Checking Amalgamability Conditions for CASL Architectural Specifications

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

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

Abstract. CASL, a specification formalism developed recently by the CoFI group, offers architectural specifications as a way to describe how simpler modules can be used to construct more complex ones. The semantics for CASL architectural specifications formulates static *amalgamation conditions* as a prerequisite for such constructions to be well-formed. These are non-trivial in the presence of subsorts due to the failure of the amalgamation property for the CASL institution. We show that indeed the static amalgamation conditions for CASL are undecidable in general. However, we identify a number of practically relevant special cases where the problem becomes decidable and analyze its complexity there. In cases where the result turns out to be **PSPACE**-hard, we discuss further restrictions under which polynomial algorithms become available. All this underlies the static analysis as implemented in the CASL tool set.

Keywords: formal specification and program development; CASL; architectural specifications; amalgamation; algorithms; decidability.

Introduction

Architectural specifications are a mechanism to support formal program development [2, 11]. They have been introduced in the CASL specification language, which provides them in addition to structured specifications [4, 3]. While the latter facilitate building global requirement specifications in a modularized fashion, the former aim at formalizing the development process: an architectural specification describes how an implementation of the global specification may be built from implementations of other (simpler) specifications. One of the operations which may be used in this process is *amalgamation*, i.e., one may build a bigger unit simply by combining (amalgamating) smaller ones. (Modules in CASL architectural specifications are called *units*; parametricity aside, they correspond semantically to models in the underlying CASL institution.)

Whenever such an amalgamation occurs in an architectural specification, the semantics requires the amalgamated unit to exist and be unique. Whether or not this holds in concrete cases is undecidable, since the CASL underlying logic is undecidable. However, in [11] we developed a somewhat stricter but entirely static *amalgamation condition* that ensures amalgamability of units. This condition is certainly decidable for CASL without subsorts. The problem of its (un)decidability for arbitrary CASL specifications with subsorts is investigated here, and the tractability of important special cases is studied.

After introducing a simplified version of the CASL institution in Section 1, we recall the institution independent amalgamation conditions as defined in [11] in Section 2 and, in Section 3, reformulate them in the simplified CASL institution. This is followed by a sound and complete *cell calculus* for discharging them. In Section 4, we show that the cell calculus and, hence, the amalgamation conditions in CASL are undecidable. This motivates the search for practically relevant special cases for which an algorithm to check amalgamation conditions can be given. First, we indicate that when the number of cells to consider is finite, the problem becomes decidable but **PSPACE**-hard. Moreover, we discuss a simple situation where the amalgamation conditions can be checked in polynomial time. In Section 5, we investigate another special case which arises naturally in the categorical framework we use, but can also be interpreted as a methodological directive on the use of subsorting in architectural specifications. Essentially, it corresponds to the requirement that we can amalgamate all units in the architectural specification, even with hidden components. No surprise here: the problem remains undecidable. However, if we once again assume that the number of cells is finite, then this problem admits a polynomial time algorithm. This algorithm is currently used by the CASL toolset, CATS[1]. In Section 6 we discuss the possibility of restricting the cell calculus to a finite domain and observe that checking the resulting amalgamation condition is decidable but **PSPACE**-hard.

1 The CASL Institution

The following considerations rely on the notion of institution [6]. An *institution* I consists of a category **Sign** of *signatures*, a *model functor* $\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbf{CAT}$, and further components which formalize sentences and satisfaction. In this context, we use only the category of signatures and the model functor. The functor **Mod** maps a signature to a category of *models* over this signature, and (contravariantly) a signature morphism to a *reduct functor* between model categories.

We say that an institution I has the *amalgamation property* if **Mod** preserves limits, i.e., maps colimits of signature diagrams to limits in **CAT**.

The underlying logic of CASL is formalized by the institution $\mathit{SubPCFOL}^{=}$ ('subsorted partial first order logic with equality and sort generation constraints', see [3]). Signatures in this institution are the usual many- and subsorted algebraic signatures with partial and total operations and predicates. In the following, for clarity of the presentation, we ignore operations and predicates and view CASL signatures simply as preorders of sorts. Although the presence of operations and predicates adds some technical complication to the issues considered, it does not change the presented results in an essential way.

[1] Accessible by WWW, at http://www.tzi.de/cofi/CASL/CATS

Formally, our simplified CASL institution consists of the category **PreOrd** of all finite preorders as the signature category. A model over a signature $\Sigma = (S, \leq)$ is an S-sorted set $\{X_s\}_{s \in S}$ together with an injective function $e_{(s,t)} : X_s \to X_t$ for all $s \leq t$ in Σ, satisfying the commutativity condition $e_{(t,u)} e_{(s,t)} = e_{(s,u)}$ for all $s \leq t \leq u$ in Σ, and such that $e_{(s,s)}$ is the identity for $s \in S$. With the usual homomorphisms, models over a signature Σ form a category $\mathbf{Mod}(\Sigma)$. Together with a mapping of signature morphisms (i.e., order-preserving functions) to the obvious reduct functors, this defines the model functor $\mathbf{Mod} : \mathbf{PreOrd}^{op} \to \mathbf{CAT}$. Unfortunately, **Mod** does not preserve limits, i.e., the CASL institution does not have the amalgamation property (see the example given below).

We will often speak of CASL signatures as (preorder, or thin) categories. Their objects, in turn, will be called *sorts*, and their morphisms *sort embeddings*. *Signature morphisms*, or shortly *morphisms* (in **PreOrd**) are functors between signatures as thin categories.

2 Amalgamation Conditions

In [11], a semantics for architectural specifications as defined in CASL [4] has been proposed. The semantics is institution independent, i.e., it is parametrized by the institution of the underlying specification formalism. We now briefly describe a problem that occurs in checking the correctness of an architectural specification with respect to the proposed semantics, and then go on to analyse this problem in the case of our simplified CASL institution.

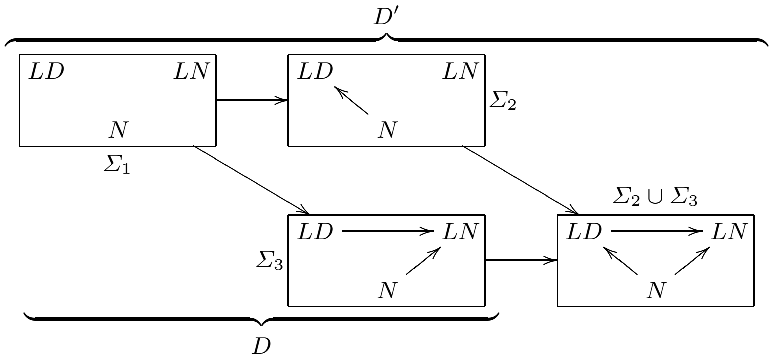
For an architectural specification to be correct, it has to satisfy certain *amalgamation conditions*. Informally, these conditions ensure that whenever two units are combined (amalgamated) into one unit, they must 'share' components from the intersection of their signatures, to allow the amalgamation to be defined unambiguously. Moreover, in the presence of subsorts, natural compatibility requirements on subsort embeddings must be ensured. One would expect the amalgamation conditions to be checked during static analysis of a specification, as in the analogous case of 'sharing conditions' in module systems of programming languages like Standard ML [8]. However, the most natural formulation of the amalgamation conditions for architectural specifications does not suggest any method of checking them statically, as it involves classes of models, possibly defined by sets of axioms of an undecidable underlying logic.

Because of this, an *extended static semantics* of architectural specifications, with the amalgamation conditions reformulated statically, has been proposed in [11] . In this approach, each part of an architectural specification is evaluated to a diagram $D : \mathbf{I} \to \mathbf{Sign}$, where $\mathbf{I}$ is a finite category, and **Sign** is the signature category of the underlying institution. A family of models $\langle M_i \rangle_{i \in \mathbf{Ob}(\mathbf{I})}$ is *compatible* with D if for all $i \in \mathbf{Ob}(\mathbf{I})$, $M_i \in \mathbf{Ob}(\mathbf{Mod}(D(i)))$, and for all $m : i \to j$ in $\mathbf{I}$, $M_i = \mathbf{Mod}(D(m))(M_j)$. For two diagrams $D : \mathbf{I} \to \mathbf{Sign}$, $D' : \mathbf{I}' \to \mathbf{Sign}$, where D' extends D, we say that D *ensures amalgamability for* D', if any model family compatible with D can be uniquely extended to a family compatible with D'.

Fig. 1. An architectural specification in Casl and its corresponding signature diagram

spec Num1 = **sorts** *Number*, *List*[*Digit*], *List*[*Number*]
ops ... **axioms** ...
spec Num2 = **sorts** *Number* <*List*[*Digit*], *List*[*Number*]
spec Num3 = **sorts** *Number* <*List*[*Number*], *List*[*Digit*]<*List*[*Number*]

arch spec A =
units U : Num1;
F : Num1 $\rightarrow$ Num2;
G : Num1 $\rightarrow$ Num3;
result $F[U]$ **and** $G[U]$
end



Example. Consider the architectural specification A presented in Fig. 1, with the simplified Casl institution as the underlying institution. To implement this specification, one has to implement a unit U — a model consisting of three sorts (we omit the expected operations and axioms here). Moreover, one has to provide two parametrized units F and G, extending any such unit with some injective functions between sorts, according to specifications Num2 and Num3. The resulting unit of the specification A is the amalgamation of both extensions applied to U.

The extended static semantics maps the specification A to the signature diagram D' shown in Fig. 1, while the diagram D corresponds to the specification A 'before' the amalgamation $F[U]$ **and** $G[U]$ ($\Sigma_1, \Sigma_2, \Sigma_3$ are the signatures of the Casl basic specifications Num1, Num2, Num3, respectively).

It is intuitively clear that amalgamation in this example is not always possible. If the units F and G of Fig. 1 are implemented independently, the injective functions they introduce may not commute (in fact, they do not commute under the informally expected realization of lists in U and of the embeddings in F and G). This is formalized in the extended static semantics by the fact that D does not ensure amalgamability for D'. This is connected to the lack of amalgamation property in the CASL institution: indeed, the model functor **Mod** does not map the pushout of D (i.e., the signature $\Sigma_2 \cup \Sigma_3$) to a pullback in **CAT**.

Now, the amalgamation conditions can be reformulated as conditions requiring that some diagrams $D : \mathbf{I} \to \mathbf{Sign}$ (corresponding to certain parts of the evaluated specification) ensure amalgamability for diagrams $D' : \mathbf{I}' \to \mathbf{Sign}$ such that $\mathbf{I}'$ results from $\mathbf{I}$ by adding a new object l and a morphism $m : k \to l$ for some $k \in \mathbf{Ob}(\mathbf{I})$, and D' extends D and maps m to an extremal epimorphism in **Sign** (the general situation can be easily reduced to this case, see [11] and the example below). In the following, the image of $m : k \to l$ under D will be denoted by $\tau : D(k) \to \Delta$.

Example (ctd.). In our example, the diagram D is extended by $\Delta = \Sigma_2 \cup \Sigma_3$ and a morphism τ from the (silently added to D) disjoint union of Σ_2 and Σ_3 to Δ. The morphism τ is an extremal epi in **PreOrd**, i.e., it is surjective on sorts and generates the preorder on its codomain.

The reformulated amalgamation conditions are 'static', because the model classes involved are not restricted by any axioms of the underlying logic. Unfortunately, in the CASL institution, even such static conditions cannot be checked directly due to the failure of the amalgamation property. As suggested in [11], this problem can be circumvented by a translation of the CASL institution to an institution with the amalgamation property:

Consider two institutions $I = (\mathbf{Sign}, \mathbf{Mod}), I' = (\mathbf{EnrSign}, \mathbf{Mod}_e)$ (the sentence and satisfaction components are omitted), and a functor $\Phi : \mathbf{Sign} \to \mathbf{EnrSign}$ such that $\mathbf{Mod}_e \circ \Phi$ is naturally isomorphic to **Mod**. Then one can show that, given two diagrams $D : \mathbf{I} \to \mathbf{Sign}$ and $D' : \mathbf{I}' \to \mathbf{Sign}$, D ensures amalgamability for D' if and only if ΦD ensures amalgamability for $\Phi D'$. Moreover, if I' satisfies the amalgamation property and certain additional conditions (see [11] for details), then ΦD ensures amalgamability for $\Phi D'$ if and only if a colimit cocone for ΦD extends to a cocone for $\Phi D'$. The latter condition is entirely 'static', in the sense that it does not directly involve models at all.

The above construction is institution independent. To reformulate the amalgamation conditions in the specific case of CASL, we need an appropriate institution $(\mathbf{EnrSign}, \mathbf{Mod}_e)$ together with a representation functor Φ. In [11], we proposed the category of left cancellable small categories **lcCat** (i.e., small categories with all morphisms being monic) as the category **EnrSign**, and the usual interpretation of preorders as thin categories as the functor Φ (when operations and predicates are present, the details are a bit more complicated, but the main idea is the same, cf. [12]). With the natural model functor $\mathbf{Mod}_e$, this construction ensures the amalgamation property and all additional requirements. Thus the amalgamation conditions for CASL architectural specifications have been reformulated as conditions involving extendability of colimit cocones for 'enriched' signature diagrams that correspond, via the extended static semantics, to architectural specifications.

In the following we present a detailed analysis of the problem of checking the latter conditions.

3 Cell Calculus

As announced in the previous section, we work with the enriched signature category **EnrSign** being the category **lcCat** of left cancellable small categories, the representation functor $\Phi : \mathbf{PreOrd} \to \mathbf{lcCat}$ interpreting preorders as thin categories, and the natural model functor $\mathbf{Mod}_e : \mathbf{lcCat}^{op} \to \mathbf{CAT}$. Note that Φ preserves extremal epimorphisms. A morphism in **lcCat** is an extremal epimorphism iff it is surjective on sorts and every embedding in its codomain is a composite of images of embeddings in its domain. To simplify the notation, we will in the following regard **PreOrd** as an actual subcategory of **lcCat**, i.e. identify preorders and thin categories (which are left cancellable), so that Φ becomes an inclusion and we can omit its application whenever convenient.

In this setting, the amalgamation conditions have the following form: given a diagram $D : \mathbf{I} \to \mathbf{lcCat}$ of thin categories, a thin category Δ, and an extremal epimorphism $\tau : D(k) \to \Delta$ for some $k \in \mathbf{Ob}(\mathbf{I})$, let $D' : \mathbf{I}' \to \mathbf{lcCat}$ be the result of adding l as a new object and $m : k \to l$ to $\mathbf{I}$, and extending D by mapping k to Δ and m to τ. The amalgamation condition now is that the colimit cocone for D extends to a cocone for D'. We can, of course, assume that $\mathbf{I}$ is finite (as it represents the structure of an architectural specification), as well as Δ and the $D(i)$ (as they represent CASL signatures).

Example (ctd.). The static incorrectness of specification A in Fig. 1 corresponds to the fact that the colimit cocone in **lcCat** for D does not extend to a cocone for D'. Indeed, fewer equations between paths of sort embeddings hold in the colimit object preseted below, than in the object $\Delta = \Sigma_2 \cup \Sigma_3$ shown in Fig. 1.

$$\boxed{\begin{tikzcd} LD \arrow[rr] & & LN \\ & N \arrow[ul] \arrow[ur, shift left] \arrow[ur, shift right] & \end{tikzcd}}$$

Informally, such an amalgamation condition states that the signature morphism τ does not 'glue' more sorts and embeddings than the colimit cocone for D. Verifying this condition requires some way of constructing colimits in **lcCat**: we introduce a *cell calculus* for this purpose. Our approach is related to the use of generalized congruences as worked out in [1] (see also [9, 10]). To construct a colimit of a signature diagram D, we collect all sorts and embedding paths occuring in D, and divide the two resulting sets by two suitable equivalence relations $\simeq$ and $\cong$, ensuring that the resulting object forms a left cancellable category and a colimit for D.

Let $\mathit{Sorts}(D) = \biguplus_{i \in \mathbf{Ob}(\mathbf{I})} \mathbf{Ob}(D(i))$ and $\mathit{Embs}(D) = \biguplus_{i \in \mathbf{Ob}(\mathbf{I})} \mathbf{Mor}(D(i))$. Thus, elements of $\mathit{Sorts}(D)$ are of the form (i, s), where $i \in \mathbf{Ob}(\mathbf{I})$ and s is a sort in $D(i)$, and elements of $\mathit{Embs}(D)$ are of the form (i, e), where $i \in \mathbf{Ob}(\mathbf{I})$ and e is a sort embedding in $D(i)$.

Now let $\simeq$ be the least equivalence on $\mathit{Sorts}(D)$ such that for all $m : i \to j$ in $\mathbf{I}$ and $s \in \mathbf{Ob}(D(i))$, $(i, s) \simeq (j, D(m)(s))$.

Consider the set of non-empty finite words over the alphabet $\mathit{Embs}(D)$. For a word $\omega = \langle (i_n, e_n), \ldots, (i_1, e_1) \rangle$, we define $\mathrm{dom}(\omega) = (i_1, \mathrm{dom}(e_1))$ and $\mathrm{cod}(\omega) = (i_n, \mathrm{cod}(e_n))$. A word $\langle (i_n, e_n), \ldots, (i_1, e_1) \rangle$ is $\simeq$-*admissible* if, for $1 \leq k < n$, $(i_k, \mathrm{cod}(e_k)) \simeq (i_{k+1}, \mathrm{dom}(e_{k+1}))$. Let $\mathit{Adm}_{\simeq}$ be the set of all $\simeq$-admissible words over $\mathit{Embs}(D)$. A pair (ω, υ) of $\simeq$-admissible words such that $\mathrm{dom}(\omega) \simeq \mathrm{dom}(\upsilon)$ and $\mathrm{cod}(\omega) \simeq \mathrm{cod}(\upsilon)$ is called a *cell over* D. For a cell $c = (\omega, \upsilon)$, we define $\mathrm{dom}(c) = \mathrm{dom}(\omega)$ and $\mathrm{cod}(c) = \mathrm{cod}(\omega)$.

Let $\cong$ be the least (equivalence) relation between $\simeq$-admissible words that satisfies the following rules:

$$\frac{m : i \to j \text{ is a morphism in } \mathbf{I}}{\langle (i,e) \rangle \cong \langle (j, D(m)(e)) \rangle}\ \textbf{(Diag)} \qquad \frac{ed \text{ is defined in } D(i)}{\langle (i,e), (i,d) \rangle \cong \langle (i, ed) \rangle}\ \textbf{(Comp)}$$

$$\frac{\omega \text{ is } \simeq\text{-admissible}}{\omega \cong \omega}\ \textbf{(Refl)} \qquad \frac{\omega \cong \upsilon}{\upsilon \cong \omega}\ \textbf{(Symm)} \qquad \frac{\omega \cong \upsilon \qquad \upsilon \cong \psi}{\omega \cong \psi}\ \textbf{(Trans)}$$

$$\frac{\omega \cong \upsilon \qquad \psi \cong \phi \qquad \mathrm{cod}(\omega) \simeq \mathrm{dom}(\psi)}{\psi\omega \cong \phi\upsilon}\ \textbf{(Cong)} \qquad \frac{\psi\omega \cong \psi\upsilon}{\omega \cong \upsilon}\ \textbf{(Lc)}$$

Given these rules, *derivations* are defined in the usual way as minimal sequences of pairs of words such that every pair in the sequence follows from the previous ones by one of the rules.

It follows by induction on the length of the derivation that if $\omega \cong \upsilon$, then (ω, υ) forms a cell over D. In particular, if the conditions for the rule **(Cong)** hold, then both $\psi\omega$ and $\phi\upsilon$ are $\simeq$-admissible. Thus, we will think of $\cong$ as a set of cells over D defined inductively by the above rules, which we therefore call the *cell calculus*; a derivation is essentially a proof that a certain cell is in $\cong$.

Now, the category $\mathbf{C}$ with $\mathit{Sorts}(D)/_{\simeq}$ and $\mathit{Adm}_{\simeq}/_{\cong}$ as the sets of objects and morphisms respectively, and with the domain, codomain, identity and composition functions defined in the obvious way, is a colimit for D in $\mathbf{lcCat}$.

Equipped with this construction, we come back to the amalgamation condition formulated as above. Consider a relation $\simeq_\tau$ on $\mathbf{Ob}(D(k))$, defined as the kernel of τ on sorts: $s \simeq_\tau t$ iff $\tau(s) = \tau(t)$. Then consider the set of all non-empty finite words over the alphabet $\mathbf{Mor}(D(k))$ and define the set of $\simeq_\tau$-admissible words in the same manner as the set of $\simeq$-admissible words. Next, define the relation $\cong_\tau$ on the set of $\simeq_\tau$-admissible words: $\omega \cong_\tau \upsilon$ if $\mathrm{dom}(\omega) \simeq_\tau \mathrm{dom}(\upsilon)$ and $\mathrm{cod}(\omega) \simeq_\tau \mathrm{cod}(\upsilon)$. Finally, consider the obvious injections $\iota_o : \mathbf{Ob}(D(k)) \to \mathit{Sorts}(D)$ and $\iota_e : \mathbf{Mor}(D(k)) \to \mathit{Embs}(D)$. We will abuse the names ι_o, ι_e to denote also the obvious extensions to words over $\mathbf{Mor}(D(k))$ and to relations on these words.

Recall that, informally, amalgamation condition requires that the morphism τ does not 'glue' more objects and embedding paths than 'glued' in the colimit of D. This is captured by the following theorem.

Theorem 1. *In the above notation, a colimit cocone for D extends to a cocone for D' iff*
(*) $\iota_o(\simeq_\tau) \subseteq \simeq$ *and*
(**) $\iota_e(\cong_\tau) \subseteq \cong$.

The condition (**) is called the *cell condition.* If the inclusion (*) holds, then ι_e maps $\simeq_\tau$-admissible words to $\simeq$-admissible words, so (**) concerns an inclusion between two sets of cells.

Thinness of the colimit of D is a stronger property than the cell condition:

Lemma 1. *In the above notation, if (*) holds and the colimit* $\boldsymbol{C}$ *of* D *is a thin category, then (**) holds as well.*

Conversely, if, for all $i \in \boldsymbol{Ob(I)}$*, there is a unique morphism* $D(i) \to \Delta$ *in the image of* D'*, and if both (*) and (**) hold, then* $\boldsymbol{C}$ *is isomorphic to* Δ *(hence thin).*

Theorem 1 reformulates the amalgamation condition using four relations: $\simeq$, $\simeq_\tau$, $\cong$, and $\cong_\tau$. The former two relations always have finite domains and hence can be represented explicitly, which makes (*) decidable.

However, the relations $\cong$ and $\cong_\tau$ have possibly infinite domains. One can always choose a finite subrelation $\cong'_\tau$ of $\cong_\tau$ such that if $\iota_m(\cong'_\tau) \subseteq \cong$ then $\iota_m(\cong_\tau) \subseteq \cong$ as well (roughly, $\cong_\tau$ can be restricted to 'loopless' words, see Sect. 6 below for a formal definition). This does not solve all problems though, since there is no obvious finite representation of $\cong$, more explicit than the cell calculus.

4 Checking the Cell Condition

We now discuss the issue of checking the cell condition, relying on the notation introduced in its formulation above and assuming that (*) holds.

In the following we will sometimes treat $\cong_\tau$ and $\cong$ as sets of cells over D (this is legal for $\cong_\tau$ because (*) holds). Moreover, we assume that $\cong_\tau$ can be represented explicitly, so the lack of a more explicit representation for $\cong$ (which is only defined inductively by the cell calculus) is the main difficulty.

4.1 Simple Case

First we discuss a simple condition that implies the cell condition and holds in many practical examples of architectural specifications.

Consider D, Δ and τ given as above, with (*) satisfied. Let $\cong_0$ be the least equivalence relation on the set of $\simeq$-admissible words that satisfies the rules **(Diag)** and **(Comp)** of the cell calculus. Obviously $\cong_0 \subseteq \cong$, so if $\iota_e(\cong_\tau) \subseteq \cong_0$, then the cell condition (**) holds.

Even though $\cong_0$ has an infinite domain, it has only finitely many non-reflexive elements, since $\mathbf{I}$ and the $D(i)$ are finite and the $D(i)$ are thin. Hence it can be explicitly represented during static analysis. Together with a method of explicit representation of $\cong_\tau$, this gives an algorithm (in fact, a polynomial time algorithm) for checking the condition $\iota_e(\cong_\tau) \subseteq \cong_0$.

Intuitively speaking, this captures the situation where each embedding cell present in the signature Δ can be traced to a cell in one of the signatures in D. This condition holds in many practical cases, including all cases where the architectural specification does not contain any subsort embeddings (then the problem is even simpler: if all signatures in D are discrete categories, then the cell condition (**) follows from (*)).

4.2 General Case

Unfortunately, the cell condition turns out to be undecidable in general:

Theorem 2. *In the above notation, and under the assumption that (*) holds, (**) is undecidable.*

The proof is by reduction from the problem of deciding whether the monoid with left cancellation presented by a given finite set of equations is trivial. For any finite set R of equations over a finite alphabet, we construct a diagram D, a category Δ and a morphism τ such that (*) holds, and such that (**) holds if and only if the monoid with left cancellation presented by R is trivial. There is a direct correspondence between the rules of the cell calculus and the axioms of monoids with left cancellation.

4.3 Special Case

Since the cell conditions are undecidable in general, we need to detect special cases of practical relevance in which the conditions are decidable. In fact, one such case is presented in Subsection 4.1.

Consider a situation where $Adm_{\simeq}$, the domain of $\cong$, is finite (leaving identity embeddings aside). In other words, there is no $\simeq$-admissible word ω such that $\mathrm{dom}(\omega) \simeq \mathrm{cod}(\omega)$, except when all letters of ω are identities. It follows that in the colimit $\mathbf{C}$ of D identities are the only embeddings with domain equal to codomain. This condition can be effectively checked, since the relation $\simeq$ can be represented explicitly, and it is enough to check the condition $\mathrm{dom}(\omega) \not\simeq \mathrm{cod}(\omega)$ for words of length smaller than the number of equivalence classes of $\simeq$.

Requiring finiteness of $Adm_{\simeq}$ rules out architectural specifications declaring mutual subsorting relations, for example, when there is an embedding $Num < List[Digit]$ (encoding a number as a list of digits) in one unit, and an embedding $List[Digit] < Num$ (computing a number from a list of digits) in another unit. It also disallows declaring two sorts to be isomorphic.

It is easy to see that if the set of $\simeq$-admissible words is finite, then the cell condition is decidable, because then the relation $\cong$ has a finite domain and hence can be computed. However, it is very unlikely that one can design a polynomial time algorithm for checking the cell condition in this case:

Theorem 3. *In the above notation, it is **PSPACE**-hard to check, knowing that (*) holds and that the set $Adm_{\simeq}$ is finite, whether (**) holds.*

The proof idea is to encode any linear bounded deterministic Turing machine $\mathcal{M}$ [7] in a signature diagram D together with its extension D' so that the cell condition holds for if and only if there is a two-way computation between two given configurations of $\mathcal{M}$.

5 Checking Colimit Thinness

In this section we discuss the following problem: given a finite diagram $D : \mathbf{I} \to \mathbf{lcCat}$ of finite thin categories, is the colimit $\mathbf{C}$ of the diagram thin? Lemma 1 shows that this condition (together with the easily decidable (*)) implies the cell condition (and is equivalent to it under reasonable further conditions).

Unfortunately, colimit thinness is again undecidable in general. Indeed, one can construct the diagram in the proof of Theorem 2 in such a way that the assumptions of both parts of Lemma 1 are satisfied, so that the cell condition is equivalent to colimit thinness there.

However, unlike the cell condition, colimit thinness is decidable by a polynomial time algorithm if the set of $\simeq$-admissible words is finite. This algorithm works roughly as follows:

We think of $\cong$ as a set of cells over D and ask whether it contains all cells, i.e., whether every cell over D is derivable in the cell calculus. Since $Adm_{\simeq}$ is finite, one can define a partial order $\leq$ on $\mathbf{Ob}(\mathbf{C})$ as follows: $\mathbf{s} \leq \mathbf{t}$ if there exists a $\simeq$-admissible word ω with $\mathrm{dom}(\omega) = (i, s)$ and $\mathrm{cod}(\omega) = (j, t)$ such that $\mu_i(s) = \mathbf{s}$ and $\mu_j(t) = \mathbf{t}$, where μ_i and μ_j are the respective colimit injections. We say that a cell (ω, υ) is an $\mathbf{s}$-cell if $\mathrm{dom}(\omega)$ is mapped to $\mathbf{s}$ by the respective colimit injection. The algorithm relies on the observation that for any $\mathbf{s}$-cell c, all derivations of c involve only $\mathbf{s}'$-cells for which $\mathbf{s} \leq \mathbf{s}'$. This implies that one can sort the set $\mathbf{Ob}(\mathbf{C})$ topologically and, for each $\mathbf{s} \in \mathbf{Ob}(\mathbf{C})$, try to prove that all $\mathbf{s}$-cells are in $\cong$ provided that for any $\mathbf{s}' > \mathbf{s}$, all $\mathbf{s}'$-cells are in $\cong$. This can be checked in polynomial time, exploiting the fact that the set of all derivable $\mathbf{s}$-cells is closed with respect to a suitable connectedness relation.

Implemented in a slightly different setting, this algorithm is used in the static analysis of Casl architectural specifications performed by the system CATS. It has polynomial time complexity, but its practical efficiency turns out to be satisfactory only for specifications of moderate size.

6 Imposing Restrictions on the Cell Calculus

There is yet another way of trying to check the cell condition, namely by simply limiting the length of words in the cells considered.

Consider D,Δ and τ as usual. We call a word $\langle (i_n, e_n), \ldots, (i_1, e_1) \rangle \in Adm_{\simeq}$ *loopless* if there are no j, l such that $1 \leq j < l \leq n$ and $(i_j, e_j) = (i_l, e_l)$. We denote the (finite) set of all loopless $\simeq$-admissible words over $Embs(D)$ by $Loopless_{\simeq}$.

Let $\cong^R$ be the least relation on $Loopless_{\simeq}$ that satisfies all the rules of the cell calculus. Obviously $\cong^R \subseteq \cong$, so the *restricted cell condition*

$$(\text{**R})\ \iota_e(\cong_\tau) \subseteq \cong^R$$

implies the original cell condition (**). Since $\cong^R$ has a finite domain, it can be effectively computed. Hence the restricted cell condition is decidable, even if the set $Adm_{\simeq}$ is infinite. However, one can see that it is **PSPACE**-hard in the general case. This follows from Theorem 3, since if $Adm_{\simeq}$ is finite, then the conditions (**R) and (**) are equivalent.

The two properties, colimit thinness and the restricted cell condition, are mutually independent: for either of them one can give a CASL architectural specification for which it fails while the other property holds. However, the restricted cell condition seems to cover all practically relevant architectural specifications. All known examples of architectural specifications that satisfy (**) but not (**R), are complicated and unlikely to occur in practice.

7 Conclusions and Future Work

In [11], a semantics of CASL architectural specifications has been proposed, with static amalgamation conditions formulated in an institution independent way. However, no method of checking the conditions in CASL is given there.

In the present paper, we discuss the problem of checking amalgamation conditions for a simplified version of the CASL institution. We show that the conditions are undecidable in general, and we discuss some important variations of the problem. The first special situation, covering many practical examples of architectural specifications, occurs when the amalgamation condition is ensured by tracing each cell to be considered to a particular signature of a given unit. This implies the cell condition and is easily decidable. Another special case we present is that the set of all cells considered is finite: here the problem is again decidable, although **PSPACE**-hard. Two further sufficient conditions for the amalgamation condition are considered. One of them is colimit thinness, which is undecidable in general, but checkable in polynomial time in an important special case. The other one is given by the restricted cell calculus, which is always decidable, but **PSPACE**-hard.

All results are presented in a somewhat simplified setting, dealing only with the subsort relation in CASL signatures and leaving aside all operations and predicates. When operations are present, the details become somewhat more involved. One extends the definition of an admissible word by adding single operation names as letters, and presents a modified version of the cell calculus dealing with extended words. This adds, for instance, some more complication to the algorithm for checking colimit thinness, but does not change its polynomial complexity. Obviously, this does not affect the negative results presented here (Theorems 2 and 3).

As a practical result of the paper, we propose a method of checking the amalgamation conditions for CASL architectural specifications. Given such a specification, we propose the following steps to establish its (static) correctness, for each of the amalgamation conditions emerging in the static analysis:

1. Compute the relations $\simeq, \simeq_\tau$ and $\cong_\tau$, as described in Section 3.
2. Check the condition (*) as described in Section 3. If this fails, the specification is incorrect.
3. Check the condition described in Section 4.1. If this succeeds, the specification is correct.

4. Check if the set $Adm_{\simeq}$, as described is Section 3, is finite. If not, go directly to step 7.
5. Check the colimit thinness condition, using the algorithm sketched in Section 5. If it succeeds, the specification is correct.
6. Check the cell condition in its full generality (it is decidable, as shown in Section 4.3). The specification is correct if and only if the check succeeds.
7. Check the restricted cell condition (it is decidable, as argued in Section 6). If the check succeeds, the specification is correct.
8. Invoke interaction with the user, generating proof obligations possibly supported by some heuristics, to prove the cell condition as described in Section 3. The specification is correct whenever a proof is found.

In practice, as it is unlikely that polynomial time algorithms for Steps 6 and 7 exist, one should go to Step 8 immediately after unsuccesful completion of Step 5. Steps 1–5 can be performed in polynomial time, keeping in mind that Step 5 is practically efficient only for specifications of moderate size. Step 8 above indicates perhaps the most important direction for our future work on this topic: developing methods to generate proof obligations for checking the amalgamation conditions, and to use theorem provers to assist the user in proving them.

References

1. M. Bednarczyk, M. Borzyszkowski, and W. Pawłowski. Generalized congruences – epimorphisms in Cat. *Theory and Applications of Categories* **5** (1999), 266–280.
2. M. Bidoit, D. Sannella and A. Tarlecki. Architectural specifications in CASL. *Algebraic Methodology and Software Technology*, *LNCS* **1548**, 341–357. Springer, 1999.
3. CoFI Semantics Task Group. CASL – The CoFI Algebraic Specification Language – Semantics. Note S-9 (v. 0.96), July 1999. Accessible by WWW[2] and FTP [3].
4. CoFI Language Design Task Group. CASL – The CoFI Algebraic Specification Language – Summary, version 1.0.1, June 2000. Accessible by WWW[2] and FTP[3].
5. M. Davis. Unsolvable Problems. in: J. Barwise (ed.), *Handbook of Mathematical Logic*, North Holland, Amsterdam, 1977.
6. J.Goguen and R.Burstall. Institutions: Abstract model theory for specifications and programming. *J. ACM* **39** (1992), 95–146.
7. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, 1979.
8. L. C. Paulson. *ML for the Working Programmer* (2nd ed.). Cambridge, 1996.
9. L. Schröder. *Composition Graphs and Free Extensions of Categories* (in German). Ph.D. thesis, University of Bremen, 1999; also: Logos, Berlin, 1999.
10. L. Schröder and H. Herrlich. Free adjunction of morphisms. *Applied Categorical Structures* **8** (2000), 595–606.
11. L. Schröder, T. Mossakowski, A. Tarlecki, B. Klin, and P. Hoffman. Semantics of Architectural Specification in CASL. *Fundamental Approaches to Software Engineering*, *LNCS* **2029**, 253-268. Springer, 2001.
12. L. Schröder, T. Mossakowski, A. Tarlecki. Amalgamation via enriched CASL signatures. ICALP 2001, to appear.

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