Section 8.

We refer to [2,27] for categorical terminology left unexplained here.

1 Architectural Specifications

The specification language CASL (*Common Algebraic Specification Language*) has been designed by CoFI, the international *Common Framework Initiative for Algebraic Specification and Development* [14]. Its underlying logical system is based on models that feature multiple (preordered) sorts, predicates, partial and total operations and subsort embeddings (see Sect. 5 for more details), first order sentences built out of equality, predicate application and term definedness as atomic formulae, and sort generation constraints. CASL then provides convenient mechanisms to built structured and architectural specifications and to group them in libraries [3,15,16].

As indicated above, architectural specifications in CASL provide a means of stating how implementation units are used as building blocks for larger components. (Dynamic interaction between modules and dynamic changes of software structure are currently beyond the scope of this approach.)

Units are represented as names to which a specification is assigned. Such a named unit is to be thought of as a given model of the specification. Units may be parametrized, where specifications are assigned to both the parameters and the result. The result specification is required to extend the parameter specifications. A parametrized unit is to be understood as a function which, given models of the parameter specifications, outputs a model of the result specification; this function is required to be *persistent* in the sense that reducing the result to the parameter signatures reproduces the parameters.

Units can be assembled via unit expressions which may contain operations such as renaming or hiding of symbols, amalgamation of units, and application of a parametrized unit. Terms containing such operations will only be defined if symbols that are identified, e.g. by renaming them to the same symbol or by amalgamating units that have symbols in common, are also interpreted in the same way in all ‘collective’ models of the units defined so far.

An architectural specification consists in declaring or defining a number of units, as well as in providing a way of assembling them to yield a result unit.

Example 1 A (fictitious) specification structure for a compiler might look roughly as follows:

$$\begin{tikzcd}
 & \textsc{Identifier} \arrow[dr] & & \textsc{List} \arrow[dl] & \\
 & & \textsc{ProgramText} \arrow[dl] \arrow[dr] & & \\
 & \textsc{AbstractSyntax} \arrow[dl] \arrow[dr] & & \textsc{SymbolTable} \arrow[dl] \arrow[dr] & \\
\textsc{Parser} \arrow[drr] & & \textsc{StaticAnalyser} \arrow[d] & & \textsc{CodeGenerator} \arrow[dll] \\
 & & \textsc{Compiler} & &
\end{tikzcd}$$

(The arrows indicate the extension relation between specifications.) An architectural specification of the compiler in CASL [15] might have the following form:

arch spec BUILDCOMPILER =
units *I* : IDENTIFIER **with sorts** *Identifier*, *Keyword*;
L : ELEM $\rightarrow$ LIST[ELEM];
IL = *L*[*I* **fit sort** *Elem* $\mapsto$ *Identifier*]
KL = *L*[*I* **fit sort** *Elem* $\mapsto$ *Keyword*]
PT : PROGRAMTEXT **given** *IL*, *KL*;
AS : ABSTRACTSYNTAX **given** *PT*;
ST : SYMBOLTABLE **given** *PT*;
P : PARSER **given** *AS*;
SA : STATICANALYSER **given** *AS*, *ST*;
CG : CODEGENERATOR **given** *ST*
result *P* **and** *SA* **and** *CG*
end

(Here, the keyword **with** is used to just list some of the defined symbols. The keyword **given** indicates imports.) According to the above specification, the parser, the static analyser, and the code generator would be constructed building upon a given abstract syntax and a given mechanism for symbol tables, and the compiler would be obtained by just putting together the former three units. Roughly speaking, this is only possible (in a manner that can be statically checked) if all symbols that are shared between the parser, the static analyser and the code generator already appear in the units for the abstract syntax or the symbol tables — otherwise, incompatibilities might occur that make it impossible to put the separately developed components together. For instance, if both StaticAnalyser and CodeGenerator declare an operation *lookup* that serves to retrieve symbols from the symbol table, then the corresponding implementations might turn out to be substantially different, so that the two components fail to be compatible. Of course, this points to an obvious flaw in the architecture: *lookup* should have been declared in SymbolTable.

In order to keep the presentation as simple as possible, we consider a modified sublanguage of Casl architectural specifications:

Architectural specifications: ASP ::= **arch spec** DD^* **result** T;
$DD ::= Dcl \mid Dfn$
An architectural specification consists of a list of unit declarations and definitions followed by a unit result term.

Unit declarations: $Dcl ::= U\colon SP \mid U\colon SP_1 \xrightarrow{\tau} SP_2$
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 τ — we assume that the definition of specifications and some syntactic means to present signature morphisms are given elsewhere. (By resorting to explicit signature morphisms, we avoid having to discuss the details of signature inclusions.)

Unit definitions: $Dfn ::= U = T$
A unit definition introduces a (non-parametrized) unit and gives its value by a unit term.

Unit terms: $T ::= U \mid U[T$ **fit** $\sigma] \mid T_1$ **with** σ_1 **and** T_2 **with** σ_2
A unit term is either a (non-parametrized) unit name, or a (parametrized) unit application with an argument that fits via a signature morphism σ, or an amalgamation of units via signature morphisms σ_1 and σ_2 required to form a sink (have a common target signature); we thus slightly generalize the amalgamation operation of Casl here, again avoiding the need to present the details of signature unions (cf. [31]). A **with**-clause or a **fit**-clause may be omitted when the associated signature morphism is an identity.

Imports as used in Example 1 can be regarded as syntactical sugar for a parametrized unit which is instantiated only once: given $U_1 : SP_1$,

$U_2 : SP_2$ **given** U_1

abbreviates

$U_2' : SP_1 \to SP_2$;
$U_2 = U_2'[U_1]$.

2 Institutions

The considerations ahead deal with aspects of the notion of *institution* [20]. An institution $\mathcal{I}$ consists of a category **Sign** of *signatures*, a *model functor*

$$\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbf{CAT},$$

where **CAT** denotes the quasicategory of categories and functors [2], and further components which formalize sentences and satisfaction. In this context, we need only the signature category and the model functor. Given a signature Σ, $\mathbf{Mod}(\Sigma)$ is called the category of (Σ-)*models*. If $\sigma : \Sigma_1 \to \Sigma_2$ is a signature morphism, then $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma_2) \to \mathbf{Mod}(\Sigma_1)$ is called a *reduct functor*; we say that a Σ_2-model M *reduces* to $\mathbf{Mod}(\sigma)(M)$, which is often denoted by $M|_\sigma$.

As indicated in the introduction, a central property that an institution may or may not satisfy is the amalgamation property. The most important special case is amalgamation for pushouts in the signature category, which are prominently used for instance in instantiations of parametrized specifications. Given a pushout

$$\begin{tikzcd}
\Sigma \arrow[r] \arrow[d] & \Sigma_1 \arrow[d] \\
\Sigma_2 \arrow[r] & \Sigma_R
\end{tikzcd}$$

in **Sign**, the amalgamation property requires that that any pair $(M_1, M_2) \in \mathbf{Mod}(\Sigma_1) \times \mathbf{Mod}(\Sigma_2)$ that is *compatible* in the sense that M_1 and M_2 reduce to the same Σ-model can be *amalgamated* to a unique Σ_R-model M (i.e., there exists a unique $M \in \mathbf{Mod}(\Sigma_R)$ that reduces to M_1 and M_2, respectively), and similarly for model morphisms. More formally, this means that the above pushout diagram is mapped by **Mod** to a pullback

$$\begin{tikzcd}
\mathbf{Mod}(\Sigma) & \mathbf{Mod}(\Sigma_1) \arrow[l] \\
\mathbf{Mod}(\Sigma_2) \arrow[u] & \mathbf{Mod}(\Sigma_R) \arrow[l] \arrow[u]
\end{tikzcd}$$

of categories.

More generally, a cocone for a diagram in **Sign** is called *amalgamable* if it is mapped to a limit under **Mod**. $\mathcal{I}$ (or **Mod**) has the *(finite) amalgamation property* if (finite) colimit cocones are amalgamable, i.e. if **Mod** preserves (finite) limits. Recall that finite limits can be constructed from pullbacks and terminal objects, so that finite amalgamation reduces to preservation of pullbacks and terminal objects — explicitly: initial signatures are mapped to the terminal (one-point) category, and pushouts of signatures are mapped to pullbacks of model categories.

If, conversely, **Mod** reflects limits, then $\mathcal{I}$ (or **Mod**) is called *definitionally complete*. It is easily seen that, under cocompleteness of the signature category, a model functor that has the amalgamation property is definitionally complete iff it reflects isomorphisms. (It will become apparent below that in cases where **Sign** is actually a 2-category, reflection of equivalences is really the more appropriate concept. Other 2-categorical notions such as 2-limits or bilimits [8] are out of the scope of this paper.) Informally speaking, definitional completeness means that identifying symbols, adding a new symbol to a signature (without constraining it by axioms), or altering 'properties' of a symbol always modifies the model category. We shall see in Section 5 (cf. Example 5) that the standard CASL institution fails to be definitionally complete, essentially because symbol profiles are not implicitly closed under subsorting. Definitional completeness, which makes the colimit property a necessary condition for amalgamability of cocones, is required, e.g., to ensure a reasonable degree of completeness for the semantics of architectural specifications with the static amalgamability conditions as formulated in Section 8.

3 Basic Architectural Semantics

We now proceed to give a *basic semantics* of the architectural language defined in Section 1 similarly as for full CASL [16]. We use the natural semantics style, by presenting rules for the *static semantics*, with judgements written as $\_ \vdash \_ \rhd \_$, and for the *model semantics*, with judgements written as $\_ \vdash \_ \Rightarrow \_$ (where the blank spaces represent, in this order, a context of some kind, a syntactical object, and a semantical object). We simplify the rules of the model semantics by assuming a successful application of the corresponding rules of the static semantics, with symbols introduced there available for the model semantics as well.

Let us stress that the semantics is given here and in Section 4 in the framework of an arbitrary institution, with the syntax for specifications and signature morphisms as used in architectural specifications given elsewhere. Similarly, we assume that the semantics for specifications is given elsewhere, with $\vdash SP \rhd \Sigma$ and $\vdash SP \Rightarrow \mathcal{M}$ implying $\mathcal{M} \subseteq \mathbf{Mod}(\Sigma)$. We will regard $\mathbf{Mod}(\Sigma)$ as a *class* of models for the purposes of the model semantics.

The static semantics for an architectural specification yields the signature of its result unit and a static context that describes the signatures of the units declared or defined within the specification. Thus, a *static context* $C_{st} = (P_{st}, B_{st})$ consists of two finite maps: P_{st} from unit names to *parametrized unit signatures*, which in turn are signature morphisms $\tau : \Sigma_1 \to \Sigma_2$, and B_{st} from unit names to signatures (for non-parametrized units). We require the domains of P_{st} and B_{st} to be disjoint. The empty static context that consists of two empty maps will be written as $C^{\emptyset}_{st}$. Given an initial static context, the static semantics for unit declarations and definitions produces a static context by adding the signature for the newly introduced unit, and the static semantics for unit terms determines the signature for the resulting unit.

In terms of the model semantics, a (non-parametrized) unit M over a signature Σ is just a model $M \in \mathbf{Mod}(\Sigma)$. A parametrized unit F over a parametrized unit signature $\tau : \Sigma_1 \to \Sigma_2$ is a persistent partial function $F : \mathbf{Mod}(\Sigma_1) \rightharpoonup \mathbf{Mod}(\Sigma_2)$ (i.e. $F(M)|_\tau = M$ for each $M \in \mathit{dom}\, F$); the domain of F is determined by the *specification* of the parameter.

The model semantics for architectural specifications involves interpretations of unit names. These are given by *unit environments* E, i.e. finite maps from unit names to units as introduced above. On the model semantics side, the analogue of a static context is a *unit context* $\mathcal{C}$, which is just a class of unit environments, and can be thought of as a constraint on the interpretation of unit names. The unconstrained unit context, which consists of all environments, will be written as $\mathcal{C}^{\emptyset}$. The model semantics for unit declarations and definitions modifies unit contexts by constraining the environments to interpret the newly introduced unit names as determined by their specification or definition.

A unit term is interpreted by a *unit evaluator* Ev, a function that yields a unit when given a unit environment in the unit context (the unit environment serves to interpret the unit names occurring in the unit term). Hence, the model semantics for a unit term yields a unit evaluator, given a unit context.

The model semantics is easily seen to be compatible with the static semantics in the following sense: we say that $\mathcal{C}$ *fits* $C_{st} = (P_{st}, B_{st})$, if, whenever $B_{st}(U) = \Sigma$ and $E \in \mathcal{C}$, then $E(U)$ is a Σ-model, and a corresponding condition holds for P_{st}. Obviously, $\mathcal{C}^{\emptyset}$ fits $C^{\emptyset}_{st}$. Now if $\mathcal{C}$ fits C_{st}, then

$$C_{st} \vdash T \rhd \Sigma \quad \text{and} \quad \mathcal{C} \vdash T \Rightarrow Ev$$

imply that $Ev(E)$ is a Σ-model for each $E \in \mathcal{C}$. Corresponding statements hold for the other syntactic categories (unit declarations and definitions, architectural specifications).

The complete semantics is given in Figure 1, where we use some auxiliary notation: given a unit context $\mathcal{C}$, a unit name U and a class of units $\mathcal{V}$,

$$\mathcal{C} \times \{U \mapsto \mathcal{V}\} := \{E + \{U \mapsto V\} \mid E \in \mathcal{C},\, V \in \mathcal{V}\},$$

where $E + \{U \mapsto V\}$ maps U to V and otherwise behaves like E. Moreover, given a unit context $\mathcal{C}$, a unit name U and a unit evaluator Ev,

$$\mathcal{C} \otimes \{U \mapsto Ev\} := \{E + \{U \mapsto Ev(E)\} \mid E \in \mathcal{C}\}.$$

We assume that the signature category is equipped with a partial *selection of pushouts* $(\sigma_R : \Sigma_2 \to \Sigma_R, \tau_R : \Sigma_1 \to \Sigma_R, \Sigma_R)$ for spans $(\sigma : \Sigma \to \Sigma_1, \tau : \Sigma \to \Sigma_2)$ of signature morphisms (where (σ, τ) may fail to have a selected pushout even when it has a pushout):

$$\begin{tikzcd} \Sigma \arrow[r, "\sigma"] \arrow[d, "\tau"'] & \Sigma_1 \arrow[d, "\tau_R"] \\ \Sigma_2 \arrow[r, "\sigma_R"'] \arrow[ur, phantom, "\ulcorner" very near end] & \Sigma_R \end{tikzcd} .$$

In CASL, the selected pushouts would be the ones that can be expressed by signature translations and simple syntactic unions.

Perhaps the only points in the semantics that require some discussion are the rules of the model semantics for unit application and amalgamation.

In the rule for application of a parametrized unit U, we have the requirement

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

where Ev denotes the unit evaluator and $\mathcal{C}$ the unit context. This is just the statement that the fitting morphism correctly 'fits' the actual parameter as an argument for the parametrized unit. To verify this requirement, one typically has to prove that σ is a specification morphism from the argument specification to the specification of the actual parameter (which, in the general case, has to be determined for the relevant unit term by means of a suitable calculus). In general, this requires some semantic or proof-theoretic reasoning.

The situation is different with the conditions marked with a $(*)$ in Figure 1. These 'amalgamability conditions' are typically expected to be at least partially discharged by some static analysis — similarly to the sharing requirements present in some programming languages (cf. e.g. Standard ML [33]). Of course, the basic static analysis given here is not suited for this purpose, since no information is stored about dependencies between units. This will be taken care of in the second level of the semantics.

$$\frac{\vdash \mathit{DD}^* \rhd C_{st} \qquad C_{st} \vdash T \rhd \Sigma}{\vdash \textbf{arch spec } \mathit{DD}^* \textbf{ result } T \rhd (C_{st}, \Sigma)} \qquad \frac{\vdash \mathit{DD}^* \Rightarrow \mathcal{C} \qquad \mathcal{C} \vdash T \Rightarrow \mathit{Ev}}{\vdash \textbf{arch spec } \mathit{DD}^* \textbf{ result } T \Rightarrow (\mathcal{C}, \mathit{Ev})}$$

$$\frac{\begin{array}{c} C^{\emptyset}_{st} \vdash \mathit{DD}_1 \rhd (C_{st})_1 \\ \cdots \\ (C_{st})_{n-1} \vdash \mathit{DD}_n \rhd (C_{st})_n \end{array}}{\vdash \mathit{DD}_1 \ldots \mathit{DD}_n \rhd (C_{st})_n} \qquad \frac{\begin{array}{c} \mathcal{C}^{\emptyset} \vdash \mathit{DD}_1 \Rightarrow \mathcal{C}_1 \\ \cdots \\ \mathcal{C}_{n-1} \vdash \mathit{DD}_n \Rightarrow \mathcal{C}_n \end{array}}{\vdash \mathit{DD}_1 \ldots \mathit{DD}_n \Rightarrow \mathcal{C}_n}$$

$$\frac{\vdash \mathit{SP} \rhd \Sigma \qquad U \notin (\mathit{dom}\, P_{st} \cup \mathit{dom}\, B_{st})}{(P_{st}, B_{st}) \vdash U\colon \mathit{SP} \rhd (P_{st}, B_{st} + \{U \mapsto \Sigma\})} \qquad \frac{\vdash \mathit{SP} \Rightarrow \mathcal{M}}{\mathcal{C} \vdash U\colon \mathit{SP} \Rightarrow \mathcal{C} \times \{U \mapsto \mathcal{M}\}}$$

$$\frac{\vdash \mathit{SP}_1 \rhd \Sigma_1 \qquad \vdash \mathit{SP}_2 \rhd \Sigma_2 \qquad \tau : \Sigma_1 \to \Sigma_2 \qquad U \notin (\mathit{dom}\, P_{st} \cup \mathit{dom}\, B_{st})}{(P_{st}, B_{st}) \vdash U\colon \mathit{SP}_1 \xrightarrow{\tau} \mathit{SP}_2 \rhd (P_{st} + \{U \mapsto \tau\}, B_{st})}$$

$$\frac{\begin{array}{c} \vdash \mathit{SP}_1 \Rightarrow \mathcal{M}_1 \qquad \vdash \mathit{SP}_2 \Rightarrow \mathcal{M}_2 \\ \mathcal{F} = \{F : \mathcal{M}_1 \to \mathcal{M}_2 \mid \text{for } M \in \mathcal{M}_1, F(M)|_\tau = M\} \end{array}}{\mathcal{C} \vdash U\colon \mathit{SP}_1 \xrightarrow{\tau} \mathit{SP}_2 \Rightarrow \mathcal{C} \times \{U \mapsto \mathcal{F}\}}$$

$$\frac{(P_{st}, B_{st}) \vdash T \rhd \Sigma \qquad U \notin (\mathit{dom}\, P_{st} \cup \mathit{dom}\, B_{st})}{(P_{st}, B_{st}) \vdash U = T \rhd (P_{st}, B_{st} + \{U \mapsto \Sigma\})} \qquad \frac{\mathcal{C} \vdash T \Rightarrow \mathit{Ev}}{\mathcal{C} \vdash U = T \Rightarrow \mathcal{C} \otimes \{U \mapsto \mathit{Ev}\}}$$

$$\frac{U \in \mathit{dom}\, B_{st}}{(P_{st}, B_{st}) \vdash U \rhd B_{st}(U)} \qquad \frac{}{\mathcal{C} \vdash U \Rightarrow \{E \mapsto E(U) \mid E \in \mathcal{C}\}}$$

$$\frac{\begin{array}{c} P_{st}(U) = \tau : \Sigma_1 \to \Sigma_2 \qquad C_{st} \vdash T \rhd \Sigma^A \qquad \sigma : \Sigma_1 \to \Sigma^A \\ (\sigma_R, \tau_R, \Sigma_R) \text{ is the selected pushout of } (\sigma, \tau) \end{array}}{(P_{st}, B_{st}) \vdash U[T \textbf{ fit } \sigma] \rhd \Sigma_R}$$

$$\frac{\begin{array}{c} \mathcal{C} \vdash T \Rightarrow \mathit{Ev}; \quad \text{for each } E \in \mathcal{C}, \mathit{Ev}(E)|_\sigma \in \mathit{dom}\, E(U) \\ \left.\begin{array}{r} \text{for each } E \in \mathcal{C}, \text{ there is a unique } M \in \mathbf{Mod}(\Sigma_R) \text{ such that} \\ M|_{\tau_R} = \mathit{Ev}(E) \text{ and } M|_{\sigma_R} = E(U)(\mathit{Ev}(E)|_\sigma) \end{array}\right\} (*) \\ \mathit{Ev}_R = \{E \mapsto M \mid E \in \mathcal{C}, M|_{\tau_R} = \mathit{Ev}(E), M|_{\sigma_R} = E(U)(\mathit{Ev}(E)|_\sigma)\} \end{array}}{\mathcal{C} \vdash U[T \textbf{ fit } \sigma] \Rightarrow \mathit{Ev}_R}$$

$$\frac{C_{st} \vdash T_i \rhd \Sigma_i, i = 1, 2; \quad \sigma_1 : \Sigma_1 \to \Sigma \text{ and } \sigma_2 : \Sigma_2 \to \Sigma}{(P_{st}, B_{st}) \vdash T_1 \textbf{ with } \sigma_1 \textbf{ and } T_2 \textbf{ with } \sigma_2 \rhd \Sigma}$$

$$\frac{\begin{array}{c} \mathcal{C} \vdash T_1 \Rightarrow \mathit{Ev}_1 \qquad \mathcal{C} \vdash T_2 \Rightarrow \mathit{Ev}_2 \\ \left.\begin{array}{r} \text{for each } E \in \mathcal{C}, \text{ there is a unique } M \in \mathbf{Mod}(\Sigma) \text{ such that} \\ M|_{\sigma_i} = \mathit{Ev}_i(E), \ i = 1, 2 \end{array}\right\} (*) \\ \mathit{Ev} = \{E \mapsto M \mid E \in \mathcal{C} \text{ and } M|_{\sigma_i} = \mathit{Ev}_i(E), \ i = 1, 2\} \end{array}}{\mathcal{C} \vdash T_1 \textbf{ with } \sigma_1 \textbf{ and } T_2 \textbf{ with } \sigma_2 \Rightarrow \mathit{Ev}}$$

Fig. 1. Basic semantics

4 Extended Static Architectural Semantics

As a solution to the problem just outlined, we now introduce an extended static analysis that keeps track of sharing among the units by means of a diagram of signatures; the idea here is that a symbol shares with any symbol to which it is mapped under some morphism in the diagram.

For our purposes, it suffices to regard a diagram as a graph morphism $D : \mathbf{I} \to \mathbf{Sign}$, where $\mathbf{I}$ is a directed graph called the *scheme* of the diagram. We use categorical terminology for $\mathbf{I}$, i.e. we call its nodes 'objects', its edges 'morphisms' etc., and we write $\mathrm{Ob}\,\mathbf{I}$ for the set of objects.

We will use the usual notion of extension for diagrams. Two diagrams D_1, D_2 *disjointly extend* D if both D_1 and D_2 extend D and moreover, the intersection of their schemes is the scheme of D. If this is the case then the union $D_1 \cup D_2$ is well-defined. Of course, disjointness can be ensured as usual by renaming the relevant components in the diagram schemes of D_1, D_2.

The judgements of the *extended static semantics* are written as $\_ \vdash \_ \rhd\!\!\rhd \_$. Most of the rules differ only formally from the rules for the static semantics; the essential differences are in the rules for unit terms. The extended static semantics additionally carries around the said diagram of signatures. Signatures for unit terms are associated to distinguished objects in the diagram scheme.

Explicitly, an *extended static context* $\mathcal{C}_{st} = (P_{st}, \mathcal{B}_{st}, D)$ consists of a map P_{st} that assigns parametrized unit signatures to parametrized unit names (as before), a signature diagram D, and a map $\mathcal{B}_{st}$ that assigns objects of the diagram scheme to (non-parametrized) unit names. As before, we require that the domains of P_{st} and $\mathcal{B}_{st}$ are disjoint. $\mathcal{C}_{st}$ determines a static context $ctx(\mathcal{C}_{st})$ formed by extracting the signature information for non-parametrized unit names from the diagram and forgetting the diagram itself. The empty extended static context, which consists of two empty maps and the empty diagram, is written as $\mathcal{C}_{st}^{\emptyset}$. The extended static semantics for unit declarations and definitions expands the given extended static context; for unit terms, it extends the signature diagram and indicates an object in the scheme that represents the result.

The diagrams enable us to restate the amalgamability conditions in a static way: for any diagram $D : \mathbf{I} \to \mathbf{Sign}$, a family $\langle M_i \rangle_{i \in \mathrm{Ob}\,\mathbf{I}}$ of models is called *D-coherent* if for each $i \in \mathrm{Ob}\,\mathbf{I}$, each $M_i \in \mathbf{Mod}(D(i))$, and for each $m : i \to j$ in $\mathbf{I}$, $M_i = M_j|_{D(m)}$; this is extended to families of model morphisms in the obvious way. Then, *D ensures amalgamability for D'*, where D' extends D, if any D-coherent model family can be uniquely extended to a D'-coherent model family, and correspondingly for coherent families of model morphisms.

$$\frac{\vdash DD^* \mathrel{\rhd\!\!\rhd} \mathcal{C}_{st} \qquad \mathcal{C}_{st} \vdash T \mathrel{\rhd\!\!\rhd} (i, D)}{\vdash \textbf{arch spec } DD^* \textbf{ result } T \mathrel{\rhd\!\!\rhd} (ctx(\mathcal{C}_{st}), D(i))}$$

$$\frac{\begin{array}{c}\mathcal{C}^{\emptyset}_{st} \vdash DD_1 \mathrel{\rhd\!\!\rhd} (\mathcal{C}_{st})_1 \\ \ldots \\ (\mathcal{C}_{st})_{n-1} \vdash DD_n \mathrel{\rhd\!\!\rhd} (\mathcal{C}_{st})_n\end{array}}{\vdash DD_1 \ldots DD_n \mathrel{\rhd\!\!\rhd} (\mathcal{C}_{st})_n}$$

$$\frac{\begin{array}{c}\vdash SP \rhd \Sigma \qquad U \notin (dom\, P_{st} \cup dom\, \mathcal{B}_{st}) \\ D' \text{ results from } D \text{ by adding a new object } i \text{ with } D'(i) = \Sigma\end{array}}{(P_{st}, \mathcal{B}_{st}, D) \vdash U \colon SP \mathrel{\rhd\!\!\rhd} (P_{st}, \mathcal{B}_{st} + \{U \mapsto i\}, D')}$$

$$\frac{\begin{array}{c}\vdash SP_1 \rhd \Sigma_1 \qquad \vdash SP_2 \rhd \Sigma_2 \qquad \tau : \Sigma_1 \to \Sigma_2 \\ U \notin (dom\, P_{st} \cup dom\, \mathcal{B}_{st})\end{array}}{(P_{st}, \mathcal{B}_{st}, D) \vdash U \colon SP_1 \xrightarrow{\tau} SP_2 \mathrel{\rhd\!\!\rhd} (P_{st} + \{U \mapsto \tau\}, \mathcal{B}_{st}, D)}$$

$$\frac{(P_{st}, \mathcal{B}_{st}, D) \vdash T \mathrel{\rhd\!\!\rhd} (i, D') \qquad U \notin (dom\, P_{st} \cup dom\, \mathcal{B}_{st})}{(P_{st}, \mathcal{B}_{st}, D) \vdash U = T \mathrel{\rhd\!\!\rhd} (P_{st}, \mathcal{B}_{st} + \{U \mapsto i\}, D')}$$

$$\frac{U \in dom\, \mathcal{B}_{st}}{(P_{st}, \mathcal{B}_{st}, D) \vdash U \mathrel{\rhd\!\!\rhd} (\mathcal{B}_{st}(U), D)}$$

$$\frac{\begin{array}{c}P_{st}(U) = \tau : \Sigma_1 \to \Sigma_2 \qquad (P_{st}, \mathcal{B}_{st}, D_0) \vdash T \mathrel{\rhd\!\!\rhd} (i, D) \qquad \sigma : \Sigma_1 \to D(i) \\ (\sigma_R, \tau_R, \Sigma_R) \text{ is the selected pushout of } (\sigma, \tau) \\ D' \text{ results from } D \text{ by adding new objects } j, k \\ \text{and new morphisms } m : j \to i, n : j \to k \text{ with } D'(m) = \sigma, D'(n) = \tau \\ D'' \text{ results from } D' \text{ by adding a new object } l \\ \text{and new morphisms } r : i \to l, s : k \to l \text{ with } D''(r) = \tau_R, D''(s) = \sigma_R \\ D' \text{ ensures amalgamability for } D''\end{array}}{(P_{st}, \mathcal{B}_{st}, D_0) \vdash U[T \textbf{ fit } \sigma] \mathrel{\rhd\!\!\rhd} (l, D'')}$$

$$\frac{\begin{array}{c}(P_{st}, \mathcal{B}_{st}, D) \vdash T_1 \mathrel{\rhd\!\!\rhd} (i_1, D_1) \qquad (P_{st}, \mathcal{B}_{st}, D) \vdash T_2 \mathrel{\rhd\!\!\rhd} (i_2, D_2) \\ \sigma_1 : D_1(i_1) \to \Sigma \qquad \sigma_2 : D_2(i_2) \to \Sigma \\ D_1 \text{ and } D_2 \text{ are disjoint extensions of } D \\ D' \text{ results from } D_1 \cup D_2 \text{ by adding a new object } j \text{ and new morphisms} \\ m_1 : i_1 \to j, m_2 : i_2 \to j \text{ with } D'(m_1) = \sigma_1,\ D'(m_2) = \sigma_2 \\ D_1 \cup D_2 \text{ ensures amalgamability for } D'\end{array}}{(P_{st}, \mathcal{B}_{st}, D) \vdash T_1 \textbf{ with } \sigma_1 \textbf{ and } T_2 \textbf{ with } \sigma_2 \mathrel{\rhd\!\!\rhd} (j, D')}$$

Fig. 2. Extended static semantics

Although we have formulated this property in terms of model families, it is essentially static: the class of model families considered is not restricted by axioms, but only by morphisms between signatures. The static nature of this condition will be made explicit in Section 8.

The rules of the extended static semantics are listed in Figure 2; given the heuristics provided above, they should be largely self-explanatory. However, the relationship between the basic static and model semantics and the extended static semantics requires a few comments.

Since, as stated at the end of the previous section, the correctness condition for arguments of parametrized units cannot be disposed of statically, one cannot expect that the extended static semantics is stronger than the model semantics, i.e. that its successful application guarantees that the model sematics will succeed as well. However, this is almost true in the sense that argument fitting is the only point that is left entirely to the model semantics. Formally, this can be captured by the statement that, assuming a successful run of the extended static semantics, the conditions marked with a $(*)$ in the rules of the model semantics (cf. Figure 1) can be removed. We denote the judgements of the thus simplified model semantics by $\_ \vdash \_ \overset{s}{\Rightarrow} \_$.

Theorem 2 *Given an architectural specification, if its extended static semantics is defined then its basic static semantics is defined and yields the same result. Moreover, then its basic model semantics is defined if and only if its simplified model semantics is defined, and when defined, they yield the same results.*

PROOF. We say that a model family $\langle M_i \rangle_{i \in \mathrm{Ob}\,\mathbf{I}}$ *witnesses* an environment E in an extended static context $\mathcal{C}_{st} = (P_{st}, \mathcal{B}_{st}, D)$ if it is D-coherent and $E(U) = M_i$ for all $U \in dom\,\mathcal{B}_{st}$ with $\mathcal{B}_{st}(U) = i \in \mathrm{Ob}\,\mathbf{I}$, where $\mathbf{I}$ is the scheme of D. From an extended static context $\mathcal{C}_{st} = (P_{st}, \mathcal{B}_{st}, D)$ we extract a unit context $ucx(\mathcal{C}_{st})$ which consists of all unit environments E such that

- $E(U)$ is a parametrized unit over τ whenever $P_{st}(U) = \tau$, and
- E is witnessed by a model family in $\mathcal{C}_{st}$.

The claim can now be made explicit (and strengthened to a form suitable for induction) for the various syntactic categories as follows:

(i) For $\mathcal{C}_{st} = (P_{st}, \mathcal{B}_{st}, D)$, if $\mathcal{C}_{st} \vdash T \mathrel{\rhd\!\!\rhd} (i, D')$ then D' extends D and $ctx(\mathcal{C}_{st}) \vdash T \rhd D'(i)$. Moreover, then:
  - for all $\mathcal{C} \subseteq ucx(\mathcal{C}_{st})$, $\mathcal{C} \vdash T \overset{s}{\Rightarrow} Ev$ if and only if $\mathcal{C} \vdash T \Rightarrow Ev$; and
  - if $\mathcal{C} \vdash T \Rightarrow Ev$ then for all $E \in \mathcal{C}$, any model family that witnesses E can be extended to a D'-coherent model family $\langle M_j \rangle_{j \in \mathrm{Ob}\,\mathbf{I}'}$ such that $M_i = Ev(E)$.

(ii) If $\mathcal{C}_{st} \vdash DD^* \mathrel{\rhd\!\!\rhd} \mathcal{C}'_{st}$ then $ctx(\mathcal{C}_{st}) \vdash DD^* \rhd ctx(\mathcal{C}'_{st})$. Moreover, then for all $\mathcal{C} \subseteq ucx(\mathcal{C}_{st})$, $\mathcal{C} \vdash DD^* \overset{s}{\Rightarrow} \mathcal{C}'$ if and only if $\mathcal{C} \vdash DD^* \Rightarrow \mathcal{C}'$. Finally, if $\mathcal{C} \subseteq ucx(\mathcal{C}_{st})$ and $\mathcal{C} \vdash DD^* \Rightarrow \mathcal{C}'$ then $\mathcal{C}' \subseteq ucx(\mathcal{C}'_{st})$.

(iii) If $\vdash$ **arch spec** DD^* **result** $T \mathrel{\rhd\!\!\rhd} (C_{st}, \Sigma)$
then $\vdash$ **arch spec** DD^* **result** $T \rhd (C_{st}, \Sigma)$.

Moreover, then $\vdash$ **arch spec** DD^* **result** $T \overset{s}{\Rightarrow} (\mathcal{C}, Ev)$ if and only if $\vdash$ **arch spec** DD^* **result** $T \Rightarrow (\mathcal{C}, Ev)$.

The proof is by induction on the length of the derivation. For the static semantics claims, the induction is easy. For the model semantics, the key observation is that each of the conditions marked by a $(*)$ in the rules of the basic model semantics follows under inductive hypothesis (i) from the other premises of the rule and from the premises of the corresponding rule of the extended static semantics. □

Calling the combination of the extended static semantics and the simplified model semantics *extended semantics*, we now have:

Corollary 3 *If the extended semantics of an architectural specification is defined, then the basic semantics is defined as well and yields the same result.*

Of course, no completeness can be expected here: even if the basic semantics is successful for a given phrase, the extended semantics may fail. This happens if the model-theoretic amalgamability conditions hold due to axioms in specifications rather than due to static properties of the involved constructions.

An additional source of failures of the extended static semantics is that, following [6], we have deliberately chosen a so-called generative static analysis: the results of applications of parametrized units 'share' with other units in the signature diagram constructed only via the morphisms from the parameter signatures to the actual arguments. Thus, two applications of the same unit to the same argument need not 'share'. As a consequence, the amalgamability condition of the extended static semantics may fail for them, while the corresponding condition in the basic model semantics would clearly hold. A 'non-generative' (or 'applicative') version of the extended static semantics is sketched in Remark 28 below.

The motivation for this choice is the fact that many typical programming languages we aim at (notably, Standard ML [33]) impose such a 'generative' semantics in their static analysis — working with more permissive conditions here would make our architectural specifications incompatible with the modularization facilities of such languages.

However — generativity issues aside — we have as much completeness as one may hope for, i.e., in general the extended static semantics detects all the amalgamation that can be established statically. To see this, note that if all the specifications considered admit all models over their signatures, then in the notation of the proof of Theorem 2, for any extended static context $\mathcal{C}_{st}$ that appears in the derivation of the extended static semantics, the corresponding unit context of the model semantics is $ucx(\mathcal{C}_{st})$, and hence, the requirements marked by a (∗) in the basic model semantics are just equivalent to the corresponding amalgamability requirements in the extended static semantics (see [24] for a formal statement of an analogous theorem).

5 Standard CASL signatures

We shall now focus our attention on the analysis of the extended static semantics, and in particular of the amalgamation conditions it imposes, in the particular context of the standard Casl institution. The amalgamation property and definitional completeness are the crucial ingredients in the reformulation of the amalgamation conditions in an entirely static form. They will be ensured in the *enriched* Casl institution, to be introduced in the next section. In order to provide a basis for the definition of the enriched institution, we sketch the definition of (standard) Casl signatures and their models; for further details see [12,15].

A Casl *signature* Σ consists of a preordered set S of sorts and sets of total and partial function symbols and predicate symbols. Function and predicate symbols are written $f : \bar{s} \to t$ and $p : \bar{s}$, respectively, where t is a sort and $\bar{s}$ is a list $s_1 \dots s_n$ of sorts (similar notation for lists is used throughout), thus determining their *name* and *profile*. Symbols with identical names are said to be in the *overloading relation* if their argument sorts have a common subsort and (in the case of function symbols) their result sorts have a common supersort. Otherwise, their overloading is just ad-hoc overloading without semantic implications. Partial function symbols may become total on subsorts of their argument sorts, but not vice versa.

A *signature morphism* consists of an order-preserving map between the associated sort preorders and maps between the symbol sets that are compatible with symbol profiles, preserve totality (i.e. may map partial to total function symbols, but not vice versa), and preserve the overloading relation. This defines the (cocomplete [30]) signature category **CASLsign**.

A *model* of a Casl signature is an interpretation of the sorts by sets and of the sort preorder by *injective maps* between these sets (in other words: a functor from the thin category associated to the sort preorder into the category of sets and injective maps), of the partial (total) function symbols by partial (total) functions between the sets specified by their profiles, and of the predicate symbols by relations. The interpretations of overloaded symbols are required to agree on common subsorts of the argument sorts via the corresponding subsort injections. *Model morphisms* are defined by the usual homomorphism condition (also w.r.t. the sort injections) and preservation of predicate satisfaction.

Thus, we have a category $\mathbf{Mod}_{CASL}(\Sigma)$ of Σ-models; this assigment extends to a model functor $\mathbf{Mod}_{CASL} : \mathbf{CASLsign}^{op} \to \mathbf{CAT}$ in the standard way. It is folklore that amalgamation fails for this model functor:

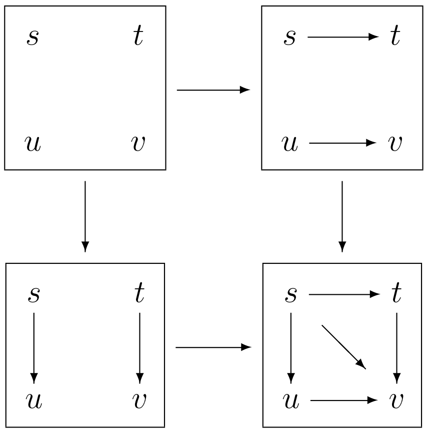
Example 4 The simplest case where amalgamation fails is the following: let Σ be the signature with sorts s and t (and no operations), and let Σ_1 be the extension of Σ by the subsort relation $s < t$. Then the pushout

$$\begin{tikzcd}
\Sigma \arrow[r] \arrow[d] & \Sigma_1 \arrow[d] \\
\Sigma_1 \arrow[r] & \Sigma_1
\end{tikzcd}$$

in **CASLsign** fails to be amalgamable (since two models of Σ_1, compatible w.r.t. the inclusion of Σ, may interpret the subsort injection differently).

The following more complex accident, being rather less readily apparent, might conceivably occur in the specification of Example 1: assume that the specification LIST[ELEM] provides a sort *List*[*Elem*] of lists of sort *Elem*. Recall that the specification of identifiers introduces two sorts *Identifier* and *Keyword*, and that the program text unit *PT* incorporates the results of application of the parametrized unit *L* to each of these two sorts.

Now suppose that the specifier of PARSER decides that keywords should be treated as identifiers, so that *Keyword* < *Identifier* and *List*[*Keyword*] < *List*[*Identifier*]. Suppose, moreover, that the specifier of STATICANALYSER finds it convenient to code simple elements as lists in some way, so that *Identifier* < *List*[*Identifier*] and *Keyword* < *List*[*Keyword*]. Singling out the union *P* **and** *SA* from the term defining the compiler in Example 1, we thus obtain a diagram of CASL signatures for the union that has the following (abstracted) form, where the arrows within the squares represent subsort embeddings:



(1)

Even though the above diagram is in fact a pushout in the category **CASLsign**, compatible models of the component signatures cannot in general be amalgamated, since the composed subsort embeddings $s < t < v$ and $s < u < v$ in the result need not be the same. (Consequently, the extended static semantics defined in the previous section would fail here — and so would the basic semantics, unless the axioms in the specifications constrained the subsort embeddings very strongly indeed.)

These observations suggest that one should enlarge the signature category in such a way that both the above examples are no longer pushouts, and that, moreover, signatures in the enlarged category would, in general, admit more than one embedding between two sorts. Thus, the 'correct' pushout signature for the diagram above would have the form

$$\boxed{\begin{tikzcd} s \arrow[r] \arrow[d] \arrow[dr, shift left] \arrow[dr, shift right] & t \arrow[d] \\ u \arrow[r] & v \end{tikzcd}} \tag{2}$$

Incidentally, $\mathbf{Mod}_{CASL}$ also fails to be definitionally complete:

Example 5 The inclusion $\Sigma_1 \hookrightarrow \Sigma_2$, where Σ_1 contains sorts $s < t$ and an operation $a : s$ and Σ_2 contains an additional symbol $a : t$, is not an isomorphism, but induces an isomorphism between the model categories $\mathbf{Mod}(\Sigma_2)$ and $\mathbf{Mod}(\Sigma_1)$.

In view of the above examples, one may wonder whether interpreting subsorts by arbitrary injections (rather than subsort inclusions, which would avoid these problems) is really a good design decision. However, the subsorts-as-inclusions approach has severe theoretical and practical drawbacks. In particular, satisfaction fails to be closed under model isomorphism (unless extra conditions like local filtration are assumed, which, however, behave badly w.r.t. colimits [23]). Moreover, 'real-life' subsort relations (such as $Int < Float$) do require coercion functions; see [34] for a detailed discussion.

6 Enriched signatures

We now introduce a category of enriched signatures in which standard signatures can be represented via a suitable functor. Moreover, we equip this signature category with a model functor which has the amalgamation property and is definitionally complete and which extends the original model functor up to a natural isomorphism; this enables us to treat amalgamability in the extended setting.

Example 4 suggests that the failure of amalgamation for standard signatures can be remedied by replacing the sort preorder by a sort category that admits more than one embedding between two sorts (similar category sorted algebras, although without a view on amalgamation, appear in [35]). This will be the main feature to distinguish enriched and standard signatures. We will certainly continue to require embeddings to be monomorphic; categories in which all morphisms are monomorphisms will be called *left cancellable.*

Moreover, there is an elegant way of handling overloading of function and predicate symbols: introduce left and right actions of the sort category on the symbols; in the case of a unary function symbol $f : s \to t$ and sort embeddings $d : u \to s$, $e : t \to v$, the left action gives rise to a function symbol $e \cdot f : s \to v$, and the right action to $f \cdot d : u \to t$:

$$\begin{tikzcd}
 & t \arrow[r, "e"] & v \\
u \arrow[r, "d"'] \arrow[ur, "f \cdot d"] & s \arrow[u, "f"'] \arrow[ur, "e \cdot f"'] &
\end{tikzcd}$$

(The right action also applies to predicate symbols.) The appropriate behaviour of models w.r.t. overloading can then be ensured by requiring that the diagrams that arise from the actions as above are translated to commutative diagrams of maps (operations) in the models. This requirement replaces the rather cumbersome overloading axioms for models needed in the case of standard signatures; similarly, overloading preservation for signature morphisms now becomes a much more straightforward equivariance condition.

Thus, we arrive at the following

Definition 6 An *enriched (*CASL*) signature* Σ consists of

(i) a left cancellable *sort category* $\mathbf{S}$ with morphisms called *embeddings*;
(ii) a class F of *function symbols* and a class P of *predicate symbols*; symbols have profiles as in the case of standard signatures;
(iii) a unary *totality* predicate on F;
(iv) a *left action* of $\mathbf{S}$ on F which assigns to each function symbol $f : \bar{s} \to t$ and each sort morphism $e : t \to u$ a function symbol $e \cdot f : \bar{s} \to u$;
(v) a *right (multi)action* of $\mathbf{S}$ on F which assigns to each function symbol $f : \bar{s} \to t$ and each list of sort embeddings $\bar{d} = (d_i : v_i \to s_i)_{i=1,\ldots,n}$ (written $\bar{d} : \bar{v} \to \bar{s}$) a function symbol $f \cdot \bar{d} : \bar{v} \to t$, and a similar right action on P.

These data are subject to the following axioms:

(i) The associative law and the identity law hold in the obvious sense (e.g., in the above notation, $id \cdot f = f$ and $(e \cdot f) \cdot \bar{d} = e \cdot (f \cdot \bar{d})$).
(ii) For a sort embedding e and function symbols f, g with appropriate profiles, $e \cdot f = e \cdot g$ implies $f = g$.
(iii) Let f, e, and $\bar{d}$ be as above. If f is total, then $f \cdot \bar{d}$ and $e \cdot f$ are total. Moreover, if $e \cdot f$ is total, then f is total.

A *signature morphism* σ between enriched signatures consists of a functor between the sort categories and a pair of maps between the classes of function and predicate symbols, respectively; all three components are denoted by σ. σ is required to be compatible with symbol profiles and to preserve totality. Moreover, σ is assumed to be *equivariant* w.r.t. the actions of the sort categories; i.e. if f, e, and $\bar{d}$ are as above, then

$$\sigma(e) \cdot \sigma(f) = \sigma(e \cdot f) \quad \text{and} \quad \sigma(f) \cdot \sigma(\bar{d}) = \sigma(f \cdot \bar{d}),$$

similarly for predicate symbols. An enriched signature is called *small* if its sort category and its symbol classes are small. Small enriched signatures and signature morphisms form a category **enrCASLsign**.

Typical examples of properly enriched signatures arise as colimits of standard signatures (the way standard signatures are regarded as enriched signatures is explained below), such as the one suggested in Diagram (2) of Section 5. Inserting this enriched signature into its proper place in Diagram (1) of the same section also provides two examples of signature morphisms as defined above.

More precisely, one should say that **enrCASLsign** is a 2-category: 2-cells between signature morphisms are natural transformations between the functor parts that satisfy the obvious naturality condition w.r.t. symbols. In particular, one has a notion of equivalence of enriched signatures defined in the usual way via 'inverses up to isomorphism'. As in the case of categories, it is straightforward to show that such a definition of equivalence amounts to the following:

Definition 7 A signature morphism σ is called *full* on symbols if, whenever f is a symbol with profile $f : \sigma(\bar{s}) \to \sigma(t)$, then there exists a symbol $\hat{f}$ with profile $\hat{f} : \bar{s} \to t$ such that $\sigma(\hat{f}) = f$. σ is *faithful* on symbols if, whenever f and g are symbols with identical profile $f, g : \bar{s} \to t$, then $\sigma(f) = \sigma(g)$ implies $f = g$. σ is an *equivalence* if the functor part of σ is an equivalence, σ is full and faithful on symbols, and σ *reflects totality*, i.e., whenever $\sigma(f)$ is a total function symbol, then so is f.

Equivalent signatures differ only in possibly having different numbers of isomorphic copies of each sort.

A crucial point is that the collection of all sets and partial maps can now be regarded as a signature. (Due to the rather different treatment of overloading, there is no obvious way to make this work for the signatures defined in [35]). More precisely: We have an enriched signature which has the category of sets and (total) injective maps as sort category and n-ary partial functions and relations as function and predicate symbols, respectively, with the obvious assignment of profiles and with totality of partial functions as the totality predicate. The actions of the sort category are given by composition (in the case of the right action, by composition with cartesian products of maps or, for predicates, by taking preimages under such products). This enriched signature will be denoted by $\mathbf{Set_p}$.

This enables us to define models in the spirit of Lawvere [26]:

Definition 8 A model of an enriched signature Σ is a signature morphism

$$\Sigma \to \mathbf{Set_p}.$$

A *morphism* between two such models σ, τ is a family ϕ of maps

$$\phi_s : \sigma(s) \to \tau(s), \text{ where } s \text{ ranges over the sorts in } \Sigma,$$

such that the usual homomorphism condition w.r.t. function symbols and embeddings holds and satisfaction of predicates is preserved.

Thus, we have a *model category* of Σ, which we denote by $\mathbf{Mod}_{enr}(\Sigma)$. A signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ induces a reduct functor $\mathbf{Mod}_{enr}(\Sigma_2) \to \mathbf{Mod}_{enr}(\Sigma_1)$ which acts on objects by composition of signature morphisms. This defines the model functor $\mathbf{Mod}_{enr} : \mathbf{enrCASLsign}^{op} \to \mathbf{CAT}$.

The representing functor $\mathsf{Enr} : \mathbf{CASLsign} \to \mathbf{enrCASLsign}$ acts on standard signatures by first forming a suitable completion of the symbol sets — closing symbol profiles under the sort preorder — to account for the actions of the sort embeddings, and then reinterpreting the data in the usual way (i.e. the sort preorder is interpreted as a thin category, and the actions of the sort category are defined using the mentioned closure of symbol profiles). Thanks to overloading preservation, morphisms between standard signatures have a well-defined extension to the respective closures. Thus, this assignment on objects extends to a functor as required. Note that Enr is faithful, but is neither full nor injective on objects (e.g., it identifies the two non-isomorphic standard signatures of Example 5) and, of course, does *not* preserve colimits (although it does preserve coproducts, which are just componentwise disjoint unions here).

Now it is easily verified that one has a natural isomorphism

$$\mathbf{Mod}_{enr} \circ \mathsf{Enr}^{op} \to \mathbf{Mod}_{CASL}$$

which maps an $\mathsf{Enr}(\Sigma)$-model to the restriction of the symbol interpretation to Σ. In particular, as indicated above,

Proposition 9 *A cocone in* **CASLsign** *is amalgamable w.r.t.* $\mathbf{Mod}_{CASL}$ *iff its image under* Enr *is amalgamable w.r.t.* $\mathbf{Mod}_{enr}$.

Thanks to the definition of models as signature morphisms, $\mathbf{Mod}_{enr}$ satisfies the amalgamation property in its most general form:

Theorem 10 **enrCASLsign** *is cocomplete, and*

$$\mathbf{Mod}_{enr} : \mathbf{enrCASLsign}^{op} \to \mathbf{CAT}$$

preserves limits.

PROOF. To build a colimit of a diagram in **enrCASLsign**, start with the set-theoretic colimits of all involved sets (sorts, embedding symbols, function symbols, predicate symbols), and assign profiles to the arising equivalence classes of symbols in the obvious way. From these equivalence classes, one can form terms according to the operations defining an enriched signature (identities and composition in the sort category and the left and right actions of the sort category). Terms may contain at most one function or predicate symbol and are classified accordingly as functions, predicates, or embeddings. In the next step, terms are identified according to the equations coming from the component signatures and the (conditional) equations for enriched signatures (identity law, associativity of composition, left cancellation of embedding symbols, laws for the actions). Finally, the totality predicate is defined as the smallest set that makes the colimit injections signature morphisms and satisfies the axioms concerning preservation and reflection of symbols in enriched signatures.

Now let $D : \mathbf{I} \to \mathbf{enrCASLsign}$ be a diagram with colimit $\langle \mu_i : D(i) \to \Sigma \rangle_{i \in \mathrm{Ob}\,\mathbf{I}}$, let $\langle \sigma_i : D(i) \to \mathbf{Set_p} \rangle_{i \in \mathrm{Ob}\,\mathbf{I}}$ and $\langle \tau_i : D(i) \to \mathbf{Set_p} \rangle_{i \in \mathrm{Ob}\,\mathbf{I}}$ be D-coherent model families, and let $\langle \phi^i : \sigma_i \to \tau_i \rangle_{i \in \mathrm{Ob}\,\mathbf{I}}$ be a D-coherent family of model morphisms. Since the *objects* of model categories are given by a representable functor (namely, the functor $hom(\_, \mathbf{Set_p})$), and since representable functors preserve limits, we can amalgamate $\langle \sigma_i \rangle_{i \in \mathrm{Ob}\,\mathbf{I}}$ and $\langle \tau_i \rangle_{i \in \mathrm{Ob}\,\mathbf{I}}$ uniquely, thus obtaining models σ and τ of Σ, i.e. signature morphisms $\sigma, \tau : \Sigma \to \mathbf{Set_p}$ (where

$$\sigma \mu_i = \sigma_i : D(i) \to \mathbf{Set_p}$$

for all $i \in \mathrm{Ob}\,\mathbf{I}$, similarly for τ). Morphisms in the model categories had to be defined in a somewhat less succinct way and hence do not lend themselves to this type of argument. However, since the colimit is set-theoretic at the level of sorts, we can (uniquely) amalgamate the ϕ^i, obtaining at least a well-defined family of maps

$$\phi_{\mu_i(s)} = \phi^i_s : \sigma(\mu_i(s)) \to \tau(\mu_i(s)),$$

where i ranges over $\mathrm{Ob}\,\mathbf{I}$ and s ranges over the sorts in $D(i)$. All that remains to be shown is that this family is indeed a model morphism from σ to τ: Since the symbols (embeddings, function symbols, predicate symbols) in Σ are generated by the images of the symbols in the $D(i)$, it suffices to prove the morphism condition for such images. For example, let $f : s_1 \dots s_n \to t$ be a function symbol in $D(i)$ for some i (predicate symbols and embeddings are treated analogously). Then

$$\begin{aligned}\phi_{\mu_i(t)}\sigma\mu_i(f) &= \phi^i_t\sigma_i(f)\\ &\subseteq \tau_i(f)(\phi^i_{s_1} \times \dots \times \phi^i_{s_n})\\ &= \tau\mu_i(f)(\phi_{\mu_i(s_1)} \times \dots \times \phi_{\mu_i(s_n)})\end{aligned}$$

(where $\subseteq$ denotes the extension relation for partial maps), i.e., ϕ satisfies the homomorphism condition for $\mu_i(f)$. □

Remark 11 In the next section, we shall construct a representation of enriched signatures as equational partial specifications, which also have the amalgation property. Thus, the intermediate step via enriched signatures could in principle be skipped. The advantage of the representation of standard signatures as enriched signatures is that, in order to actually perform a static analysis of architectural specifications by verifying the colimit property in concrete cases, one does not need to apply full first order partial equational reasoning, but only a rather simpler calculus (called the *cell calculus* in [25]) that relates only to enriched signatures; cf. Section 8 for details.

7 Definitional completeness

To reformulate the static amalgamability conditions of the extended static semantics to a more manageable form and to establish that the reformulation does not strengthen the conditions unnecessarily, we need not only the amalgamation property, but also definitional completeness for enriched CASL. To prove definitional completeness we introduce a further representation of enriched signatures in a suitable class of small categories which is easily seen to have this property; we then go on to show that the representation is sufficiently well-behaved to transfer definitional completeness back to the setting of enriched signatures. In an intermediate step, we introduce a coding of enriched signatures as equational partial specifications in the sense of [13], thus essentially providing an extension of the existing embedding of the CASL logic into partial first-order logic with equality [12].

Let **Lex** denote the category of small left exact (lex) categories (i.e. small categories with finite limits) and left exact (lex, finite limit preserving) functors. The *model category* $\mathbf{Mod}_{lex}(\mathbf{A})$ of a small lex category $\mathbf{A}$ is the category $\mathbf{Lex}(\mathbf{A}, \mathbf{Set})$ of lex functors $\mathbf{A} \to \mathbf{Set}$ and natural transformations. By composition of lex functors, this assignment extends to a functor

$$\mathbf{Mod}_{lex} : \mathbf{Lex}^{op} \to \mathbf{CAT}.$$

$\mathbf{Mod}_{lex}$ trivially has the amalgamation property, i.e. preserves limits. Moreover, being a 2-functor, $\mathbf{Mod}_{lex}$ preserves equivalences. Conversely,

Theorem 12 $\mathbf{Mod}_{lex}$ *reflects equivalences.*

PROOF. For any lex category $\mathbf{A}$, the finitely presentable objects in $\mathbf{Mod}_{lex}(\mathbf{A})$ are precisely the representable functors [28]. Thus, any equivalence $\mathbf{Mod}_{lex}(\mathbf{A}) \to \mathbf{Mod}_{lex}(\mathbf{B})$ restricts to an equivalence between the respective subcategories of representable functors (since equivalences preserve all categorical properties [2], in particular finite presentability). The latter are equivalent to $\mathbf{A}$ and $\mathbf{B}$, respectively, via the Yoneda Embedding. $\square$

The above theorem captures an essential component of the definitional completeness for enriched CASL; as far as we can see, even a direct proof of Corollary 20 below would have to somehow incorporate the nontrivial properties shown here succinctly by categorical means.

An (equational partial) *specification* $\mathcal{S} = (\Omega, \mathcal{A})$ is defined (slightly extending the definition in [13]) as follows:

Ω is a *signature* consisting of sorts and (partial) operation symbols with profiles as before. This signature gives rise to a notion of sorted terms in context in the usual way, where a context is a list $x_1 : s_1, \dots, x_n : s_n$ of sort assignments for variables, abbreviated as $\bar{x} : \bar{s}$, where $\bar{x} = (x_1, \dots, x_n)$ and $\bar{s} = (s_1, \dots, s_n)$. The judgement 'the term α has sort t in context Γ' is written

$$\Gamma \rhd \alpha : t.$$

Lists of terms are called *multi-terms*; if $\Gamma \rhd \alpha_i : t_i$ for $i = 1, \dots, m$, then we write $\Gamma \rhd \alpha : \bar{t}$, where $\alpha = (\alpha_1, \dots, \alpha_m)$. In fact, application of an operator f is regarded as forming a term $f(\alpha)$ from a multi-term α. As a slight twist, the empty multi-term $()$ doubles as a term of 'sort' $()$. Given a judgement $\bar{y} : \bar{t} \rhd \beta : \bar{u}$, the term obtained by simultaneously substituting α_i for y_i, $i = 1, \dots, m$ is (somewhat inaccurately) denoted $\beta\alpha$.

$$\textbf{(var)}\ \frac{x \text{ in } \Gamma}{x \overset{e}{=} x} \qquad \textbf{(sym)}\ \frac{\alpha \overset{e}{=} \beta}{\beta \overset{e}{=} \alpha} \qquad \textbf{(tr)}\ \frac{\begin{array}{c}\alpha \overset{e}{=} \beta \\ \beta \overset{e}{=} \gamma\end{array}}{\alpha \overset{e}{=} \gamma} \qquad \textbf{(cong)}\ \frac{\begin{array}{c}\alpha \overset{e}{=} \beta : \bar{t} \\ f : \bar{t} \to u \\ \operatorname{def} f(\alpha)\end{array}}{f(\alpha) \overset{e}{=} f(\beta)}$$

$$\textbf{(str)}\ \frac{\operatorname{def} f(\alpha)}{\operatorname{def} \alpha} \qquad \textbf{(ax)}\ \frac{\begin{array}{c}\phi \Rightarrow_{\bar{y}:\bar{t}} \psi \in \mathcal{A} \\ \Gamma \rhd \alpha : \bar{t} \\ \phi\alpha \qquad \operatorname{def} \alpha\end{array}}{\psi\alpha} \qquad \textbf{(unit)}\ \frac{\operatorname{def} \alpha : ()}{\alpha \overset{e}{=} ()}$$

Fig. 3. Deduction rules for existential equality in context Γ

$\mathcal{A}$ is a set of *axioms* that take the form of implications: an *existential equation* in context Γ is a pair $\phi = (\alpha^1, \alpha^2)$, consisting of two terms $\Gamma \rhd \alpha^i : t$, $i = 1, 2$, for some sort t. Such an equation is written $\alpha^1 \overset{e}{=} \alpha^2$ or, explicitly indicating the result sorts, $\alpha^1 \overset{e}{=} \alpha^2 : t$. We will also write equations between multi-terms, thereby meaning just the obvious *sets* of equations between terms; such equations are also referred to as existential equations. We use notations like $\phi \wedge \psi$ and `true` to denote the union of sets of equations and the empty set of equations, respectively. An *implication* in context Γ is a sentence of the form $\phi \Rightarrow \psi$, where ϕ and ψ are existential equations (between multi-terms) in context Γ. The context may be explicitly indicated by writing $\phi \Rightarrow_\Gamma \psi$. $\alpha \overset{e}{=} \alpha$ is sometimes abbreviated as $\operatorname{def} \alpha$. For $\Gamma \rhd \alpha : \bar{t}$ and an existential equation $\psi = (\beta^1, \beta^2)$ in context $\bar{y} : \bar{t}$, $\psi\alpha$ denotes the equation $\beta^1\alpha \overset{e}{=} \beta^2\alpha$ in context Γ.

In Figure 3, we present the rules of a deduction system for existential equality associated to a specification $\mathcal{S} = (\Omega, \mathcal{A})$. The rules given in the figure are parametrized over a *fixed* context Γ (which appears explicitly in the rules (var) and (ax)); proofs in this system are best thought of as beginning with the words 'Let $x_1 : s_1$, $x_2 : s_2 \ldots$'. Both the congruence rule and the strictness rule readily generalize to arbitrary terms in place of basic operations. The system is obviously sound w.r.t. the notion of model defined below in the sense that, whenever a valuation of the context variables in a given model satisfies a set of equations, then it satisfies all equations that can be deduced from that set. The system will turn out to be complete as well. We write $\phi \vdash_\Gamma \psi$ if ψ can be deduced from ϕ in context Γ by means of these rules; in this case, we say that the implication $\phi \Rightarrow_\Gamma \psi$ is a *theorem*.

A morphism between specifications is defined as a theory morphism in the usual sense, i.e. a signature morphism that transforms axioms into theorems. Thus, specifications form a category **epSpec**.

A *model* of the signature Ω is an algebra that interprets sorts as sets and symbols as partial operations in the usual way. Given such a model, a *valuation* (for a context) is an interpretation of the variables in the context by elements of the appropriate sorts; valuations extend to terms as usual. A valuation η *satisfies* an existential equation (in the same context) iff both terms in the equation are defined and equal under η; the satisfaction of implications is defined correspondingly. A model of a specification $\mathcal{S} = (\Omega, \mathcal{A})$ is a model of Ω in which all implications in $\mathcal{A}$ are satisfied by all valuations for their context; it is clear how this leads to a model functor $\mathbf{Mod}_{eps} : \mathbf{epSpec}^{op} \to \mathbf{CAT}$.

Using the deduction system of Figure 3, we can now construct a lex category $\mathsf{Th}_{eps}(\mathcal{S})$ from $\mathcal{S}$: the objects of $\mathsf{Th}_{eps}(\mathcal{S})$ are pairs $A = (\Gamma, \phi)$ consisting of a context Γ and an existential equation ϕ in that context. Morphisms $(\Gamma, \phi) \to (\bar{y} : \bar{t}, \psi)$ are terms in context $\Gamma \rhd \alpha : \bar{t}$ such that

$$\phi \vdash_\Gamma \psi\alpha \wedge \operatorname{def} \alpha,$$

taken modulo existential equality deducible from ϕ in context Γ. The identity on (Γ, ϕ) is represented by $\bar{x}$, where $\Gamma = (\bar{x} : \bar{s})$. Composition is defined via simultaneous substitution of representing terms. This is a well-defined operation thanks to the generalized congruence rule and the following meta-theorem, which also shows that the composite is indeed a morphism:

Proposition 13 *If ϕ and ψ are existential equations in context $\bar{y} : \bar{t}$ and $\Gamma \rhd \alpha : \bar{t}$, then $\phi \vdash_{\bar{y}:\bar{t}} \psi$ implies $\phi\alpha \wedge \operatorname{def} \alpha \vdash_\Gamma \psi\alpha$.*

PROOF. Induction over the length of the derivation of $\phi \vdash_{\bar{y}:\bar{t}} \psi$. □

As expected, 'concatenation' of contexts (which may require variable renaming) together with conjunction of existential equations defines finite products, and

$$\bar{x} : (\bar{x} : \bar{s}, \phi \wedge \alpha^1 \overset{e}{=} \alpha^2) \to (\bar{x} : \bar{s}, \phi)$$

is an equalizer of $\alpha^1, \alpha^2 : (\bar{x} : \bar{s}, \phi) \to B$ in $\mathsf{Th}_{eps}(\mathcal{S})$. Thus, $\mathsf{Th}_{eps}(\mathcal{S})$ is indeed a lex category. It is easily seen that Th_{eps} extends to a functor

$$\mathsf{Th}_{eps} : \mathbf{epSpec} \to \mathbf{Lex}$$

(since signature morphisms act in the obvious way on contexts and terms, and preserve deduction when further extended to existential equations). Similarly as in [13], one easily verifies that one has an equivalence of categories

$$\mathbf{Mod}_{lex}(\mathsf{Th}_{eps}(\mathcal{S})) \to \mathbf{Mod}_{eps}(\mathcal{S})$$

(natural in $\mathcal{S}$) and concludes, using the fact that representable functors are models of $\mathsf{Th}_{eps}(\mathcal{S})$, that the deduction system of Figure 3 is complete.

$$\left.\begin{array}{r}\mathrm{def}(e\cdot f)(\bar{x}) \Rightarrow_{\bar{x}:\bar{s}} (e\cdot f)(\bar{x}) \stackrel{e}{=} e(f(\bar{x}))\\ \mathrm{def}\, e(f(\bar{x})) \Rightarrow_{\bar{x}:\bar{s}} (e\cdot f)(\bar{x}) \stackrel{e}{=} e(f(\bar{x}))\end{array}\right\}\begin{array}{l} f:\bar{s}\to t \text{ function symbol,}\\ e:t\to u \text{ embedding;}\end{array}$$

$$\left.\begin{array}{r}\mathrm{def}(f\cdot \bar{d})(\bar{x}) \Rightarrow_{\bar{x}:\bar{v}} (f\cdot \bar{d})(\bar{x}) \stackrel{e}{=} f(\bar{d}(\bar{x}))\\ \mathrm{def}\, f(\bar{d}(\bar{x})) \Rightarrow_{\bar{x}:\bar{v}} (f\cdot \bar{d})(\bar{x}) \stackrel{e}{=} f(\bar{d}(\bar{x}))\end{array}\right\}\begin{array}{l} f \text{ function or predicate symbol,}\\ \bar{d}:\bar{v}\to \bar{s} \text{ embeddings;}\end{array}$$

$$\begin{array}{rl} \mathsf{true} \Rightarrow_{x:s} e(d(x)) \stackrel{e}{=} (e\circ d)(x), & d:s\to t,\ e:t\to u \text{ embeddings;}\\ \mathsf{true} \Rightarrow_{x:s} id_s(x) \stackrel{e}{=} x, & s \text{ sort;}\\ \mathsf{true} \Rightarrow_{\bar{x}:\bar{s}} \mathrm{def}\, f(x), & f:\bar{s}\to t \text{ total function symbol;}\\ e(x) \stackrel{e}{=} e(y) \Rightarrow_{x:s,\,y:s} x \stackrel{e}{=} y, & e:s\to t \text{ embedding.}\end{array}$$

Fig. 4. Axioms associated to an enriched CASL signature

It is straightforward to translate an enriched CASL signature Σ into an equational partial specification $\mathsf{Spec}(\Sigma) = (\Omega, \mathcal{A})$: the sorts in Ω are the sorts of Σ; the symbols in Ω are the function and predicate symbols of Σ, where the range of all predicate symbols is the 'sort' (), and the embedding symbols with the obvious unary profiles. The axioms are given in Figure 4 (note: totality of embedding symbols is a consequence of the axiom for composition of embeddings). This obviously extends to a functor

$$\mathsf{Spec} : \mathbf{enrCASLsign} \to \mathbf{epSpec}.$$

It is easily verified that $\mathbf{Mod}_{eps} \circ \mathsf{Spec}^{op}$ is naturally isomorphic to $\mathbf{Mod}_{enr}$.

Remark 14 The translation functor Spec can be modified to retain predicates and include an elementhood predicate and partial projection functions for subsorts, thus providing an extension to enriched CASL of the existing embedding of the CASL logic $SubPFOL^=$ (subsorted partial first-order logic with equality) into $PFOL^=$ (partial first-order logic with equality) [12].

Putting the two translations together, we have a representation

$$\mathsf{Th}_{enr} := \mathsf{Th}_{eps} \circ \mathsf{Spec} : \mathbf{enrCASLsign} \to \mathbf{Lex}$$

of enriched signatures as lex categories.

Theorem 15 Th_{enr} *reflects equivalences.*

The proof relies on two statements (Lemmas 17 and 18 below) concerning deduction in $\mathsf{Spec}(\Sigma) = \mathcal{S}$:

Definition 16 A (multi-)term α in $\mathcal{S}$ is said to *reduce* to an embedding d (list $\bar{d}$ of embeddings) in Σ if all symbols in α are embeddings, and their composite is d ($\bar{d}$), with the composite of an empty chain of embeddings taken to be the identity. In this case we write $\alpha \twoheadrightarrow d$ ($\alpha \twoheadrightarrow \bar{d}$). Reduction to a function or predicate symbol f in Σ ($\alpha \twoheadrightarrow f$) is defined analogously, using also the left and right actions of the sort category.

Lemma 17 *Let $\bar{x} : \bar{s} \rhd \alpha, \beta : t$, and let $h : \bar{s} \to t$ be a symbol in $\mathcal{S}$ such that*

$$\operatorname{def} \alpha \vdash_{\bar{x}:\bar{s}} \beta \overset{e}{=} h(\bar{x}).$$

Then $\beta \twoheadrightarrow h$.

PROOF. Call an existential equation $\alpha_1 \overset{e}{=} \alpha_2$ *reductively equivalent* if, for each $a \in \Omega$, $\alpha_1 \twoheadrightarrow a$ is equivalent to $\alpha_2 \twoheadrightarrow a$. The set of reductively equivalent equations is closed under application of the deduction rules of Figure 3 (with the axioms given in Figure 4). Thus, since $\operatorname{def} \alpha$ is reductively equivalent, so is $\beta \overset{e}{=} h(x_w)$; this implies the claim. □

Lemma 18 *If*

$$\mathsf{true} \vdash_{\bar{x}:\bar{s}} \operatorname{def} f(\bar{x})$$

in $\mathcal{S}$ for a function symbol $f : \bar{s} \to t$ in Σ, then f is a total symbol.

PROOF. Call an existential equation ϕ *leaf-total* if, whenever a term of the form $g(\beta)$, where g is a function symbol and $\beta \twoheadrightarrow \bar{d}$ for a list $\bar{d}$ of embeddings, occurs as a subterm in ϕ, then $g \cdot \bar{d}$ is a total symbol. The set of leaf-total equations is closed under deduction. Thus, $\operatorname{def} f(\bar{x})$ as in the statement is leaf-total, which implies the claim. □

In a nutshell, this means that the only terms that are provably equal to symbols are the obvious candidates, and that only total function symbols are provably total.

Remark 19 In this context, it is interesting to note that, even for standard CASL signatures, there are provably total *terms* that cannot be expressed using only total function symbols. Consider the CASL specification

spec SP =
sorts $s < t$;
$u, v < w$
ops $c : u$;
$c : v$;
$f : w \to ? \; t$;
$f : u \to ? \; s$;
$f : v \to t$;
$g : s \to s$

The term $() \triangleright g(f(c)) : s$ is provably defined, but can only be expressed using the partial profile $f : u \to ?\ s$. The reason for this phenomenon appears to be the absence of intersection types in CASL.

PROOF (Theorem 15)

Let $\sigma : \Sigma_1 \to \Sigma_2$ be a morphism of enriched signatures such that $\mathsf{Th}_{enr}(\sigma)$ is an equivalence.

σ is isomorphism-dense: Let s be a sort in Σ_2. Then there exists an isomorphism $\alpha : (x : s, \mathsf{true}) \to \mathsf{Th}_{enr}(\sigma)(B)$ for some object B of $\mathsf{Th}_{enr}(\Sigma_1)$. Lemma 17 implies that both α and its inverse reduce to embeddings, so that s is isomorphic to an object of the form $\sigma(t)$ in the sort category of Σ_2.

σ is faithful: Let $f, g : \bar{s} \to t$ be function symbols in Σ_1 (predicate symbols and embeddings are treated analogously) such that $\sigma(f) = \sigma(g)$. Then by fullness of $\mathsf{Th}_{enr}(\sigma)$, the morphism

$$\bar{x} : (\bar{x} : \bar{s}, \operatorname{def} f(\bar{x}) \wedge \operatorname{def} g(\bar{x})) \to (\bar{x} : \bar{s}, \operatorname{def} f(\bar{x}))$$

in $\mathsf{Th}_{enr}(\Sigma_1)$ is an isomorphism, the inverse of which is necessarily $\bar{x}$. Thus, we have morphisms

$$f(\bar{x}), g(\bar{x}) : (\bar{x} : \bar{s}, \operatorname{def} f(\bar{x})) \to (z : t, \mathsf{true})$$

in $\mathsf{Th}_{enr}(\Sigma_1)$. By faithfulness of $\mathsf{Th}_{enr}(\sigma)$, these morphisms are equal, i.e.

$$\operatorname{def} f(\bar{x}) \vdash_{\bar{x}:\bar{s}} f(\bar{x}) \stackrel{e}{=} g(\bar{x}).$$

This implies $f = g$ by Lemma 17.

σ is full on embeddings: Let $e : \sigma(s) \to \sigma(t)$ be an embedding in Σ_2. By fullness of $\mathsf{Th}_{enr}(\sigma)$, there exists a morphism

$$\alpha : (x : s, \mathsf{true}) \to (y : t, \mathsf{true})$$

in $\mathsf{Th}_{enr}(\Sigma_1)$ such that $\mathsf{Th}_{enr}(\sigma)(\alpha) = e$, i.e.

$$\vdash_{x:\sigma(s)} \sigma(\alpha) \stackrel{e}{=} e(x).$$

By Lemma 17, $\sigma(\alpha) \twoheadrightarrow e$; hence, $\alpha \twoheadrightarrow d$ for some embedding $d : s \to t$, and $\sigma(d) = e$.

σ is full on symbols: Let $f : \sigma(\bar{s}) \to \sigma(t)$ be a function symbol in Σ_2 (predicate symbols are treated analogously). Let $\Gamma = (\bar{x} : \sigma(\bar{s}))$, and let

$A = (\Gamma, \operatorname{def} f(\bar{x}))$. By isomorphism-density of $\mathsf{Th}_{enr}(\sigma)$, we have an isomorphism $\beta : A \to \mathsf{Th}_{enr}(\sigma)(B)$ in $\mathsf{Th}_{enr}(\Sigma_2)$ for some $B = (\bar{y} : \bar{u}, \phi)$ in $\mathsf{Th}_{enr}(\Sigma_1)$. By Lemma 17, β must be equal to a list of embeddings. Thus, by fullness of σ on embeddings, $\beta = \sigma(\bar{d})(\bar{x})$ for some list $\bar{d} : \bar{s} \to \bar{u}$ of sort isomorphisms.

The morphism

$$f(\bar{x}) \circ \beta^{-1} : \mathsf{Th}_{enr}(\sigma)(B) \to (z : \sigma(t), \mathsf{true})$$

has a preimage $\alpha : B \to (z : t, \mathsf{true})$ under $\mathsf{Th}_{enr}(\sigma)$. Then $\sigma(\alpha)\beta = f(\bar{x})$ as morphisms $A \to (z : \sigma(t), \mathsf{true})$, i.e.

$$\operatorname{def} f(\bar{x}) \vdash_\Gamma f(\bar{x}) \overset{e}{=} \sigma(\alpha)\sigma(\bar{d})(\bar{x}),$$

so that $\sigma(\alpha\bar{d}(\bar{x})) \twoheadrightarrow f$ (where now $\bar{x} : \bar{s}$) by Lemma 17. This implies $\alpha\bar{d}(\bar{x}) \twoheadrightarrow g$ for some symbol g, and $\sigma(g) = f$.

σ reflects totality: If $f : \bar{s} \to t$ is a function symbol such that $\sigma(f)$ is total, then in $\mathsf{Th}_{enr}(\Sigma_2)$ we have

$$\bar{x} : (\bar{x} : \sigma(\bar{s}), \mathsf{true}) \to (\bar{x} : \sigma(\bar{s}), \operatorname{def} \sigma(f)(\bar{x})),$$

and so by fullness of $\mathsf{Th}_{enr}(\sigma)$,

$$\bar{x} : (\bar{x} : \bar{s}, \mathsf{true}) \to (\bar{x} : \bar{s}, \operatorname{def} f(\bar{x}))$$

is a morphism in $\mathsf{Th}_{enr}(\Sigma_1)$, which implies $\vdash_{\bar{x}:\bar{s}} \operatorname{def} f(\bar{x})$. By Lemma 18, f is total. □

By Theorems 12 and 15, $\mathbf{Mod}_{enr}$ reflects equivalences. Since it is trivial to show that a signature morphism σ is bijective on sorts if $\mathbf{Mod}_{enr}(\sigma)$ is an isomorphism, this implies

Corollary 20 $\mathbf{Mod}_{enr}$ *is definitionally complete.*

In particular, we now have a necessary and sufficient criterion for amalgamability in **CASLsign**:

Corollary 21 *A cocone in* **CASLsign** *is amalgamable iff its image under* Enr *is a colimit in* **enrCASLsign**.

8 Static Analysis via Enriched Signatures

We are now ready to translate the amalgamation conditions that appear in the rules for unit application and amalgamation in the extended static semantics to entirely static conditions. To this end, we assume that we have a cocomplete category **EnrSign** of enriched signatures with a model functor $\mathbf{Mod}_{enr} : \mathbf{EnrSign}^{op} \to \mathbf{CAT}$ which has the amalgamation property and is definitionally complete, and a functor $\Phi : \mathbf{Sign} \to \mathbf{EnrSign}$ such that $\mathbf{Mod}_{enr} \circ \Phi^{op}$ and $\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbf{CAT}$ are naturally isomorphic. For the standard CASL institution, these data have been constructed in Section 6, with definitional completeness proved in Section 7.

Recall that the amalgamation conditions of Section 4 required diagrams to ensure amalgamability for certain extensions, which was defined as unique extendability of coherent families of models (and their morphisms). By the assumption on the model functors, this requirement is equivalent to the corresponding statement for the translations of the diagrams via Φ. By the amalgamation property, a D-coherent family of models for a diagram D in **EnrSign** is essentially the same as a model of the colimit signature $\operatorname{colim} D$. Thanks to definitional completeness, we thus have:

Proposition 22 *Let D' be a diagram in* **EnrSign** *that extends D. D ensures amalgamability for D' iff the induced morphism* $\operatorname{colim} D \to \operatorname{colim} D'$ *is an isomorphism.*

This condition can be checked by means of a factorization property in the cases of interest here:

Definition 23 Let $\mathbf{A}$ be a category, and let $D' : \mathbf{I}' \to \mathbf{A}$ be a diagram that extends $D : \mathbf{I} \to \mathbf{A}$. Then D *covers* D' if, for each $j \in \operatorname{Ob} \mathbf{I}'$, the sink of all $D'(m) : D(i) \to D'(j)$, where $i \in \operatorname{Ob} \mathbf{I}$ and $m : i \to j$ in $\mathbf{I}'$, is an episink.

Proposition 24 *Let D and D' be diagrams in a cocomplete category, where D' extends D. If D covers D', then the induced morphism* $\operatorname{colim} D \to \operatorname{colim} D'$ *is an isomorphism iff the colimit cocone for D extends to a cocone for D'.*

PROOF. Let f denote the induced morphism $\operatorname{colim} D \to \operatorname{colim} D'$. If f is an isomorphism, then the extension $D' \to \operatorname{colim} D$ is obtained by composing the colimit cocone for D' with f^{-1}. Conversely, if $\nu : D' \to \operatorname{colim} D$ extends the colimit $\kappa : D \to \operatorname{colim} D$, then ν factors through the colimit $\mu : D' \to \operatorname{colim} D'$ by a morphism $g : \operatorname{colim} D' \to \operatorname{colim} D$. Since κ is an episink, $gf = id$. The covering condition ensures $f\nu = \mu$, so that $fg = id$, since $\nu = g\mu$ and μ is an episink. $\square$

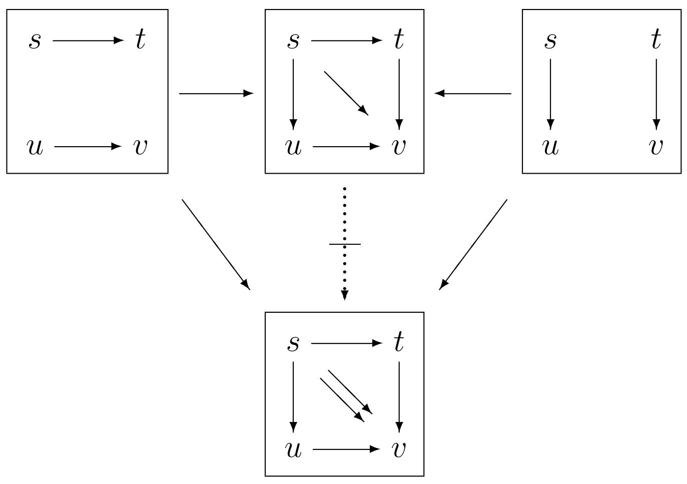
Remark 25 In case $\mathbf{Mod}_{enr}$ has the amalgamation property, but fails to be definitionally complete, the conditions above still provide a *sufficient* amalgamability criterion (more precisely: the criterion given in Proposition 22 is, in the weaker setting, sufficient but not in general necessary, while Proposition 24 remains unaffected).

The covering condition is satisfied when amalgamability is checked in the rules of the extended static semantics in the two cases that correspond to CASL constructs: unit application, where the diagram is extended by a selected pushout (cf. Section 3), and CASL amalgamation, where the diagram is extended by a sink consisting of two inclusions of standard CASL signatures into their union. The factorization condition concerns then the translation under Φ of a sink (τ_1, τ_2) in **CASLsign**, which in both cases yields an episink $(\Phi(\tau_1), \Phi(\tau_2))$ in **enrCASLsign**:

$$\begin{tikzcd}
\Phi(D(i_1)) \arrow[r, "\Phi(\tau_1)"] \arrow[dr, "\mu_{i_1}"'] & \Phi(\Sigma) \arrow[d, dotted, "\theta"'] & \Phi(D(i_2)) \arrow[l, "\Phi(\tau_2)"'] \arrow[dl, "\mu_{i_2}"] \\
& \operatorname{colim} \Phi \circ D &
\end{tikzcd} \tag{3}$$

(D denotes the original diagram in **CASLsign**, and the μ_i denote the colimit injections in **enrCASLsign**).

Example 26 The simple union of sort preorders presented in Example 4, Diagram (1), fails to admit a factorization as above: the colimit will have two different sort embeddings $s \to v$ as depicted in Diagram (2). Thus, the above diagram specializes to



and it is clear that the morphism indicated by the dotted arrow fails to exist.

Thus we have essentially reduced the amalgamation problem to proving the existence of the factorizations required in the above proposition.

In order to provide a construction for the factorization θ in **enrCASLsign**, we note additionally that in the two cases of interest the images of $\Phi(\tau_1)$ and $\Phi(\tau_2)$ jointly generate $\Phi(\Sigma)$ (categorically: $(\Phi(\tau_1), \Phi(\tau_2))$ is an extremal episink), i.e. that each symbol in $\Phi(\Sigma)$ can be built up from symbols coming from the $\Phi(\tau_i)$ by means of the operations defining enriched signatures (composition and identities, left and right actions), and the totality predicate is the smallest one with the property that the τ_i preserve totality. It is now clear how θ has to be defined if it exists, namely by extending the effect of μ_{i_1} and μ_{i_2} to composite symbols (e.g., $\theta(\tau_1(e) \cdot \tau_2(f)) = \mu_{i_1}(e) \cdot \mu_{i_2}(f)$). The task that remains is to check whether this yields a well-defined map (which is, then, automatically a signature morphism). This requires a calculus for proving equality of morphisms and symbols in the colimit; see [25]. Due to the way enriched signatures are defined, this problem is at least as hard as the corresponding one for colimits of left cancellable categories, which is in fact shown to be undecidable in general, but decidable by a polynomial algorithm in practically relevant cases, see [25].

Remark 27 Using coproducts of signatures, the factorization condition for sinks is, in fact, easily reducible to the case where D' extends D by a single new morphism. It is easy to check that coproducts of standard CASL signatures are preserved by the representation functor to enriched CASL signatures. Therefore, in the cases of interest as depicted by Diagram (3), one may always replace D by a diagram D^+ extending D by a new node i for the coproduct of $D(i_1)$ and $D(i_2)$ together with edges for the coproduct injections; preservation of coproducts implies that the colimits of $\Phi \circ D$ and of $\Phi \circ D^+$ coincide. Let then $\tau : D^+(i) \to \Sigma$ be the factorization of the sink (τ_1, τ_2) through the coproduct. $\Phi(\tau)$ is an (extremal) epimorphism iff $(\Phi(\tau_1), \Phi(\tau_2))$ is an (extremal) episink. Now the colimit injection $\mu_i : \Phi(D^+(i)) \to \operatorname{colim} \Phi \circ D^+$ factors (μ_{i_1}, μ_{i_2}) through the coproduct $\Phi(D^+(i))$ of $\Phi(D(i_1))$ and $\Phi(D(i_2))$. Therefore, the factorization condition for the sink as depicted by Diagram (3) is equivalent to the existence of a factorization θ for μ_i through $\Phi(\tau)$:

$$\begin{tikzcd}
\Phi(D^+(i)) \arrow[r, "\Phi(\tau)"] \arrow[dr, "\mu_i"'] & \Phi(\Sigma) \arrow[d, dotted, "\theta"'] \\
 & \operatorname{colim} \Phi \circ D^+
\end{tikzcd}$$

This simplification is exploited in [25].

Remark 28 In the construction of a non-generative static semantics for unit application (cf. Section 4), the above machinery provides an easy criterion for the equivalence of two instantiations: let U be a parametrized unit over $\tau : \Sigma_1 \to \Sigma_2$. Two actual argument models are considered to be (partially) equivalent if they reduce (via fitting morphisms $\sigma_i : \Sigma_1 \to D(j_i)$, $i = 1, 2$, where D denotes the present context diagram with nodes j_1, j_2 given by the extended static analysis of the respective argument terms) to the same model of the parameter signature Σ_1. This will be the case for all pairs of models that appear in D-coherent families if

$$\mu_{j_1} \circ \Phi(\sigma_1) = \mu_{j_2} \circ \Phi(\sigma_2),$$

where μ is the colimit cocone for $\Phi \circ D$. In this case, we can use the same edge of the diagram scheme to represent τ in both applications of U; this has the effect that the two results with signatures Σ_R^1 and Σ_R^2 share to exactly the right degree via the maps $\Sigma_2 \to \Sigma_R^i$, $i = 1, 2$, that appear in the defining pushouts. The resulting semantics would capture all the aspects of 'non-generativity' that can be detected statically, without considering the implications of the actual semantic model identity.

9 Conclusions and future work

We have used a small and modified but quite representative subset of Casl architectural specifications to present and discuss its complete semantics given in an institution-independent way. Besides the basic static and model semantics, we have laid out an extended static analysis, where sharing information between models is stored as a diagram of signatures. This has allowed us to formulate the required amalgamability conditions 'almost' statically, i.e. without referring to particular models constructed. Given a representation (preserving the model categories) of the underlying institution in one that has the amalgamation property, these conditions can be replaced by literally static ones. Moreover, in the case that also the converse of the amalgamation property holds, the static conditions may in a suitable sense be regarded as 'complete' w.r.t. the model-theoretic ones.

The results presented here are independent of the logical structure of institutions — sentences and satisfaction do not play any explicit role here (except for being used implicitly in basic specifications, of course). However, the sentences become relevant as soon as we discuss further issues of verification in architectural specifications (represented here by the remaining fitting condition in the semantics of unit applications). As proposed in [24], formal proof obligations can be extracted from such conditions using colimits of specification diagrams, but only if the underlying institution has the amalgamation property. The technique proposed here should allow us to circumvent this requirement: specification diagrams can be translated to the enriched signature category and put together there, opening a way also for the development of tools supporting validation and verification of Casl (architectural) specifications.

Applying this semantics to the case of the Casl institution required the construction of a representation of the Casl institution (the extension to sentences, disregarded here, is straightforward) in enriched Casl, an institution with a category of enriched signatures that satisfies the amalgamation property (and also its converse). This representation carries the additional benefit of making a number of results (e.g. concerning normal forms and proof systems) about institution-independent specification languages applicable to CASL.

In more detail, we have modified Reynolds' approach [35] to subsorting via sort categories by using actions of the sort category on function and predicate symbols. In this way we elegantly deal with the problems of both overloading and amalgamation. Moreover, the associated logic admits a reduction to partial conditional equational logic. For the latter, we provide a sound and complete proof system, extending and simplifying the work of [13].

Typically, the use of enriched CASL will be as follows. Specifications are written in ordinary CASL. In situations where the user inputs a certain combination (colimit) of signatures (e.g. when writing an instantiation of a parameterized specification), the natural requirement will be to check whether this combination remains a colimit in the category of enriched CASL signatures, thus guaranteeing amalgamability of models. At this stage, enriched CASL remains completely hidden from the user. In contrast to this, there are also situations where a combination of signatures (colimit) is automatically produced by a tool (e.g. during a proof in a development graph or during static analysis of architectural specifications). In these situations, it is advisable to use the properly enriched signatures that may crop up as intermediate results rather than to reject them immediately. Theorem proving in the enriched CASL logic is eased by the fact that this logic can be embedded into partial first-order logic with equality in much the same way as the CASL logic. Algorithms related to the actual computation of colimits of enriched signatures (a prerequisite for the development of tools for architectural specifications and proof support) are discussed in [25].

Future lines of research include the generalization of the techniques developed in this work to arbitrary institutions, resulting in particular in a generic procedure for 'making institutions amalgamable'. It appears that institutions where finite amalgamation fails fall into two classes — one where preservation of pushouts fails (with the CASL institution as an example), and one where preservation of initial objects (and coproducts) fails. The institutions with unnamed universes mentioned in the introduction belong to the latter class. It is plausible to assume that these institutions can — in analogy to the extension of unsorted to multisorted logic — be made amalgamable by introducing 'multiple universes'.

At a level that stays closer to CASL-related problems, one may wonder whether the techniques applied here work for derived signature morphisms, i.e. ones that may map symbols to arbitrary terms, as well (or even for the yet more general derived signature morphisms discussed in [7]). The first complication that arises here is that, in order to ensure cocompleteness of the signature category, one will have to include axioms of some limited form (at least strong equations) as constituents of generalized signatures. It is, then, easy to find a sufficient criterion for amalgamability of cocones by means of an encoding of such generalized signatures as equational partial specifications in much the same sense as above. However, it is an open problem how to obtain a criterion that is also necessary, i.e. how to find an intermediate step similar to the enriched signatures defined above that allows one to also establish definitional completeness. Furthermore, the notion of derived signature morphism itself requires, in a setting with partial functions, rather more care than usual; cf. [41].

We conjecture that not only the amalgamation property, but also the Craig interpolation property (in the weakened form valid in multisorted logic [9]), which fails in standard CASL, holds in enriched CASL. This property, which seems to be related to amalgamation in some way [37], is required by the institution-independent proof calculus for ASL-like specifications [10].

Acknowledgements

Partial support by the CoFI Working Group (ESPRIT WG 29432) is gratefully acknowledged, as well as the work of the CoFI Language and Semantics Task Groups. Moreover, we wish to thank Michel Bidoit and Donald Sannella for collaboration on architectural specifications, Fernando Orejas for reference [35], and Maura Cerioli for pointing out the failure of amalgamation for CASL models. This paper is a revised and extended version of [42,43].

References

[1] CoFI, *Catalogue of existing frameworks*, in [14], 1996.

[2] J. Adámek, H. Herrlich, and G. E. Strecker, *Abstract and Concrete Categories*, 2nd ed., Wiley Interscience, New York, 1990.

[3] E. Astesiano, M. Bidoit, H. Kirchner, B. Krieg-Brückner, P. D. Mosses, D. Sannella, and A. Tarlecki, CASL*: the Common Algebraic Specification Language*, Theoret. Comput. Sci. (2003), to appear.

[4] H. Baumeister, *Relations between abstract datatypes modeled as abstract datatypes*, Ph.D. thesis, Universität des Saarlandes, 1998.

[5] J.A. Bergstra, J. Heering, and P. Klint, *Module algebra*, J. ACM **37** (1990), 335–372.

[6] M. Bidoit, D. Sannella, and A. Tarlecki, *Architectural specifications in* CASL, Algebraic Methodology and Software Technology, LNCS, vol. 1548, Springer, 1999, pp. 341–357, full version to appear in Formal Aspects of Computing.

[7] ______, *Global development via local observational construction steps*, Mathematical Foundations of Computer Science, LNCS, Springer, 2002, to appear.

[8] F. Borceux, *Handbook of Categorical Algebra 1*, Cambridge, 1994.

[9] T. Borzyszkowski, *Generalized interpolation in* CASL, Inform. Process. Lett. **76** (2000), 19–24.

[10] ______, *Logical systems for structured specifications*, Theoret. Comput. Sci. (2003), to appear.

[11] R. M. Burstall and J. A. Goguen, *Putting theories together to make specifications.*, Proc. 5th Intl. Joint Conf. on Artificial Intelligence, Carnegie-Mellon University, 1977, pp. 1045–1058.

[12] M. Cerioli, A. Haxthausen, B. Krieg-Brückner, and T. Mossakowski, *Permissive subsorted partial logic in* CASL, Algebraic Methodology and Software Technology, LNCS, vol. 1349, Springer, 1997, pp. 91–107.

[13] I. Claßen, M. Große-Rhode, and U. Wolter, *Categorical concepts for parameterized partial specifications*, Math. Struct. Comput. Sci. **5** (1995), 153–188.

[14] CoFI, *The Common Framework Initiative for algebraic specification and development, electronic archives*, accessible by WWW[4] and FTP[5].

[15] CoFI Language Design Task Group, CASL *Summary, version 1.0*, Documents/CASL/Summary, in [14], July 1999.

[16] CoFI Semantics Task Group, CASL – *The* CoFI *Algebraic Specification Language – Semantics*, Note S-9 (version 0.96), in [14], July 1999.

[17] R. Diaconescu, J. Goguen, and P. Stefaneas, *Logical support for modularisation*, Logical Environments, Cambridge, 1993, pp. 83–130.

[18] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification 1*, Springer, 1985.

[19] ______, *Fundamentals of Algebraic Specification 2*, Springer, 1990.

[20] J. Goguen and R. Burstall, *Institutions: Abstract model theory for specification and programming*, J. ACM **39** (1992), 95–146.

[21] J. V. Guttag, J. J. Horning, S. J. Garland, K. D. Jones, A. Modet, and J. M. Wing, *Larch: Languages and Tools for Formal Specification*, Springer, 1993.

[22] R. Harper, F. Honsell, and G. D. Plotkin, *A framework for defining logics*, J. ACM **40** (1993), 143–184.

[4] http://www.cofi.info

[5] ftp://ftp.brics.dk/Projects/CoFI

[23] A. E. Haxthausen and F. Nickl, *Pushouts of order-sorted algebraic specifications*, Algebraic Methodology And Software Technology, LNCS, vol. 1101, Springer, 1996, pp. 132–147.

[24] P. Hoffman, *Verifying architectural specifications*, 15th Workshop on Algebraic Development Techniques, LNCS, vol. 2267, Springer, 2001, pp. 152–175.

[25] B. Klin, P. Hoffman, A. Tarlecki, L. Schröder, and T. Mossakowski, *Checking amalgamability conditions for* CASL *architectural specifications*, Mathematical Foundations of Computer Science, LNCS, vol. 2136, Springer, 2001, pp. 451–463.

[26] F. W. Lawvere, *Functorial semantics of algebraic theories*, Proc. Natl. Acad. Sci. USA **50** (1963), 869–872.

[27] S. Mac Lane, *Categories for the Working Mathematician*, 2nd ed., Springer, 1997.

[28] M. Makkai and A. M. Pitts, *Some results on locally finitely presentable categories*, Trans. Amer. Math. Soc. **299** (1987), 473–496.

[29] K. Meinke and J. V. Tucker (eds.), *Many-sorted Logic and its Applications*, Wiley, 1993.

[30] T. Mossakowski, *Colimits of order-sorted specifications*, 12th Workshop on Algebraic Development Techniques, LNCS, vol. 1376, Springer, 1998, pp. 316–332.

[31] ______, *Specification in an arbitrary institution with symbols*, 14th Workshop on Algebraic Development Techniques, LNCS, vol. 1827, Springer, 2000, pp. 252–270.

[32] T. Mossakowski, S. Autexier, and D. Hutter, *Extending development graphs with hiding*, Fundamental Approaches to Software Engineering, LNCS, vol. 2029, Springer, 2001, pp. 269–283.

[33] L. C. Paulson, *ML for the Working Programmer*, 2nd ed., Cambridge, New York, 1996.

[34] B. Pierce, *Types and Programming Languages*, MIT Press, 2002.

[35] J. C. Reynolds, *Using category theory to design implicit conversions and generic operators*, Semantics-Directed Compiler Generation, LNCS, vol. 94, Springer, 1980, pp. 211–258.

[36] M. Roggenbach and L. Schröder, *Towards trustworthy specifications I: Consistency checks*, 15th Workshop on Algebraic Development Techniques, LNCS, vol. 2267, Springer, 2001, pp. 305–327.

[37] A. Salibra and G. Scollo, *Interpolation and compactness in categories of pre-institutions*, Math. Struct. Comput. Sci. **6** (1996), 261–286.

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

[39] D. Sannella and A. Tarlecki, *Specifications in an arbitrary institution*, Inform. and Comput. **76** (1988), 165–210.

[40] D. Sannella and M. Wirsing, *Specification languages*, Algebraic Foundations of Systems Specifications (A. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, eds.), Springer, 1999, pp. 243–272.

[41] L. Schröder, *Classifying categories for partial equational logic*, Category Theory and Computer Science, Electron. Notes Comput. Sci., 2002, to appear.

[42] L. Schröder, T. Mossakowski, and A. Tarlecki, *Amalgamation via enriched CASL signatures*, Automata, Languages and Programming, Lect. Notes Comp. Sci., vol. 2076, Springer, 2001, pp. 993–1004.

[43] L. Schröder, T. Mossakowski, A. Tarlecki, P. Hoffman, and B. Klin, *Semantics of architectural specifications in* Casl, Fundamental Approaches to Software Engineering, LNCS, vol. 2029, Springer, 2001, pp. 253–268.

[44] M. Wirsing, *Structured algebraic specifications: A kernel language*, Theoret. Comput. Sci. **42** (1986), 123–249.