Semantics of Architectural Specifications in CASL

Lutz Schröder[1], Till Mossakowski[1], Andrzej Tarlecki[2,3], Bartek Klin[4], and Piotr Hoffman[2]

[1] BISS, Department of Computer Science, Bremen University
[2] Institute of Informatics, Warsaw University
[3] Institute of Computer Science, Polish Academy of Sciences
[4] BRICS, Arhus 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 has amalgamation, 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.

Introduction

A common feature of present-day algebraic specification languages (see e.g. [Wir86,EM85,GHG$^+$93,CoFI96,SW99]) is the provision of operations for building large specifications in a structured fashion from smaller and simpler ones [BG77]. For the quite different purpose of describing the modular structure of software systems under development [SST92], architectural specifications have been introduced as a comparatively novel feature in the algebraic specification language CASL recently developed by the CoFI group [CoF,CoF99a,Mos99].

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 [BST99].

The aim of the present paper is to outline the semantics of architectural specifications at various levels. We use a simple subset of CASL architectural specifications, which is expressive enough to study the main mechanisms and features of the semantics. A crucial prerequisite for a semantics of architectural specifications is the amalgamation property, which allows smaller models to be combined into larger ones under statically checkable conditions. Somewhat informally, the amalgamation property ensures that whenever two models 'share' components in the intersection of their signatures, they can be unambiguously put together to form a model of the union of their signatures. Many standard logical systems (like multisorted equational [EM85] and first-order logic [MT93] with the respective standard notions of model) admit amalgamation, so quite often this property is taken for granted in work on specification formalisms (cf. e.g. [ST88]). However, the expected amalgamation property fails to hold for the logical system underlying CASL.

We develop a three-step semantics that circumvents this problem. The first step is a purely model-theoretic semantics. Here, amalgamability is just *required* whenever it is needed. This makes the definition of the semantics as straightforward and permissive as possible, but leaves 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. A suitable amalgamation property is needed to make the static character of these conditions explicit. The trick used in the third step to achieve this is to embed the CASL logic into a richer logic that does have amalgamation. This makes it possible to restate the amalgamability conditions as entirely static factorization properties of signature morphisms.

These three steps of the semantics are in fact independent of the details of the underlying CASL logical system: we present them in the framework of an arbitrary logical system formalized as an institution [GB92]. As a result, the factorization properties to which we reduce the amalgamation conditions are still relatively abstract. A calculus for checking these factorization properties in the case of the specificic logic underlying CASL is developed in a separate paper [KHT$^+$].

We refer to [Mac97,AHS90] for categorical terminology left unexplained here.

1 Architectural Specifications

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*, i.e. 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 [CoF99a] might have the following form:

arch spec BUILDCOMPILER =
units *I* : IDENTIFIER **with sorts** *Identifier*, *Keyword*;
L : ELEM $\rightarrow$ LIST[ELEM];
PT = *L*[*I* **fit sort** *Elem* $\mapsto$ *Identifier*]
and *L*[*I* **fit sort** *Elem* $\mapsto$ *Keyword*];
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.

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

Architectural specifications: $ASP ::= \textbf{arch spec}\ UDD^*\ \textbf{result}\ T;$
$UDD ::= Dcl \mid Dfn$
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: $T ::= U \mid U[T\ \textbf{fit}\ \sigma] \mid T_1\ \textbf{with}\ \sigma_1\ \textbf{and}\ T_2\ \textbf{with}\ \sigma_2$
A unit term is either a (non-parametrized) unit name, or a unit application with an argument that fits via a signature morphism σ, or an amalgamation of units via signature morphisms σ_1 and σ_2. We require that σ_1 and σ_2 form an episink (have a common target signature and are jointly epi); we thus slightly generalize the amalgamation operation of CASL here, again avoiding the need to present the details of signature unions (cf. [Mos00]).

Imports as used in Example 1 can be regarded as syntactical sugar for a parametrized unit which is instantiated only once.

2 Institutions and Amalgamation

The semantic considerations ahead rely on the notion of *institution* [GB92]. An institution 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 [AHS90], and further components which formalize sentences and satisfaction. In this context, we need only the model functor. For a signature Σ, $\mathbf{Mod}(\Sigma)$ is referred to as the category of *models for* Σ, and for a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$, $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma_2) \to \mathbf{Mod}(\Sigma_1)$ is called the *reduct functor*. $\mathbf{Mod}(\sigma)(M)$ is often written as $M|_\sigma$.

A cocone for a diagram in **Sign** is called *amalgamable* if it is mapped to a limit under **Mod**. I has the *(finite) amalgamation property* if (finite) colimit cocones are amalgamable, i.e. if **Mod** preserves (finite) limits.

The underlying logic of CASL is formalized by the institution *SubPCFOL* (for 'subsorted partial first order logic with sort generation constraints'); the associated signature category is denoted by **CASLsign** [CoF99b,Mos]. As mentioned above, *SubPCFOL* does not have the finite amalgamation property (cf. Example 3).

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 [CoF99b]. 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. Moreover, 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 a static context describing the signatures of the units declared or defined within the specification and the signature of its result unit. 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 argument *specification.*

The model semantics for architectural specifications yields a *unit context* $\mathcal{C}$, which is a class of *unit environments* E, i.e. finite maps from unit names to units as introduced above, and a *unit evaluator* UEv, a function that yields a unit when given a unit environment in the unit context. The unconstrained unit context, which consists of all environments, will be written as $\mathcal{C}^{\emptyset}$. The model semantics for unit declarations and definitions enlarges unit contexts as expected. Finally, the model semantics for a unit term yields a unit evaluator, given a unit context.

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 UEv,

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

We assume that the signature category is equipped with a partial *selection of pushouts* $(\sigma_R : \Sigma_1 \to \Sigma_R, \tau_R : \Sigma_2 \to \Sigma_R, \Sigma_R)$ for spans $(\sigma : \Sigma \to \Sigma_1, \tau : \Sigma \to$

Σ_2) of signature morphisms (where (σ, τ) may fail to have a selected pushout even when has a pushout). In CASL, the selected pushouts would be the ones that can be expressed by signature translations and simple syntactic unions. We also assume that the semantics for specifications is given elsewhere, with $\vdash SP \rhd \Sigma$ and $\vdash SP \Rightarrow \mathcal{M}$ implying $\mathcal{M} \subseteq \mathbf{Mod}(\Sigma)$.

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

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

where UEv 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 constructed from 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 [Pau96]). 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.

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 $D : \mathbf{I} \rightarrow \mathbf{Sign}$, 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.

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.

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

$$\frac{\begin{array}{c} C_{st}^{\emptyset} \vdash UDD_1 \rhd (C_{st})_1 \\ \cdots \\ (C_{st})_{n-1} \vdash UDD_n \rhd (C_{st})_n \end{array}}{\vdash UDD_1 \ldots UDD_n \rhd (C_{st})_n} \qquad \frac{\begin{array}{c} \mathcal{C}^{\emptyset} \vdash UDD_1 \Rightarrow \mathcal{C}_1 \\ \cdots \\ \mathcal{C}_{n-1} \vdash UDD_n \Rightarrow \mathcal{C}_n \end{array}}{\vdash UDD_1 \ldots UDD_n \Rightarrow \mathcal{C}_n}$$

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

$$\frac{\vdash SP_1 \rhd \Sigma_1 \qquad \vdash SP_2 \rhd \Sigma_2 \qquad \tau : \Sigma_1 \to \Sigma_2 \qquad U \notin (dom\, P_{st} \cup dom\, B_{st})}{(P_{st}, B_{st}) \vdash U{:}\, SP_1 \xrightarrow{\tau} SP_2 \rhd (P_{st} + \{U \mapsto \tau\}, B_{st})}$$

$$\frac{\begin{array}{c} \vdash SP_1 \Rightarrow \mathcal{M}_1 \qquad \vdash SP_2 \Rightarrow \mathcal{M}_2 \\ \mathcal{F} = \{F : \mathcal{M}_1 \to \mathcal{M}_2 \mid \text{for } M \in \mathcal{M}_1, F(M)|_{\tau} = M\} \end{array}}{\mathcal{C} \vdash U{:}\, SP_1 \xrightarrow{\tau} SP_2 \Rightarrow \mathcal{C} \times \{U \mapsto \mathcal{F}\}}$$

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

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

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

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

$$\frac{\begin{array}{c} C_{st} \vdash T_1 \rhd \Sigma_1 \qquad C_{st} \vdash T_2 \rhd \Sigma_2 \\ \sigma_1 : \Sigma_1 \to \Sigma \text{ and } \sigma_2 : \Sigma_2 \to \Sigma \text{ form an episink} \end{array}}{(P_{st}, B_{st}) \vdash T_1 \textbf{ with } \sigma_1 \textbf{ and } T_2 \textbf{ with } \sigma_2 \rhd \Sigma}$$

$$\frac{\begin{array}{c} \mathcal{C} \vdash T_1 \Rightarrow UEv_1 \qquad \mathcal{C} \vdash T_2 \Rightarrow UEv_2 \\ \left.\begin{array}{r} \text{for each } E \in \mathcal{C}, \text{ there is a unique } M \in \mathbf{Mod}(\Sigma) \text{ such that} \\ M|_{\sigma_i} = UEv_i(E),\ i = 1, 2 \end{array}\right\} (*) \\ UEv = \{E \mapsto M \mid E \in \mathcal{C} \text{ and } M|_{\sigma_i} = UEv_i(E),\ i = 1, 2\} \end{array}}{\mathcal{C} \vdash T_1 \textbf{ with } \sigma_1 \textbf{ and } T_2 \textbf{ with } \sigma_2 \Rightarrow UEv}$$

Fig. 1. Basic semantics

Explicitly, an *extended static context* $\mathcal{C}_{st} = (P_{st}, \mathcal{B}_{st}, D)$ consists of a map P_{st} that assigns 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 $\mathcal{C}^{\emptyset}_{st}$. 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 $D : \mathbf{I} \to \mathbf{Sign}$, $\langle M_i \rangle_{i \in \mathrm{Ob}\,\mathbf{I}}$ is called *consistent* with D 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 $\mathbf{I}$, $M_i = M_j|_{D(m)}$. We denote the class of all model families that are consistent with D by $\mathbf{Mod}(D)$. Then D *ensures amalgamability for* D', where D' extends D, if any family in $\mathbf{Mod}(D)$ can be uniquely extended to a family in $\mathbf{Mod}(D')$.

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 6.

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.

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

Theorem 2. *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 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

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

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

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

$$\frac{\begin{array}{c}\vdash \mathit{SP}_1 \rhd \Sigma_1 \qquad \vdash \mathit{SP}_2 \rhd \Sigma_2 \qquad \tau : \Sigma_1 \to \Sigma_2 \\ U \notin (\mathit{dom}\, P_{st} \cup \mathit{dom}\, \mathcal{B}_{st})\end{array}}{(P_{st}, \mathcal{B}_{st}, D) \vdash U\colon \mathit{SP}_1 \xrightarrow{\tau} \mathit{SP}_2 \mathrel{\rhd\!\!\!>} (P_{st} + \{U \mapsto \tau\}, \mathcal{B}_{st}, D)}$$

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

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

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

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

Fig. 2. Extended static semantics

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 11 below.

The motivation for this choice is the fact that many typical programming languages we aim at (notably, Standard ML [Pau96]) 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.

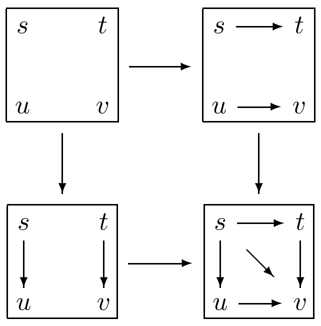
However, we will see in Section 5 (Th. 6) that — generativity issues aside — we have as much completeness as one may hope for, i.e. that the extended static semantics detects all the amalgamation that can be established statically.

5 Enriched Signatures

The actual verification of the amalgamability conditions in the extended static semantics is precisely the point where the fact that amalgamation fails in the CASL institution begins to cause difficulties:

Example 3. Assume that the specification LIST[ELEM] that appears in Example 1 provides a type *List*[*Elem*] of lists of type *Elem*. Recall that the specification of identifiers introduces two sorts *Identifier* and *Keyword*, and that the parametrized unit L ('list') is applied to these two sorts in the 'program text' unit PT.

Now suppose that the specifier of PARSER decides that key words 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, i.e. *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 fails here.)

This observation suggests that one should enlarge the signature category in such a way that the above square is no longer a pushout, and that, moreover, the enlarged category would have to take the fact into account that there may, in general, be more than one embedding between two sorts. Thus, the 'correct' pushout signature would have the form

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

Motivated by this example, we introduce a category **enrCASLsign** of *enriched signatures* in which **CASLsign** can be represented via a functor

$$\Phi : \mathbf{CASLsign} \to \mathbf{enrCASLsign}.$$

Moreover, we equip this category with a model functor

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

which has the amalgamation property (i.e. preserves limits) and which 'extends' the model functor **Mod** of the CASL institution *SubPCFOL*, i.e. $\mathbf{Mod}_e \circ \Phi^{op}$ and **Mod** are naturally isomorphic. One can build an institution around **enrCASLsign** and define an institution representation of *SubPCFOL* therein in the sense of [Tar96]. At any rate, we shall use terms like 'amalgamable' w.r.t. $\mathbf{Mod}_e$. The details of the definitions and full proofs will be presented separately [SMT$^+$]; the basic concepts are outlined below.

As suggested by the example, the step from **CASLsign** to **enrCASLsign** chiefly consists in replacing the preorder on the sorts by a sort category (category sorted algebras, although without a view on amalgamation, go back to [Rey80]). The fact that all embeddings are actually interpreted as injective maps is reflected by the requirement that all morphisms in the sort category are monomorphisms. There is an elegant way to handle overloading of function and predicate symbols in this setting using left and right actions of the sort category on the symbols; see [SMT$^+$] for details.

Now sets and partial maps form a (rather large) enriched signature $\mathbf{Set_p}$: take injective maps as the sort category, relations as predicate symbols, and total (partial) functions as total (partial) function symbols.

Signature morphisms are defined in the obvious way, i.e. as consisting of a functor between the sort categories and maps between the respective symbol classes which are compatible with symbol profiles and 'overloading' (i.e. with the mentioned actions of the sort category). These data define the signature category **enrCASLsign**, where objects have to be restricted to *small* signatures (i.e.

signatures where the sort category and the symbol classes are small) in order to actually obtain a category. It is easily seen that **enrCASLsign** is cocomplete, and that, just as with small and large categories, colimits in **enrCASLsign** remain colimits in the world of 'large enriched signatures' such as $\mathbf{Set_p}$.

Models of an enriched signature Σ can now be defined as signature morphisms

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

Model morphisms are defined by the standard homomorphy conditions. Signature morphisms induce reduct functors in the opposite direction via composition; this defines the model functor $\mathbf{Mod}_e$.

The functor $\Phi : \mathbf{CASLsign} \to \mathbf{enrCASLsign}$ acts on CASL signatures by interpreting the sort preorder as a thin category (and by completing the symbol sets as required by the mentioned actions; cf. [SMT+]).

Now it is easily verified that one indeed has a natural isomorphism $\mathbf{Mod}_e \circ \Phi^{op} \to \mathbf{Mod}$. Thus:

Proposition 4. *A cocone in* **CASLsign** *is amalgamable in SubPCFOL iff its image under* Φ *is amalgamable w.r.t.* $\mathbf{Mod}_e$.

Thanks to the way models are defined, amalgamation for enriched signatures comes almost for free: models are given by a representable functor (namely, $hom(\_, \mathbf{Set_p})$), which automatically preserves limits; little more consideration has to be given to model morphisms. Explicitly:

Proposition 5. $\mathbf{Mod}_e$ *has the amalgamation property.*

Moreover, the 'converse' of amalgamation holds as well:

Theorem 6. $\mathbf{Mod}_e$ *reflects isomorphisms.*

Since $\mathbf{Mod}_e$ preserves limits and $\mathbf{enrCASLsign}^{op}$ is complete, it follows that $\mathbf{Mod}_e$ reflects limits. Explicitly, a cocone in **enrCASLsign** is amalgamable iff it is a colimit. This is essentially what is meant by the 'optimal degree of completeness' statement in Section 4: a cocone in **CASLsign** is amalgamable iff it is mapped to a colimit under Φ.

6 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 $\mathbf{Mod}_e : \mathbf{EnrSign}^{op} \to \mathbf{CAT}$ which has the amalgamation property and reflects isomorphisms (limits) and a functor $\Phi : \mathbf{Sign} \to \mathbf{EnrSign}$ such that $\mathbf{Mod}_e \circ \Phi^{op}$ and $\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbf{CAT}$ are naturally isomorphic. For the CASL institution *SubPCFOL*, these data have been constructed above.

Recall that the said amalgamation conditions required diagrams to ensure amalgamability for certain extensions, which was defined as unique extendability of consistent families of models. 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 consistent family of models for a diagram D in **EnrSign** is essentially the same as a model of the colimit signature $\operatorname{colim} D$. Thus we have

Proposition 7. *Let D' be a diagram in* **EnrSign** *that extends D. D ensures amalgamation 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 8. 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 9. *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'.*

In the two cases where amalgamation is required in the rules of the extended static semantics, the covering condition is satisfied. Thus we have essentially reduced the amalgamation problem to proving the existence of the factorizations required in the above proposition. In both cases, the factorization condition concerns a sink (τ_1, τ_2) in **Sign** (in the case of amalgamation, the two injections into the union, and in the case of application, the pushout cocone):

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

(D denotes the original diagram, and the μ_i denote the colimit injections).

Example 10. The simple union of sort preorders presented in Example 3, Diagram (1), fails to admit a factorization as above, since the colimit will have two different sort embeddings $s \to v$ as depicted in Diagram (2).

In order to provide a construction for the factorization θ in the concrete case of **enrCASLsign**, we have to require additionally that $(\Phi(\tau_1), \Phi(\tau_2))$ is an extremal episink, i.e. that the images of $\Phi(\tau_1)$ and $\Phi(\tau_2)$ jointly generate $\Phi(\Sigma)$. This is the case for pushouts and unions in **CASLsign** (although unions need not be extremal in **CASLsign**!).

Under this additional condition, it is clear how θ has to be defined if it exists, namely by extending the effect of the μ_i to terms formed from the generators. The task that remains is to check if this results in a well-defined signature morphism. This requires a calculus for proving equality of morphisms and symbols in the colimit; such a calculus is discussed in [KHT$^+$].

Remark 11. In the construction of a non-generative semantics for unit application (cf. Section 4), the amalgamation property provides an easy criterion for the equivalence of two instantiations: let U be a parametrized unit over the signature $\tau : \Sigma_1 \to \Sigma_2$. 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) to the same model of the parameter signature Σ_1. This will be the case for all pairs of models that appear in consistent families for D 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.

7 Conclusions and Future Work

We have presented and discussed the semantics of a small and modified but quite representative subset of Casl architectural specifications 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. In institutions with amalgamation, these conditions can be replaced by literally static ones; this may require representing the given institution in one that has the amalgamation property.

We have demonstrated how such a representation can be defined for the standard Casl institution; the main point here was that one has to admit categories of subsort embeddings instead of just preorders in the signatures. Computational aspects of the amalgamability conditions for this specific institution are discussed separately [KHT$^+$]. It is known that these conditions are in general undecidable. However, decision procedures can be developed for important special cases; for instance, orders (rather than preorders) of sorts admit deciding the amalgamability conditions by a polynomial algorithm [Kli00]. For the general case, it seems that an approximative algorithm that restricts the length of 'cells' involved in the proofs will suffice in practically relevant cases.

The technique for the extended static analysis with colimits of enriched signatures providing a way to statically reformulate amalgamability conditions is general enough to be readily available for other design specification frameworks where amalgamation causes problems. In fact, we conjecture that any institution can be canonically extended to an institution with the amalgamation property by adding formal colimits of signatures.

The work presented here is 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 [Hof00], 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.

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 and Maura Cerioli for pointing out the lack of amalgamation for CASL models.

References

[CoFI96] CoFI. Catalogue of existing frameworks. Catalogue.html, in [CoF], 1996.

[AHS90] J. Adámek, H. Herrlich, and G. E. Strecker. *Abstract and Concrete Categories*. Wiley Interscience, New York, 2nd edition, 1990.

[BG77] R. M. Burstall and J. A. Goguen. Putting theories together to make specifications. In *Proc. 5th Intl. Joint Conf. on Artificial Intelligence*, pages 1045–1058. Carnegie-Mellon University, 1977.

[BST99] M. Bidoit, D. Sannella, and A. Tarlecki. Architectural specifications in CASL. In *7th Intl. Conference on Algebraic Methodology and Software Technology (AMAST'98)*, volume 1548 of *LNCS*, pages 341–357. Springer, 1999.

[CoF] CoFI. The Common Framework Initiative for algebraic specification and development, electronic archives. Notes and Documents accessible by WWW[1] and FTP[2].

[CoF99a] CoFI Language Design Task Group. CASL – The CoFI Algebraic Specification Language – Summary, version 1.0. Documents/CASL/Summary, in [CoF], July 1999.

[CoF99b] CoFI Semantics Task Group. CASL – The CoFI Algebraic Specification Language – Semantics. Note S-9 (version 0.96), in [CoF], July 1999.

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

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

[GHG$^+$93] 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.

[1] http://www.brics.dk/Projects/CoFI
[2] ftp://ftp.brics.dk/Projects/CoFI

[Hof00] P. Hoffman. Semantics of architectural specifications (in Polish). Master's thesis, Warsaw University, 2000.

[KHT+] B. Klin, P. Hoffman, A. Tarlecki, T. Mossakowski, and L. Schröder. Checking amalgamability conditions for CASL architectural specifications. Work in progress; see also [Kli00].

[Kli00] B. Klin. An implementation of static semantics for architectural specifications in CASL (in Polish). Master's thesis, Warsaw University, 2000.

[Mac97] S. Mac Lane. *Categories for the Working Mathematician*. Springer, Berlin, 2nd edition, 1997.

[Mos] T. Mossakowski. Relating CASL with other specification languages: the institution level. *Theoretical Computer Science*. To appear.

[Mos99] P. D. Mosses. CASL: A guided tour of its design. In *13th Workshop on Abstract Datatypes WADT 98*, volume 1589 of *LNCS*, pages 216–240. Springer, 1999.

[Mos00] T. Mossakowski. Specification in an arbitrary institution with symbols. In *14th Workshop on Abstract Datatypes WADT 99*, volume 1827 of *LNCS*, pages 252–270. Springer, 2000.

[MT93] K. Meinke and J. V. Tucker, editors. *Many-sorted Logic and its Applications*. Wiley, 1993.

[Pau96] L. C. Paulson. *ML for the Working Programmer*. Cambridge University Press, New York, 2nd edition, 1996.

[Rey80] J. C. Reynolds. Using category theory to design implicit conversions and generic operators. In *Semantics-Directed Compiler Generation*, volume 94 of *LNCS*, pages 211–258. Springer, 1980.

[SMT+] L. Schröder, T. Mossakowski, A. Tarlecki, P. Hoffman, and B. Klin. Amalgamation via enriched signatures. Work in progress.

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

[ST88] D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76:165–210, 1988.

[SW99] D. Sannella and M. Wirsing. Specification languages. In A. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors, *Algebraic Foundations of Systems Specifications*, pages 243–272. Springer, 1999.

[Tar96] A. Tarlecki. Moving between logical systems. In *11th Workshop on Specification of Abstract Data Types*, volume 1130 of *LNCS*, pages 478–502. Springer, 1996.

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