Contents

Introduction

Architecture is an ambiguous term. In the context of software engineering, it may mean either the structure of a software system, or the structure of the process leading to the development of such a system.

With the first meaning, specifying architecture amounts to describing the various components of a system and how they are connected and interact. This decomposition into components and connections is, typically, not observable for a user of the system. Nevertheless, it is considered a property of the system, since it may be observed when components are modified, added or removed. In other words, it has no importance from the point of view of a user-oriented specification, but is important from the point of view of maintainance or further development. This notion of architecture as organization of a system into components and connections is used, e.g., in [AG97]. A similar notion is also used in the approach based on CommUnity, developed in [FL97].

In this thesis the second meaning is used: specifying architecture means specifying the software development process, that is, the process in which a software system is developed. This notion of architecture is based on the notions of program development process and of constructor implementation originating from [ST88b, SST92] and summarized in [ST97]; see also the discussion of various software structures in [PCW85]. In the thesis, *architectural specifications* are defined, analyzed and verified. Architectural specifications are formal descriptions of software development processes. By the software development process a process is meant at the beginning of which a requirements specification is given and at the end of which a system (program) satisfying the requirements specification is obtained. Architectural specifications have been first introduced in the specification language CASL [BM04, CoF04b] and in [BST98].

The above two meanings of architecture are not unrelated. It is natural that if, in a requirements specification, a certain structure of components is fixed, then, in the process leading to the development of a program satisfying that specification, a similar structure may be identified. Conversely, the structure of a sufficiently modular program development process may influence what the components of the developed program are.

In the context of architectural specifications, the slogan is often employed that "the structure of implementations may differ from the structure of specifications" and that while the latter is described by *structured specifications*, for the former the distinct notion of architectural specifications should be used [BST02a]. Structured specifications, introduced in the language CLEAR [BG77, BG80] and, in an institution-independent fashion, in [ST88a], specify the same properties as flat specifications, but in a structured way. Such a structure is indispensable, since, for large, practical software projects, a flat specification which is just a list of all the required properties of the system would become unmanageable. However, this structure is devised only in the interest of making the specification easier to understand and use. It is intended to describe neither the decomposition of the program into components and connections, nor the structure of the process of developing a program which satisfies the specification. Therefore it is certainly true that the architecture of a program may differ from the structure of its specification, whereby architecture may be understood in either of the meanings discussed above.

However, because the term implementation is, like architecture, equivocal, usually meaning a program satisfying a given specification, but sometimes meaning the process leading to the development of such a program (as in [BM04]), it should be stressed that architectural specifications describe the structure of implementations in the second meaning, i.e., they describe the structure of the program development process, while structured specifications describe the, possibly distinct, structure of the requirements specification itself. Examples in which these structures actually differ may be found in [FJ90, CoF04a].

To recapitulate, architectural specifications are formal descriptions of the process of program development. More precisely, and in accord with existing usage of the term, architectural specifications describe *steps* in such a process, while the whole process is described by what we call *refinement dags* (directed acyclic graphs).

This thesis is devoted to the study of architectural specifications and, save for the preliminaries of Chapter I, consists of two main parts: Chapter II and Chapter III. In Chapter II, architectural specifications and refinement dags are defined and analyzed. Specifically, in Sections 4 and 5 architectural specifications are introduced and defined formally. Refinement dags and notions of correctness for them and for architectural specifications are defined in Section 6, where it is also proved that refinement dags may be flattened into single architectural specifications (Proposition 5). In the next section, a new construct is added to architectural specifications, namely imports, and refinement and flattening are extended to architectural specifications with imports. Section 8 introduces the distinction between generative and non-generative semantics of architectural specifications, a distinction of great importance for their properties and for their verification.

Architectural specifications and refinement dags may be defined and analyzed on two distinct levels: on the level of models (algebras) of a certain logic, on the one hand, or on the level of programs on the other hand. Since we assume that the semantics of every program is given as a model of the considered logic, these levels are closely related. Only at first glance is this relationship unproblematic. In the context of this relationship, Section 9 strongly suggests that instead of the original semantics of architectural specifications, defined after [CoF04b], a modified version, called partial model semantics, should be used. In Section 10, it is shown that the studied relationship between correctness on the level of models and on the level of programs is as expected if and only if the architectural specifications concerned satisfy certain static assumptions, which we call fully static correctness (see Theorems 20 and 28 and Proposition 25). Also, in these results and in Theorem 29, the importance of the amalgamation and iso-reflection properties of the considered logic becomes apparent. In Section 11 it is then shown that similar results cannot be expected for the so-called non-generative semantics (Proposition 30). The final section of Chapter II discusses the existing literature on the problems considered in the previous sections and contains conclusions. In particular, it is advocated that the semantics of architectural specifications should be partial and generative and that fully static correctness should be enforced.

Chapter III of the thesis is dedicated to the verification of architectural specifications and refinement dags, that is, to checking their correctness; only verification for the generative semantics is considered in detail. After some introductory remarks, in Section 14 we provide a sound and complete proof calculus for checking that fully statically correct architectural specifications and refinement dags are correct (Theorem 35). Completeness is relative in the sense that proof obligations in the underlying specification language are produced.

Then the verification problem for architectural specifications and refinement dags in specification languages without amalgamation is considered. Perhaps the most notable such language is CASL [BM04, CoF04b, ABK+02], designed as a common platform for research on algebraic specification. CASL was intended to consolidate existing work in the area and be a focal point in developing new ideas and techniques. A wide spectrum of features has been included in CASL, allowing research in many areas to be conducted within one formalism. A formal definition of its semantics makes it possible to reason rigorously about CASL specifications. For all these reasons it is natural to consider verification in the context of CASL specifications, and this is what we do in this thesis. CASL's complexity makes this both a challenging and worthwhile task, since a verification method which can deal with CASL may be expected to be able to deal with other specification languages as well. One of the important properties of CASL is its lack of amalgamation. Unfortunately, since for languages without amalgamation the notion of fully static correctness is not meaningful, other techniques have to be developed. This is done in the next two sections.

Specifically, in Section 15 we consider verification under the assumption of extended static correctness, a generalization of fully static correctness to languages without amalgamation. A sound and complete verification method under this assumption is provided (Theorem 42), as is a sound and complete, though not necessarily decidable, method of checking extended static correctness itself (Theorem 41). Both these methods assume that the specification language concerned is represented in a "total" way in some language with amalgamation and, in the latter case, also iso-reflection. Finally, in Section 16, a general sound and complete system for proving correctness is presented (Theorems 43 and 47). Here, it is assumed that the specification language concerned is represented, not necessarily in a total way, in some language with amalgamation. The downside is that additional rather complex assumptions are needed and that a large number of proof obligations is generated.

Next, verification methods developed in Sections 15 and 16 are applied to the specification language CASL. First, in Section 17, a non-total representation of CASL in a sublanguage of CASL that has amalgamation is presented, and it is shown that this representation satisfies all the required assumptions. In Section 18, the specification language ENRICHEDCASL is presented, in which CASL may be represented in a total way, and it is proved that it has amalgamation. Proof of iso-reflection for ENRICHEDCASL is deferred to the next two sections, as it requires applying some more elaborate mathematical techniques.

Thus, in Section 19, which does not depend on other parts of the thesis, we introduce some amalgamation theory for monoids, in particular Isbell's Zigzag Theorem, and generalize it to the cases of categories, operads and multicategories. We also prove certain important consequences regarding iso-reflection for the functor taking any (multi)category C to the class of (multi)functors $C \to \mathbf{Set}$. Using the introduced theory, in Section 20 it is then shown that ENRICHEDCASL indeed has iso-reflection.

In Section 21, the problem of verification for a non-generative semantics is briefly considered.

Finally, Section 22 contains information on existing work on verification of architectural specifications and representations of CASL in other specification languages, as well as conclusions.

The most important results of the thesis are recapitulated in a Summary; the developed verification methods and situations in which they should be applied are also presented there. These methods are then applied to the so-called Steam-Boiler Example, the running example of the CASL User Manual [BM04].

Presentation and Notation

Overall, it has been our concern to make the thesis as natural and easy to read as possible. For this reason, references to the literature and comparisons with definitions and results found there are concentrated in Sections 12 and 22. If this were not the case, the sheer number of versions of certain notions could hamper the readibility. For the same reason, if the proof of a proposition is lengthy and its content is not needed in the following considerations, this proof is usually to be found in the Appendices. Most proofs in the Appendices are not overly difficult; an exception is, perhaps, the proof of Lemma 26, used in the proof of Proposition 25. Note that it should be possible to read the thesis omitting the proofs altogether.

The presentation is mostly from first principles and most of the mathematical concepts and notation are defined before use. However, knowledge of basic category theory is assumed and notions such as functor, limit, completeness, natural transformation, etc., are used freely. For this categorical background, see [Mac71]. Composition in categories is always written in diagrammatical order, i.e., if $f : a \to b$ and $g : b \to c$, then their composition is written as $f;g : a \to c$. The same convention applies to the composition of set-theoretic functions. The domain and codomain of a morphism f are written as $\mathrm{dom}(f)$ and $\mathrm{cod}(f)$, respectively. If C is a category, then $c \in C$ may either mean that c is an object, or that it is a morphism in C, depending on the context. By a *sink* in a category we mean an object Δ together with a family of morphisms having codomain Δ; sinks are written as $\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I}$. In case I is the empty set, the sink σ still defines the signature Δ. Thus, any colimit, including the colimit of an empty diagram, is a sink.

The notation $[y/x]$ is used extensively. If f is a function, then $f[y/x]$ denotes a function that behaves like f on the domain of f save for x and that takes x to y. This notation may be used both if x is and if it is not in the domain of f. If F is a set of functions and Y is a set, then $F[Y/x]$

denotes the set of all $f[y/x]$ such that $f \in F$ and $y \in Y$.

In certain cases the notation $(\_)^*$ is used to denote that some object is lifted to sequences or tuples. So, for example, if S is a set, then S^* is the set of all tuples or sequences of elements of S; similarily, if f is an unary function and $x = (x_1, \ldots, x_n)$ is a tuple, then $f^*(x)$ denotes the tuple $(f(x_1), \ldots, f(x_n))$.

The distinction between sets and classes ("large sets"), is usually retained. However, foundational questions regarding classes are disregarded, since they do not influence in any interesting way the problems considered in this thesis.

Acknowledgements

I am indebted to my supervisor, Professor Andrzej Tarlecki, for his interest in my work and for inspiring suggestions. It was his encouragement that made me choose architectural specifications as my research topic. Without his guidance neither starting, nor finishing this thesis would have been possible. I would also like to thank Bartek Klin, Till Mossakowski and Lutz Schröder for joint discussions and work on architectural specifications and amalgamation; these discussions led to many ideas presented in this thesis. Special thanks go to Bartek Klin for convincing me that a Zigzag Theorem for operads does not hold in general.

Chapter I

Preliminaries

1 Introduction

If the program development process is to be formalized, then a framework must be provided in which notions such as "program" or "specification" are defined. This framework must have some rudimentary structure, which will later be exploited to give structure to the development process itself. Such a framework is described below.

We assume that a notion of *signature* is given, equivalent to what in programming is usually called an *interface*. The intuitive meaning of a signature is that it defines the tests that an observer may perform on a system. For instance, in Standard ML [Pau96], the only test an observer may conduct on a program over the signature `sig val sqrt : real -> real end` is take a real number, apply `sqrt` and check what the result is. Thus, a signature defines a language of possible observations on a program, usually in the form of operations which may be performed. Signatures are denoted by Σ, Δ, etc.

Apart from Standard ML signatures, a standard example of signatures are *algebraic signatures*. An algebraic signature is a pair (S, p), where S is a set of *sorts* and p is a function into $S^* \times S$. An element f of the domain of p is called a *function symbol* and, if $p(f) = ((s_1, \ldots, s_n), s)$, then its *arity* is said to be n and its *profile* $s_1 \times \cdots \times s_n \to s$; we then simply write $f : s_1 \times \cdots \times s_n \to s$ or, if $n = 0$, $f : s$.

It is also assumed that, for any signature Σ, a class of *programs over* Σ or, equivalently, Σ-programs, is given. It is denoted by $\mathit{Prog}(\Sigma)$.

We could stop here and specify programs directly. However, it is preferable to add a degree of indirectness and specify programs through their semantics. Thus, we assume additionally that, for any signature Σ, a class of *models over* Σ or, equivalently, Σ-models, is given. It is denoted by $\mathbf{Mod}(\Sigma)$

and its elements are denoted by M, N, etc. Models over Σ represent the possible behaviors of a program with interface Σ. Now, programs also have a *semantics*, that is, for any signature Σ, a function $[\![\_]\!]_\Sigma : \mathit{Prog}(\Sigma) \to \mathbf{Mod}(\Sigma)$, taking any Σ-program P to the Σ-model $[\![P]\!]_\Sigma$ which represents its behavior. The program P is said to *implement* the model $[\![P]\!]_\Sigma$. If no implementation of a certain Σ-model M exists, then M is called *unimplementable*. The semantics need neither be surjective, so that unimplementable models may exist, nor injective, so that the same model may be implemented by many programs.

A standard example of a model over an algebraic signature is a partial algebra. For any algebraic signature $\Sigma = (S, p)$, the class $\mathbf{PAlg}(\Sigma)$ of *partial algebras over* Σ consists of pairs (M^S, M^F), where M^S is a function taking any sort s in Σ to a set $M^S(s)$, called the *carrier of* s, and M^F is a function taking any function symbol $f : s_1 \times \cdots \times s_n \to s$ in Σ to a partial function $M^F(f) : M^S(s_1) \times \cdots \times M^S(s_n) \to M^S(s)$, called the *interpretation of* f.

For simple programming languages, a partial algebra semantics, perhaps extended in some way, may be appropriate. But, even for functional programs, many other kinds of models are also considered, in which the behavior of a program is described in a way more abstract or less abstract than by a partial algebra. For example, a more abstract notion is one in which observationally equivalent algebras are considered equal [ST87]; a less abstract notion of model, on the other hand, could be one in which the interpretation of any function symbol is a function which returns not only a result, but also the number of steps used to compute that result (see, e.g., the semantics of Grail [ABH+04]).

To the notions of signature, program and model we now add a notion of *specification*, i.e., an object describing a class of programs. As already announced, this class of programs is defined indirectly, via the semantics. So, it is assumed that, for any signature Σ, a class of *specifications over* Σ or, equivalently, Σ-specifications, is given. It is denoted by $\mathit{Spec}(\Sigma)$ and its elements are denoted by SP, SP', etc. Moreover, these specifications have their own semantics, each specification SP yielding a signature $\mathit{sig}[SP]$ and a class of allowed behaviors $[\![SP]\!] \subseteq \mathbf{Mod}(\mathit{sig}[SP])$, called the *model class of* SP. Then, any specification SP defines a class of $\mathit{sig}[SP]$-programs, written $[\![\![SP]\!]\!]$, namely the class

$$\big\{P \in \mathit{Prog}(\mathit{sig}[SP]) \mid [\![P]\!]_{\mathit{sig}[SP]} \in [\![SP]\!]\big\}$$

The assumptions we have imposed on specifications, even though very basic, do not always hold in the real world, since, for instance, specifications may be incorrect and thus not yield a signature or class of models, or they may depend on some additional parameters, like a global environment of previously defined specifications. In fact, both of these problems arise in CASL. Nevertheless, we stick by these assumptions, since giving them up would complicate the presentation, while any gains in generality would be quite superficial.

A specification is called *inconsistent* if its model class is empty; it is called *total* if its model class consists of all models over the given signature. A total specification over Σ is denoted by $\top_\Sigma$ and it is assumed that, for any signature Σ, such a specification exists. Note that a total specification may be inconsistent. Two specifications are said to be *equivalent* if their signatures and model classes are equal.

An example of a specification in the framework of partial algebras is an equational specification, i.e., for any algebraic signature $\Sigma = (S, p)$, a pair consisting of the signature Σ and of a finite set of Σ-*equations*, that is, pairs (t_1, t_2), written $t_1 = t_2$, of Σ-terms of the same sort and over a common set of variables. A partial algebra is said to *satisfy an equation* $t_1 = t_2$, if the inductively defined interpretations of t_1 and t_2 coincide for any valuation of the variables, meaning that either both are undefined, or both are defined and equal (this is so-called *strong equality*). Then, the model class of a Σ-specification SP is the class of all partial algebras over Σ which satisfy all the equations from SP. The class of all equational specifications over Σ is denoted by $EqSpec(\Sigma)$.

Up to now, we have not imposed any real structure on signatures, models or programs. We do this now, requiring that signatures form a category *Sig* (the *signature category*) and that $Prog : Sig^{op} \to \mathbf{Class}$ and $\mathbf{Mod} : Sig^{op} \to \mathbf{Class}$ are functors, the *program functor* and the *model functor*, respectively. We also require that the semantics of programs $[\![\_]\!]$ be a natural transformation from *Prog* to **Mod**. Morphisms in *Sig* are called *signature morphisms* and denoted by σ, τ, etc. They represent, e.g., renaming of signature elements or inclusion of a smaller signature in a bigger one.

The requirement that **Mod** be a contravariant functor, taking, for any signature morphism $\sigma : \Sigma \to \Delta$, any Δ-model M to a Σ-model written as $M|_\sigma$, may be explained as follows. If a Δ-model M and a signature morphism $\sigma : \Sigma \to \Delta$ are given, then the behavior $M|_\sigma$ is defined by the property that, when tested by a Σ-observation o, it returns the result that M would return under the Δ-observation $\sigma(o)$, obtained by translating o along σ. For partial algebras, a signature morphism $\sigma : (S, p) \to (T, q)$ may be defined as a pair of functions $(\sigma^S : S \to T, \sigma^F : \mathrm{dom}(p) \to \mathrm{dom}(q))$ such that $q(\sigma^F(f)) = ((\sigma^S(s_1), \ldots, \sigma^S(s_n)), \sigma^S(s))$, where $p(f) = ((s_1, \ldots, s_n), s)$. For any partial algebra M over (T, q), the reduct $N = M|_\sigma$ is defined by $N^S = \sigma^S; M^S$ and $N^F = \sigma^F; M^F$. The functoriality of **Mod** is obvious.

The requirement that *Prog* be a contravariant functor as well, and that the semantics functions form a natural transformation, is more limiting. It means that, for any signature morphism $\sigma : \Sigma \to \Delta$, it is possible not only to transform behaviors over Δ to behaviors over Σ, but also that, having implemented a Δ-behavior M by a program P, we are able to transform this program P to a program $P|_\sigma$, which implements the Σ-behavior $M|_\sigma$. In fact, this is an assumption regarding the modularity of the programming language used; though, of course, a non-modular programming language also fits into this framework if, as its signature category, a discrete category is taken.

For our purposes it is convenient to assume that in the signature category a notion of *signature inclusion* is defined. An *inclusion category* for a signature category *Sig* is a subcategory $\mathcal{I}$ of *Sig* such that all signatures are objects in $\mathcal{I}$ and between any two signatures Σ and Δ there exists at most one morphism in $\mathcal{I}$, called the *inclusion of* Σ *into* Δ and denoted by $\iota_{\Sigma\subseteq\Delta}$. If such an inclusion exists, Σ is called a *subsignature* of Δ. Reducts $\_|_{\iota_{\Sigma\subseteq\Delta}}$ along inclusions are written as $\_|_\Sigma$ if Δ is clear from the context. In the framework of partial algebras, an inclusion is defined as a signature morphism (σ^S, σ^F) such that both components are set-theoretic inclusions; this category of inclusions is denoted by $\mathcal{I}_{PA}$.

The notions presented above may be summarized by two definitions. A *specification language* $\mathcal{L}$ is a tuple $(\mathit{Sig}, \mathcal{I}, \mathbf{Mod}, \mathit{Spec}, \mathit{sig}[\_], [\![\_]\!])$, where:

- *Sig* is a category,
- $\mathcal{I}$ is an inclusion category for *Sig*,
- $\mathbf{Mod} : \mathit{Sig}^{op} \to \mathbf{Class}$ is a functor,
- $\mathit{Spec} : \mathit{Sig} \to \mathbf{Class}$ is a function,
- $\mathit{sig}[\_]$ and $[\![\_]\!]$ form a semantics of specifications, that is, for any $\mathit{SP} \in \mathit{Spec}(\Sigma)$, $\mathit{sig}[\mathit{SP}] = \Sigma$ and $[\![\mathit{SP}]\!] \subseteq \mathbf{Mod}(\Sigma)$; moreover, for any signature Σ, there is a total Σ-specification $\top_\Sigma$.

Clearly, partial algebras with equational specifications form a specification language $\mathcal{PA} = (\mathit{AlgSig}, \mathcal{I}_{PA}, \mathbf{PAlg}, \mathit{EqSpec}, \mathit{sig}[\_], [\![\_]\!])$.

A *programming language* $\mathcal{P}$ is a tuple $(\mathit{Sig}, \mathbf{Mod}, \mathit{Prog}, [\![\_]\!])$, where:

- *Sig* is a category,
- $\mathbf{Mod} : \mathit{Sig}^{op} \to \mathbf{Class}$ is a functor,
- $\mathit{Prog} : \mathit{Sig}^{op} \to \mathbf{Class}$ is a functor,

- $[\![\_]\!]$ is a natural transformation from *Prog* to **Mod**, that is, a family of functions $\{[\![\_]\!]_\Sigma\}_{\Sigma \in Sig} : Prog(\Sigma) \to \mathbf{Mod}(\Sigma)$ natural in $\Sigma \in Sig$.

Before we provide an example of a programming language, we present a slightly more complex specification language. As has been already mentioned, a signature morphism $\sigma : \Sigma \to \Delta$ is, in principle, a mechanism of defining Σ-models from Δ-models. In the language $\mathcal{PA}$ this mechanism is very rudimentary, since it only allows one to define the interpretations of elements of Σ as interpretations of elements of Δ. This mechanism may be rendered richer by allowing *derived signature morphisms.* The notion of derived signature morphism is quite general, usually meaning signature morphisms more complex than the standard ones. We now present a simple version of derived signature morphisms for partial algebras; a more complicated example, in the context of observational equivalence, may be found in [BST02b].

The category $AlgSig^{der}$ of algebraic signatures with derived morphisms has, as objects, algebraic signatures; signature morphisms $\sigma : (S,p) \to (T,q)$ in this category, which we call *derived signature morphisms*, are pairs (σ^S, σ^F), with σ^S, as before, a function from S to T and with σ^F a function taking any function symbol $f : s_1 \times \cdots \times s_n \to s$ in (S,p) to a term $\sigma^F(f)$ over (T,q) of sort $\sigma^S(s)$ and with variables $x_1 : \sigma^S(s_1), \ldots, x_n : \sigma^S(s_n)$. Composition of signature morphisms is defined naturally via substitution. Note that *AlgSig* will be treated as a subcategory of $AlgSig^{der}$, under the obvious embedding. This allows us to treat $\mathcal{I}_{PA}$ as an inclusion category for $AlgSig^{der}$.

The functor $\mathbf{PAlg}^{der}$ works like **PAlg** on objects and the reduct $N = M|_\sigma$ is defined by setting $N^S = \sigma^S; M^S$ and N^F to a map which takes any function symbol f as discussed above to a function which, applied to a tuple $(v_1, \ldots, v_n)$ of elements of the appropriate carriers, returns the value of the term $\sigma^F(f)$ in M under the valuation $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$. Observe that, for morphisms from *AlgSig*, the functors **PAlg** and $\mathbf{PAlg}^{der}$ coincide.

Then $\mathcal{PA}^{der} = (AlgSig^{der}, \mathcal{I}_{PA}, \mathbf{PAlg}^{der}, Spec, sig[\_], [\![\_]\!])$ is a specification language. Now, let Σ_{nat} be the algebraic signature

sort *nat*;
op $0 : nat$;
ops $succ, pred : nat \to nat$

and let M_{nat} be a model of Σ_{nat}, in which the carrier of *nat* is $\mathbb{N}$, the interpretation of 0 is 0, the interpretation of *succ* is the successor and the interpretation of *pred* is a (total) predecessor, taking $n+1$ to n and 0 to 0.

For any algebraic signature Δ, let Δ-programs be all derived signature morphisms $P : \Delta \to \Sigma_{nat}$. For any signature morphism $\sigma : \Sigma \to \Delta$ in

AlgSig, let $P|_\sigma = \sigma; P$. This defines a functor $\mathit{Prog} : \mathit{AlgSig}^{op} \to \mathbf{Class}$. The semantics of programs may then be defined by the rule $[\![P]\!]_\Sigma = M_{nat}|_P$. This gives a programming language $\mathcal{PPA} = (\mathit{AlgSig}, \mathbf{PAlg}, \mathit{Prog}, \{[\![\_]\!]_\Sigma\}_{\Sigma \in \mathit{AlgSig}})$.

The idea of derived morphisms may be taken even further, allowing one to use not only simple algebraic terms, as above, but also fixed-points, etc., with full programming power. In fact, with a stretched definition, any parametric program (e.g., Standard ML functor) can be treated as a "highly derived" signature morphism from the signature of its result to the signature of its parameter.

A specification language $\mathcal{L} = (\mathit{Sig}, \mathcal{I}, \mathbf{Mod}, \mathit{Spec}, \mathit{sig}[\_], [\![\_]\!])$ and a programming language $\mathcal{P} = (\mathit{Sig}, \mathbf{Mod}, \mathit{Prog}, \{[\![\_]\!]_\Sigma\}_{\Sigma \in \mathit{Sig}})$ are called *compatible*; thus, a specification and a programming language are compatible if they share the signature category and model functor. For compatible languages, one can construct the specification language $(\mathit{Sig}, \mathcal{I}, \mathit{Prog}, \mathit{Spec}, \mathit{sig}[\_], [\![[\![\_]\!]]\!])$. This language is denoted by $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$. It is the specification language for programs defined in $\mathcal{P}$, obtained from the specification language $\mathcal{L}$ for models. This language is of special importance, since, ultimately, we are interested in building programs, not abstract models.

2 The CASL Specification Language

In this section, the Common Algebraic Specification Language CASL [BM04, CoF04b, ABK+02], as a specification language in the sense of the previous section, is defined. This is necessary for two reasons.

First of all, even though in all of Chapter II and in much of Chapter III we quantify over all specification languages with certain properties, while reading it is important to have non-trivial examples of such languages in mind. The languages $\mathcal{PA}$ and $\mathcal{PA}^{der}$ for partial algebras are not sufficient in this respect. The language CASL, on the other hand, is not only complex enough, but it also has interesting sublanguages, so that the reader can instantiate propositions and theorems not to one CASL language, but to several sublanguages with various properties.

Because of this, first an informal description of CASL is provided, intended to give the reader an idea of what a complex specification language might look like.

A second reason for including a definition of CASL is that, in Sections 17–20, we are concerned with representing CASL in specification languages with better properties. In order to define these representations, a formal definition of CASL itself is indispensable. It should be noted that the definition provided below is much simpler than the official definition of CASL, since it abstracts away from many details and focuses on the most important aspects.

For this reason, after the informal description of CASL, we provide a rather formal definition, intended to be used in sections of Chapter III concerned with representations of CASL. It is commendable to skip this rather cumbersome definition now and return to it only before Section 17.

One additional remark is necessary here. From our perspective, CASL consists of two, to some extent independent, formalisms. The first one is a specification language; this is what is described and defined below. The second is the architectural specifications formalism, which is, of course, intended to be applied to the CASL specification language as its underlying language. In Chapter II, we provide our own definition of architectural specifications, referring to the version defined in CASL as to CASL architectural specifications. Both versions of architectural specifications are very similar and, in principle, they are both parametrized by an underlying specification language.

The CASL specification language is defined in a sequence of steps. First, there are CASL *basic (flat) specifications*, that is, specifications without structuring constructs.

These come in two flavors. The simpler one is the specification language of basic multisorted CASL, denoted by BASICMULTICASL. This is, in many respects, similar to the specification language $\mathcal{PA}$. Signatures are multisorted, defining sorts, partial and total function symbols, and predicate symbols. Note that a function symbol here consists of a name and a profile: two symbols with the same name may still be considered distinct (the same applies to predicate symbols). Signature morphisms are standard (non-derived). As for models, they are defined naturally, defining carriers of sorts and interpretations of partial and total function symbols and predicate symbols. One point worth noting is that all carriers are required to be non-empty. Finally, specifications are just signatures together with sets of first-order sentences with (strong) equality and sort-generation constraints. Sort-generation constraints are used to force the model to be generated from a certain submodel using certain function symbols.

It will be of importance later that not only models of BASICMULTICASL, but also homomorphisms between models over a common signature are defined, as are reducts of such homomorphisms. In BASICMULTICASL, these homomorphisms are weak, that is, they preserve, but are not required to reflect the definedness of partial function application or the satisfaction of predicates.

The more sophisticated flavor of basic specifications is the language of basic subsorted CASL, denoted by BASICSUBCASL. The only difference in comparison to BASICMULTICASL is that signatures here define a preorder on the sorts, called the *subsort preorder*. The fact that $s \leq t$ in this preorder, written, in specifications, as $s < t$, is to be interpreted as stating that s is a subsort of t. Models in BasicSubCasl contain, for any $s \leq t$ in the preorder, an injection from the carrier of s to the carrier of t, called the interpretation of the embedding $s < t$. Profiles of function and predicate symbols and the use of subsort injections in terms may be implicit. For this reason, it is necessary, in order to get a sound semantics, to impose certain consistency conditions on the injections. Thus, the interpretation of $s < s$ must be the identity and the interpretation of $s < t$ composed with the interpretation of $t < u$ must be equal to the interpretation of $s < u$. Moreover, if two function symbols have the same name and there is a common subsort of the sort of their arguments and a common supersort of the sort of their results, then they must also satisfy certain consistency conditions (a similar rule applies to predicate symbols). Details may be found in the formal definition below.

Casl also contains *structuring operations*, that is, mechanisms for building bigger specifications from smaller ones. These operations are defined along the lines of [ST88a] and may be applied to any specification language. They include sum, translation, reduction (hiding) and a freeness construct. For the freeness construct, the homomorphisms between models, mentioned above, are used. Parametric specifications, i.e., specifications parametrized by specifications, are also provided; however, we do not present them here, not even in the formal definition, since any specification using them may be syntactically transformed into an equivalent specification without them; due to the parameter fitting mechanism in parametric specifications, such a syntactic transformation may give rise to proof obligations.

By applying the structuring operations to BasicMultiCasl and BasicSubCasl, respectively, the specification languages of multisorted Casl, denoted by MultiCasl, and of subsorted Casl, denoted by SubCasl, are obtained.

In fact, SubCasl is the full Casl specification language. However, we prefer to use the name SubCasl instead, so that no misunderstanding can occur as to which version of Casl is meant. One additional distinction is needed here. No cardinality restrictions are enforced on the specification languages defined in this section. This is in contrast to the official semantics of Casl, where it is required that, first, all signatures are finite, i.e., there is a finite number of sorts and of function and predicate symbols, and that, second, basic specifications may contain only a finite number of sentences. This finiteness requirement is, of course, reasonable; however, for most of this thesis, enforcing it would involve cluttering definitions and propositions with finiteness assumptions. Therefore, all specification languages are defined without finiteness assumptions and when these assumptions are of importance (Section 16), we speak of *finite* SubCasl, *finite* MultiCasl, etc.

We now proceed to formally define the specification languages briefly introduced above.

A notion very similar to that of a specification language as defined in Section 1 is that of an *institution* [GB92, Tar99]. An institution **Ins** is a tuple $(Sig, \mathbf{Mod}, Sen, \{\models_\Sigma\}_{\Sigma \in Sig})$, where:

- Sig is a category,
- $\mathbf{Mod} : Sig^{op} \to \mathbf{CAT}$ is a functor (note that its codomain is the category of all categories),
- $Sen : Sig \to \mathbf{Set}$ is a functor, called the *sentence functor*,
- $\{\models_\Sigma\}_{\Sigma \in Sig}$ is a family of relations $\models_\Sigma \subseteq Sen(\Sigma) \times \mathbf{Mod}(\Sigma)$, called *satisfaction relations*.

Additionally, in an institution, the *satisfaction condition* must hold, that is, for any signature morphism $\sigma : \Sigma \to \Delta$, any model $M \in \mathbf{Mod}(\Delta)$ and any sentence $\phi \in Sen(\Sigma)$, we must have

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

In an institution, as in a specification language, the reduct $\mathbf{Mod}(\sigma)(M)$ is written $M|_\sigma$; also, instead of $Sen(\sigma)(\phi)$ one simply writes $\sigma(\phi)$ and calls it the *translation of ϕ along σ*.

As shown below, except for **CAT** being the codomain of **Mod**, the main differences between a specification language and an institution are that the former has a notion of inclusion (an idea already existing in the literature), that *Spec* is a function and not a functor, that any specification defines a signature and that there are total specifications over all signatures. Note that, in the definition of Casl, yet another notion is used, namely that of an *institution with symbols* and with *raw symbols* (see [Mos00]); there, the "symbol" structure is used to explicitly define signature morphisms, something we treat as given here.

Any institution $\mathbf{Ins} = (Sig, \mathbf{Mod}, Sen, \models)$ together with an inclusion category $\mathcal{I}$ for Sig gives rise to a specification language of *presentations over* **Ins** *with inclusions* $\mathcal{I}$, written $\mathrm{Pres}_\mathcal{I}(\mathbf{Ins})$, namely $(Sig, \mathcal{I}, \mathbf{Mod}, Spec, sig[\_], [\![\_]\!])$, where, for any signature Σ, $Spec(\Sigma) = \{(\Sigma, \Phi) \mid \Phi \subseteq Sen(\Sigma)\}$, $sig[(\Sigma, \Phi)] = \Sigma$ and $[\![(\Sigma, \Phi)]\!] = \{M \in \mathbf{Mod}(\Sigma) \mid M \models_\Sigma \phi \text{ for all } \phi \in \Phi\}$. Note that, in a finite version of the specification language, the sets of sentences Φ in presentations (Σ, Φ) are required to be finite. It is clear that this specification language has a richer structure than a regular specification language in that its model functor has as codomain not **Class**, but **CAT**; such objects will be called *enriched specification languages*.

The institution **MultiCasl** of basic multisorted CASL is a tuple $(Sig, \mathbf{Mod}, Sen, \{\models_\Sigma\}_{\Sigma \in Sig})$, the components of which are defined as follows.

A signature Σ in Sig is a tuple (S, TF, PF, P), where:

- S is a set of *sorts*,
- $TF = \{TF_{w,s}\}_{w \in S^*, s \in S}$ is a family of sets of *names of total function symbols*,
- $PF = \{PF_{w,s}\}_{w \in S^*, s \in S}$ is a family of sets of *names of partial function symbols*,
- $P = \{P_w\}_{w \in S^*}$ is a family of sets of *names of predicate symbols.*

It is additionally required that always $TF_{w,s} \cap PF_{w,s} = \emptyset$.

A *function symbol* takes the form $f_{w,s}$, where $f \in TF_{w,s} \cup PF_{w,s}$. It is *total* if $f \in TF_{w,s}$ and *partial* otherwise. A *predicate symbol* takes the form b_w, where $b \in P_w$.

A signature morphism $\sigma : (S, TF, PF, P) \to (S', TF', PF', P')$ in Sig is a triple $(\sigma^S, \sigma^F, \sigma^P)$, where:

- $\sigma^S : S \to S'$ is a function,
- $\sigma^F = \{\sigma^F_{w,s}\}_{w \in S^*, s \in S}$ is a family of functions $\sigma^F_{w,s} : TF_{w,s} \cup PF_{w,s} \to TF'_{(\sigma^S)^*(w), \sigma^S(s)} \cup PF'_{(\sigma^S)^*(w), \sigma^S(s)}$, preserving the totality of symbols,
- $\sigma^P = \{\sigma^P_w\}_{w \in S^*}$ is a family of functions, with $\sigma^P_w : P_w \to P'_{(\sigma^S)^*(w)}$.

Signature morphisms compose naturally.

For any signature Σ, $\mathbf{Mod}(\Sigma)$ is the category of models having:

- as models M, triples (M^S, M^F, M^P) of functions, taking:
  - any sort s in Σ to a non-empty carrier set $M^S(s)$,
  - any function symbol $f_{w,s}$ in Σ, where $w = s_1 \ldots s_n$, to a partial function $M^F(f_{w,s}) : M^S(s_1) \times \cdots \times M^S(s_n) \to M^S(s)$, the function being total if $f_{w,s}$ is,
  - any predicate symbol b_w in Σ, where $w = s_1 \ldots s_n$, to a set $M^P(b_w) \subseteq M^S(s_1) \times \cdots \times M^S(s_n)$.
- as morphisms $h : M \to M'$, families $\{h_s\}_{s \in S^*}$ of (total) functions $h_s : M^S(s) \to (M')^S(s)$ such that, for any tuple of sorts $w = s_1 \ldots s_n$ in Σ and any tuple $(x_1, \ldots, x_n) \in M^S(s_1) \times \cdots \times M^S(s_n)$:

- for any function symbol $f_{w,s}$ in Σ, if $h_s(M^F(f_{w,s})(x_1,\ldots,x_n))$ is defined, then so is $(M')^F(f_{w,s})(h_{s_1}(x_1),\ldots,h_{s_n}(x_n))$ and they are both equal,
- for any predicate symbol b_w in Σ, if $(x_1,\ldots,x_n) \in M^P(b_w)$, then $(h_{s_1}(x_1),\ \ldots,\ h_{s_n}(x_n)) \in (M')^P(b_w)$.

Homomorphisms compose naturally.

For any signature morphism $\sigma = (\sigma^S, \sigma^F, \sigma^P) : \Sigma = (S, TF, PF, P) \to \Sigma' = (S', TF', PF', P')$, the reduct functor $\_|_\sigma : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ takes:

- any model $M' \in \mathbf{Mod}(\Sigma')$ to the model $M \in \mathbf{Mod}(\Sigma)$ defined by:
  - for all sorts s in Σ,
  $$M^S(s) = (M')^S(\sigma^S(s)),$$
  - for all function symbols $f_{w,s}$ in Σ,
  $$M^F(f_{w,s}) = (M')^F(\sigma^F_{w,s}(f)_{(\sigma^S)^*(w),\sigma^S(s)}),$$
  - for all predicate symbols b_w in Σ,
  $$M^P(b_w) = (M')^P(\sigma^P_w(b)_{(\sigma^S)^*(w)}).$$
- any Σ'-homomorphism $h' = \{h'_{s'}\}_{s' \in S'} : M' \to N'$ to the Σ-homomorphism $h = h'|_\sigma : M'|_\sigma \to N'|_\sigma$ defined by $h = \{h'_{\sigma^S(s)}\}_{s \in S}$.

For any signature Σ, $Sen(\Sigma)$ is the set of all closed first-order formulas, with atomic formulas built using variables of sorts in Σ, function symbols and predicate symbols in Σ, and equality. Additionally, triples (S_0, F_0, θ), called *sort-generation constraints*, are also in this set, provided that:

- $\theta : \Sigma_0 \to \Sigma$ is a signature morphism,
- S_0 is the set of *generated sorts*, i.e., a set of sorts from Σ_0,
- F_0 is the set of *constructors*, i.e., a set of function symbols from Σ_0.

For any signature morphism $\sigma : \Sigma \to \Sigma'$, the function $Sen(\sigma) : Sen(\Sigma) \to Sen(\Sigma')$ is defined naturally on first-order sentences and, on sort-generation constraints, by the rule $\sigma((S_0, F_0, \theta)) = Sen(\sigma)((S_0, F_0, \theta)) = (S_0, F_0, \theta; \sigma)$.

The satisfaction relation $M \models_\Sigma \phi$ for first-order ϕ is defined as the usual first-order satisfaction, with equality interpreted strongly and predicates on non-defined terms being false. For sort-generation constraints we define $M \models_\Sigma (S_0, F_0, \theta)$ if, for any $s_0 \in S_0$ and any $x_0 \in (M|_\theta)^S(s_0)$, there exists a term t_0 over $\mathrm{dom}(\theta)$ of sort s_0, built using function symbols from F_0 and with no variables of sorts from S_0, and a valuation v_0 into $M|_\theta$ such that t_0 under valuation v_0 has value x_0.

The satisfaction condition does indeed hold and so **MultiCasl** is an institution (see [CoF04b]).

A signature morphism σ in *Sig* is considered an inclusion if σ^S is a set-theoretic inclusion, σ^F is a family of inclusions and σ^P is a family of inclusions. The above-defined category of inclusions for *Sig* is denoted by $\mathcal{MI}$.

We now form the (enriched) specification language $\mathrm{Pres}_{\mathcal{MI}}(\mathbf{MultiCasl})$. This is the specification language of *basic multisorted* CASL; it is denoted by BASICMULTICASL.

However, when writing down specifications in this language, we do not give signatures as quadruples, signature morphisms as triples, and so on. Instead, we use special CASL syntax developed for this purpose. This syntax is mostly self-explanatory. Its semantics is also fully and precisely defined in [CoF04b]. The same remark applies to BASICSUBCASL, described below.

Basic subsorted CASL is built by extending basic multisorted CASL with subsorts. Subsorted CASL supports a rather general form of subsorting, in which the interpretation of a subsort relation $s < t$ may be an arbitrary injection, not necessarily an inclusion. The rationale behind such a decision is that this is the case in the real world, e.g., integers are typically a subsort of the reals, but they have a different representation. A second reason is that, in the subsorts-by-inclusions approach, the relation of being a model of a specification is not closed under model isomorphism. To see this, one has to use the structuring operation $\_|_{\sigma:\Sigma\to\Delta}$, usually called hiding, defined towards the end of this section. Consider the specification $\top_\Delta|_{\iota_{\Sigma\subseteq\Delta}}$, where $\Sigma = \{\mathbf{sorts}\ s, t\}$ and $\Delta = \{\mathbf{sorts}\ s < t\}$. Its models are models in which the carrier of s is a subset of the carrier of t. This class of models is, obviously, not closed under isomorphism, at least for standard notions of homomorphism.

The institution **SubCasl** of *basic subsorted* CASL is a tuple $(\mathit{Sig}_\leq, \mathbf{Mod}_\leq, \mathit{Sen}_\leq, \{\models^\leq_\Sigma\}_{\Sigma\in \mathit{Sig}_\leq})$, with the components defined as follows.

A signature Σ in $\mathit{Sig}_\leq$ is a tuple $(S, \mathit{TF}, \mathit{PF}, P, \leq)$ such that $(S, \mathit{TF}, \mathit{PF}, P)$ is a signature in *Sig* and $\leq$ is a *subsort preorder* on $S \times S$, i.e., a reflexive-transitive relation. For such a signature, the *overloading relations* $\sim_F$ and $\sim_P$ on function and predicate symbols are defined by:

- $f_{w_1,s_1} \sim_F f_{w_2,s_2}$ if there exist w and s with $w \leq^* w_1$, $w \leq^* w_2$, $s_1 \leq s$ and $s_2 \leq s$,

- $b_{w_1} \sim_P b_{w_2}$ if there exists w with $w \leq^* w_1$ and $w \leq^* w_2$.

A signature morphism $\sigma : \Sigma \to \Sigma'$ in $Sig_{\leq}$, where $\Sigma = (S, TF, PF, P, \leq)$ and $\Sigma' = (S', TF', PF', P', \leq')$, is a signature morphism $\sigma = (\sigma^S, \sigma^F, \sigma^P) : (S, TF, PF, P) \to (S', TF', PF', P')$ in Sig such that:

- $s \leq s'$ implies $\sigma^S(s) \leq' \sigma^S(s')$ for all sorts s, s' in Σ,
- $f_{w_1,s_1} \sim_F f_{w_2,s_2}$ implies $\sigma^F_{w_1,s_1}(f)_{(\sigma^S)^*(w_1),\sigma^S(s_1)} \sim'_F \sigma^F_{w_2,s_2}(f)_{(\sigma^S)^*(w_2),\sigma^S(s_2)}$ for all function symbols f_{w_1,s_1}, f_{w_2,s_2} in Σ,
- $b_{w_1} \sim_P b_{w_2}$ implies $\sigma^P_{w_1}(b)_{(\sigma^S)^*(w_1)} \sim'_P \sigma^P_{w_2}(b)_{(\sigma^S)^*(w_2)}$ for all predicate symbols b_{w_1}, b_{w_2} in Σ.

For any signature $\Sigma = (S, TF, PF, P, \leq)$ in $Sig_{\leq}$, by $\Sigma^{\#}$ denote the signature (S, TF', PF', P') in Sig, where:

- $TF'_{w,s} = TF_{w,s} \cup \{em_{s' \leq s} \mid w = s' \leq s\}$ (the *embeddings*),
- $PF'_{w,s} = PF_{w,s} \cup \{prj_{s' \geq s} \mid s \leq s' = w\}$ (the *projections*),
- $P'_w = P_w \cup \{in_{s,s'} \mid s' \leq s = w\}$ (the *membership predicates*).

Here, it is assumed that the embeddings and projections are distinct from each other and from other function symbols and the membership predicates are distinct from other predicate symbols. The operation $(\_)^{\#}$ on signatures naturally extends to a functor $(\_)^{\#} : Sig_{\leq} \to Sig$. In the sequel, by embedding, projection and membership predicate we will mean not only the respective function and predicate symbols, but also their interpretations.

By $\Sigma^{\#\#}$ denote the specification $(\Sigma^{\#}, \Phi)$ in BasicMultiCasl, where Φ consists of what we call *defining sentences*, i.e., sentences over $\Sigma^{\#}$ which state that:

1. an embedding of s into s is the identity,
2. an embedding of s into s' is injective,
3. the composition of embeddings of s into s' and s' into s'' is equal to the embedding of s into s',
4. a projection of s' onto s is the least right-sided inverse of the embedding of s into s',
5. a membership predicate $in_{s,s'}$ holds on an element x of sort s iff the projection of s onto s' is defined on x,

6. if $f_{w_1,s_1} \sim_F f_{w_2,s_2}$, $w \leq^* w_1$, $w \leq^* w_2$, $s_1 \leq s$ and $s_2 \leq s$, then for any tuple x^* of elements of sorts w, embedding x^* in w_1, applying f_{w_1,s_1} and embedding the result in s gives the same result as embedding x^* in w_2, applying f_{w_2,s_2} and embedding the result in s,

7. if $b_{w_1} \sim_P b_{w_2}$, $w \leq^* w_1$ and $w \leq^* w_2$, then for any tuple x^* of elements of sorts w, embedding x^* in w_1 and checking, whether it is in b_{w_1}, gives the same result as embedding x^* in w_2 and checking, whether it is in b_{w_2}.

Observe that, if the signature Σ is finite, then the set Φ of defining sentences is finite as well; this will allow us to construct the specification language of finite BasicSubCasl and, later, finite SubCasl.

The model functor $\mathbf{Mod}_{\leq}$ is defined as the composition $((\_)^{\#\#})^{op}; \mathbf{Mod}$, where $\sigma^{\#\#} = \sigma^{\#}$ for all signature morphisms σ.

The sentence functor $Sen_{\leq}$ is defined as $(\_)^{\#}; Sen$.

The satisfaction relation $\models^{\leq}$ is defined by the rule $M \models^{\leq}_{\Sigma} \phi$ if $M \models_{\Sigma^{\#}} \phi$, for $M \in \mathbf{Mod}_{\leq}(\Sigma)$ and $\phi \in Sen_{\leq}(\Sigma)$.

The satisfaction condition obviously holds and so **SubCasl** is an institution.

A signature morphism $\sigma : \Sigma \to \Sigma'$ is considered an inclusion in $Sig_{\leq}$ if it is, after removing the subsort preorder from its domain and codomain, an inclusion in $\mathcal{MI}$. The above-defined category of inclusions for $Sig_{\leq}$ is denoted by $\mathcal{SI}$.

As before, the (enriched) specification language $\mathrm{Pres}_{\mathcal{SI}}(\mathbf{SubCasl})$ may now be formed. This is the specification language of *basic subsorted* Casl and it is denoted by BasicSubCasl.

Having defined two enriched specification languages of basic specifications, we now present a general method of adding structuring operations to an enriched specification language. It is modeled after [ST88a]; the form found here follows [Hof03].

Assume that an arbitrary enriched specification language $\mathcal{L} = (Sig, \mathcal{I}, \mathbf{Mod}, Spec, sig[\_], [\![\_]\!])$, which we call a *language of basic specifications* and a class $\mathcal{D}$ of signature morphisms in Sig, which we call the *class of reduction morphisms*, are given. The specification language $\mathrm{Str}_{\mathcal{D}}(\mathcal{L})$ of *structured specifications over $\mathcal{L}$ with $\mathcal{D}$-reductions* is the tuple $(Sig, \mathbf{Mod}, Spec', [\![\_]\!]')$, where, for any signature Σ, specifications in $Spec'(\Sigma)$ and their semantics $[\![\_]\!]'$ are inductively defined to be of the following forms:

- SP, where $SP \in Spec(\Sigma)$; then

$$[\![SP]\!]' = [\![SP]\!]$$

- SP **with** $\sigma : \Delta \to \Sigma$, where $SP \in \mathit{Spec}'(\Delta)$; then

$$[\![SP \textbf{ with } \sigma : \Delta \to \Sigma]\!]' = \{M \in \mathbf{Mod}(\Sigma) \mid M|_\sigma \in [\![SP]\!]'\}$$

- $SP_1 \cup SP_2$, where $SP_1, SP_2 \in \mathit{Spec}'(\Sigma)$; then

$$[\![SP_1 \cup SP_2]\!]' = [\![SP_1]\!]' \cap [\![SP_2]\!]'$$

- $SP|_{\sigma:\Sigma\to\Delta}$, where $SP \in \mathit{Spec}'(\Delta)$ and $\sigma \in \mathcal{D}$; then

$$[\![SP|_{\sigma:\Sigma\to\Delta}]\!]' = \{M|_\sigma \mid M \in [\![SP]\!]'\}$$

- **free** SP **along** $\sigma : \Delta \to \Sigma$, where $SP \in \mathit{Spec}'(\Sigma)$; $[\![\textbf{free } \phi \textbf{ along } \sigma]\!]'$ is then defined to be the class of all Σ-models $M \in [\![SP]\!]'$ such that, for any model $N \in [\![SP]\!]'$ and any morphism $h : M|_\sigma \to N|_\sigma$ in $\mathbf{Mod}(\Delta)$, there exists a unique morphism $g : M \to N$ in $\mathbf{Mod}(\Sigma)$ such that $g|_\sigma = h$.

The specification languages of multisorted and subsorted CASL now arise by applying the above method to the specification languages BASICMULTICASL and BASICSUBCASL, respectively. So, MULTICASL is simply the specification language $\mathrm{Str}_{\mathcal{MI}}$(BASICMULTICASL), while SUBCASL is the specification language $\mathrm{Str}_{\mathcal{SI}}$(BASICSUBCASL).

It is worth noting that, while adding derived signature morphisms to MULTICASL may be performed in a manner similar to what has been done for the language of partial algebras $\mathcal{PA}$ in Section 1, the same is not true for SUBCASL. Apart from interesting problems concerning translation of function symbols to terms in which subsort embeddings are used, the main difficulty stems from the fact that, in order for the reduct of models to function properly, one has to require signature morphisms to preserve the overloading relations. But, in the context of derived signature morphisms, this would mean that overloading relations on terms would have to be introduced, and it is unclear how this could be realized cleanly.

3 Amalgamation and Iso-reflection

While many of the formalisms introduced in this thesis do not require the considered specification language to satisfy any special properties, these properties will often be needed to prove certain theorems or the correctness of certain verification techniques. The most important of these properties, amalgamation and iso-reflection, are described in this section; the notion of having consistent signatures is also mentioned.

Amalgamation, iso-reflection and consistent signatures are all properties of a functor $F : Sig^{op} \to \mathbf{Class}$. A specification language is said to have amalgamation if its model functor does; a programming language is said to have amalgamation if its program functor does (even though programming languages also have a model functor!). The properties of iso-reflection and consistent signatures will be considered only for specification languages. Below, the name **Mod** is used instead of the more general F.

A functor $\mathbf{Mod} : Sig^{op} \to \mathbf{Class}$ is said to *have amalgamation*, if it takes colimits in *Sig* to limits in **Class**. It is said to have *iso-reflection* if it reflects isomorphisms, that is, if, for any signature morphism σ, if $\_|_\sigma$ is a bijection (isomorphism of classes), then σ is an isomorphism. It is said to have *consistent signatures* if it never produces the empty class, that is, if, for any signature Σ, $\mathbf{Mod}(\Sigma)$ is non-empty.

We now analyze these notions in more detail.

Let $\sigma = \{\sigma_i : \Sigma_i \to \Sigma\}_{i \in I}$ be a sink in *Sig* and let M be an *I-family of models*, i.e., a family $M = \{M_i\}_{i \in I}$ such that $M_i \in \mathbf{Mod}(\Sigma_i)$ for all $i \in I$. The model family M is said to be *σ-amalgamable* if there exists a unique Σ-model N such that $N|_{\sigma_i} = M_i$ for all $i \in I$. Then N is called the *σ-amalgamation* of M and denoted by $\bigoplus_{i \in I} \sigma_i(M_i)$ or, for an n-element and somehow ordered set $I = \{i_1, \ldots, i_n\}$, $\sigma_{i_1}(M_{i_1}) \oplus \cdots \oplus \sigma_{i_n}(M_{i_n})$. The importance of amalgamation stems from the fact that the whole family M may be uniquely represented by a single model N.

By a *diagram in Sig with shape I*, where I is a small category, a functor $D : I \to Sig$ is understood. An I-family of models $\{M_i\}_{i \in I}$ is said to be *consistent with D* if, for all morphisms $m : i \to j$ in I, $M_j|_m = M_i$ holds.

The following proposition explains what taking colimits in *Sig* to limits in **Class** means:

Proposition 1. *Assume $D : I \to Sig$ is a diagram and $\sigma = \{\sigma_i : D(i) \to \Sigma\}_{i \in I}$ is its colimit. Then the following conditions are equivalent:*

1. $\mathbf{Mod}(\sigma) = \{\_|_{\sigma_i}\}_{i \in I}$ *is a limit of* $D; \mathbf{Mod}$ *in* **Class**,
2. *any model family consistent with D is σ-amalgamable,*
3. *the map $\Phi = \lambda M \in \mathbf{Mod}(\Sigma) \cdot \{M|_{\sigma_i}\}_{i \in I}$ is a bijection onto the class of all model families consistent with D.*

Proof. (1) $\Rightarrow$ (2): Let M be a model family consistent with D. Consider the set $\{*\}$ and, for all $i \in I$, functions $f_i : \{*\} \to \mathbf{Mod}(D(i))$ defined by $f_i(*) = M_i$. Then $\{f_i : \{*\} \to \mathbf{Mod}(D(i))\}_{i \in I}$ is a cone over $D; \mathbf{Mod}$. Thus, there exists a unique function $f : \{*\} \to \mathbf{Mod}(\Sigma)$ such that $f_i(*) = f(*)|_{\sigma_i}$. This means that $f(*)$ is the σ-amalgamation of M.

(2) $\Rightarrow$ (3): Since σ is a cocone, $\mathbf{Mod}(\sigma)$ is a cone over $D;\mathbf{Mod}$ in **Class**. Thus Φ is into the class of all model families consistent with D. Let Ψ take any model family consistent with D to its σ-amalgamation. Then Φ and Ψ are inverses of each other: clearly, $\Psi;\Phi$ is the identity, and, by the uniqueness of the amalgamation, so is $\Phi;\Psi$.

(3) $\Rightarrow$ (1): Because $\mathbf{Mod}(\sigma)$ is a cone over $D;\mathbf{Mod}$, only the universal property has to be checked. Let $\{f_i : A \to \mathbf{Mod}(D(i))\}_{i\in I}$ be a cone over $D;\mathbf{Mod}$. For any $a \in A$, let $f(a) = \Phi^{-1}(\{f_i(a)\}_{i\in I})$. Then $f : A \to \mathbf{Mod}(\Sigma)$ factorizes the cone through $D;\mathbf{Mod}$, since $\Phi(f(a)) = \{f_i(a)\}_{i\in I}$. Moreover, because Φ is injective, any f with this property must be equal $\Phi^{-1}(\{f_i(a)\}_{i\in I})$. $\square$

When proving that a model functor has amalgamation the following well-know fact is often handy:

Proposition 2. *If Sig is cocomplete then* **Mod** *has amalgamation iff it takes coproducts in Sig to products in* **Class** *and coequalizers in Sig to equalizers in* **Class**. $\square$

It is worth noting that not taking coproducts to products and not taking coequalizers to equalizers are two quite distinct reasons for which a specification language may not have amalgamation. For instance, single-sorted logics generally do not have amalgamation, because the model functor does not take coproducts to products. On the other hand, subsorted CASL does not have amalgamation, as will shortly be seen, since it does not take coequalizers to equalizers.

Of course, the value of amalgamation drops if few colimits exist in *Sig*. It is highest if *Sig* is cocomplete. All the non-derived signature categories presented up to now, i.e., categories of algebraic signatures, multisorted CASL signatures and subsorted CASL signatures, are cocomplete (see [Mos98]). Also, the three model functors, for partial algebras, for multisorted CASL and for subsorted CASL, all take coproducts to products.

However, not all three take other colimits to limits. The model functors for partial algebras and multisorted CASL do. Unfortunately, the model functor of subsorted CASL does not:

Example 1. *Consider the signatures* $\Sigma = \{\,\mathbf{sorts}\ s, t;\,\}$ *and* $\Sigma' = \{\,\mathbf{sorts}\ s < t;\,\}$ *and a diagram* D

$$\begin{array}{ccc} A & \xrightarrow{\iota} & B \\ {\scriptstyle\iota}\downarrow & & \\ C & & \end{array}$$

where ι is the inclusion of Σ into Σ'. The colimit (pushout) σ of this diagram is simply Σ', with $\iota : D(A) \to \Sigma$ and $\mathrm{id}_{\Sigma'} : D(B) = D(C) \to \Sigma$.

But **Mod** *does not transform this pushout into a pullback. Consider a model family N consistent with D in which the sorts s and t have, in all three models, the carriers $\{*\}$ and $\{0,1\}$, respectively, and in which N_B interprets the embedding $s < t$ by the function $* \mapsto 0$ and N_C interprets it by the function $* \mapsto 1$. Clearly, this family is consistent with D, but is not σ-amalgamable, which, by Proposition 1, proves that subsorted* CASL *does not have amalgamation.*

As for the case of derived signatures, the signature category $AlgSig^{der}$ is not cocomplete. As a matter of fact, in many cases even cocones over diagrams do not exist:

Example 2. *Consider the diagram D:*

$$A \underset{f \mapsto x_2}{\overset{f \mapsto x_1}{\rightrightarrows}} B$$

where $D(A)$ is the signature $\{\textbf{sort } s;\ \textbf{op } f : s \times s \to s\}$ and $D(B)$ is the signature $\{\textbf{sort } s\}$. Here, the signature morphisms define f as a left or right projection. Clearly, these morphisms cannot possibly be (co)equalized.

Moreover, the model functor $\mathbf{PAlg}^{der}$ does not have amalgamation:

Example 3. *Consider the diagram D:*

$$\begin{tikzcd} A \arrow[r, "f \mapsto g(g(x_1))"] \arrow[d, "f \mapsto h(h(x_1))"'] & B \\ C & \end{tikzcd}$$

where $D(A)$, $D(B)$ and $D(C)$ are all signatures with one sort s and one unary function symbol, f, g and h, respectively. The pushout σ of this diagram consists of the signature $\{\textbf{sort } s;\ \textbf{op } l : s \to s\}$ and obvious cocone injections.

Consider the model family N consistent with D in which the carrier of s is $\{0,1\}$ in all three models, the interpretation of f in N_A is the identity, the interpretation of g in N_B is the identity as well, and the interpretation of h in N_C is the function $0 \mapsto 1, 1 \mapsto 0$. Clearly, the family N is compatible with D, but is not σ-amalgamable.

As for amalgamation for programming languages, the programming language $\mathcal{PPA}$ does have amalgamation. This may be shown as follows. Observe that a colimit in the category *AlgSig* is, at the same time, a colimit in the category $AlgSig^{der}$. Now, any model family consistent with a diagram D in *AlgSig* is simply a cocone over D. Consider the colimit of D in *AlgSig*. This colimit is also a colimit of D in $AlgSig^{der}$, as stated above. Hence, there exists a unique morphism η from the colimit to Σ_{nat} which factorizes the cocone through the colimit. Then η is the amalgamation of the family along the colimit. Thus, $\mathcal{PPA}$ has amalgamation.

Weaker forms of cocompleteness and amalgamation, applicable to categories of derived morphisms, can be used. For example, in [MHAH04], weak amalgamation is used, which means that colimits in the signature category are required to be mapped to weak limits in the category of all classes. In [Hof01], I have used a still weaker form of amalgamation, called *logical amalgamation*. This not only allows weak limits to arise, but it allows the "limit" to be defined using the sentences of the considered institution. This makes any analysis undecidable, but the upside is that signature categories with derived morphisms may be dealt with easily.

The second important property we are interested in is iso-reflection. In principle, iso-reflection means that changing a signature in a non-isomorphic way must change the class of models over that signature.

In fact, this property is connected to amalgamation:

Proposition 3. *If Sig is cocomplete and* **Mod** *has amalgamation, then the following conditions are equivalent:*

1. **Mod** *has iso-reflection,*

2. *for any cocone σ over a diagram $D : I \to Sig$, σ is a colimit of D iff* $\mathbf{Mod}(\sigma)$ *is a limit of* $D; \mathbf{Mod}$.

Proof. (1) $\Rightarrow$ (2): The "only if" part follows from the amalgamation property of **Mod**, without using iso-reflection. Assume $\mathbf{Mod}(\sigma)$ is a limit of $D; \mathbf{Mod}$. Let σ' be the colimit of D and η the unique universal morphism factorizing σ' through σ. Because of amalgamation, $\mathbf{Mod}(\sigma')$ is a limit of $D; \mathbf{Mod}$. Thus, $\_|_\eta$ is the unique function factorizing $\mathbf{Mod}(\sigma)$ through $\mathbf{Mod}(\sigma')$. Because $\mathbf{Mod}(\sigma)$ and $\mathbf{Mod}(\sigma')$ are both limits, $\_|_\eta$ must be an isomorphism in **Class**. Thus, by iso-reflection, η is an isomorphism in *Sig* and so σ must have been a colimit.

(2) $\Rightarrow$ (1): Let $\sigma_1 : \Sigma_1 \to \Sigma$ be a signature morphism such that $\_|_{\sigma_1}$ is a bijection. Then σ_1 is a cocone over a diagram D with one object labeled Σ_1. Also, $\mathbf{Mod}(\sigma_1) = \_|_{\sigma_1}$ is a limit of $D; \mathbf{Mod}$, since $\_|_{\sigma_1}$ is a bijection (an isomorphism in **Class**). Hence, σ_1 must be a colimit, that is, an isomorphism. □

Therefore, one may, as a slogan, say that while amalgamation guarantees soundness of representing consistent model families by single models, iso-reflection guarantees completeness of such a representation.

It is clear that partial algebras and multisorted CASL have iso-reflection. However, subsorted CASL fails here again:

Example 4. *Consider the signatures* $\Sigma = \{\mathbf{sorts}\ s < t;\ \mathbf{op}\ c : s\}$ *and* $\Sigma' = \{\mathbf{sorts}\ s < t;\ \mathbf{op}\ c : s, c : t;\}$ *and the inclusion* ι *of* Σ *in* Σ'. *Obviously,* ι *is not an isomorphism. But, because of the overloading relations,* $\_|_\iota$ *is a bijection, since in all* Σ'*-models the interpretation of* $c : t$ *depends only on the interpretations of* $c : s$ *and of the embedding* $s < t$.

For consistent signatures, the following trivial fact may be stated:

Proposition 4. *If Sig is cocomplete, then the following conditions are equivalent:*

1. **Mod** *has consistent signatures,*
2. *for any diagram* $D : I \to Sig$, *there exists a model family consistent with* D.

Proof. (1) $\Rightarrow$ (2): Consider a colimit $\sigma = \{\sigma_i : D(i) \to \Sigma\}_{i \in I}$ of D. Then $\mathbf{Mod}(\Sigma)$ contains at least one element, say, M. Then, because $\mathbf{Mod}(\sigma)$ is a cone over $D; \mathbf{Mod}$, the family $\{M|_{\sigma_i}\}_{i \in I}$ is consistent with D.

(2) $\Rightarrow$ (1): Obvious, since (1) is a specialization of (2) to one-element diagrams. □

As is easy to check, all the specification languages considered in this thesis have consistent signatures.

The notion of amalgamation as defined here has an extensive literature (see, e.g., [SMT$^+$05], as well as [Ore99], where it is called exactness, and [Tar99], where it is called composability of signatures); surprisingly, it is closely connected to amalgamation theory in algebra, e.g., semigroup theory, see Section 19. In the context of architectural specifications it has been discussed, e.g., in [SMT$^+$01b]. Iso-reflection has been discussed in [SMT$^+$05], where it is called definitional completeness. However, it is important to note that, in [SMT$^+$05], both amalgamation and iso-reflection are defined in a framework where the model functor has as codomain **CAT**. Consequently, amalgamation of homomorphisms between models is considered. Also, only isomorphisms in **CAT** are required to be reflected. Thus, the notions used there are, in general, distinct from those used in this thesis.

Chapter II

Architectural Specifications

4 Introduction

A program development process begins with a requirements specification SP and its ultimate target is producing a program P such that $P \in [\![SP]\!]$ (or equivalently $[\![P]\!] \in [\![SP]\!]$). Then $\mathcal{T} = SP$ is called a *task* and P an *implementation* of $\mathcal{T}$. An obvious idea for dealing with this task is to construct some simpler tasks $\mathcal{T}_1, \ldots, \mathcal{T}_n$ and, having completed them, put their implementations together to build an implementation of $\mathcal{T}$. Such a procedure may be formally expressed using *architectural specifications*.

An architectural specification ASP lists and names, under the keyword **units**, tasks $X_1 : \mathcal{T}_1, \ldots, X_n : \mathcal{T}_n$ and, under the keyword **result**, defines a procedure for putting their implementations together. The original task $\mathcal{T}$ is not stated explicitly in ASP. We will later introduce statements of the form:

$$ASP :: \mathcal{T}$$

with the intended meaning that, if all the tasks $\mathcal{T}_1, \ldots, \mathcal{T}_n$ within ASP are completed, then ASP defines a correct procedure of putting their implementations together to build an implementation of $\mathcal{T}$.

Two questions must be asked here: first, what are the tasks allowed to be, and, second, what mechanisms may be used to "put together" the tasks' implementations?

As for the first question, what is certain is that any task of the form "produce a program in $[\![SP]\!]$" must be allowed. However, if we want to decompose the task of producing a program into subtasks, then it is not always possible to decompose it into tasks of producing smaller programs. For instance, suppose we want to implement an auxiliary function and then, using that function, our main function. Then the latter task cannot be expressed in the form "produce a program in $[\![SP]\!]$", but rather in the form "assuming that you have a program in $[\![SP_{aux}]\!]$, produce a program in $[\![SP]\!]$". Thus, we end up with at least two kinds of tasks: one takes the form SP and requires producing a so-called *non-parametric unit*, i.e., a program in $[\![SP]\!]$, and the other takes the form $SP \to SP'$ and requires producing a so-called *parametric unit*, i.e., a function from programs in $[\![SP]\!]$ to programs in $[\![SP']\!]$.

Actually, an arbitrary implementation of a task of the form $SP \to SP'$ will not serve our purposes. Typically, in SP there will be some type, say, `elem`, and in SP' there will be the same type and some new types and functions, for example types `elem, stack` and functions `empty : stack`, `push : stack * elem -> stack` and `pop : stack -> elem`. Now, if arbitrary implementations are allowed, then it might happen that the type `elem` in the implementation of SP' has nothing to do with the type `elem` in the implementation of SP. This is not what we want, since we need stacks over the *given* type `elem`, not over *any* type called `elem`; thus, a *sharing condition* has to be imposed to link the parameter and the result of an implementation of this task. For these reasons, we require that, first, a task of the form $SP \to SP'$ is a legal task only if the signature Σ of SP is a subsignature of the signature Δ of SP' and, second, an implementation of such a task must be *persistent*, i.e., take any program P in $[\![SP]\!]$ to a program P' in $[\![SP']\!]$ such that P' reduced along the inclusion $\iota_{\Sigma\subseteq\Delta}$ is equal to P.

As for the second question, we will be using a minimal set of four operations for putting units together; all of these operations return a non-parametric unit as a result; using them, *terms* may be built, which are denoted by T, T', etc. Three of these operations are fairly simple:

- "X", returning the non-parametric unit named X (*variable*),
- "$X(T)$", applying the parametric unit named X to the non-parametric unit defined by the term T (*application*),
- "$T|_{\sigma:\Sigma\to\Delta}$", returning the σ-reduct of the non-parametric unit over Δ defined by the term T (*reduction*).

A fourth operation, the only one which really links multiple entities into one bigger whole, is *amalgamation*, written "$\bigcup_{i\in I}\sigma_i(T_i)$", where $\sigma = \{\sigma_i : \Sigma_i \to \Delta\}_{i\in I}$ is a sink in *Sig* and the T_i are terms over Σ_i. The intended meaning is that this construct returns the σ-amalgamation of the I-family of non-parametric units defined by the T_i, $i \in I$.

We describe now a special, practically important case of amalgamation. For any signatures Σ_1 and Σ_2, one may consider their least upper bound with respect to the inclusion order. If the least upper bound exists, Σ_1 and Σ_2 are

$$
\begin{aligned}
ASP &::= \textbf{units } UDS \textbf{ result } UE\\
UDS &::= \epsilon \mid\\
&\quad\; UDS\ X : USP\\
USP &::= SP \mid\\
&\quad\; SP \to SP\\
UE &::= T \mid\\
&\quad\; \lambda\, X : SP \bullet T\\
T &::= X \mid\\
&\quad\; X(T) \mid\\
&\quad\; T|_{\sigma:\Sigma\to\Delta} \mid\\
&\quad\; \textstyle\bigcup_{i\in I} \sigma_i(T_i)
\end{aligned}
$$

Figure 1: Formal syntax of architectural specifications.

called *compatible*, the least upper bound is denoted by $\Sigma_1 \cup \Sigma_2$ and called the *sum* of both signatures. The amalgamation construct $\bigcup_{i\in\{1,2\}} \iota_{\Sigma_i \subseteq \Sigma_1\cup\Sigma_2}(T_i)$ will be often used and written simply as $T_1 + T_2$. If the unit described by it exists for programs P_1 and P_2, then they are called *amalgamable*, without mentioning the implicit sink σ, and the amalgamation itself is written as $P_1 \oplus P_2$. The same notational shortcut may be applied to models.

For the languages $\mathcal{PA}$ and $\mathcal{PPA}$ for partial algebras, the operation "$\cup$" simply puts together elements from both signatures. This may be impossible; for example, the algebraic signatures Σ_1 = **sorts** s; **ops** $f : s \to s$ and Σ_2 = **sorts** s; **ops** $f : s$ do not have a sum, because they disagree on the arity of f. But if we consider Σ_3 = **sorts** s; **ops** $f : s \to s,\ g : s$, then the sum $\Sigma = \Sigma_1 \cup \Sigma_3$ does exist. Also, not all pairs M_1, M_3 of Σ_1- and Σ_3-models are amalgamable; in fact, they are amalgamable iff the carriers of s and the interpretations of f in both are the same, i.e., iff $M_1^S(s) = M_3^S(s)$ and $M_1^F(f) = M_3^F(f)$. An analogical remark applies to pairs P_1 and P_3 of Σ_1- and Σ_3-programs. Note that to guarantee amalgamability of P_1 and P_3 it is not sufficient that $[\![P_1]\!]_{\Sigma_1}$ and $[\![P_3]\!]_{\Sigma_3}$ are amalgamable; the actual "type definition" used to define the sort s and the actual "code" used to define the function symbol $f : s \to s$ must also be the same.

Expressions describe the result unit of an architectural specification; the unit may either be non-parametric, and then the expression is simply a term, or it may be parametric, and then the expression is a λ-abstraction of a term. An architectural specification is called *parametric* or *non-parametric*, depending on whether the expression is parametric or non-parametric.

The formal syntax of architectural specifications is summarized in Figure 1.

5 Semantics

Before the semantics of architectural specifications is given, one issue needs mentioning. In the previous section, architectural specifications have been described as implicitly interpreted in some specification language of the form $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$; that is, specifications used in the architectural specification denoted classes of programs and all architectural constructs operated on programs. In CASL, and in architectural specifications as they will be from now on presented in this thesis, the semantics is defined solely on the level of models, that is, in an arbitrary specification language $\mathcal{L}$. Thus, everywhere above, the word "program" should actually be replaced by "model", and the symbol "$[\![\_]\!]$" by "$[\![\_]\!]$".

There is a number of reasons for such a choice. First, this frees us from coming up with a functor *Prog*, the definition of which would, in many cases, be very difficult to provide. Second, it allows us to choose a programming language $\mathcal{P}$ later, after having designed the architecture of the system; though, of course, this will not be *any* language: it must be compatible with $\mathcal{L}$, so that the specification language $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ may later be formed. For further considerations on the relation between model-based architectural specifications and obtaining the final result in the form of a program, see Sections 9–11.

After this introduction, the definition of the semantics of architectural specifications follows.

A *unit signature* $\mathrm{U}\Sigma$ is either a (regular) signature Σ or a *parametric unit signature* $\Sigma \to \Delta$, where Σ is a subsignature of Δ. The class of all units over $\mathrm{U}\Sigma$ is denoted by $\mathbf{Unit}(\mathrm{U}\Sigma)$, that is, $\mathbf{Unit}(\Sigma) = \mathbf{Mod}(\Sigma)$ and $\mathbf{Unit}(\Sigma \to \Delta)$ is the set of all partial functions $F : \mathbf{Mod}(\Sigma) \to \mathbf{Mod}(\Delta)$ which are *persistent*, that is, $F(M)|_\Sigma = M$ for all $M \in \mathrm{dom}(F)$.

A *unit specification USP* is either a (regular) specification *SP* or a *parametric unit specification* $SP \to SP'$, where $sig[SP]$ is a subsignature of $sig[SP']$. The unit signature of the unit specification *USP* is denoted $sig[USP]$, that is, if $USP = SP$, then $sig[USP] = sig[SP]$, and if $USP = SP \to SP'$, then $sig[USP] = sig[SP] \to sig[SP']$. By $[\![USP]\!]$ the class of all units satisfying *USP* is denoted, i.e., if $USP = SP$, then $[\![USP]\!] = [\![SP]\!]$, and if $USP = SP \to SP'$, then $[\![USP]\!]$ consists of all units $F \in \mathbf{Unit}(sig[USP])$ such that, for all $M \in [\![SP]\!]$, $M \in \mathrm{dom}(F)$ and $F(M) \in [\![SP']\!]$. A unit specification *USP* is said to be *inconsistent* if $[\![USP]\!]$ is empty.

Note that the term "parametric unit signature" should be parsed as "signature of parametric unit" and the term "parametric unit specification" as "specification of parametric unit" (it is not a parametric specification!).

A *static context* C_s assigns unit signatures to a finite number of unit names. An *environment ϱ fitting C_s* assigns an element of $\mathbf{Unit}(C_s(X))$ to any unit name X in C_s. A unit name X in C_s is called *parametric* if $C_s(X)$ is a parametric signature; otherwise it is called *non-parametric*. A *context C fitting C_s* is any class of environments fitting C_s. A *model evaluator MEv from C to Σ* is a (total) function from C to $\mathbf{Mod}(\Sigma)$. Similarily, a *unit evaluator UEv from C to* $\mathrm{U}\Sigma$ is a (total) function from C to $\mathbf{Unit}(\mathrm{U}\Sigma)$.

For architectural specifications, a static and a model semantics are provided. In the static semantics, the role of which is to check that the architectural specification *ASP* is well-formed, objects such as static contexts and signatures are computed. The semantic function for this semantics is denoted by $[\_]$, possibly with parameters. The static semantics $[ASP]$ of an architectural specification is a pair $(C_s, \mathrm{U}\Sigma)$, where C_s is a static context describing the required units and $\mathrm{U}\Sigma$ is the unit signature of the result unit. In the model semantics, the run of which assumes a previous successful run of the static semantics, objects such as contexts, units and unit evaluators are computed. Its semantic function is denoted by $[\![\_]\!]$. The model semantics $[\![ASP]\!]$ of an architectural specification is a unit evaluator from a context fitting C_s to $\mathrm{U}\Sigma$, where $[ASP] = (C_s, \mathrm{U}\Sigma)$; this evaluator maps any appropriate environment of required units to the newly built result unit.

In the semantic equations it is assumed that any undefined value on the right hand side propagates, producing an undefined value on the left hand side as well. This goes for values in λ-expressions as well: λ-expressions denote total functions. Equations for any syntactic category are preceded by a header describing the types of the semantic functions, that is, the parameters they may be given and the result they will then produce. In particular, the header defines what is meant by saying that running the model semantics requires a previous successful run of the static semantics. The headers are thus both an assumption and a form of proposition provable by induction. For example, the header for expressions says that $[\![UE]\!]_\varrho$ may only be computed if $[UE]_{C_s}$ is defined and if ϱ is an environment fitting C_s; moreover, in this case $[\![UE]\!]_\varrho$, if defined, is a unit over the unit signature $[UE]_{C_s}$.

$$\boxed{[ASP] = (C_s, \mathrm{U}\Sigma) \qquad\qquad [\![ASP]\!] = UEv}$$

The context $\mathrm{dom}(UEv)$ fits C_s and UEv is a unit evaluator to $\mathrm{U}\Sigma$.

$$\begin{array}{rcl} [\mathbf{units}\ UDS\ \mathbf{result}\ UE] & = & ([UDS], [UE]_{[UDS]}) \\ [\![\mathbf{units}\ UDS\ \mathbf{result}\ UE]\!] & = & \lambda\varrho \in [\![UDS]\!] \cdot [\![UE]\!]_\varrho \end{array}$$

$$\boxed{[UDS] = C_s \qquad\qquad [\![UDS]\!] = C}$$

C_s is a static context and the context C fits C_s.

$$\begin{array}{lcl}
[\epsilon] & = & \emptyset \\
[UDS\ X : USP] & = & [UDS][sig[USP]/X] \quad \text{if } X \notin \mathrm{dom}([UDS]) \\
[\![\epsilon]\!] & = & \{\emptyset\} \\
[\![UDS\ X : USP]\!] & = & [\![UDS]\!][[\![USP]\!]/X]
\end{array}$$

$$\boxed{[UE]_{C_s} = \mathrm{U}\Sigma \qquad\qquad [\![UE]\!]_\varrho = U}$$

The environment ϱ should fit C_s. Then U is a unit over $\mathrm{U}\Sigma$.

$$\begin{array}{lcll}
[\lambda A : SP \bullet T]_{C_s} & = & sig[SP] \to [T]_{C_s[sig[SP]/A]} & \text{if } A \notin \mathrm{dom}(C_s) \text{ and } sig[SP] \\
 & & & \text{is a subsignature} \\
 & & & \text{of } [T]_{C_s[sig[SP]/A]} \\
[\![\lambda A : SP \bullet T]\!]_\varrho & = & \lambda M \in [\![SP]\!] \cdot [\![T]\!]_{\varrho[M/A]} & \text{if, for all } M \in [\![SP]\!], \\
 & & & [\![T]\!]_{\varrho[M/A]}|_{sig[SP]} = M
\end{array}$$

$$\boxed{[T]_{C_s} = \Sigma \qquad\qquad [\![T]\!]_\varrho = M}$$

The environment ϱ should fit C_s. Then M is a Σ-model.

$$\begin{array}{lcll}
[A]_{C_s} & = & C_s(A) & \text{if } A \in \mathrm{dom}(C_s) \text{ and } C_s(A) \\
 & & & \text{is a non-parametric signature} \\
[A(T)]_{C_s} & = & \Delta & \text{if } C_s(A) = [T]_{C_s} \to \Delta \\
[\bigcup_{i\in I} \sigma_i(T_i)]_{C_s} & = & \Delta & \text{if } \sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i\in I} \text{ and, for all} \\
 & & & i \in I,\ \Delta_i = [T_i]_{C_s} \\
[T|_{\sigma:\Sigma\to\Delta}]_{C_s} & = & \Sigma & \text{if } [T]_{C_s} = \Delta \\
[\![A]\!]_\varrho & = & \varrho(A) & \\
[\![A(T)]\!]_\varrho & = & \varrho(A)([\![T]\!]_\varrho) & \text{if } [\![T]\!]_\varrho \in \mathrm{dom}(\varrho(A)) \\
[\![\bigcup_{i\in I} \sigma_i(T_i)]\!]_\varrho & = & \bigoplus_{i\in I} \sigma_i([\![T_i]\!]_\varrho) & \text{if the } [\![T_i]\!]_\varrho \text{ are } \sigma\text{-amalgamable,} \\
 & & & \text{where } \sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i\in I} \\
[\![T|_{\sigma:\Sigma\to\Delta}]\!]_\varrho & = & ([\![T]\!]_\varrho)|_\sigma &
\end{array}$$

Note that, strictly speaking, the reduction $\_|_{sig[SP]}$ in the side condition of the model semantics equation for λ-expressions is ambiguous, since the signature Σ of the term T cannot be inferred form the data at hand. Thus, formally, it is necessary to either have, as an additional parameter of the semantic function, a static context C_s, which is the approach taken in the official semantics of CASL, or to require that this signature is expressed in the syntax. We prefer to take neither approach and simply assume that Σ is given implicitly.

Architectural specifications, as defined in CASL, differ somewhat from what has been presented above. We now describe some rather minor differences, the more important ones being described in Sections 7 and 8.

Reduction and amalgamation constructs in CASL architectural specifications are less general than in our presentation. In CASL, a reduction may be performed only along an inclusion; moreover, not all inclusions are allowed in this context (the inclusion is required to reflect the subsort preorder). Also, only two forms of amalgamation are allowed: one is the already introduced form $T_1 + T_2$, the other is the form T_1 **with** $\sigma_1 : \Delta_1 \to \Delta$, which is equivalent to the general amalgamation $\bigcup_{i \in \{1\}} \sigma_i(T_i)$ and in which certain additional surjectivity constraints are enforced on σ_1.

On the other hand, CASL architectural specification provide a general form of reduction in the context of parametric unit application. Namely, a parametric unit F over the unit signature $\Sigma \to \Delta$, may be applied to a non-parametric unit M over Σ' via a *fitting morphism* $\sigma : \Sigma \to \Sigma'$. The result of this application is obtained by computing the pushout $(\sigma' : \Delta \to \Delta', \tau : \Sigma' \to \Delta')$ of the span of morphisms $(\iota_{\Sigma \subseteq \Delta}, \sigma)$ and then taking the amalgamation $\sigma'(F(M|_\sigma)) \oplus \tau(M)$.

In architectural specifications as defined in CASL, parametric units are more general in that they may take multiple parameters. Still, it is required that the parameters' signatures are compatible and that, for any tuple $(M_1, \dots, M_n)$ in the domain of a parametric unit, the models $M_1, \dots, M_n$ are amalgamable. These requirements mean that, except for syntactical differences, one may just as well change all such units into one-parameter units, amalgamating the actual arguments into one model $M_1 \oplus \dots \oplus M_n$ (to conduct this change, however, it is necessary to have reductions along arbitrary inclusions).

6 Correctness and Refinement

An architectural specification is *statically correct* if it has a denotation under the static semantics; it is *correct* if it is statically correct and it has a denotation under the model semantics. In this case, the architectural specification describes a correct decomposition of the task of implementing the result unit into tasks of implementing the declared units.

Of course, if architectural specifications are to be used to express actual implementation steps, then there must be some way of stating not only that ASP describes a certain procedure of constructing some unit, but also that this unit will satisfy a given unit specification. As announced in Section 4, to this end we introduce statements of the form:

$$ASP :: USP$$

The above statement is one of the four *refinement statements* that will be used in the program development process. The idea is that a program development process starts with a requirements specification SP, which is then *refined* to some simpler tasks, these are further refined, and so on, with final tasks being either library modules, or tasks which are self-implementing (this is explained below). In this way, a direct acyclic graph (dag) representing the program development process is built. The nodes in this graph are tasks $\mathcal{T}$ and the edges represent refinements.

A refinement statement takes the form:

$$\mathcal{IT} :: \mathcal{OT}$$

where $\mathcal{OT}$ is the *original task*, i.e., the task being refined, and $\mathcal{IT}$ is the *implementing task*. The original task may either be a unit specification USP, or a unit declared in an architectural specification, written $ASP.X$, where X is a unit declared in ASP. An implementing task may be either a unit specification USP, or an architectural specification ASP.

The semantics of refinement statements is defined as follows.

The *signature of a task* is denoted by $\Sigma[\_]$. We have:

- $\Sigma[USP] = sig[USP]$,
- $\Sigma[ASP] = \mathrm{U}\Sigma$, where $[ASP] = (C_s, \mathrm{U}\Sigma)$,
- $\Sigma[ASP.X] = C_s(X)$, where $[ASP] = (C_s, \mathrm{U}\Sigma)$, if $X \in \mathrm{dom}(C_s)$.

A task $\mathcal{T}$ is *statically correct* if $\Sigma[\mathcal{T}]$ is defined. A refinement statement $\mathcal{IT} :: \mathcal{OT}$ is *statically correct* if $\Sigma[\mathcal{IT}]$ and $\Sigma[\mathcal{OT}]$ are defined and coincide. If the signature of both tasks is non-parametric, then the whole statement is called *non-parametric*; otherwise it is called *parametric*.

The *unit class* of a statically correct task is denoted by $\mathcal{M}[\_]$. We have:

- $\mathcal{M}[USP] = [\![USP]\!]$,
- $\mathcal{M}[ASP] = \{[\![ASP]\!](\varrho) \mid \varrho \in \mathrm{dom}([\![ASP]\!])\}$,

- $\mathcal{M}[ASP.X] = [\![USP]\!]$, if ASP is correct and contains the declaration $X : USP$.

A task $\mathcal{T}$ is *correct* if it is statically correct and $\mathcal{M}[\mathcal{T}]$ is defined. A refinement statement $\mathcal{IT} :: \mathcal{OT}$ is *component-correct* if it is statically correct and $\mathcal{IT}$ and $\mathcal{OT}$ are correct.

A refinement statement $\mathcal{IT} :: \mathcal{OT}$ is *correct*, or *holds*, if it is component-correct and $\mathcal{M}[\mathcal{IT}] \subseteq \mathcal{M}[\mathcal{OT}]$.

The fact that a refinement statement is correct, expressed simply as:

$$\mathcal{IT} :: \mathcal{OT}$$

has to be proved in some proof calculus. Calculi for proving such statements are developed in Chapter III; the derivability of a statement using such a calculus is expressed as:

$$\vdash \mathcal{IT} :: \mathcal{OT}$$

It is then shown that, under certain assumptions, the latter implies the former (soundness) and the former implies the latter (completeness).

A refinement statement describes one step in a program development process. The whole development process may be described by refinement dags, that is, directed acyclic graphs in which edges represent single steps in the development process.

A *refinement dag* is a connected directed acyclic graph r together with a root, written $root(r)$. Nodes in r are labeled by unit specifications and statically correct architectural specifications; the label of a node n is denoted by $L(n)$. Some edges in r are unlabeled and some are labeled by unit names. The existence of an edge from n_1 to n_2 is denoted by $n_1 \rightarrow n_2$ or $n_1 \xrightarrow{X} n_2$, depending on whether the edge is unlabeled or labeled; we then write $n_1 \rightarrow r'$ and $n_1 \xrightarrow{X} r'$, which means that r' is the part of r reachable from n_2. A refinement dag r must also satisfy the following properties:

- if a node n_1 is labeled by USP, then it has at most one outgoing edge which is unlabeled; moreover, if $n_1 \rightarrow n_2$ is an edge in r, then $L(n_2) :: L(n_1)$ is statically correct,

- if a node n_1 is labeled by ASP, then all its outgoing edges must be labeled by names of units declared in ASP; the same label may not occur on two edges; moreover, if $n_1 \xrightarrow{X} n_2$ is an edge in r, then $L(n_2) :: L(n_1).X$ is statically correct.

A refinement dag r is *component-correct* if all the labels of its nodes are correct tasks, and it is *correct*, or *holds*, if it is component-correct and:

- if $n_1 \to n_2$ is an edge in r, then $L(n_2) :: L(n_1)$ is correct,
- if $n_1 \xrightarrow{X} n_2$ is an edge in r, then $L(n_2) :: L(n_1).X$ is correct.

A node is called *implemented*, if it has the maximum possible number of outgoing edges. This means that all the subtasks needed to implement it are also present in the dag. An implemented node with no outgoing edges can also occur: it must be labeled by an architectural specification without any declared units. Such a specification may even define a non-parametric unit, if it uses the amalgamation construct over an empty sink (this will usually generate the only model over the initial signature; for instance, a model of the built-in datatypes of the language). Some nodes or parts of nodes are not intended to be implemented; rather, they contain specifications of library modules, which are assumed to be given in the development process.

The intention is, of course, that, if a refinement dag with root labeled USP is correct, then it represents a function taking models of all the unimplemented nodes and producing a model of USP. This may be formalized via a flattening operation, which we denote by $\mathcal{A}(r)$ and which, for any statically correct refinement dag r, produces an architectural specification representing the same development process as r, but in a flattened manner.

First, a definition of substitution on terms and expressions is needed. Substitution is written as $UE[UE'/X]$ and defined inductively as follows:

$$
\begin{array}{lcll}
(\lambda Y : SP \bullet T_\lambda)[UE/X] & = & \lambda Y : SP \bullet T_\lambda[UE/X] & \text{if } X \neq Y \text{ and } Y \\
 & & & \text{does not appear} \\
 & & & \text{in } UE \\
X[T/X] & = & T & \\
Y[UE/X] & = & Y & \text{if } X \neq Y \\
X(T_0)[\lambda Z : SP \bullet T_\lambda/X] & = & T_\lambda[T_0[\lambda Z : SP \bullet T_\lambda/X]/Z] & \\
Y(T_0)[UE/X] & = & Y(T_0[UE/X]) & \text{if } X \neq Y \\
\bigcup_{i\in I} \sigma_i(T_i)[UE/X] & = & \bigcup_{i\in I} \sigma_i(T_i[UE/X]) & \\
T_0|_{\sigma:\Sigma\to\Delta}[UE/X] & = & (T_0[UE/X])|_{\sigma:\Sigma\to\Delta} &
\end{array}
$$

Second, if r is a dag and there is an edge e from n_1 to n_2 in r, then by *r with the edge e removed* we understand the dag r without e and, if there is no other edge in r ending in n_2, without the node n_2 and with all edges starting in n_2 removed; note that this is an inductive definition. In other words, removing an edge from a dag also removes any edges and nodes below it, unless these edges or nodes are pointed to from other parts of the dag.

In the definition of $\mathcal{A}(r)$, we assume that all unit names used in architectural specifications in r, including λ-bound variables, are distinct; if this is not the case, one may first perform an appropriate renaming (α-conversion) and then apply the flattening. The flattening is defined inductively as follows:

- if r contains only the root, which is labeled USP, then $\mathcal{A}(r)$ is (here, A and F are newly generated names):
  - **units** $A : SP$ **result** A, if $USP = SP$,
  - **units** $F : SP \to SP'$ **result** $\lambda A : SP \bullet F(A)$, if $USP = SP \to SP'$.
- if r contains only the root, which is labeled by ASP, then $\mathcal{A}(r) = ASP$,
- if r contains an edge $root(r) \to r'$, then $\mathcal{A}(r) = \mathcal{A}(r')$,
- if r contains an edge $root(r) \xrightarrow{X} r'$ and X is the last declaration in the architectural specification $L(root(r))$ for which an edge exists, then proceed as follows. If

  $L(root(r)) =$ **units** $UDS\ X : USP\ Y_1 : USP_1 \ldots Y_n : USP_n$ **result** UE

  and

  $$\mathcal{A}(r') = \textbf{units}\ UDS'\ \textbf{result}\ UE'$$

  then produce the refinement dag r'', which is equal r except that

  - the edge $root(r) \xrightarrow{X} r'$ is removed, and
  - $root(r)$ is labeled by

    units $UDS\ UDS'\ Y_1 : USP_1 \ldots Y_n : USP_n$ **result** $UE[UE'/X]$

  Now set $\mathcal{A}(r) = \mathcal{A}(r'')$.

Observe that this is a correct inductive definition, since in each non-final step the number of edges in the refinement dag concerned drops at least by one. Also, the static correctness of the architectural specification placed at the root of r'' and containing the substitution $UE[UE'/X]$ is guaranteed by the distinctness and freshness of all variables and by the fact that r is a refinement dag; in the process of defining $\mathcal{A}(\_)$, the latter property is preserved.

That refinement dags have the expected properties may then be expressed as follows:

Proposition 5. *If a refinement dag r is correct and*

$$L(root(r)) :: \mathcal{OT}$$

then

$$\mathcal{A}(r) :: \mathcal{OT}$$

Proof. First, a lemma concerning substitution is needed. A proof is given in Appendix 1.

Lemma 6. *Assume that $[UE']_{C_s}$ is defined and that Z and unit names from C_s are not λ-bound in UE. If $[UE]_{C_s[[UE']_{C_s}/Z]}$ is defined, then it is equal $[UE[UE'/Z]]_{C_s}$.*

Assume that the above expressions are defined and that, additionally, ϱ fits C_s and $[\![UE']\!]_\varrho$ is defined. If $[\![UE]\!]_{\varrho[[\![UE']\!]_\varrho/Z]}$ is defined, then it is equal $[\![UE[UE'/Z]]\!]_\varrho$. □

Next, we consider the effects of flattening one edge in a node labeled by an architectural specification. Again, a proof, which uses Lemma 6, is given in Appendix 1.

Lemma 7. *Assume that $ASP =$ **units** $UDS\ X : USP\ Y_1 : USP_1 \ldots Y_n : USP_n$ **result** UE is correct, $ASP' =$ **units** UDS' **result** UE' is correct and that $ASP' :: ASP.X$ holds.*

*Let $ASP'' =$ **units** $UDS\ UDS'\ Y_1 : USP_1 \ldots Y_n : USP_n$ **result** $UE[UE'/X]$. Then ASP'' is correct and, moreover:*

1. $\pi_2([ASP'']) = \pi_2([ASP])$,
2. *for any $\varrho' \in \mathrm{dom}([\![ASP'']\!])$, there exists an environment ϱ such that $[\![ASP'']\!](\varrho') = [\![ASP]\!](\varrho)$.* □

The proof of Proposition 5 is now by induction over the number of edges in the refinement dag r.

If r consists of a single node, and so has no edges, the proposition clearly holds. If r contains an unlabeled edge $root(r) \to r'$, then $\mathcal{A}(r) = \mathcal{A}(r')$, for which, by the induction hypothesis, $\mathcal{A}(r') :: \mathcal{OT}$ holds, because, the refinement dag r being correct, $L(root(r')) :: L(root(r)) :: \mathcal{OT}$.

The last case is r containing an edge $root(r) \xrightarrow{X} r'$, where X is the last declaration in $ASP = L(root(r'))$, for which such an edge exists. By the induction hypothesis, the architectural specification $ASP' = \mathcal{A}(r')$ satisfies $ASP' :: ASP.X$, since, the refinement dag r being correct, we have $L(root(r')) :: ASP.X$. Let $ASP'' = L(root(r''))$. Then, by Lemma 7, the architectural specification ASP'' is correct and $\pi_2([ASP'']) = \pi_2([ASP])$;

this implies that the statement $ASP'' :: \mathcal{OT}$ is component-correct, since $ASP :: \mathcal{OT}$ was. By the same lemma, for any $\varrho' \in \mathrm{dom}([\![\mathcal{A}(r)]\!])$, there exists an environment ϱ such that $[\![ASP'']\!](\varrho') = [\![ASP]\!](\varrho)$. Because $ASP :: \mathcal{OT}$, this implies that $ASP'' :: \mathcal{OT}$ holds. By the induction hypothesis, we then have $\mathcal{A}(r) :: \mathcal{OT}$, since $\mathcal{A}(r) = \mathcal{A}(r'') = ASP''$. □

Proposition 5 is of special value if $\mathcal{OT} = SP$ and r contains only implemented nodes, since then r, via the semantics of architectural specifications, defines a model $[\![\mathcal{A}(r)]\!](\emptyset) \in [\![SP]\!]$, where $\emptyset$ is the empty environment.

7 Imports

A major construct that has been omitted up to now, but which exists in CASL architectural specifications, are *imports*. In CASL, a declaration may take not only the form $X : USP$, but also $X : SP$ **given**$_\Sigma$ T. The idea is that this defines an implicit parametric unit $F : SP_I \to SP$, where $[\![SP_I]\!]$ is the class of all Σ-models which may arise as the result of T, and then sets $X = F(T)$. Methodological arguments in favor of imports may be found in Section 12. Adding imports requires just adding one production to the syntax, as shown in Figure 2, and two equations to the semantics. Note that in CASL, as a logical consequence of having multiple parameters in units, parametric units are also allowed to have imports. However, since the parameters of a parametric unit with imports are required to be amalgamable with the imports, such units are rather difficult to use.

$UDS\ X : SP\ \mathbf{given}_\Sigma\ T$

Figure 2: Architectural specifications syntax for imports.

The new semantic equations are as follows:

$$[UDS\ X : SP\ \mathbf{given}_\Sigma\ T] \quad = \quad [UDS][sig[SP]/X] \quad \text{if } X \notin \mathrm{dom}([UDS]),\ \Sigma \text{ is a subsignature of } sig[SP] \text{ and } [T]_{[UDS]} = \Sigma$$

$$[\![UDS\ X : SP\ \mathbf{given}_\Sigma\ T]\!] \quad = \quad \begin{cases} \emptyset & \text{if, for some } \varrho \in [\![UDS]\!], \text{ there exists no } M \in [\![SP]\!] \text{ with } M|_\Sigma = [\![T]\!]_\varrho \\ \{\varrho[M/X] \mid \varrho \in [\![UDS]\!],\ M \in [\![SP]\!],\ M|_\Sigma = [\![T]\!]_\varrho\} & \text{otherwise} \end{cases}$$

Note that, in the model semantics equation, it is assumed that if, for some $\varrho \in [\![UDS]\!]$, $[\![T]\!]_\varrho$ is undefined, then so is the left-hand side.

In the model semantics equation for an *import declaration*, that is, for a declaration of a unit with imports, the context returned may be the empty set, if the implicit declaration of a parametric unit would have been inconsistent, thus also generating the empty context. In this case the import declaration is said to be *inconsistent*. This is necessary in order to ensure the equivalence between an import declaration and the implicit declaration of a parametric unit. As will be seen in Section 9, inconsistencies in parametric unit declarations and in imports cause considerable trouble, since they make an architectural specification correct "for free". At the same time, this correctness does not make much methodological sense, since it only states that the development step, being impossible to implement, will never give us incorrect results.

One problem with imports is that the specification SP_I of the argument of the implicit declaration cannot, in general, be computed. The model class $\mathcal{M} = \{[\![T]\!]_\varrho \mid \varrho \in [\![UDS]\!]\}$ is the intended domain of the implicit parametric unit, which means that SP_I should have the property that $[\![SP_I]\!] = \mathcal{M}$. Unfortunately, the class $\mathcal{M}$ need not be of the form $[\![SP]\!]$ for any specification SP. This is especially true if parametric units are allowed to be applied many times in the same architectural specification, see, e.g., the architectural specification in Figure 4, Section 8. Because a simple logical description of $\mathcal{M}$ does not always exist, the idea of a syntactical translation of architectural specifications with imports to architectural specifications without them has to be abandoned.

Another problem is refinement of architectural specifications with imports. Because, as mentioned above, it is not possible to rephrase the import declaration as a parametric declaration plus an application, refinement statements and refinement dags introduced in Section 6 have to be extended, so that imports are also taken into account.

Thus, an original task $\mathcal{OT}$ is, additionally, allowed to take the form $ASP.X$, for ASP of the form **units** $UDS\ X : SP$ $\textbf{given}_\Sigma\ T\ UDS'$ **result** UE. Such a task defines:

- the signature $\Sigma[ASP.X] = \Sigma \to sig[SP]$, if ASP is statically correct,
- the unit class $\mathcal{M}[ASP.X] =$

$$\{F \in \mathbf{Unit}(\Sigma \to sig[SP]) \mid \text{for all } M \in \mathcal{M},\ M \in \mathrm{dom}(F) \text{ and } F(M) \in [\![SP]\!]\}$$

  if ASP is correct and $\mathcal{M} = \{[\![T]\!]_\varrho \mid \varrho \in [\![UDS]\!]\}$.

Most of the notions previously defined for refinement statements apply to them equally well when imports are added to architectural specifications. One exception is the operation $\mathcal{A}(r)$, which has to be adjusted in the case where $root(r) = ASP$, since imports which may appear in ASP have to be taken into account.

A refinement dag r is said to be *flattenable*, if, for any node n_1 labeled by an architectural specification ASP, if X is an import declaration in ASP, then there exists an edge $n_1 \xrightarrow{X} n_2$ in r. Thus, in a flattenable refinement dag all import declarations must be refined. The flattening operation $\mathcal{A}(r)$ is defined only for flattenable r. This ensures that the architectural specification $\mathcal{A}(r)$ never contains any imports.

The definition of $\mathcal{A}(r)$ for refinement dags with imports is an extension of the previous definition by the case of r containing an edge $root(r) \xrightarrow{X} r'$, where X is the last unit name declared, for which such an edge exists, and where the declaration of X is an import declaration. Note that, in the other cases, the operation $\mathcal{A}(\_)$ may only be applied to flattenable dags and that, when so applied, it recursively calls itself only for flattenable dags, and always returns an architectural specification without imports. In the new case, $L(root(r))$ must take the form **units** $UDS\ X : SP$ $\mathbf{given}_\Sigma$ $T\ Y_1 : USP_1 \ldots Y_n : USP_n$ **result** UE, since r is flattenable; for the same reason, r' is flattenable. Assume $\mathcal{A}(r') =$ **units** UDS' **result** $\lambda Y : SP_I \bullet T_\lambda$; the architectural specification $\mathcal{A}(r')$ must be parametric, since otherwise the refinement statement $root(r) \xrightarrow{X} r'$ could not have been statically correct. Also, it does not contain imports. Now produce the refinement dag r'', which is the same as r except that

- the edge $root(r) \xrightarrow{X} r'$ is removed, and
- $root(r)$ is labeled by

$$\mathbf{units}\ UDS\ UDS'\ Y_1 : USP_1 \ldots Y_n : USP_n\ \mathbf{result}\ UE[T_\lambda[T/Y]/X]$$

Observe that r'' is flattenable. Thus, we may set $\mathcal{A}(r) = \mathcal{A}(r'')$ and this architectural specification will not contain any imports.

Proposition 5 may then be extended:

Proposition 8. *If a flattenable refinement dag r, possibly involving imports, is correct and*

$$L(root(r)) :: \mathcal{OT}$$

then

$$\mathcal{A}(r) :: \mathcal{OT}$$

Proof. The following lemma, concerning import declarations, is analogous to Lemma 7 from the proof of Proposition 5. Its proof may be found in Appendix 1.

Lemma 9. *Assume* $ASP = \textbf{units}\ UDS\ X : SP\ \textbf{given}_\Sigma\ T\ Y_1 : USP_1 \ldots Y_n : USP_n\ \textbf{result}\ UE$ *is correct, that* $ASP' = \textbf{units}\ UDS'\ \textbf{result}\ \lambda Y : SP_I \bullet T_\lambda$ *is correct and that* $ASP' :: ASP.X$ *holds.*

The architectural specification $ASP'' = \textbf{units}\ UDS\ UDS'\ Y_1 : USP_1 \ldots \ldots Y_n : USP_n\ \textbf{result}\ UE[T_\lambda[T/Y]/X]$ *is then correct and, moreover:*

1. $\pi_2([ASP'']) = \pi_2([ASP])$,
2. *for any* $\varrho' \in \mathrm{dom}([\![ASP'']\!])$*, there exists an environment* ϱ *such that* $[\![ASP'']\!](\varrho') = [\![ASP]\!](\varrho)$. □

The proof of Proposition 8 is by induction over the number of edges in the flattenable refinement dag r. It is an extension of the proof of Proposition 5 by the case of r containing an edge $root(r) \xrightarrow{X} r'$, where X is an import declaration and where the declaration of X is the last declaration in $ASP = L(root(r'))$, for which an edge exists.

By the induction hypothesis, the architectural specification $ASP' = \mathcal{A}(r')$ satisfies $ASP' :: ASP.X$, since, the refinement dag r being correct, we have $L(root(r')) :: ASP.X$. Let $ASP'' = L(root(r''))$. Then, by Lemma 9, the architectural specification ASP'' is correct and $\pi_2([ASP'']) = \pi_2([ASP])$; this implies that the statement $ASP'' :: \mathcal{OT}$ is component-correct, since $ASP :: \mathcal{OT}$ was. By the same lemma, for any $\varrho' \in \mathrm{dom}([\![\mathcal{A}(r)]\!])$, there exists an environment ϱ such that $[\![ASP'']\!](\varrho') = [\![ASP]\!](\varrho)$. Because $ASP :: \mathcal{OT}$, this implies that $ASP'' :: \mathcal{OT}$ holds. By the induction hypothesis, we then have $\mathcal{A}(r) :: \mathcal{OT}$, since $\mathcal{A}(r) = \mathcal{A}(r'') = ASP''$. □

It should be noted that the restriction to flattenable refinement dags is necessary, though it could be slightly relaxed. The reason is the behavior of inconsistent import declarations. Consider an architectural specification ASP of the form $\textbf{units}\ \ldots X : USP \ldots Y : SP\ \textbf{given}_\Sigma\ T \ldots \textbf{result}\ UE$. Clearly, if $ASP.X$ is refined, then the unit class $\mathcal{M}[ASP.Y]$ may also change: the domain of units $F \in \mathcal{M}[ASP.Y]$ may, and usually will, become smaller. This is not a problem as such. But if the import declaration was, originally, inconsistent and becomes consistent after refining $ASP.X$, then the expression UE, previously correct "for free", will undergo normal semantical analysis and may turn out to be incorrect. Therefore allowing such refinements would mean that the operation $\mathcal{A}(\_)$ may return an incorrect architectural specification. As is clear from the above, the problem only exists if a refined declaration precedes an unrefined import declaration or if such a situation may arise while computing $\mathcal{A}(\_)$. Note that, as Theorem 10 shows, the problem ceases to exist in the partial model semantics proposed in Section 9.

8 Generativity

An important feature of architectural specification semantics as defined both in CASL as well as in this thesis is the *non-generativity*, also called *applicativity*, of the semantics. Non-generativity means that, in the semantics, parametric units are treated as functions, returning the same result when fed the same parameter. The opposite of a non-generative semantics is a *generative* one, in which two applications of the same parametric unit to the same argument may give different results. In fact, using a generative semantics is quite common in programming languages, as exemplified by the generative functors of Standard ML; it also serves a methodological purpose, making the design more transparent (it is clear that what is correct with respect to a generative semantics is even more so with respect to a non-generative one).

The generative semantics may be defined either by allowing units to be multi-valued functions or, syntactically, by treating each declaration $F : USP$ such that F is applied $n > 0$ times in the given architectural specification as a list of declarations $F_1, \ldots, F_n : USP$ and then treating the ith application of F as an application of F_i. We are only interested whether an architectural specification has a denotation at all with respect to the generative semantics and therefore it makes no difference for us which of the above approaches is taken. Note that, if no parametric unit in ASP is applied more than once, then the generative and non-generative semantics are equivalent. An architectural specification is called *generatively correct* if it has a denotation with respect to the generative semantics. A refinement dag, refinement statement, architectural specification, declaration, expression or term is called *generative* if no parametric unit in it is applied more than once. It should be noted that the flattening operation $\mathcal{A}(\_)$ preserves the property of being generative in the sense that, if r is a generative refinement dag, then $\mathcal{A}(r)$ is a generative architectural specification.

In CASL, the semantics is function-based and thus non-generative, just like the semantics presented in Section 5. Still, use of generatively correct architectural specifications is recommended. Moreover, an additional semantics for architectural specifications is provided, the so-called *extended static semantics*, which is also not obligatory; this semantics, described in detail in Section 15, enforces a number of conditions on the architectural specifications used and among them generative correctness. Therefore it may be said that, although formally architectural specifications in CASL have a non-generative semantics, in practice a generative one is enforced.

In many cases non-generativity is unproblematic. If we write $F(T)$ twice in an architectural specification, then it is only natural that the result be twice the same. Had we chosen a generative semantics, we would see that an amalgamation as simple as $F(T) + F(T)$ may be incorrect, since the left $F(T)$ may differ from the right one. Because, in practice, it is frequently necessary to use a model more than once (e.g., extracting different groups of useful elements from a model), in a generative semantics it is necessary to introduce *local definitions* with an obvious syntax (Figure 3) and semantics.

$T ::= \ldots \mid \textbf{local } X = T \textbf{ within } T$

Figure 3: Local definitions in architectural specifications.

Thanks to the local definition, instead of $F(T) + F(T)$ one may now write **local** $X = F(T)$ **within** $X + X$, which is correct. Of course, for non-generative architectural specifications this is redundant; nevertheless, such a construct exists in CASL in an even more general form which allows parametric units to be locally defined as well.

If architectural specifications with imports are used, then local definitions do not suffice, since they do not allow the same $F(T)$ to be used in more than one import. For this reason, the possibility is introduced of mixing declarations with *global definitions*, i.e., definitions of the form $X = T$ in the **units** section. We do not provide a formal semantics for local or global definitions, since both may be translated into "dummy" imports; for example, a definition of the form $X = T$ may be substituted for by the declaration $X : \top_\Sigma \textbf{ given}_\Sigma\ T$, where Σ is the signature of T (this technique does not work in case the local definition is part of a λ-expression and a λ-bound variable A appears in T; this does not cause problems, since one may first transform the λ-expression of the form $\lambda A : SP \bullet T_\lambda$ into the unit declaration $A : SP$ and the term $T_\lambda + A$).

The impression might arise that differences between generative and non-generative architectural specifications are only formal, since by using local definitions of subterms one can transform the latter into the former, while preserving their correctness and semantics. This turns out not to be the case; to see this, consider the architectural specification ASP defined in Figure 4 and define $SP = \{\ \textbf{sort } s;\ \textbf{ops } a, b, a', b' : s;\ \textbf{axioms } a = a' \Rightarrow b = b';\ \}$. In a non-generative semantics, the refinement $ASP :: SP$ holds; in a generative semantics, it does not hold, and there are no common subterms in ASP the local definition of which could change this situation.

units

$A : \{ \textbf{ sort } s; \}$
$F : \{ \textbf{ sort } s; \} \rightarrow \{ \textbf{ sort } s; \textbf{ op } a : s; \}$
$G : \{ \textbf{ sort } s; \} \rightarrow \{ \textbf{ sort } s; \textbf{ op } a : s; \}$
$H : \{ \textbf{ sort } s; \textbf{ op } a : s; \} \rightarrow \{ \textbf{ sort } s; \textbf{ ops } a, b : s; \}$

result

local

$B = F(A)$
$C = G(A)$

within

$H(B) + (H(C)|_{\textbf{sort } s \mapsto s, \textbf{ ops } a' \mapsto a, b' \mapsto b})$

Figure 4: Example of non-trivial use of non-generativity.

For reasons that will become apparent in the following sections and some of which are summarized in Section 12, in this thesis we focus on the generative semantics. The non-generative semantics is considered more closely only in Sections 11 and 21.

9 Programs and Partial Model Semantics

A crucial question regarding refinement dags is whether they, if they hold, really define a correct program development process? That is, if a refinement dag in a language $\mathcal{L}$ with root labeled *USP* has been verified to be correct, and if implementations of its unimplemented nodes in a compatible programming language $\mathcal{P}$ are provided, is it possible to extract an implementation of *USP* in that language?

Of course, it may be discussed whether the above property should be required for *any* compatible programming language $\mathcal{P}$, or perhaps for a limited class of languages. Still, without some form of this property the whole idea of formalizing program development with architectural specifications and refinement dags would fail, since one would not get an inch closer to a final implementation.

Because of Proposition 8, we may confine ourselves to single refinement statements, since any refinement dag may be presented in that form. In fact, we also may consider only refinement statements in which the implementing task is an architectural specification *ASP*: this is because an implementing task of the form *USP* may be represented by an architectural specification, as shown in the definition of the operation $\mathcal{A}(\_)$.

Thus we reformulate our property as follows: if $\mathcal{IT} :: \mathcal{OT}$ holds in a language $\mathcal{L}$, does it necessarily hold in the language $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$, for any $\mathcal{P}$ compatible with $\mathcal{L}$?

In general the answer to the above question is unfortunately "no". We will now investigate why this is so, as well as circumstances, in which the answer is positive.

First, one should observe that while, for any specification *SP*, if $[\![SP]\!]$ is empty, then so is $[\![SP]\!]$, this is not the case for parametric unit specifications *USP*. For example, if we specify that we want a parametric unit that takes a model with a function `does_program_terminate`, satisfying the obvious specification SP_{term}, and returns a model satisfying the specification $\perp$ (i.e., the inconsistent specification), then certainly there will be no such parametric unit. However, assuming that our programming language does not perform miracles and so cannot implement SP_{term}, such a parametric unit on *programs* does exist, namely the empty function.

This trivial observation causes considerable trouble. A refinement statement $\mathcal{IT} :: \mathcal{OT}$ may hold simply because $\mathcal{IT}$ is unimplementable in $\mathcal{L}$, e.g., it is the unit specification $SP_{term} \to \perp$. If the same statement is interpreted in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$, then it may turn out that $\mathcal{IT}$ has an implementation, in this example the empty function. Unfortunately, this implementation need not be an implementation of $\mathcal{OT}$; hence, the refinement statement interpreted in the programming language may turn out to be incorrect. The same situation will arise if $\mathcal{IT}$ is an architectural specification in which a declaration of the above form is included.

To solve this, somewhat artificial, problem, refinement statements have to be given a slightly modified model semantics, called *partial model semantics*. In fact, this modified semantics at the same time solves other problems connected to inconsistency, e.g., problems with refinement in the presence of inconsistent import declarations, described in Section 7, or problems with completeness of proof calculi (see, e.g., the form of completeness in Section 3 of [Hof01]).

The fundamental idea of the partial model semantics is that if, in $ASP =$ **units** *UDS* **result** *UE*, the unit $X : USP$ is declared, where *USP* is inconsistent, then *ASP* should not be declared correct "for free", whithout checking the correctness of the fragment coming after this inconsistent declaration. Note that this regards all kinds of declarations in the **units** section, parametric, non-parametric, and with imports, as well as parameter declarations in λ-expressions.

Thus, the point of the partial model semantics is that, e.g., removing a declaration that is never used in the architectural specification should preserve its correctness. For parametric declarations, the matter is a bit more subtle. Observe that a declaration $F : SP \to SP'$ may be inconsistent because some model $M \in [\![SP]\!]$ has no extension in $[\![SP']\!]$. But, in the architectural specification, the unit F, even though it is applied, may, in fact, never be applied to such a model M. In such a case, the point of the partial model semantics is that removing such an M from the domain of F should preserve correctness of the architectural specification.

From a technical point of view, the idea of partial model semantics, described above, is realized by adding a special object $\perp$ which represents the lack of a model. All non-parametric units may be $\perp$; parametric units are required to preserve $\perp$, but may also return it when fed a proper model. This forces all unit specifications to be consistent.

Let $\mathbf{Mod}_\perp(\Sigma) = \mathbf{Mod}(\Sigma) \cup \{\perp\}$, $[\![SP]\!]_\perp = [\![SP]\!] \cup \{\perp\}$, $\mathbf{Unit}_\perp(\Sigma) = \mathbf{Mod}_\perp(\Sigma)$ and $\mathbf{Unit}_\perp(\Sigma \to \Sigma')$ be the set of all partial functions $F : \mathbf{Mod}_\perp(\Sigma) \to \mathbf{Mod}_\perp(\Sigma')$ such that $\perp \in \mathrm{dom}(F)$, $F(\perp) = \perp$ and, for all $M \in \mathrm{dom}(F)$, $F(M) \neq \perp$ implies $F(M)|_\Sigma = M$. The latter property of F is called, as before, *persistence*. Finally, for any parametric unit specification $SP \to SP'$, by $[\![SP \to SP]\!]_\perp$ we denote the set of all $F \in \mathbf{Unit}_\perp(sig[SP] \to sig[SP'])$ such that, for all $M \in [\![SP]\!]_\perp$, $M \in \mathrm{dom}(F)$ and $F(M) \in [\![SP']\!]_\perp$. For $\sigma : \Sigma \to \Sigma'$, let $\_|^\perp_\sigma : \mathbf{Mod}_\perp(\Sigma') \to \mathbf{Mod}_\perp(\Sigma)$ denote the standard reduct $\_|_\sigma$, which additionally takes $\perp$ to $\perp$.

Notions of environment, context and model or unit evaluator will be used for the partial case as well. An environment ϱ *partially fits* a static context C_s if $\varrho(X) \in \mathbf{Unit}_\perp(C_s(X))$, for all X in C_s, and a context C *partially fits* C_s if, for all $\varrho \in C$, ϱ partially fits C_s.

The partial model semantics of architectural specifications is essentially the same as the model semantics, except that, in declarations, $[\![\_]\!]_\perp$ is used instead of $[\![\_]\!]$ and that the special element $\perp$ gets propagated. To avoid any confusion, below all the equations of the partial model semantics are provided (the static semantics remains as before). The semantic function is denoted by $[\![\_]\!]^\perp$.

$$\boxed{[ASP] = (C_s, \mathrm{U}\Sigma) \qquad\qquad [\![ASP]\!]^\perp = UEv}$$

The context $\mathrm{dom}(UEv)$ fits C_s and UEv is a unit evaluator into $\mathrm{U}\Sigma$.

$$[\![\mathbf{units}\ UDS\ \mathbf{result}\ UE]\!]^\perp \;=\; \lambda\varrho \in [\![UDS]\!]^\perp \cdot [\![UE]\!]^\perp_\varrho$$

$$\boxed{[UDS] = C_s \qquad\qquad [\![UDS]\!]^{\perp} = C}$$

The context C fits C_s.

$$
\begin{array}{lcl}
[\![\epsilon]\!]^{\perp} & = & \{\emptyset\} \\
[\![UDS\ X : SP\ \mathbf{given}_{\Sigma}\ T]\!]^{\perp} & = & \{\varrho[M/X] \mid \varrho \in [\![UDS]\!]^{\perp},\ M \in [\![SP]\!]_{\perp}, \\
 & & M \neq \perp \text{ implies } M|_{\Sigma} = [\![T]\!]^{\perp}_{\varrho}\} \\
[\![UDS\ X : USP]\!]^{\perp} & = & [\![UDS]\!]^{\perp}[[\![USP]\!]_{\perp}/X]
\end{array}
$$

Note that, in the rule for imports, the equality $M|_{\Sigma} = [\![T]\!]^{\perp}_{\varrho}$ means that both sides must be defined and equal.

$$\boxed{[UE]_{C_s} = \mathrm{U}\Sigma \qquad\qquad [\![UE]\!]^{\perp}_{\varrho} = U}$$

The environment ϱ should partially fit C_s. Then $U \in \mathbf{Unit}_{\perp}(\mathrm{U}\Sigma)$.

$$
\begin{array}{lcll}
[\![\lambda A : SP \bullet T]\!]^{\perp}_{\varrho} & = & \lambda M \in [\![SP]\!]_{\perp} \cdot [\![T]\!]^{\perp}_{\varrho[M/A]} & \text{if, for all } M \in [\![SP]\!]_{\perp}, \\
 & & & [\![T]\!]^{\perp}_{\varrho[M/A]} \neq \perp \text{ implies} \\
 & & & [\![T]\!]^{\perp}_{\varrho[M/A]}|_{sig[SP]} = M
\end{array}
$$

Observe that the side-condition forces $[\![T]\!]^{\perp}_{\varrho[\perp/A]} = \perp$.

$$\boxed{[T]_{C_s} = \Sigma \qquad\qquad [\![T]\!]^{\perp}_{\varrho} = M}$$

The environment ϱ should partially fit C_s. Then $M \in \mathbf{Mod}_{\perp}(\Sigma)$.

$$
\begin{array}{lcll}
[\![A]\!]^{\perp}_{\varrho} & = & \varrho(A) & \\
[\![A(T)]\!]^{\perp}_{\varrho} & = & \varrho(A)([\![T]\!]^{\perp}_{\varrho}) & \text{if } [\![T]\!]^{\perp}_{\varrho} \in \mathrm{dom}(\varrho(A)) \\
[\![\bigcup_{i \in I} \sigma_i(T_i)]\!]^{\perp}_{\varrho} & = & \begin{cases} \perp & \text{if } [\![T_i]\!]^{\perp}_{\varrho} = \perp \text{ for some } i \in I \\ \bigoplus_{i \in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho}) & \text{otherwise, if additionally} \\ & \text{the } [\![T_i]\!]_{\varrho} \text{ are } \sigma\text{-amalgamable,} \\ & \text{where } \sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I} \end{cases} & \\
[\![T|_{\sigma:\Sigma \to \Delta}]\!]^{\perp}_{\varrho} & = & ([\![T]\!]^{\perp}_{\varrho})|^{\perp}_{\sigma} &
\end{array}
$$

An architectural specification is said to be *partially correct* if it is statically correct and has a denotation under the partial model semantics.

The partial model semantics induces a *partial semantics* of tasks, written $\mathcal{M}^{\perp}[\_]$. This semantics is exactly the same as the regular model semantics, except that it uses the partial semantics of units, unit specifications and architectural specifications. Refinement statements with respect to the partial semantics are the same as before; they are written

$$\mathcal{IT} ::_{\perp} \mathcal{OT}$$

and such a statement is *partially component-correct* if it is statically correct and $\mathcal{IT}$ and $\mathcal{OT}$ are partially correct. It is *partially correct*, or *partially holds*, if it is partially component-correct and

$$\mathcal{M}^{\perp}[\mathcal{IT}] \subseteq \mathcal{M}^{\perp}[\mathcal{OT}]$$

The qualification "partially" is omitted if the symbol "$::_{\perp}$" is used. Refinement dags with respect to the partial semantics are also the same as before. Their semantics is influenced by the change of semantics of refinement statements, which results in the new notions of a refinement dag being *partially component-correct* and *partially correct* or *partially holding*.

The operation $\mathcal{A}(\_)$ has a slightly changed, simpler definition. Because two versions of this operation have already been presented, below the full definition of the third and final version is provided, even though the first three cases are handled exactly as before.

To handle imports cleanly, a substitution on declarations has to be introduced. So, by $\mathit{UDS}[\mathit{UE}/X]$ denote the same list of declarations as UDS, except that any $Y : \mathit{SP}\ \mathbf{given}_{\Sigma}\ T$ is replaced by $Y : \mathit{SP}\ \mathbf{given}_{\Sigma}\ T[\mathit{UE}/X]$.

Let r be a refinement dag in which all unit names are distinct. Then

- if r contains only the root, which is labeled USP, then $\mathcal{A}(r)$ is:
  - $\mathbf{units}\ A : \mathit{SP}\ \mathbf{result}\ A$, if $\mathit{USP} = \mathit{SP}$,
  - $\mathbf{units}\ F : \mathit{SP} \to \mathit{SP}'\ \mathbf{result}\ \lambda A : \mathit{SP} \bullet F(A)$, if $\mathit{USP} = \mathit{SP} \to \mathit{SP}'$,

  where A and F are newly generated,
- if r contains only the root, which is labeled by ASP, then $\mathcal{A}(r) = \mathit{ASP}$,
- if r contains an edge $\mathit{root}(r) \to r'$, then $\mathcal{A}(r) = \mathcal{A}(r')$,
- if r contains an edge $\mathit{root}(r) \xrightarrow{X} r'$ and X is the last declaration in $L(\mathit{root}(r))$ for which an edge exists, then proceed as follows. Assume

$$L(\mathit{root}(r)) = \mathbf{units}\ \mathit{UDS}_1\ \mathit{UD}\ \mathit{UDS}_2\ \mathbf{result}\ \mathit{UE}$$

where UD is a declaration of the form $X : USP$ or $X : USP$ $\mathbf{given}_\Sigma$ T, and

$$\mathcal{A}(r') = \mathbf{units}\ UDS'\ \mathbf{result}\ UE'$$

Observe that if UD is an import declaration, then UE' must be of the form $\lambda Y : SP \bullet T_\lambda$. Let S be the substitution $[UE'/X]$ if UD is not an import declaration and $[T_\lambda[T/Y]/X]$ if UD is $X : USP$ $\mathbf{given}_\Sigma$ T. Now produce the refinement dag r'', which is equal r except that

- the edge $root(r) \xrightarrow{X} r'$ is removed, and
- $root(r)$ is labeled by

$$\mathbf{units}\ UDS_1\ UDS'\ UDS_2[S]\ \mathbf{result}\ UE[S]$$

and set $\mathcal{A}(r) = \mathcal{A}(r'')$.

Note that the architectural specification $\mathcal{A}(r)$ may contain import declarations, if some such declarations have not been refined in r. Also, the flattenability requirements have gone away, by virtue of the introduced partiality, so that a more general version of Propositions 5 and 8 holds:

Theorem 10. *If a refinement dag r, possibly with imports, is partially correct and*

$$L(root(r)) ::_\perp \mathcal{OT}$$

then

$$\mathcal{A}(r) ::_\perp \mathcal{OT}$$

Proof. For the partial model semantics, Lemma 11, a version of Lemma 6, still holds. The proof is very similar to that of Lemma 6 and we omit it. Note, however, that even the statement of this lemma would *not* be possible for less radical versions of the partial model semantics, such as the one in [MHAH04], in which non-parametric units may not be replaced by $\perp$. This is because, in such a semantics, $\varrho[[\![UE']\!]^\perp_\varrho/Z]$ would not be a correct environment if $[\![UE']\!]^\perp_\varrho$ was $\perp$.

Lemma 11. *Assume that $[UE']_{C_s}$ and $[UE]_{C_s[[UE']_{C_s}/Z]}$ are defined and that Z and unit names from C_s are not λ-bound in UE.*

Assume that ϱ partially fits C_s and $[\![UE']\!]^\perp_\varrho$ is defined. If $[\![UE]\!]^\perp_{\varrho[[\![UE']\!]^\perp_\varrho/Z]}$ is defined, then it is equal $[\![UE[UE'/Z]]\!]^\perp_\varrho$. □

As before, separate lemmas for the cases of a regular declaration and an import declaration are needed. A proof of Lemma 12 may be found in Appendix 1. The proof of Lemma 13 is similar to the proofs of Lemmas 9 and 12 and therefore omitted.

Lemma 12. *Assume* $ASP = \textbf{units}\ UDS_1\ X : USP\ UDS_2\ \textbf{result}\ UE$ *is partially correct, that* $ASP' = \textbf{units}\ UDS'\ \textbf{result}\ UE'$ *is partially correct and that* $ASP' ::_\perp ASP.X$ *holds.*

Then $ASP'' = \textbf{units}\ UDS_1\ UDS'\ UDS_2[UE'/X]\ \textbf{result}\ UE[UE'/X]$ *is partially correct and, moreover:*

1. $\pi_2([ASP'']) = \pi_2([ASP])$,
2. *for any* $\varrho' \in \mathrm{dom}([\![ASP'']\!])$, *there exists an environment* ϱ *such that* $[\![ASP'']\!](\varrho') = [\![ASP]\!](\varrho)$. □

Lemma 13. *Assume* $ASP = \textbf{units}\ UDS_1\ X : SP\ \textbf{given}_\Sigma\ T\ UDS_2\ \textbf{result}\ UE$ *is partially correct, that* $ASP' = \textbf{units}\ UDS'\ \textbf{result}\ \lambda Y : SP_I \bullet T_\lambda$ *is partially correct and that* $ASP' ::_\perp ASP.X$ *holds.*

Let $ASP'' = \textbf{units} UDS_1\ UDS'\ UDS_2[T_\lambda[T/Y]/X]\ \textbf{result} UE[T_\lambda[T/Y]/X]$. *Then* ASP'' *is partially correct and, moreover:*

1. $\pi_2([ASP'']) = \pi_2([ASP])$,
2. *for any* $\varrho' \in \mathrm{dom}([\![ASP'']\!])$, *there exists an environment* ϱ *such that* $[\![ASP'']\!](\varrho') = [\![ASP]\!](\varrho)$. □

The proof of the theorem is now analogous to that of Propositions 5 and 8 with the exception that, instead of Lemmas 7 and 9, it uses Lemmas 12 and 13. Therefore we omit it. □

Comparison of the above theorem with the analogous proposition for the regular model semantics (Proposition 8) clearly shows the superiority of the partial model semantics, which guarantees that the natural flattening operation always works as expected. Still, one could raise the doubt that, perhaps, the result of the flattening under the partial model semantics is of less value, since the denotation of the resulting architectural specification could return $\perp$ instead of the strived for model or program. However, any impression that, by introducing the object $\perp$, requirements on correct refinements have been relaxed, is totally false. Actually, the contrary is true; this fact is stated in Propositions 14 and 17.

For any environment ϱ (non-partially) fitting C_s let $\perp(\varrho)$ be the same as ϱ except that, for all parametric units X, we have $\perp(\varrho)(X) = \varrho(X)[\perp/\perp]$, that is, $\perp(\varrho)(X)$ is the same as the parametric unit $\varrho(X)$ except that it additionally maps $\perp$ to $\perp$; then $\perp(\varrho)$ partially fits C_s.

A correct architectural specification ASP is said to contain an *inconsistent declaration* if $[\![ASP]\!] = \emptyset$. More generally, it contains an *inconsistent specification* if it contains an inconsistent declaration or it is parametric and

$[\![ASP]\!](\varrho)$ is the empty function for all $\varrho \in \mathrm{dom}([\![ASP]\!])$. These notions can be easily seen to correspond to the appropriate informal notions.

For any declaration, expression or term, by $np(\_)$ the set of non-parametric unit names appearing freely in that syntactic object is denoted. If an expression UE is of the form $\lambda X : SP \bullet T$, then *UE uses its λ-argument* if X is an element of $np(T)$; if the result unit of an architectural specification ASP is a unit expression, then *ASP uses its λ-argument* if its result unit does so.

The consequences of the partial correctness of an architectural specification for its denotation under the (regular) model semantics are described in the following proposition. In particular, it states the fundamental fact that partial correctness implies correctness. A proof may be found in Appendix 1.

Proposition 14. *If ASP is partially correct, then it is correct and either it contains an inconsistent declaration, or*

$$\mathrm{dom}([\![ASP]\!]) = C$$

where $C = \{\varrho \text{ fits } C_s \mid \bot(\varrho) \in \mathrm{dom}([\![ASP]\!]^{\bot})\}$ and $C_s = \pi_1([ASP])$, and, for all $\varrho \in C$, if ASP is non-parametric, then

$$[\![ASP]\!](\varrho) = [\![ASP]\!]^{\bot}(\bot(\varrho))$$

and if it is parametric, then it uses its λ-argument and

$$[\![ASP]\!](\varrho)[\bot/\bot] = [\![ASP]\!]^{\bot}(\bot(\varrho))$$

□

It might be surprising that, in light of Proposition 14, partially correct parametric architectural specifications with consistent declarations have to use their λ-argument. To see this, assume that a partially correct architectural specification ASP is of the form **units** UDS **result** $\lambda A : SP \bullet T$ and that UDS is consistent. It can be shown that then there exists some non-partial environment $\varrho \in [\![UDS]\!]^{\bot}$, where "non-partial" means that $\bot$ only appears in it as the result of feading $\bot$ to parametric units. Now consider ϱ extended by the mapping of A to $\bot$. If ASP did not use its λ-argument, then the result of evaluating T in this environment could not be $\bot$. But this would mean that ASP is not partially correct, since the partial model semantics rule for λ-expressions requires that $[\![T]\!]^{\bot}_{\varrho[\bot/A]}$ be $\bot$.

An immediate corollary of Proposition 14 regarding refinement statements is:

Corollary 15. *If the refinement statement $\mathcal{IT} ::_{\bot} \mathcal{OT}$ is partially component-correct, then the refinement statement $\mathcal{IT} :: \mathcal{OT}$ is component-correct.* □

Conversely, conditions in which correctness implies partial correctness are the subject of the following proposition, a proof of which may be, again, found in Appendix 1.

Proposition 16. *If an architectural specification is correct, contains no inconsistent specification and, if parametric, uses its λ-argument, then it is partially correct.* □

Thus, requiring refinement statements and, as a result, refinement dags to be partially correct is not at all a liberal policy. Quite on the contrary, it is more demanding than the traditional approach. This is stated below:

Proposition 17. *If*

$$\mathcal{IT} ::_{\perp} \mathcal{OT}$$

then

$$\mathcal{IT} :: \mathcal{OT}$$

Proof. Assume that $\mathcal{IT} ::_{\perp} \mathcal{OT}$. We now have two cases.

Suppose first that the statement is non-parametric and that $M \in \mathcal{M}[\mathcal{IT}]$. Then we also have $M \in \mathcal{M}^{\perp}[\mathcal{IT}]$: this is clear if $\mathcal{IT}$ has the form SP; it is also true if it has the form ASP, since then $M = [\![ASP]\!](\varrho)$, for some environment ϱ, and then, by Proposition 14, we have $M = [\![ASP]\!]^{\perp}(\perp(\varrho))$, i.e., $M \in \mathcal{M}^{\perp}[ASP]$. Because $\mathcal{IT} ::_{\perp} \mathcal{OT}$, we may conclude that $M \in \mathcal{M}^{\perp}[\mathcal{OT}]$. Observe that $\mathcal{M}^{\perp}[\mathcal{OT}] = \mathcal{M}[\mathcal{OT}] \cup \{\perp\}$, since $\mathcal{OT}$ may be of the form SP or of the form $ASP.X$, where $X : SP$ is a declaration in ASP; in both cases $\mathcal{M}^{\perp}[\mathcal{OT}] = [\![SP]\!]_{\perp}$ and $\mathcal{M}[\mathcal{OT}] = [\![SP]\!]$. Since $M \neq \perp$, this implies $M \in \mathcal{M}[\mathcal{OT}]$.

Suppose now that the statement is parametric and that $F \in \mathcal{M}[\mathcal{IT}]$. Then also $F[\perp/\perp] \in \mathcal{M}^{\perp}[\mathcal{IT}]$: again, this is clear if $\mathcal{IT}$ has the form USP; but it is also true for $\mathcal{IT}$ of the form ASP, since then $F = [\![ASP]\!](\varrho)$, for some environment ϱ, and so, by Proposition 14, $F[\perp/\perp] = [\![ASP]\!]^{\perp}(\perp(\varrho))$, which means that $F[\perp/\perp] \in \mathcal{M}^{\perp}[ASP]$. Because $\mathcal{IT} ::_{\perp} \mathcal{OT}$, $F[\perp/\perp] \in \mathcal{M}^{\perp}[\mathcal{OT}]$.

If $\mathcal{OT}$ is of the form USP or $ASP.X$, where $X : USP$ is a declaration in ASP, then $F[\perp/\perp] \in [\![USP]\!]_{\perp}$ and so, clearly, $F \in [\![USP]\!] = \mathcal{M}[\mathcal{OT}]$, since F never returns $\perp$. If, on the other hand, $\mathcal{OT}$ is of the form $ASP.X$, where the declarations of ASP take the form $UDS\ X : SP\ \mathbf{given}_{\Sigma}\ T\ UDS'$, then, for any $M' \in \mathcal{M}'$, $M' \in \mathrm{dom}(F[\perp/\perp])$ and $F[\perp/\perp](M') \in [\![SP]\!]_{\perp}$, where $\mathcal{M}' = \{[\![T]\!]^{\perp}_{\eta'} \mid \eta' \in [\![UDS]\!]^{\perp}\} \cup \{\perp\}$; also, $F[\perp/\perp]$ is persistent in the sense that $F[\perp/\perp](M')|_{\Sigma} = M'$ for $M' \neq \perp$ in its domain. Let $\mathcal{M} = \{[\![T]\!]_{\eta} \mid \eta \in [\![UDS]\!]\}$; this is defined, because ASP is partially correct and, hence, correct. Observe that, for any $\eta \in [\![UDS]\!]$, by Lemmas 78 and 79 (see Appendix 1) and because ASP is correct, $[\![T]\!]_{\eta} = [\![T]\!]^{\perp}_{\perp(\eta)}$ and so $\mathcal{M} \subseteq \mathcal{M}'$. Hence, for any $M \in \mathcal{M}$, $M \in \mathrm{dom}(F)$, since $M \neq \perp$, and $F(M) \in [\![SP]\!]$, since F never produces $\perp$; also, F is persistent, i.e., $F(M)|_\Sigma = M$ for all M in its domain. This means that $F \in \mathcal{M}[ASP.X] = \mathcal{M}[\mathcal{OT}]$. $\square$

The above considerations may be concluded by a simple proposition, which shows that the partial model semantics solves the problem it was designed to solve. Even though it is an equivalence, the crucial implication is the one going downwards. This implication guarantees that the step from models to programs is correct in the, rather trivial, case that all tasks concerned are just unit specifications. More precisely, this implication shows that a refinement dag built of unit specifications, which partially holds in $\mathcal{L}$, continues to partially hold in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$. In the next section, this result is, under additional assumptions, extended to all refinement dags, i.e., also those containing architectural specifications.

Proposition 18. *For any unit specifications USP and USP′ over a common signature in $\mathcal{L}$:*

$$USP ::_\perp USP' \textit{ in } \mathcal{L}$$

$$\textit{iff}$$

$$\textit{for any programming language } \mathcal{P} \textit{ compatible with } \mathcal{L}$$

$$USP ::_\perp USP' \textit{ in } \mathrm{Pr}(\mathcal{L}, \mathcal{P})$$

Proof. Let $\mathcal{L} = (\mathit{Sig}, \mathcal{I}, \mathbf{Mod}, \mathit{sig}[\_], [\![\_]\!])$.

"$\Uparrow$": It is enough to consider a programming language $\mathcal{P}$ in which "programs" are simply models from $\mathcal{L}$. Formally, let $\mathcal{P}$ be the programming language $(\mathit{Sig}, \mathbf{Mod}, \mathbf{Mod}, \{\mathrm{id}_{\mathbf{Mod}(\Sigma)}\}_{\Sigma \in \mathit{Sig}})$. Then $\mathcal{P}$ is a programming language compatible with $\mathcal{L}$ and, moreover, $\mathrm{Pr}(\mathcal{L}, \mathcal{P}) = \mathcal{L}$.

"$\Downarrow$": Assume $\mathcal{P}$ is a programming language compatible with $\mathcal{L}$. If $USP = SP$ and $USP' = SP'$, then $SP ::_\perp SP'$ in $\mathcal{L}$ means that $[\![SP]\!] \subseteq [\![SP']\!]$, which implies $[\![SP]\!] \subseteq [\![SP']\!]$, that is, $SP ::_\perp SP'$ in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$.

Suppose that $USP = SP_0 \to SP_1$ and $USP' = SP'_0 \to SP'_1$ and that $P \in [\![SP_0 \to SP_1]\!]_\perp$. Let $\Sigma_0 = \mathit{sig}[SP_0]$ and $\Sigma_1 = \mathit{sig}[SP_1]$.

Take any $Q \in [\![SP'_0]\!]$. We have $[\![Q]\!]_{\Sigma_0} \in [\![SP'_0]\!]$ and, since $SP_0 \to SP_1 ::_\perp SP'_0 \to SP'_1$ holds in $\mathcal{L}$, one may conclude that $[\![Q]\!]_{\Sigma_0} \in [\![SP_0]\!]$. Hence, $[\![SP'_0]\!] \subseteq \mathrm{dom}(P)$.

Define the parametric unit F_Q by setting $F_Q([\![Q]\!]_{\Sigma_0}) = [\![P(Q)]\!]_{\Sigma_1}$ and $F_Q(M) = \perp$ for all Σ_0-models M other than $[\![Q]\!]_{\Sigma_0}$. Of course, F_Q is an element of $[\![SP_0 \to SP_1]\!]_\perp$. Because $SP_0 \to SP_1 ::_\perp SP'_0 \to SP'_1$ holds in $\mathcal{L}$, the unit F_Q is also an element of $[\![SP'_0 \to SP'_1]\!]_\perp$. Therefore $[\![P(Q)]\!]_{\Sigma_1} = F_Q([\![Q]\!]_{\Sigma_0})$ is an element of $[\![SP'_1]\!]_\perp$. Hence, $P(Q)$ is an element of $[\![SP'_1]\!]_\perp$.

Thus, we have shown that, for any $Q \in [\![SP'_0]\!]$, the program Q is an element of the domain of P and, moreover, $P(Q)$ is an element of $[\![SP'_1]\!]_\perp$. This proves that P is an element of $[\![SP'_0 \to SP'_1]\!]_\perp$. Hence, $SP_0 \to SP_1 ::_\perp SP'_0 \to SP'_1$ holds in $\mathcal{P}$. □

The reason why this proposition and, especially, its crucial downward implication, concerns only tasks not involving architectural specifications is that an architectural specification partially correct in $\mathcal{L}$ may, because of problems with amalgamability, cease to be partially correct in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$, as described in the next section.

Note that the above proposition would not hold, had the regular (non-partial) semantics of refinement statements been used. A counterexample involving a `does_program_terminate` function has been presented at the beginning of this section.

10 Programs and Amalgamability

As mentioned above, even with partial model semantics the transition from models to programs need not preserve correctness. More precisely, a refinement dag partially correct on the model level might not be partially correct on the level of programs. The reason is that while operations such as variable, application and reduction can be easily transferred from models to programs, this is not the case with amalgamation: even if some models M and N have a unique common extension along some morphisms, the same need not be true for programs P and Q such that $[\![P]\!] = M$ and $[\![Q]\!] = N$.

This problem is much deeper than the problem considered in the previous section and cannot be dealt with by a small fix in the semantics. The solution is based on two restrictions: first, restricting the spectrum of programming languages for which the transition is required to work, and, second, restricting the class of refinement dags for which the transition is required to work. Note that the first restriction means that, effectively, we are enforcing a weaker property and the second restriction means that the property will hold for a smaller class of refinement dags. Of course, membership in this class has to be possible to check, so that an abstract definition does not suffice here.

For a transition from models to programs to be possible, amalgamations in the architectural specification concerned cannot be correct "by accident". For, in such a case, it would be easy to come up with a programming language for which this accident does not happen. It should by now be obvious that a natural nonaccidental reason is the amalgamation property. That is, as the first restriction, we consider only programming languages with amalgamation. Such a restriction is natural, since the amalgamation property says precisely that certain reasonable amalgamations are always correct.

What are these "reasonable" amalgamations? Well, the amalgamation property guarantees that model families consistent with a diagram may be amalgamated along the colimit of that diagram (Proposition 1). We recognize two cases of "reasonable" amalgamations. The first case is an amalgamation which is the colimit of some diagram (such an amalgamation is called regular, see below); one then only checks whether the architectural specification ensures that any family of models to be amalgamated is consistent with the diagram. The second case we recognize is an amalgamation which is part of the colimit of signatures of all non-parametric units in the architectural specification. This case encompasses the previous one, as will become clear when all these notions are precisely defined. The point, however, is that, to even consider the second case, a colimit of the signatures of all non-parametric units in the architectural specification has to exist. So, with a cocomplete signature category one can always use the second approach, while for not cocomplete signature categories the regular amalgamation approach may be employed. Note that the approach through regular amalgamations has the benefit that it may be easier to check correctness with respect to it. If both approaches may be used, they are equivalent, see Proposition 19.

We now proceed as follows. A notion of *fully static correctness* is defined, which checks whether all amalgamations in an architectural specification fall in one of the two categories discussed above. This notion is defined using rules presented in natural semantics style. The rules are essentially deterministic (though, technically speaking, lack of cocompleteness of the signature category can cause nondeterministic behavior) and so they define an algorithm for checking fully static correctness. Note, however, that this algorithm treates, as primitive operations, computation of certain colimits and universal morphisms in the signature category. In certain specification languages, such as EnrichedCasl, presented in Section 18, these operations are, in fact, not computable (see [KHT+01]). The considered form of correctness is called *fully static* because it operates on signatures only, with absolutely no reference to models or specifications. It is a more elaborate form of static analysis than the standard static semantics, though it does assume that the standard static semantics has been run successfully.

Notions needed to define fully static correctness are now presented.

A sink $\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I}$ is said to be *regular*, if there exists a diagram D with shape O in Sig such that $I \subseteq O$, for all $i \in I$ we have $D(i) = \Delta_i$, and for any object $o \in O$ there exists a morphism m in O with $\mathrm{dom}(m) = o$ and $\mathrm{cod}(m) \in I$, and, moreover, some sink $\tau = \{\tau_o : D(o) \to \Delta\}_{o \in O}$ satisfying $\tau_i = \sigma_i$ for $i \in I$ and $\tau_o = D(m); \sigma_i$ for $o \in O \setminus I$, $i \in I$ and $m : o \to i$, is a colimit of the diagram D. Note that the colimit sink τ, if it exists, is unique. The diagram D is called the *witness* of the regularity of σ. An architectural specification, declaration, expression or term is called *regular* if all amalgamations in it are regular.

Examples of regular sinks are both **with** and "+" amalgamations in CASL. As already explained, a T **with** σ construct is allowed in CASL only if certain surjectivity constraints are enforced on σ which, in principal, force it to be a coequalizer and, hence, a regular sink. As for amalgamations of the form $T_1 + T_2$ where the signatures of T_1 and T_2 are Σ_1 and Σ_2, they are pushouts of a diagram:

$$\begin{tikzcd} \Sigma_1 & \\ \Sigma_1 \cap \Sigma_2 \arrow[u, "\iota_1"] \arrow[r, "\iota_2"'] & \Sigma_2 \end{tikzcd}$$

where $\Sigma_1 \cap \Sigma_2$ is an *intersection* and ι_1 and ι_2 are inclusions of the intersection in Σ_1 and Σ_2, respectively; in $\mathcal{PA}$, the intersection of compatible signatures is defined as $(S, p) \cap (T, q) = (S \cap T, p \cap q)$; a similar definition may be applied to multisorted and subsorted CASL.

If the considered specification language has amalgamation, then the correctness of regular amalgamations is particularily easy to check; we now explain this informally. Suppose that D is a witness of the regularity of a sink $\{\sigma_i : \Delta_i \to \Delta\}_{i \in I}$. Suppose also that the amalgamated units have denotations $M_i = [\![T_i]\!]_{\varrho}^{\perp}$ for $i \in I$, and that none of them is $\perp$. Recall that the shape O of D is required to be of a special form, namely, for any object $o \in O$, there must exist a morphism $m : o \to i$ with $i \in I$. Now consider an extension of the family of models $\{M_i\}_{i \in I}$ to a family $\{M_o\}_{o \in O}$ consistent with D. The special form of the shape O implies that the extension, if it exists, is unique. It turns out that this extension exists precisely if the original family $\{M_i\}_{i \in I}$ is σ-amalgamable. To see this, observe that if the family is amalgamable and N is its amalgamation, then the extension may be defined by setting $M_o = N|_{m;\sigma_i}$, where $m : o \to i$ is a morphism in O; the correctness of this definition is guaranteed by the fact that σ is part of a cocone over D. On the other hand, if an extension exists, then the amalgamation property of the language $\mathcal{L}$ guarantees that this extension is amalgamable along a colimit cocone on D that extends the sink σ. Hence, one may construct the amalgamation N, which is also a σ-amalgamation of the original family $\{M_i\}_{i \in I}$.

In the absence of regularity of amalgamations one can proceed in a more global manner, namely first compute the colimit of the signatures of all non-parametric units in the architectural specification and amalgamate along the colimit sink, obtaining a "global" model $M_\star$ over the "global" signature $\Sigma_\star$. Provided that they are not $\perp$, all the units $[\![T_i]\!]_\varrho^\perp$ may now be obtained by taking $M_\star|_{g_i}$, for some morphisms $g_i : [T_i]_{C_s} \to \Sigma_\star$. To check σ-amalgamability, one now has to construct a colimit $\gamma : \Delta \to \Sigma'_\star$, $\iota : \Sigma_\star \to \Sigma'_\star$, where $\Sigma'_\star$ will be the new "global" signature. If ι is an isomorphism, then the σ-amalgamation may be defined as $M_\star|_{\gamma;\iota^{-1}}$. It can also be shown that if, for any language $\mathcal{L}$ with amalgamation, the $M_\star|_{g_i}$ are σ-amalgamable, then ι must be an isomorphism.

Below, the above ideas are presented formally and in detail.

Let $\mathcal{J}$ be subcategory of *Sig*. A *full $\mathcal{J}$-context* is a set Γ of declarations of the form

$$\{\star : \Sigma,\ A : \Sigma_A, B : \Sigma_B, \ldots,\ j_A : A \to \star, j_B : B \to \star, \ldots\}$$

where

- $\star$ is a special symbol representing the "global" signature,
- A, B, etc. are distinct unit names; they are called *nodes* of Γ and the set of all nodes is denoted by $\mathrm{dom}(\Gamma)$ (it does not contain $\star$). For any declaration $X : \Sigma_X$ in Γ, by $\Gamma(X)$ the signature Σ_X is denoted.
- $j_A : A \to \star$, $j_B : B \to \star$, etc. are morphisms in $\mathcal{J}$ with domain $\Gamma(A)$, $\Gamma(B)$, etc., respectively, and with codomain $\Gamma(\star)$; they are called *edges* of Γ. For any node X in Γ, we write $\Gamma(A \to \star)$ for the unique morphism j_X such that Γ contains the declaration $j_X : X \to \star$.

The qualification "$\mathcal{J}$-context" is often dropped, so that by "full context" implicitly a full $\mathcal{J}$-context is understood. If Γ_1 and Γ_2 are full $\mathcal{J}$-contexts such that $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$ and $\Gamma_1(\star) = \Gamma_2(\star)$, then their *simple sum*, written as $\Gamma_1 + \Gamma_2$, is defined by taking all the declarations from both contexts together.

Full contexts are used to represent families of models over $\mathrm{dom}(\Gamma)$ by a single $\star$-model. Thus, $\Gamma(\star)$ is the "global" signature of, e.g., a considered term. The subcategory $\mathcal{J}$ is used for technical reasons, so that the required cocompleteness of the signature category is, to some extent, limited to $\mathcal{J}$. In particular, in the case of a signature category with derived morphisms, one would usually take as $\mathcal{J}$ the subcategory of non-derived signature morphisms. In order to understand the presentation below, it is easiest to assume that $\mathcal{J} = \mathit{Sig}$.

Composing a full $\mathcal{J}$-context $\Gamma = \{\star : \Sigma,\ X_1 : \Delta_1, \ldots, X_n : \Delta_n,\ j_1 : X_1 \to \star, \ldots, j_n : X_n \to \star\}$ with a morphism $j : \Sigma \to \Sigma'$ in $\mathcal{J}$ produces the full $\mathcal{J}$-context $\{\star : \Sigma',\ X_1 : \Delta_1, \ldots, X_n : \Delta_n,\ (j_1; j) : X_1 \to \star, \ldots, (j_n; j) : X_n \to \star\}$. This context is denoted by $\Gamma; j$.

The signature category *Sig* and inclusion category $\mathcal{I}$ for *Sig* are said to be $\mathcal{J}$*-cocomplete* if $\mathcal{J}$ is a subcategory of *Sig* containing all inclusions and such that:

- for any span (t, ι) in *Sig*, where ι is an inclusion, there exists a pushout (j, t') in *Sig* such that j is a morphism in $\mathcal{J}$,
- all colimits of diagrams built of morphisms from $\mathcal{J}$ exist in *Sig* and the colimit injections all are morphisms from $\mathcal{J}$.

The first of these assumptions is needed to allow one to build full $\mathcal{J}$-contexts for applications of parametric units and for import declarations. The second assumption is a restricted cocompleteness assumption on the signature category: colimits must only exist for diagrams of morphisms from $\mathcal{J}$; but for this property to be used inductively, we also need to assume that the obtained colimit is also built of morphisms from $\mathcal{J}$.

Contexts that might arise as denotations of unit declarations in an architectural specifications are described by pairs ζ,χ, called *context descriptions* and formally defined below. The information in such a pair should be interpreted as follows. The component χ takes any parametric unit name to the signature of a parametric unit and any non-parametric unit name to a morphism $t : \Sigma \to \Sigma_\star$, which describes how the concerned unit may be obtained from a "global" unit as a reduct. Thus, Σ is the actual signature of the unit itself. The component ζ is defined only for names of non-parametric units. The idea is that under $\zeta(A)$ a full context is kept which describes all the non-parametric units that have to be used in order for A to be defined and not equal $\perp$. These units are described as a full context $\zeta(A)$, i.e., a global signature $\zeta(A)(\star)$ and morphisms $\zeta(A)(X \to \star)$ for any unit name X needed to build A; note that $\mathrm{dom}(\zeta(A))$ is the set set of names of units needed to build the unit A. The signature $\zeta(A)(\star)$ is only relatively global, that is, it is a "global signature for A", since it only describes non-parametric units needed to build A. The morphism $\zeta(A)(X \to \star)$ tells us how a global signature for X can be extracted from a global signature for A; the reduct $\_|_{\zeta(A)(X \to \star)}$ takes models that may be denoted by A to models that may be denoted by X. The requirements on ζ, found below, guarantee that, first, there are no circularities (no unit needs itself to be built) and, second, transitivity holds, that is, if Z is needed to build Y and Y is needed to build X, then Z is needed to build X; moreover, if one takes the global signature for X, $\zeta(X)(\star)$, then extracting the global signature for Z from it is equivalent to extracting the global signature for Y and from that extracting the global signature for Z.

Formally, a *context description* is a pair ζ, χ, where:

- χ is a map taking any unit name X in its domain to:
  - a parametric unit signature $\Sigma \to \Sigma'$, in which case X is called *parametric*, or
  - a signature morphism t, in which case X is called *non-parametric*,
- ζ is a map taking any non-parametric unit name $X \in \mathrm{dom}(\chi)$ to a full $\mathcal{J}$-context $\zeta(X)$ such that $\zeta(X)(\star) = \mathrm{cod}(\chi(X))$, $X \notin \mathrm{dom}(\zeta(X))$ and, for all $Y \in \mathrm{dom}(\zeta(X))$, $Y \in \mathrm{dom}(\zeta)$ and $\zeta(X)(Y) = \zeta(Y)(\star)$. Moreover, for any $Z \in \mathrm{dom}(\zeta(Y))$, we must have $Z \in \mathrm{dom}(\zeta(X))$ and $\zeta(Y)(Z \to \star); \zeta(X)(Y \to \star) = \zeta(X)(Z \to \star)$.

By $C_s(\chi)$ the *static context induced by* χ is denoted, i.e., the static context taking any $X \in \mathrm{dom}(\chi)$ to $\chi(X)$ if X is parametric and to $\mathrm{dom}(\chi(X))$ otherwise.

While evaluating terms, we will build a context Γ which is similar to the contexts of the form $\zeta(A)$, with the single exception that it does not yet have a name (and so no unit can depend on it). The notion of this context fitting ζ, χ is thus used to enforce properties which will later be needed in order for Γ to be added to ζ, χ.

A full $\mathcal{J}$-context Γ *fits* a context description ζ, χ if, for all $X \in \mathrm{dom}(\Gamma)$, $\Gamma(X) = \zeta(X)(\star)$ and, for all $Y \in \mathrm{dom}(\zeta(X))$, we have $Y \in \mathrm{dom}(\Gamma)$ and the equality $\Gamma(Y \to \star) = \zeta(X)(Y \to \star); \Gamma(X \to \star)$ holds.

Finally, for use in amalgamations, we need to define a way of joining contexts Γ_i which, for $i \in I$, describe terms T_i. Informally, to do this we first construct a graph which is the sum of all the Γ_i, obtaining a bipartite graph. It is crucial that if a unit name A appears in two contexts, it is interpreted as the same A in the sum (this does not apply to the $\star$). We then construct a colimit of this bipartite graph, the colimit signature being the new "global" signature; this colimit is called the *relevant transformation* and the new context is obtained by composing the old contexts with the relevant transformation. All this is formalized below.

Assume *Sig* and $\mathcal{I}$ are $\mathcal{J}$-cocomplete. Assume Γ_i, for all $i \in I$, are full $\mathcal{J}$-contexts fitting a context description ζ, χ. Let $\mathcal{N} = \bigcup_{i \in I} \mathrm{dom}(\Gamma_i)$. The *sum of contexts*, denoted by $\Gamma = \bigcup_{i \in I} \Gamma_i$, is built as follows. Consider the (bipartite) graph consisting of nodes labeled by objects $\{\zeta(X)(\star)\}_{X \in \mathcal{N}}$ and $\{\Gamma_i(\star)\}_{i \in I}$ and edges labeled by morphisms $\{\Gamma_i(X \to \star)\}_{i \in I, X \in \mathrm{dom}(\Gamma_i)}$. All edges in this graph are labeled by morphisms from $\mathcal{J}$. By the $\mathcal{J}$-cocompleteness assumption, the free category generated by the graph has a colimit $\{j_i : \Gamma_i(\star) \to \Sigma\}_{i \in I}$ in *Sig* such that all the morphisms j_i come from $\mathcal{J}$. Let Γ contain the nodes $X : \zeta(X)(\star)$ for all $X \in \mathcal{N}$ and the special node

$\star : \Sigma$, and let it contain edges $(\Gamma_i(X \to \star); j_i) : X \to \star$ for $X \in \mathcal{N}$, where i is such that $X \in \mathrm{dom}(\Gamma_i)$. Such an i must exist, by definition of $\mathcal{N}$, and it does not matter which is picked, since the morphisms j_i form a cocone. The colimit $\{j_i : \Gamma_i(\star) \to \Gamma(\star)\}_{i \in I}$ is called the *relevant transformation* for the sum $\Gamma = \bigcup_{i \in I} \Gamma_i$.

The statement that an architectural specification ASP is fully statically correct is written $\vdash ASP \rhd\!\!\rhd \Box$. The rules for deriving such statements, presented in a natural semantics style, assume that a category $\mathcal{J}$ is picked such that Sig and $\mathcal{I}$ are $\mathcal{J}$-cocomplete. For any specification language $\mathcal{L} = (Sig, \mathcal{I}, \mathbf{Mod}, Spec, sig[\_], [\![\_]\!])$, by its *static language*, written $(\mathcal{L})_s$, the tuple $(Sig, \mathcal{I}, Spec, sig[\_])$ is understood. Generally, by a *static language* $\mathcal{L}_s$ a static language of some specification language is meant. Clearly, the static semantics of an architectural specification depends only on the static language in which the architectural specification is to be interpreted. Therefore one may speak of architectural specifications "over $\mathcal{L}_s$" and of static correctness "in $\mathcal{L}_s$"; this applies to any task $\mathcal{IT}$ or $\mathcal{OT}$, too. Similarily, because the rules for deriving fully static correctness do not depend on anything except $\mathcal{L}_s$ and $\mathcal{J}$, one may speak of fully static correctness "in $\mathcal{L}_s$ and $\mathcal{J}$".

Note that fully static correctness and, in fact, any notion of correctness for architectural specifications, may be extended to refinement statements and refinement dags in a natural way. Fully static correctness and other forms of correctness are therefore, in the sequel, freely applied to refinement statements and dags.

The process of deriving fully static correctness is called *fully static checking*. It is assumed, as is clear from the headers of the rules, that the static semantics has had a successful run before. Note that the rule for amalgamation is really two rules in one, i.e., one rule with all the premises before the **or** and one rule with all the premises except those between **either** and **or**.

$$\boxed{\vdash ASP \rhd\!\!\rhd \Box}$$

ASP must be statically correct.

$$\frac{\begin{array}{c}\vdash UDS \rhd\!\!\rhd \zeta, \chi \\ \zeta, \chi \vdash UE \rhd\!\!\rhd \Box\end{array}}{\vdash \textbf{units}\ UDS\ \textbf{result}\ UE \rhd\!\!\rhd \Box}$$

$$\boxed{\zeta, \chi \vdash UE \rhd\!\!\rhd \Box}$$

ζ, χ must be a context description and $[UE]_{C_s(\chi)}$ must be defined.

$$\frac{\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma,\ t}{\zeta, \chi \vdash T \rhd\!\!\rhd \Box}$$

$$\frac{\begin{array}{c}\zeta[\{\star : sig[SP]\}/A], \chi[\mathrm{id}_{sig[SP]}/A] \vdash T \rhd\!\!\rhd \Gamma,\ t \\ A \in \mathrm{dom}(\Gamma) \text{ and } \Gamma(A \to \star) = \iota_{sig[SP] \subseteq \mathrm{dom}(t)}; t\end{array}}{\zeta, \chi \vdash \lambda A : SP \bullet T \rhd\!\!\rhd \Box}$$

$$\boxed{\vdash UDS \rhd\!\!\rhd \zeta, \chi}$$

$[UDS]$ must be defined. Then ζ, χ is a context description and $[UDS] = C_s(\chi)$.

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

$$\frac{\vdash UDS \rhd\!\!\rhd \zeta, \chi}{\vdash UDS\ X : SP \rhd\!\!\rhd \zeta[\{\star : sig[SP]\}/X], \chi[\mathrm{id}_{sig[SP]}/X]}$$

$$\frac{\vdash UDS \rhd\!\!\rhd \zeta, \chi}{\vdash UDS\ X : SP \to SP' \rhd\!\!\rhd \zeta, \chi[sig[SP] \to sig[SP']/X]}$$

$$\frac{\begin{array}{c}\vdash UDS \rhd\!\!\rhd \zeta, \chi \\ \zeta, \chi \vdash T \rhd\!\!\rhd \Gamma,\ t \\ (j : \Gamma(\star) \to \Sigma', \tau : sig[SP] \to \Sigma') \text{ is a pushout of } (t, \iota_{\Delta \subseteq sig[SP]}) \text{ and } j \in \mathcal{J}\end{array}}{\vdash UDS\ X : SP\ \mathbf{given}_\Delta\ T \rhd\!\!\rhd \zeta[(\Gamma; j)/X], \chi[\tau/X]}$$

$$\boxed{\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma,\ t}$$

ζ, χ must be a context description and $[T]_{C_s(\chi)}$ must be defined. Then Γ is a full $\mathcal{J}$-context fitting ζ, χ and $t : [T]_{C_s(\chi)} \to \Gamma(\star)$.

$$\frac{}{\zeta, \chi \vdash A \rhd\!\!\rhd \zeta(A) + \{A : \zeta(A)(\star), \mathrm{id}_{\zeta(A)(\star)} : A \to \star\},\ \chi(A)}$$

$$\frac{\begin{array}{c}\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma_0,\ t_0 \\ \chi(A) = \mathrm{dom}(t_0) \to \Delta \\ (j : \Gamma_0(\star) \to \Sigma,\ t : \Delta \to \Sigma) \text{ is a pushout of } (t_0, \iota_{\mathrm{dom}(t_0) \subseteq \Delta}) \text{ and } j \in \mathcal{J}\end{array}}{\zeta, \chi \vdash A(T) \rhd\!\!\rhd (\Gamma_0; j),\ t}$$

$$
\frac{\begin{array}{c}
\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I} \\
\zeta, \chi \vdash T_i \rhd\!\!\rhd \Gamma_i,\ t_i \text{ for all } i \in I \\
\Gamma = \bigcup_{i \in I} \Gamma_i \text{ and the relevant transformation is } \{\tau_i : \Gamma_i(\star) \to \Gamma(\star)\}_{i \in I} \\
\textbf{either} \\
D : O \to \mathit{Sig} \text{ is a witness of } \sigma\text{'s regularity} \\
\text{for all } m_1 : o \to i_1 \text{ and } m_2 : o \to i_2 \text{ in } O \text{ with } i_1, i_2 \in I, \\
\text{we have } D(m_1); t_{i_1}; \tau_{i_1} = D(m_2); t_{i_2}; \tau_{i_2} \\
t : \Delta \to \Gamma(\star) \text{ is universal for the colimit of } D \text{ defined by } \sigma \\
\text{and the cocone over } D \text{ defined by } \{t_i; \tau_i : \Delta_i \to \Gamma(\star)\}_{i \in I} \\
\textbf{or} \\
\{\gamma : \Delta \to \Sigma', \iota : \Gamma(\star) \to \Sigma'\} \text{ define a colimit} \\
\text{of } \{\sigma_i : \Delta_i \to \Delta, (t_i; \tau_i) : \Delta_i \to \Gamma(\star)\}_{i \in I} \\
\iota \text{ is an isomorphism and } t = \gamma; \iota^{-1}
\end{array}}{\zeta, \chi \vdash \bigcup_{i \in I} \sigma_i(T_i) \rhd\!\!\rhd \Gamma,\ t}
$$

$$
\frac{\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma_0,\ t_0}{\zeta, \chi \vdash T|_{\sigma : \Delta_0 \to \Delta} \rhd\!\!\rhd \Gamma_0,\ (\sigma; t_0)}
$$

As promised, we first show that the rules are essentially deterministic, which is clear except for the rule for amalgamation, where one can choose between the **either** and **or** branches. But if one makes some rudimentary assumptions which guarantee that both these branches potentially could be used, that is, for the **either** branch the assumption that the amalgamation σ is regular, and for the **or** branch the assumption that the colimit (γ, ι) exists, then both branches turn out equivalent. This is shown in the following proposition:

Proposition 19. *Assume that ζ, χ is a context description, $[\bigcup_{i \in I} \sigma_i(T_i)]_{C_s(\chi)}$ is defined, and that the first three premises of the rule for amalgamation hold. Assume also that σ is regular and that there exists a colimit of $\{\sigma_i : \Delta_i \to \Delta, (t_i; \tau_i) : \Delta_i \to \Gamma(\star)\}_{i \in I}$. Then, independent of the choice of regularity witness, universal morphism and colimit, the* **either** *premises hold iff the* **or** *premises hold, and the resulting morphism t is the same in both cases.*

Proof. Suppose the three **either** premises hold. Let $\gamma : \Delta \to \Sigma'$, $\iota : \Gamma(\star) \to \Sigma'$ be the colimit of $\{\sigma_i : \Delta_i \to \Delta, (t_i; \tau_i) : \Delta_i \to \Gamma(\star)\}_{i \in I}$; by assumption, the colimit exists. For any $i \in I$, we have $\sigma_i; \gamma = t_i; \tau_i; \iota = \sigma_i; t; \iota$. Because σ defines a colimit and so is jointly epi, this implies that $\gamma = t; \iota$, so $t; \iota$ and ι form a colimit. We claim that t and $\mathrm{id}_{\Gamma(\star)}$ form a colimit as well (they certainly form a cocone). Assume $\alpha : \Delta \to \Sigma''$ and $\beta : \Gamma(\star) \to \Sigma''$ form a cocone over the considered diagram. Then there is a (unique) morphism

$a : \Sigma' \to \Sigma''$ such that $\alpha = t; \iota; a$ and $\beta = \iota; a$. Let $h = \beta : \Gamma(\star) \to \Sigma''$. Then $\alpha = t; h$ and $\beta = \mathrm{id}_{\Gamma(\star)}; h$. Moreover, h is unique with these properties, since if $h' : \Gamma(\star) \to \Sigma''$ satisfies $\alpha = t; h'$ and $\beta = \mathrm{id}_{\Gamma(\star)}; h'$, then $h = \beta = h'$. This proves that t and $\mathrm{id}_{\Gamma(\star)}$ is a colimit. Since ι factors one colimit through another colimit, it must be an isomorphism. Moreover, $\gamma; \iota^{-1} = t; \iota; \iota^{-1} = t$, as required.

Suppose now that both **or** premises hold. Let $D : O \to \mathit{Sig}$ be a witness of σ's regularity. Take any $m_1 : o \to i_1$ and $m_2 : o \to i_2$ in D. We then have $D(m_1); t_{i_1}; \tau_{i_1} = D(m_1); t_{i_1}; \tau_{i_1}; \iota; \iota^{-1} = D(m_1); \sigma_{i_1}; \gamma; \iota^{-1}$, where the latter equation holds, since γ and ι form a cocone and so $\sigma_i; \gamma = t_i; \tau_i; \iota$. This is further equal $D(m_2); \sigma_{i_2}; \gamma; \iota^{-1}$, since σ forms a cocone over D. This in turn equals $D(m_2); t_{i_2}; \tau_{i_2}$. It is easy to see that $\gamma; \iota^{-1}$ factors the colimit defined by $\{t_i; \tau_i : \Delta_i \to \Gamma(\star)\}_{i \in I}$ through the colimit defined by σ, since $\sigma_i; (\gamma; \iota^{-1}) = (t_i; \tau_i; \iota); \iota^{-1} = t_i; \tau_i$ for all $i \in I$. Therefore, by the uniqueness of the the universal morphism, we have $t = \gamma; \iota^{-1}$. $\square$

Fully static checking is said to be *runnable* on ASP in $\mathcal{L}_s$ and $\mathcal{J}$ if the signature category of $\mathcal{L}_s$ is $\mathcal{J}$-cocomplete and either all amalgamations in ASP are regular, or the signature category of $\mathcal{L}_s$ is cocomplete. Runnability is defined analogously for declarations, expressions and terms. The notion of runnability ensures that fully static checking will not fail due to lack of sufficient cocompleteness. This notion is used in propositions in which an equivalence between fully static correctness and some other property is stated, since otherwise the implication from that property to fully static correctness would, in general, not hold.

A fundamental result on fully static correctness is:

Theorem 20. *Assume ASP is generative and statically correct in a static language $\mathcal{L}_s$ and that fully static checking is runnable on ASP in $\mathcal{L}_s$ and $\mathcal{J}$. Then*

ASP is fully static correct in $\mathcal{L}_s$ and $\mathcal{J}$

iff

ASP, if parametric, uses its $\lambda-$argument

and

for any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ such that ASP is partially correct in $\mathcal{L}$,

for any programming language $\mathcal{P}$ compatible with $\mathcal{L}$ and with amalgamation,

ASP is partially correct in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$

This theorem states that fully static correctness is exactly the notion we were looking for. First, it guarantees that a specification partially correct in $\mathcal{L}$ will remain partially correct in $\Pr(\mathcal{L}, \mathcal{P})$, for any programming language $\mathcal{P}$ with amalgamation. This is the, most crucial, implication downwards (a form of soundness). Second, it says that lack of fully static correctness implies, in fact, that in some cases partial correctness is *not* preserved in transition from models to programs (a form of completeness); for architectural specifications which return parametric units, this form of completeness is weakened by the additional assumption that ASP uses its λ-argument.

The, more important, downward implication holds even without the assumption of runnability of fully static checking. Such a strengthened form of implication is used, towards the end of this section, in Theorem 28, where a result analogous to Theorem 20 is proved for refinement statements.

The proof of Theorem 20 uses two very important propositions, namely Propositions 21 and 25. Both use a *stripping* operation $|\_|$ to strip any architectural specification, declaration or expression of all its specifications; more precisely, $|\_|$ replaces any specification SP in the given language construct by the total specification $\top_{sig[SP]}$ (the obvious formal definition is omitted). It is clear that this operation preserves the static semantics and the fully static correctness of all constructs.

Recall that $np(\_)$ denotes the set of names of non-parametric units appearing freely in a syntactic construct. Similarily, for any architectural specification, declaration, expression or term, $p(\_)$ denotes the set of all parametric unit names applied in the given syntactic construct. Note that solely declaring the name does *not* suffice; it has to be actually applied to some argument. For any declaration UDS, by $\overline{p}(UDS)$, on the other hand, the set of all parametric names declared and *not* applied in UDS is denoted.

The first of the mentioned propositions does not even refer to fully static checking. It states that to be certain that a generative architectural specification ASP is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$, it suffices that, first, all amalgamations in $|ASP|$ and, hence, in ASP are correct in $\Pr(\mathcal{L}, \mathcal{P})$ and, second, all applications in ASP are correct in $\mathcal{L}$ (instead of $\Pr(\mathcal{L}, \mathcal{P})$).

Proposition 21. *If $\mathcal{P}$ is a programming language compatible with $\mathcal{L}$, ASP is generative, $|ASP|$ is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$ and ASP is partially correct in $\mathcal{L}$, then ASP is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$.*

Proof. Here, we only provide the notions and lemmas needed to prove the proposition. Proofs of these lemmas and a proof of the whole proposition using the lemmas are given in Appendix 2.

In the proof, in the notation $[\![P]\!]$, where P is a program, the subscript Σ of the signature of P will be omitted. Also, this notation is extended to the case where $P = \bot$; in such a case it is understood that $[\![\bot]\!] = \bot$.

The most important notion in the proof is that of a *model* of an environment ϱ in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$. The model is a "corresponding" environment in $\mathcal{L}$. For non-parametric units, it is clear what this correspondence means. For parametric ones, certain problems may arise. This is formalized and discussed below.

For ϱ partially fitting C_s in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$, an environment ϱ' partially fitting C_s in $\mathcal{L}$ is a *model* of ϱ, if, for all unit names X in C_s:

- $\varrho'(X) = [\![\varrho(X)]\!]$, if X is non-parametric,
- if $C_s(X) = \Sigma \to \Sigma'$, then:
  - for all Σ-programs P, $[\![P]\!] \in \mathrm{dom}(\varrho'(X))$ iff $P \in \mathrm{dom}(\varrho(X))$,
  - for all $P \in \mathrm{dom}(\varrho(X))$, there exists $Q \in \mathrm{dom}(\varrho(X))$ such that $[\![P]\!] = [\![Q]\!]$ and $\varrho'(X)([\![P]\!]) = [\![\varrho(X)(Q)]\!]$.

The environment ϱ' is an *exact model* if, additionally, for all parametric X in C_s, $M \in \mathrm{dom}(\varrho'(X))$ implies that $M = \bot$ or M is implementable.

A unit $F \in \mathbf{Unit}_\bot(\Sigma \to \Sigma')$ in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ is *domain-stable* if, for all Σ-programs P and Q such that $[\![P]\!] = [\![Q]\!]$, if $P \in \mathrm{dom}(F)$, then $Q \in \mathrm{dom}(F)$; it is *stable* if, for all such P and Q, additionally $[\![F(P)]\!] = [\![F(Q)]\!]$. An environment ϱ partially fitting C_s in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ is *domain-stable* if, for all parametric X in C_s, the unit $\varrho(X)$ is domain-stable; it is stable if, for all parametric X in C_s, the unit $\varrho(X)$ is stable.

The following connection between models and stability may be established:

Lemma 22. *Let ϱ partially fit C_s in* $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$. *Then:*

- *ϱ has a model iff ϱ is domain-stable,*
- *ϱ has an exact model iff ϱ has a model,*
- *ϱ has exactly one exact model iff ϱ is stable.* □

Let ν be a set of parametric unit names in C_s and let ϱ partially fit C_s in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$. Models ϱ_1 and ϱ_2 of ϱ are *ν-equivalent* if they coincide on non-parametric units and, for any parametric X in C_s, if $X \in \nu$ then $\varrho_1(X)$ and $\varrho_2(X)$ differ only on unimplementable models. Note that ν-equivalence is an equivalence relation on all models of ϱ.

The intuition throughout, and especially in premise (6) of Lemma 23 below, is that if one comes up with a certain model ϱ of an environment ϱ^P that is an element of the denotation of *UDS* in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$, then even though ϱ itself need not be an element of the denotation of *UDS* in $\mathcal{L}$, there exists a model η which is such an element and does not differ too much from ϱ; more precisely, it is the same on non-parametric unit names, and on parametric unit names, if they come from ν, it differs only slightly, namely on unimplementable models. The idea is that in ν all parametric units may be found which have not been yet applied. Because of generativity, the change on already applied units is not dangerous, since they cannot be applied any more.

Thanks to the partial model semantics, it is possible to assume that all parametric units are total functions. Formally, for any ϱ partially fitting C_s in $\Pr(\mathcal{L},\mathcal{P})$, the *totalization* of ϱ, written $\top(\varrho)$, is a map such that, for any unit name X in C_s:

- if X is non-parametric, then $\top(\varrho)(X) = \varrho(X)$,
- if $C_s(X) = \Sigma \to \Sigma'$, then $\top(\varrho)(X) =$

$$\lambda P \in \mathit{Prog}(\Sigma) \cup \{\bot\} \cdot \text{ if } P \in \mathrm{dom}(\varrho(X)) \text{ then } \varrho(X)(P) \text{ else } \bot$$

Note that $\top(\varrho)$ partially fits C_s in $\Pr(\mathcal{L},\mathcal{P})$.

The proof of the whole proposition then rests on the following two lemmas, proved by a lengthy induction in Appendix 2.

Lemma 23. *Assume that ν is a set of parametric unit names in C_s and that $[T]_{C_s}$ is defined. Assume further that ϱ^P partially fits C_s in $\Pr(\mathcal{L},\mathcal{P})$ and that the context E partially fits C_s in $\mathcal{L}$.*

If

1. *T is generative,*
2. *$p(T) \subseteq \nu$,*
3. *ϱ^P is domain-stable,*
4. *$[\![T]\!]^{\bot}_{\top(\varrho^P)}$ is defined,*
5. *$[\![T]\!]^{\bot}_{\eta}$ is defined for all $\eta \in E$, and*
6. *for any model ϱ of ϱ^P, there exists a ν-equivalent model $\eta \in E$,*

then

(*) $[\![T]\!]^{\bot}_{\varrho^P}$ is defined and

$$[\![T]\!]^{\bot}_{\varrho^P} = [\![T]\!]^{\bot}_{\top(\varrho^P)}$$

(**) for any model ϱ of ϱ^P, there exists a $(\nu \setminus p(T))$-equivalent model $\eta' \in E$ such that, for any subterm T' of T,

$$[\![[\![T']\!]^{\perp}_{\varrho^P}]\!] = [\![T']\!]^{\perp}_{\eta'}$$

$\square$

Lemma 24. *Assume UDS is generative, $[UDS]$ is defined, $C = [\![|UDS|]\!]^{\perp}$ is defined in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ and $E = [\![UDS]\!]^{\perp}$ is defined in $\mathcal{L}$.*

Then $C^P = [\![UDS]\!]^{\perp}$ is defined in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$.

Assume, additionally, that $\varrho^P \in C^P$. Then:

1. *ϱ^P is domain-stable,*
2. *$\top(\varrho^P) \in C$,*
3. *for any model ϱ of ϱ^P, there exists a $\overline{p}(UDS)$-equivalent model $\eta \in E$.* $\square$

$\square$

The second proposition needed to prove Theorem 20 is a characterization of fully static correctness in terms of the stripping operation $|\_|$. The idea here is that partial correctness of $|ASP|$ is really equivalent to the correctness of all amalgamations; the proposition states that, if we quantify over all specification languages with amalgamation, this is exactly what fully static correctness means. The proposition then additionally states that, instead of the quantification, one may consider just one specification language with amalgamation, provided that it has iso-reflection and consistent signatures. This additional statement is not needed in the proof of Theorem 20, but will be used later, when Theorem 29 is stated.

Proposition 25. *Assume ASP is generative and statically correct in a static language $\mathcal{L}_s$ and that fully static checking is runnable on ASP in $\mathcal{L}_s$ and $\mathcal{J}$. Then*

ASP is fully statically correct in $\mathcal{L}_s$ and $\mathcal{J}$

iff

for any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation,

$|ASP|$ is partially correct in $\mathcal{L}$

Moreover, if the signature category of $\mathcal{L}_s$ is cocomplete and $\mathcal{L}_0$ is a specification language with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation, iso-reflection and consistent signatures, then the above statements both hold iff $|ASP|$ is partially correct in $\mathcal{L}_0$.

Proof. As with the proof of the previous proposition, here only the concepts and lemmas used are presented. The proofs of these lemmas and the proof of the proposition using these lemmas may be found in Appendix 3.

Any full $\mathcal{J}$-context Γ defines a static context *induced by* Γ, written $C_s(\Gamma)$, namely the one taking any unit name $X \in \mathrm{dom}(\Gamma)$ to the signature $\Gamma(X)$. A $\Gamma(\star)$-model M Γ-*represents* an environment η partially fitting $C_s(\Gamma)$ if $\eta(X) = M|_{\Gamma(X \to \star)}$ for all X in $C_s(\Gamma)$.

Let ζ, χ be a context description. An environment η *satisfies* ζ if $\mathrm{dom}(\eta) = \mathrm{dom}(\zeta)$ and, for all $X \in \mathrm{dom}(\eta)$, if $\eta(X) \neq \perp$, then $\eta(X)$ is a $\zeta(X)(\star)$-model which $\zeta(X)$-represents $\eta|_{\mathrm{dom}(\zeta(X))}$; in other words, if $\eta(X) \neq \perp$ then, for any $Y \in \mathrm{dom}(\zeta(X))$, $\eta(Y) = \eta(X)|_{\zeta(X)(Y \to \star)}$ must hold.

An environment η satisfying ζ χ-*represents* an environment ϱ if ϱ partially fits $C_s(\chi)$ and, for any $X \in \mathrm{dom}(\eta)$, we have $\varrho(X) = \eta(X)|^{\perp}_{\chi(X)}$. Note that, for any η satisfying ζ, there exists a *minimal* environment ϱ χ-represented by η; namely, for any $X \in \mathrm{dom}(\chi)$, if X is non-parametric, then $\varrho(X) = \eta(X)|^{\perp}_{\chi(X)}$, and if X is parametric, then $\varrho(X)$ is the total unit constantly equal $\perp$.

A parametric unit $F \in \mathbf{Unit}_{\perp}(\Sigma \to \Sigma')$ is *total* if $\mathrm{dom}(F) = \mathbf{Mod}_{\perp}(\Sigma)$. An environment ϱ partially fitting C_s is *total* if, for all parametric X in C_s, $\varrho(X)$ is total. A context C partially fitting C_s is *total* if all environments in C are total.

The proof of the proposition is based on the following two lemmas, which show the equivalence of fully static checking **(FS)** and partial model semantics **(PM)**. Under additional assumptions, these are both equivalent to an existential version of the partial model semantics **(EPM)**.

Lemma 26. *Let $\mathcal{L}_s$ be a static language.*

Assume T is generative, ζ, χ is a context description such that $[T]_{C_s(\chi)}$ is defined in $\mathcal{L}_s$ and that fully static checking is runnable on T in $\mathcal{L}_s$ and $\mathcal{J}$.

The following conditions are equivalent:

(FS) $\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma, t$ *in $\mathcal{L}_s$ and $\mathcal{J}$,*

(PM) *for any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation, for any total environment ϱ χ-represented by an environment η satisfying ζ, $[\![T]\!]^{\perp}_{\varrho}$ is defined in $\mathcal{L}$.*

Moreover, if **(FS)** *holds, $(\mathcal{L})_s = \mathcal{L}_s$, $\mathcal{L}$ has amalgamation and η satisfies ζ, then:*

$$\{[\![T]\!]^{\perp}_{\varrho} \mid \varrho \textit{ is total and } \chi\textit{-represented by } \eta\} \setminus \{\perp\} \\ = \{M|_t \mid M \ \Gamma\textit{-represents } \eta|_{\mathrm{dom}(\Gamma)}\}$$

If the signature category of $\mathcal{L}_s$ is cocomplete and $\mathcal{L}_0$ is a language with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation and iso-reflection, then **(FS)** *and* **(PM)** *are also equivalent to:*

(EPM) *for any total environment ϱ χ-represented by an environment η satisfying ζ, $[\![T]\!]^{\perp}_{\varrho}$ is defined in $\mathcal{L}_0$.* $\square$

For any static context C_s, we write $np(C_s)$ for the static context C_s restricted to non-parametric unit names. For any environment ϱ partially fitting C_s, let $np(\varrho)$ denote the environment ϱ restricted to non-parametric unit names in C_s; then $np(\varrho)$ partially fits $np(C_s)$. For any context C partially fitting C_s, let $np(C)$ consist of environments of the form $np(\varrho)$, with $\varrho \in C$; then $np(C)$ partially fits $np(C_s)$.

A context description ζ, χ *describes* a context C partially fitting $C_s(\chi)$, if $np(C)$ consists of all environments of the form $np(\varrho)$, where ϱ is an environment χ-represented by some η satisfying ζ.

Let C_s be a static context and ν a set of parametric unit names in C_s. A total context C partially fitting C_s is *ν-volatile* if, for any $\varrho \in C$, any unit name $X \in \nu$ and any total unit $F \in \mathbf{Unit}_{\perp}(C_s(X))$, we have $\varrho[F/X] \in C$.

Lemma 27. *Let $\mathcal{L}_s$ be a static language.*

If UDS is generative, $[UDS]$ is defined in $\mathcal{L}_s$ and fully static checking is runnable on UDS in $\mathcal{L}_s$ and $\mathcal{J}$, then the following conditions are equivalent:

(FS) $\vdash UDS \rhd\!\!\rhd \zeta, \chi$ *in $\mathcal{L}_s$ and $\mathcal{J}$,*

(PM) *for any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation, $C = [\![|UDS|]\!]^{\perp}$ is defined in $\mathcal{L}$.*

Moreover, if **(FS)** *holds and $\mathcal{L}$ and C are as in* **(PM)**, *then C is total and $\overline{p}(UDS)$-volatile and ζ, χ describes C.*

If the signature category of $\mathcal{L}_s$ is cocomplete and $\mathcal{L}_0$ is a language with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation and iso-reflection, then **(FS)** *and* **(PM)** *are equivalent to:*

(EPM) *the context $C = [\![|UDS|]\!]^{\perp}$ is defined in $\mathcal{L}_0$.* $\square$

$\square$

Armed with Propositions 21 and 25, we may now prove Theorem 20.

Proof of Theorem 20. Assume ASP is generative and statically correct in a static language $\mathcal{L}_s$ and that fully static checking is runnable on ASP in $\mathcal{L}_s$ and $\mathcal{J}$.

"$\Downarrow$": Assume ASP is fully statically correct in $\mathcal{L}_s$ and $\mathcal{J}$ and that it is partially correct in $\mathcal{L}$, where $\mathcal{L}$ is a specification language with $(\mathcal{L})_s = \mathcal{L}_s$. Clearly, if ASP returns a parametric unit, then, because of the premise "$A \in \mathrm{dom}(\Gamma)$" in the rule for deriving the fully static correctness of λ-abstractions, it must use its λ-argument. Assume $\mathcal{P}$ is a programming language compatible with $\mathcal{L}$ and with amalgamation; note that the latter assumption means that $\Pr(\mathcal{L}, \mathcal{P})$ has amalgamation. By Proposition 25, the architectural specification $|ASP|$ is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$. Since, additionally, ASP is partially correct in $\mathcal{L}$, by Proposition 21 we conclude that ASP is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$.

"$\Uparrow$": Assume that, for any specification language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$, if ASP is partially correct in $\mathcal{L}$ and $\mathcal{P}$ is a programming language $\mathcal{P}$ compatible with $\mathcal{L}$ and with amalgamation, then ASP is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$.

Take any specification language $\mathcal{L} = (\mathit{Sig}, \mathcal{I}, \mathbf{Mod}, \mathit{Spec}, \mathit{sig}[\_], [\![\_]\!])$ with amalgamation and such that $(\mathcal{L})_s = \mathcal{L}_s$. By Proposition 25, to show that ASP is fully statically correct, it is enough to show that $|ASP|$ is partially correct in $\mathcal{L}$.

Let $\mathcal{L}'$ be such that $(\mathcal{L}')_s = \mathcal{L}_s$, which defines the signature and inclusion categories, the specifications Spec and the function $\mathit{sig}[\_]$, and let the model functor $\mathbf{Mod}_\bullet$ of $\mathcal{L}'$ be constantly equal $\{\bullet\}$, i.e., there be exactly one Σ-model, namely $\bullet$, for any signature Σ in Sig. Of course, all reducts are simply identities on $\{\bullet\}$. Let all specifications in $\mathcal{L}'$ be interpreted as total, i.e., for any Σ-specification SP, define $[\![SP]\!]_\bullet = \mathbf{Mod}_\bullet(\Sigma)$.

It is obvious that ASP is partially correct in $\mathcal{L}'$. To see this, observe that all applications must be correct, since all parametric units are total. Also, all amalgamations must be correct, since, in $\mathcal{L}'$, all amalgamations exist: the unique model with the required properties is always $\bullet$. Finally, if ASP returns a parametric unit, then the persistency requirement in the equation for λ-abstraction holds for arguments not equal $\perp$. If the argument is $\perp$, then the semantics of the term must be $\perp$ as well, since ASP uses its λ-argument.

Consider the programming language $\mathcal{P} = (\mathit{Sig}, \mathbf{Mod}_\bullet, \mathbf{Mod}, \{[\![\_]\!]_\Sigma\}_{\Sigma \in \mathit{Sig}})$, where the semantics $[\![P]\!]_\Sigma$ of any Σ-program P is simply $\bullet$. Of course, $\mathcal{P}$ is compatible with $\mathcal{L}'$ and has amalgamation, since $\mathcal{L}$ does. Since ASP is partially correct in $\mathcal{L}'$, it must be also, by assumption, be partially correct in $\Pr(\mathcal{L}', \mathcal{P})$. But the latter language is simply $(\mathit{Sig}, \mathcal{I}, \mathbf{Mod}, \mathit{Spec}, \mathit{sig}[\_], [\![\_]\!])$, where $[\![SP]\!] = \{M \in \mathbf{Mod}(\mathit{sig}[SP]) \mid [\![M]\!]_{\mathit{sig}[SP]} \in [\![SP]\!]_\bullet\}$, which is further equal to $\mathbf{Mod}(\mathit{sig}[SP])$, since the semantics in $\mathcal{P}$ of any M is the model $\bullet$ and the semantics in $\mathcal{L}'$ of any SP is the class $\{\bullet\}$. Thus, all specifications in $\Pr(\mathcal{L}', \mathcal{P})$ are total and, except for that, $\Pr(\mathcal{L}', \mathcal{P})$ is equal $\mathcal{L}$. Therefore, partial correctness of ASP is $\Pr(\mathcal{L}', \mathcal{P})$ is equivalent to partial correctness of $|ASP|$ in $\mathcal{L}$. Hence, $|ASP|$ is partially correct in $\mathcal{L}$, which completes the proof. $\square$

As already announced, Theorem 20 may be, in one direction, generalized to refinement statements, showing that fully static correctness guarantees that a correct development process on models induces a correct development process on programs. More precisely, if r is a refinement dag partially correct on the level of models, then fully static correctness of r guarantees that it is partially correct on the level of programs, for any programming language with amalgamation. This is a straightforward corollary of the following result:

Theorem 28. *Assume that $\mathcal{IT}$ and $\mathcal{OT}$ are tasks in a static language $\mathcal{L}_s$ such that, if they contain an architectural specification, then that specification is generative and fully statically correct in $\mathcal{L}_s$ and $\mathcal{J}$. If $\mathcal{L}$ is a specification language with $(\mathcal{L})_s = \mathcal{L}_s$ and*

$$\mathcal{IT} ::_\perp \mathcal{OT} \quad \textit{in} \quad \mathcal{L}$$

then, for any programming language $\mathcal{P}$ compatible with $\mathcal{L}$ and with amalgamation,

$$\mathcal{IT} ::_\perp \mathcal{OT} \quad \textit{in} \quad \mathrm{Pr}(\mathcal{L}, \mathcal{P})$$

Proof. In the proof, the downward direction of the equivalence stated in Theorem 20 is used, but in a slightly strengthened form, i.e., without the assumption that fully stating checking is runnable on the architectural specification ASP in $\mathcal{L}_s$ and $\mathcal{J}$. It is easy to check that, in the proof of Theorem 20, this assumption is only used when the opposite direction of the equivalence is proved. More precisely, this assumption is used in Lemma 26 in the proof that if **(PM)** holds then **(FS)** holds for the amalgamation construct; as a consequence, the assumption is also needed to show that **(PM)** implies **(FS)** in Lemma 27 and to show the upward implication in Proposition 25 and in Theorem 20.

It is obvious that, under the assumptions of this theorem, Theorem 20 in its strengthened form guarantees that, if $\mathcal{IT} ::_\perp \mathcal{OT}$ is (partially) correct in $\mathcal{L}$, then it is (partially) correct in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ as well.

In the proof, the case of an original task $\mathcal{OT}$ of the form $ASP.X$, where X is not declared with imports, need not be considered, since, with ASP partially correct in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$, it is equivalent to an original task of the form USP. Similarily, the case of an implementing task $\mathcal{IT}$ of the form USP need not be considered either, since an equivalent task of the form ASP may easily be constructed.

What remains are thus refinement statements $ASP ::_\perp USP$ and $ASP_1 ::_\perp ASP_2.X$, where X is declared with imports in ASP_2. Throughout, it is assumed that the refinement statements involved are all (partially) correct, in particular ASP, ASP_1 and ASP_2 are partially correct architectural specifications.

In the first case, two subcases need to be considered. If $ASP =$ **units** UDS **result** T and $USP = SP$, then $ASP ::_{\perp} USP$ holds iff $ASP' =$ **units** UDS $F : SP \to SP$ **result** $F(T)$ is partially correct, where F is a new unit name. Moreover, ASP' is fully statically correct and generative. If, on the other hand, $ASP =$ **units** UDS **result** $\lambda A : SP_A \bullet T$ and $USP = SP \to SP'$, then $ASP ::_{\perp} USP$ holds iff $SP ::_{\perp} SP_A$ holds and $ASP' =$ **units** UDS $A : SP_A$ $F : SP' \to SP'$ **result** $F(T)$ is partially correct, where F is a new unit name. Again, ASP' is fully statically correct and generative. Hence, if $ASP ::_{\perp} USP$ holds in $\mathcal{L}$, then ASP' is partially correct in $\mathcal{L}$ and, in the parametric case, $SP ::_{\perp} SP_A$ holds in $\mathcal{L}$. By Theorem 20 and, in the parametric subcase, Proposition 18, ASP' is partially correct in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ and, moreover, in the parametric case $SP ::_{\perp} SP_A$ holds in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$. Hence, $ASP ::_{\perp} USP$ holds in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$.

In the second case, let $ASP_1 =$ **units** UDS_1 **result** $\lambda A : SP_1 \bullet T_1$ and $ASP_2 =$ **units** UDS_2 $X : SP_2$ $\mathbf{given}_{\Sigma}$ T_2 UDS_2' **result** UE_2. Without loss of generality, it may be assumed that unit names in these architectural specifications are distinct. Consider the architectural specifications $ASP' =$ **result** UDS_2 $F : SP_1 \to SP_1$ **result** $F(T_2)$ and $ASP'' =$ **result** UDS_1 UDS_2 $A : \top_{\Sigma}$ $\mathbf{given}_{\Sigma}$ T_2 $F : SP_2 \to SP_2$ **result** $F(T_1)$, where F is a new unit name. Then $ASP_1 ::_{\perp} ASP_2.X$ holds iff ASP' and ASP'' are partially correct. Moreover, ASP' and ASP'' both are fully statically correct and generative. Hence, if $ASP_1 ::_{\perp} ASP_2.X$ holds in $\mathcal{L}$, then ASP' and ASP'' are partially correct in $\mathcal{L}$ and so, by Theorem 20, they are partially correct in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$. Therefore, $ASP_1 ::_{\perp} ASP_2.X$ holds in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$, which completes the proof. □

In the following theorem, a characterization of fully static correctness in terms of a single specification language with amalgamation is developed. It is shown that this characterization is valid iff the language has iso-reflection and consistent signatures. Apart from theoretical knowledge this theorem gives on architectural specifications, it has two practical applications. First, it is used in Section 15, in particular in Theorem 41, to translate the problem of checking so-called extended static correctness from a specification language without amalgamation to a language with amalgamation, iso-reflection and consistent signatures. Second, it shows that, in languages with iso-reflection and consistent signatures, fully static correctness has the expected intuitive meaning. Thanks to this, the effects of fully static checking will not look like "magic" to a person using them: if fully static correctness fails, it will be possible to see, within the specification language used, what went wrong.

Theorem 29. *Assume that $\mathcal{L}$ is a specification language with amalgamation and a signature category which is cocomplete and $\mathcal{J}$-cocomplete.*

The model functor of $\mathcal{L}$ has iso-reflection and consistent signatures iff, for any generative architectural specification ASP statically correct in $(\mathcal{L})_s$, the following conditions are equivalent:

- *ASP is fully statically correct in $(\mathcal{L})_s$ and $\mathcal{J}$,*
- *$|ASP|$ is partially correct in $\mathcal{L}$.*

Proof. "$\Leftarrow$": Suppose that there exists a non-isomorphism $\sigma_1 : \Delta_1 \to \Delta$ such that $\_|_{\sigma_1}$ is a bijection. Consider the architectural specification $ASP =$ **units** $A_1 : \top_{\Delta_1}$ **result** $\bigcup_{i\in\{1\}} \sigma_i(A_i)$

Clearly, $|ASP|$, which coincides with ASP, is partially correct, since its declarations evaluate to $C = \emptyset[\mathbf{Mod}_\perp(\Delta_1)/A_1]$ and, for all $\varrho \in C$ such that $\varrho(A_1) \neq \perp$, there exists a unique Δ_1-model N such that $N|_{\sigma_1} = \varrho(A_1)$.

At the same time, ASP is not fully statically correct, since, in the rule for amalgamation, one gets $\Gamma_1 = \{A_1 : \Delta_1, \star : \Delta_1, \mathrm{id}_{\Delta_1} : A \to \star\}$ and $t_1 = \mathrm{id}_{\Delta_1}$, which implies $\Gamma = \Gamma_1$ and $\tau_1 = \mathrm{id}_{\Delta_1}$, and so $(\mathrm{id}_{\Delta_1}, \sigma_1)$ defines a colimit over $\{\sigma_1 : \Delta_1 \to \Delta, \mathrm{id}_{\Delta_1} : \Delta_1 \to \Delta_1\}$. But, by assumption, σ_1 is not an isomorphism.

Suppose now that there exists a signature Σ with $\mathbf{Mod}(\Sigma) = \emptyset$. Consider the architectural specification $ASP =$ **units** $B : \top_\Sigma$ **result** $\lambda A : \top_\Sigma \bullet B$.

Of course, ASP is partially correct, since the only environment in the context defined by the declarations of ASP takes A to $\perp$ and so the defined parametric unit is a function taking the only element $\perp$ of its domain to $\perp$.

But, again, ASP is not fully statically correct, because the premise $A \in \mathrm{dom}(\Gamma)$ in the rule for λ-abstraction is not satisfied.

"$\Rightarrow$": This just the the "moreover" part of Proposition 25. □

11 Programs and Non-generativity

In the previous section, it has been assumed that the architectural specifications concerned are generative (which is equivalent to assuming that a generative partial model semantics is used). This was obviously necessary, since the fully static semantics, as defined there, does not keep track of consecutive applications of the same parametric unit. As a result, amalgamations of the form $F(T) + F(T)$ are, in general, not fully statically correct.

The idea might thus arise to produce a more general fully static semantics, one which would work for non-generative architectural specifications as well. The intention is that a non-generative counterpart of Theorem 20 should hold. Also, this more general version of the fully static semantics should be "static" at least in the sense that it would not depend on the specifications used in an architectural specification. In other words, this semantics should only, as is the case with the fully static semantics, depend on $|ASP|$.

The following proposition shows that the, discussed above, fully static semantics for non-generative architectural specifications cannot exist or, more precisely, that for such a semantics Theorem 20 cannot hold.

Proposition 30. *There exists a static language $\mathcal{L}_s$ with a cocomplete signature category, and non-parametric architectural specifications ASP_1 and ASP_2 statically correct in $\mathcal{L}_s$ and such that $|ASP_1| = |ASP_2|$ and:*

1. *for any specification language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$, if ASP_1 is partially correct in $\mathcal{L}$, then, for any programming language $\mathcal{P}$ compatible with $\mathcal{L}$ and with amalgamation, ASP_1 is partially correct in* $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$,

2. *there exists a specification language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ such that ASP_2 is partially correct in $\mathcal{L}$ and there exists a programming language $\mathcal{P}$ compatible with $\mathcal{L}$ and with amalgamation such that ASP_2 is not partially correct in* $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$.

Proof. Let the static language $\mathcal{L}_s$ be the static language of the specification language $\mathcal{PA}$, but instead of equational specifications, allow *conditional equational specifications*. Thus, the signature category of $\mathcal{L}_s$ is simply *AlgSig*, the inclusion category is $\mathcal{I}_{PA}$, and specifications consist of a signature Σ and a finite set of implications of the form $t_1 = t'_1, \ldots, t_n = t'_n \Rightarrow t = t'$, where $t_1 = t'_1, \ldots, t_n = t'_n, t = t'$ are Σ-equations. As mentioned in Section 3, *AlgSig* is cocomplete.

Let ASP_2 be:

units

$N : \top_{\Sigma_0}$

$A : \{\textbf{sort}\ nat\ \textbf{op}\ a : nat\}\ \textbf{given}_{\Sigma_0}\ N$

$B : \{\textbf{sort}\ nat\ \textbf{op}\ a : nat\}\ \textbf{given}_{\Sigma_0}\ N$

$F : \{\textbf{sort}\ nat\ \textbf{op}\ a : nat\} \to \{\textbf{sort}\ nat\ \textbf{ops}\ a, fa : nat\}$

$G : \{\textbf{sort}\ nat\ \textbf{ops}\ a, b, fa, fb : nat\ \textbf{axiom}\ a = b \Rightarrow fa = fb\} \to \top_\Sigma$

result

$G(F(A) + (F(B)|_{\textbf{sort}\ nat \mapsto nat\ \textbf{ops}\ b \mapsto a, fb \mapsto fa}))$

where Σ_0 is the signature $\{\textbf{sort}\ nat\}$ and Σ is the signature $\{\textbf{sort}\ nat\ \textbf{ops}\ a, b, fa, fb : nat\}$. Let ASP_1 be $|ASP_2|$, that is, be the same as ASP_2, except that the specification of the argument of G is the total specification $\top_\Sigma$. Obviously, ASP_1 and ASP_2 are statically correct in $\mathcal{L}_s$ and $|ASP_1| = |ASP_2|$.

Observe that the architectural specification ASP_1 is partially correct in any specification language $\mathcal{L}'$ with $(\mathcal{L}')_s = \mathcal{L}_s$ and amalgamation. This is because, in all applications, the domain of the applied unit is unrestricted and because the amalgamation is an amalgamation along the pushout of a colimit of the span ι, ι', where ι extends Σ_0 by the operations a and fa and ι' extends Σ_0 by the operations b and fb, and the Σ_0-model named N is always in the "source" vertex of the span. Therefore ASP_1 is partially correct, in particular, in the language $\mathcal{L}' = \Pr(\mathcal{L}, \mathcal{P})$, where $(\mathcal{L})_s = \mathcal{L}_s$ and $\mathcal{P}$ has amalgamation. This proves point (1).

Let $\mathcal{L}$ be a language such that $(\mathcal{L})_s = \mathcal{L}_s$, its model functor is **PAlg** and the semantics of specifications is defined as satisfaction of the conditional equations, where equations are interpreted strongly. The architectural specification ASP_2 is partially correct in $\mathcal{L}$. The correctness of the amalgamation is established as for ASP_1, since **PAlg** has amalgamation (see Section 3). The correctness of the applications of F is obvious, since the domain of F is unrestricted. As for the application of G, two cases arise. If, informally, $A = B$, then $F(A) = F(B)$ and so, in the argument of G, the equation $fa = fb$ holds, which makes the application correct. If $A \neq B$, then, in the argument of G, $a \neq b$ holds, which, again, makes the application correct.

Consider the programming language $\mathcal{PPA}$, introduced in Section 1; of course, $\mathcal{PPA}$ is compatible with $\mathcal{L}$. As mentioned in Section 3, it also has amalgamation. But ASP_2 is not partially correct in $\Pr(\mathcal{L}, \mathcal{P})$. To see this, consider an environment ϱ such that the unit $\varrho(A)$ takes a to the term 0, the unit $\varrho(B)$ takes a to the term $pred(succ(0))$, and the unit $\varrho(F)$, when fed $\varrho(A)$, takes fa to the term 0 and, when fed anything else, takes fa to $succ(0)$. Then the argument of G is a program P, i.e., a morphism into Σ_{nat}, in which a goes to 0, fa goes to 0, b goes to $pred(succ(0))$ and fb goes to $succ(0)$. Does P, as a program, satisfy the axiom $a = b \Rightarrow fa = fb$? That is, does the algebra $[\![P]\!]$ satisfy $a = b \Rightarrow fa = fb$? In the algebra $[\![P]\!]$, the carrier of nat is $\mathbb{N}$, the constants a, fa and b are all equal 0 and the constant fb equals 1. Thus $a = b$ holds and $fa = fb$ does not. This implies that the application of G, and with it the whole specification ASP_2, is not partially correct in $\Pr(\mathcal{L}, \mathcal{P})$. This proves point (2). $\square$

The architectural specification ASP_2 of the above proposition is also a counterexample to a non-generative version of Proposition 21, since $|ASP_2| = ASP_1$ is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$ and ASP_2 is partially correct in $\mathcal{L}$, but ASP_2 is not partially correct in $\Pr(\mathcal{L}, \mathcal{P})$.

A closer look at the architectural specification ASP_2 and the environment ϱ used in the proof reveals the real story behind the above failure: the problem lies in the parametric unit $\varrho(F)$. Observe that the semantics of A

and B are the same, $[\![\varrho(A)]\!] = [\![\varrho(B)]\!]$. But $\varrho(F)(\varrho(A))$ and $\varrho(F)(\varrho(B))$ have different semantics, since in the former fa is mapped to a term interpreted as the number 0, and in the latter to one interpreted as the number 1. In other words, the parametric unit $\varrho(F)$ is not stable.

It may be thus conjectured that a non-generative version of Proposition 21 and Theorem 20 would hold, if the semantics in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ was changed in such a way that only environments stable with respect to the language $\mathcal{L}$ were considered legal. Note, however, that this would require the introduction of yet another semantics; moreover, this semantics would have to be parametrized by a language $\mathcal{L}$ *and* a compatible programming language $\mathcal{P}$.

Of course, this does not mean that one cannot develop a useful notion of non-generative fully static correctness. One proposal of such a notion, made in [SMT$^+$05], is to consider two applications of F to T_1 and T_2 as giving the same result if T_1 and T_2 can *statically* be shown to be equal. This is certainly a reasonable restriction on the use of non-generativity and should not cause any problems in practice. However, this is a very superficial form of non-generativity and, in fact, one could claim that it treats non-generativity as syntactical sugar: with such a semantics, two applications $F(T_1)$ and $F(T_2)$ of $F : SP \to SP'$ are treated as a shortcut for saying that T_1 and T_2 are the same and so $F(T_1)$ and $F(T_2)$ are the same as well, but, in case the equality of T_1 and T_2 cannot be established statically, they are to be treated as $F_1(T_1)$ and $F_2(T_2)$, where $F_1 : SP \to SP'$ and $F_2 : SP \to SP'$. While such shortcuts may be of value in practice, when using architectural specifications, it seems that they do not involve any further insight and, therefore, they are not considered in this thesis. Because of the non-existence of an appropriate notion of fully static correctness for non-generative architectural specifications, such specifications are, in general, not considered either. Ideas on how verification problems may be solved in the non-generative case are described very informally in Section 21, which is mostly based on [Hof01].

12 Literature and Conclusions

Architectural specifications have been introduced as an element of the specification formalism CASL [BM04, CoF04b] following motivation and presentation in [BST98, BST02a]. The semantics used there is essentially the same as the static and model semantics described in Section 5: a non-generative semantics without partiality.

A notion of refinement based on architectural specifications has not been discussed in much detail in any published work. The problem has been mentioned in Section 9 of [BST02a], where the general idea is considered that specifications used in the **units** section of an architectural specification may be further refined by other architectural specifications, i.e., that a tree-like structure of refinements could be used. Then, however, it is argued that one could refine the architectural specification itself by refining any unit specification USP appearing in it to another unit specification USP'; so, for example, a declaration $X : SP$ $\mathbf{given}_\Sigma$ T may be refined to $X : SP'$ $\mathbf{given}_\Sigma$ T, if $[\![SP']\!] \subseteq [\![SP]\!]$. A theorem (Theorem 2) then states that, under the technical assumption that no import declaration in the original architectural specification ASP is inconsistent, the so obtained architectural specification ASP' is correct, if ASP was, and that $\mathcal{M}[ASP'] \subseteq \mathcal{M}[ASP]$. The mentioned theorem is, in effect, a very basic version of a flattening theorem such as Proposition 8; the flattening of a tree-like structure of refinements is not considered in it.

The notion of refinement has been thus only sketched in [BST02a] and no satisfactory solution of the problem of refining import declarations has been searched for there. However, as mentioned above, the very existence of import declarations forced the introduction of a technical assumption analogous to the flattenability assumption of Proposition 8. Note that an assumption of this form, requiring that certain parametric unit specifications are consistent, is very difficult to discharge; on the other hand, one might argue that there is no need of discharging it, since a development process based on an inconsistent specification cannot end with an actual implementation and so cannot cause any harm. Because the model semantics used in [BST02a] is not a partial one, this technical assumption could not have been given up, as has been done in Theorem 10.

In a yet unpublished paper, [MST04], the authors have proposed a much more developed notion of refinement. In the paper, refinements are allowed to have a tree-like structure and syntax for expressing this structure is introduced. For reasons which are not fully clear, refinements consist of what has been, in Section 6, called implementing and original tasks and of a morphism σ such that, for any model M of the implementing task, $M|_\sigma$ is a model of the original task (for this to work in the parametric case, the simplifying assumption is made that specifications of the parameters do not change). Refinement of import declarations is not considered in [MST04], except for the, somewhat imprecise, remark that import declarations may be transformed into parametric unit declarations. No flattening theorem is stated, but Theorem 2 of [BST02a] is cited as holding under the same technical assumption as before. This assumption could not have been dropped, since the model semantics used is not partial.

The treatment of refinement in Section 6 is thus the first full discussion of refinement via architectural specifications. A tree-like structure of refinements, refinement of import declarations and a flattenability theorem (Theorem 10) without any technical assumptions are developed. The last of these is possible, because the partial model semantics of architectural specifications has been adopted instead of the standard, non-partial model semantics. Of course, there are many other reasons for this choice of semantics and these reasons are considered below.

The problem of the relationship between architectural specifications, as defined on models, and the development process on programs has received even less attention in the literature than the problem of refinement, though, in the context of algebraic specification in general, this relationship has been discussed in [ST96]. The reason might have been that it seemed obvious that an architectural specification correct on the model level describes a correct development process on programs.

As the considerations of Sections 9-11 show, this is not the case.

The notion of a programming language adopted in Section 4 is similar to what may be found in [BST02a], though the treatment there is more general, the existence of a reduct functor on programs and of the equality $[\![P|_\sigma]\!] = [\![P]\!]|_\sigma$ not being postulated.

What is crucial, however, is that the implication between an architectural specification being a correct implementation on the model level, which may be expressed as $ASP ::_\perp \mathcal{OT}$ in $\mathcal{L}$, and the correctness of the development process on programs, which may be expressed as $ASP ::_\perp \mathcal{OT}$ in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$, has never been investigated. In fact, elements of the solution, such as amalgamation in the context of architectural specifications, have been the target of extensive research [SMT+01b, SMT+05]. The same is true for fully static correctness, a slightly more general version of which is called extended static correctness or semantics (see Section 15 and the literature in Section 22). A version even closer to fully static correctness may be found in [Tar03]. A concept similar to iso-reflection, called *definitional completeness*, has been considered in [SMT+05]. Nevertheless, a connection between these concepts and correctness of the development process on the level of programs, in the form found in Propositions 21 and 25 and Theorems 20, 28 and 29, has not been established before.

In my view, the following lessons are to be learned from the flattening theorem and the propositions and theorems regarding the relationship between the levels of models and of programs.

First, the partial model semantics of Section 9 should replace the standard, non-partial model semantics, as defined in Casl and Section 5. The fundamental difference between these semantics is that the partial one never declares an object correct "for free", simply because some previous declarations turned out inconsistent. The superiority of this approach becomes apparent, if it is considered that partiality is needed to ensure that a flattening theorem holds without reservations (Theorem 10) and to ensure that a refinement holding on models also holds on programs (Theorem 28). These two properties are of such a crucial importance that a semantics without them should be deemed unacceptable. Benefits are also to be found in the verification process, where, under the standard semantics, additional assumptions regarding the consistency of parametric unit specifications were required (compare the completeness theorems of Chapter III with those of [Hof01, Hof03]). Because partial model semantics is a more restrictive version of the standard model semantics (Proposition 14), any property guaranteed by standard correctness is guaranteed by partial correctness as well. Note that partial model semantics has been introduced by myself in [MHAH04], in a version where non-parametric units were not allowed to be represented by $\perp$; this version had, from a proof-theoretic point of view, certain benefits, but did not ensure that a flattening theorem holds without further assumptions.

Second, non-generative semantics should be replaced by a generative one. Moreover, the use of imports instead of parametric unit declarations should be encouraged. Proposition 30 shows that, for non-trivial cases of non-generativity, i.e., cases where it is more than just a syntactical short-cut, a reasonable concept of fully static correctness, which would guarantee preservation of correctness in the transition from models to programs, cannot be offered. From a methodological point of view, architectural specifications which, in a non-trivial way, make use of non-generativity, such as ASP_2 defined in the proof of Proposition 30, are not examples of good design. For these reasons, generativity should be enforced and imports, as a way of guaranteeing generativity, should be preferred over parametric unit declarations. An import declaration is, by definition, generative, since it implicitly declares a parametric unit and applies it exactly once; the same unit may never be applied again. A purely methodological reason for using import declarations is that they allow to split the process of defining a subtask in the development process into two separate stages. In the first stage, the developer just says for what kind of unit, represented by a term T, an extension is needed, and what specification SP that extension should satisfy. In the second stage, the developer has to decide which properties of the term T are needed to produce an efficient implementation of SP; these properties are then captured by a specification SP_0 and the import declaration is refined to a parametric unit specification $SP_0 \to SP$. Note that the second stage is an abstraction stage and, in fact, does not have to be performed when the architectural specification is written. If a parametric unit specification were to be used instead of the import declaration, the developer would have to perform both stages at once and, as a result, produce a declaration of the form $F : SP_0 \to SP$ and an application $F(T)$.

Third, fully static correctness should be enforced on architectural specifications.

This conclusion is based on the assumption that since, in general, it cannot be guaranteed that the transition from the level of models to the level of programs preserves correctness, the most reasonable restriction on the languages involved is the restriction to programming languages with amalgamation. The rationale behind such a choice is that this is the simplest and most natural restriction which guarantees that amalgamations in architectural specifications are correct. Moreover, many programming languages do, in fact, have the amalgamation property or, at least, the weak amalgamation property, as shown in Section 3 for the case of the programming language $\mathcal{PPA}$.

Having made this assumption, it is clear that only fully statically correct architectural specifications should be considered. The fundamental reason is that fully static correctness ensures, by Theorem 28, that the development process described by an architectural specification is correct for any programming language with amalgamation. Moreover, Theorem 20 shows that fully static correctness is, in fact, the weakest possible property that guarantees such an effect. Note that although, formally, the specification language itself need not have amalgamation, this is usually the case if a programming language compatible with it does. For, if the specification language does not have amalgamation and the programming language does, then either models, for which an amalgamation does not exist, are not implementable in the programming language, in which case they could be removed from the specification language as well, or these models have more than one potential amalgamation and exactly one of these is implementable. In this case, again, the unimplementable would-be amalgamations could be removed from the specification language.

This conclusion shows that the form of subsorting which appears in SubCasl is somewhat unfortunate, since SubCasl does not, in contrast to MultiCasl, have amalgamation. One way out of this problem would be to require all subsort embeddings to be set-theoretic inclusions; this choice, as explained in Section 2, has intentionally been rules out, inter alia because, in programming languages, subtypes quite often use different representations of objects than the supertypes, so that the embeddings are, in fact, not inclusions. Another way out is to, instead of SubCasl, use the language EnrichedCasl, which contains SubCasl with its subsorting features and has amalgamation. Problems regarding the embedding of SubCasl in a language with better properties, in particular EnrichedCasl, are discussed in Sections 17 and 18, in the context of verifying architectural specifications over specification languages without amalgamation, such as SubCasl.

Fourth, in view of Theorem 29, it is preferable for the specification language used to have iso-reflection and consistent signatures. If the language has iso-reflection, consistent signatures and amalgamation, then fully static correctness of an architectural specification is exactly equivalent to that architectural specification being partially correct after erasing all specifications in it to total specifications. In this case, fully static correctness turns out to be sound and complete with respect to a natural and intuitively easy to check notion. This is especially important, since, as already mentioned, it allows the person writing architectural specifications to control their fully static correctness, instead of finding out that, surprisingly, fully static checking failed. As mentioned in Section 3, the specification language SubCasl, in contrast to MultiCasl, does not have iso-reflection. As for iso-reflection in EnrichedCasl, see Sections 19 and 20.

The conclusion of the thorough analysis of architectural specifications and their properties performed in Chapter II must thus be that architectural specifications should be interpreted in the partial model semantics, required to be generative and fully statically correct, and that the specification language used should have amalgamation and, preferably, iso-reflection and consistent signatures. During the verification of architectural specifications, which is discussed in the following sections, it will be, therefore, first assumed that these properties hold. Later, some of them, especially those which do not hold for the specification language SubCasl, will be given up and verification without them will be considered.

Chapter III

Verification

13 Introduction

The main reason for formalizing a program development process and expressing it in the form of architectural specifications and refinement dags is that one can then *prove* that the process is correct. There are, of course, other reasons as well: formally writing down a proposed development process may be useful even if one does not intend to prove its correctness, for example since it forces one to present the design in an unambiguous way. Such reasons may even lead to the use of architectural specifications with informal, natural language specifications (axioms).

Proving that a refinement dag is correct is called its *verification*. Analogously, one may speak of verification of refinement statements and of architectural specifications. It is obvious that any verification method requires methods for checking whether statements of the form

$$[\![SP]\!] \subseteq [\![SP']\!]$$

hold. Thus, verification methods should be parametrized by a method for checking such statements. Moreover, only soundness (soundness and completeness) of the latter method can guarantee soundness (soundness and completeness) of verification. However, instead of assuming explicitly that a sound and complete method of checking these statements, i.e., a sound and complete entailment relation between specifications, is given, it seems more elegant to simply treat these statements, in their semantical form, as "primitive" operations, which may be "called" from any verification algorithm.

In fact, statements of a slighlty more complicated kind are needed in the verification methods considered in this thesis. By $\mathcal{SP}$, $\mathcal{SP}'$, etc. sets of specifications over a common signature Σ are denoted; implicitly, it is assumed that, even for an empty set, the signature Σ may be recovered. For any such set, $[\![\mathcal{SP}]\!]$ denotes the class of all Σ-models M such that $M \in [\![SP]\!]$, for all $SP \in \mathcal{SP}$. Note that, if $\mathcal{SP}$ is empty, then $[\![\mathcal{SP}]\!]$ is $\mathbf{Mod}(\Sigma)$. The notation $[\![\mathcal{SP}]\!]_\perp$ has an analogous meaning. By a *proof obligation* in a specification language, a statement of the form

$$[\![\mathcal{SP}]\!] \subseteq [\![SP]\!]$$

is understood, where $\mathcal{SP}$ is a set of specifications over the signature of SP. Typically, the set $\mathcal{SP}$ is finite, but this need not be the case if amalgamations along infinite sinks are considered.

The above form of proof obligation is a natural and practical choice. It is natural, since it asks one of the simplest possible questions about the specifications of a given specification language. It is practical, since checking whether such obligations hold has been an area of intensive research. This is true, of course, for "basic" logics, i.e., specification languages without structuring operations; but it is also true for languages of structured specifications [BCH99, Bor02].

Verification is considered only for refinement statements, since it is obvious how to verify refinement dags if one is given a method of verifying particular statements. For refinement statements, verification methods are presented as *proof calculi*, in which judgements of the form

$$\vdash^{calc} \mathcal{IT} ::_\perp \mathcal{OT}$$

may be derived; here $\_^{calc}$ is a decoration which says which of the calculi is meant. The calculi considered in this thesis are defined by rules written in a natural semantics style, similar to the rules for deriving fully static correctness. A crucial property of the rules used in this thesis is that they are, in a certain sense, deterministic or, more precisely, unambiguous. That is, any given judgement may have at most one derivation; though, of course, for a given $\mathcal{IT}$ there may be many tasks $\mathcal{OT}_i$ such that $\vdash^{calc} \mathcal{IT} ::_\perp \mathcal{OT}_i$ is derivable. Moreover, whether a judgement is derivable depends only on certain static properties, i.e., properties of signatures, and on whether the proof obligations "invoked" return true or false. This implies that, to any refinement statement which statisfies all the static properties, one may assign a set of proof obligations such that the refinement statement is correct iff the proof obligations in this set hold. Therefore, the proof calculi presented in the following sections may be treated as algorithms generating appropriate sets of proof obligations.

The applicability of a given proof calculus to a refinement statement may depend on the refinement statement having some properties, such as generativeness or fully static correctness. For this reason, a number of proof calculi is presented. The first one, denoted by $\vdash^f$, is applicable to generative and fully statically correct refinement statements in a language with amalganation. The second one, denoted by $\vdash^e$, is applicable to generative and extended statically correct refinement statements; extended static correctness is a generalized form of fully static correctness for languages without amalgamation. Note that this calculus is defined via a translation into the calculus $\vdash^f$, so that no new rules need to be defined. The third one, denoted by $\vdash^g$, is applicable to generative refinement statements, with certain restrictions on the amalgamations. Note that both $\vdash^e$ and $\vdash^g$ use representations of the concerned specification language in some other language. These representations are discussed in Section 17–20. Finally, in Section 21, the problem of verification for non-generative refinement statements is discussed.

In the respective sections, the above calculi are only defined for deriving judgements of the form $\vdash^{calc} ASP ::_\perp USP$. This is because there exists a general method of extending such a calculus to a calculus for all refinement statements. In order for this method to work, two preconditions have to be satisfied.

First, our general method "invokes" the original proof calculus on certain constructed architectural specifications. But, as has been said, it may happen that the original proof calculus works only under the assumption that it gets, as input, architectural specifications with a given property, e.g., fully static correctness. So, it is necessary that the extension of $\vdash^{calc}$ to other refinement statements is *closed* under this property, i.e., assuming that it gets, as input, architectural specifications with the required property, it will invoke the more primitive calculus only on architectural specifications with the same property.

Second, this method and, in fact, also the proof calculi presented in the following sections, require the considered specification language to have an important feature, namely *translations*. If SP is a Σ-specification and $\sigma : \Sigma \to \Delta$ a signature morphism, then a *translation of SP along σ*, written $\sigma(SP)$, is a Δ-specification such that

$$[\![\sigma(SP)]\!] = \{M \in \mathbf{Mod}(\Delta) \mid M|_\sigma \in [\![SP]\!]\}$$

A specification language *has translations* if all translations $\sigma(SP)$ exist. If $\mathcal{SP}$ is a set of Σ-specifications, then $\sigma(\mathcal{SP})$ denotes the set of translations along σ of all specifications in $\mathcal{SP}$. Implicitly, the assumption is made here that the translations are, in some sense, constructive or computable; this remark applies equally to operations on signatures and signature morphisms and, in reality, may fail (see remarks on extended static correctness in Section 15). A specification language with translations is very close to an institution, the only relaxation being that the equations $(\sigma;\tau)(SP) = \tau(\sigma(SP))$ and

$\mathrm{id}(SP) = SP$ need not hold, so that the translation of specifications (i.e., sentences of the institution) need not be functorial.

The *extension* of a proof calculus for refinement statements of the form $\vdash^{calc} ASP ::_\perp USP$ to a calculus for all statically correct refinement statements is defined by the following rules, in which it is assumed that the specification language concerned has translations. The notation $\vdash^{calc} ASP ::_\perp \square$ means $\vdash^{calc} ASP ::_\perp S$, where, for $ASP = \textbf{units } UDS \textbf{ result } UE$, $S = \top$ if $UE = T$ and $S = SP \to \top$ if $UE = \lambda A : SP \bullet T$. This notation is used to express the fact that ASP is partially correct.

$$\frac{[\![SP_1]\!] \subseteq [\![SP_2]\!]}{\vdash^{calc} SP_1 ::_\perp SP_2}$$

$$\frac{\begin{array}{c}[\![SP_2]\!] \subseteq [\![SP_1]\!] \\ [\![\{\iota_{sig[SP_1] \subseteq sig[SP'_1]}(SP_2), SP'_1\}]\!] \subseteq [\![SP'_2]\!]\end{array}}{\vdash^{calc} SP_1 \to SP'_1 ::_\perp SP_2 \to SP'_2}$$

$$\frac{\begin{array}{c}ASP = \textbf{units } UDS_1\ X : USP\ UDS_2 \textbf{ result } UE \\ \vdash^{calc} ASP ::_\perp \square \\ \vdash^{calc} \mathcal{IT} ::_\perp USP\end{array}}{\vdash^{calc} \mathcal{IT} ::_\perp ASP.X}$$

$$\frac{\begin{array}{c}ASP = \textbf{units } UDS_1\ X : SP \textbf{ given}_\Sigma\ T\ UDS_2 \textbf{ result } UE \\ \vdash^{calc} ASP ::_\perp \square \\ \vdash^{calc} \textbf{units } UDS_1\ F : SP_1 \to SP'_1 \textbf{ result } F(T) ::_\perp SP, \text{ where } F \text{ is new}\end{array}}{\vdash^{calc} SP_1 \to SP'_1 ::_\perp ASP.X}$$

$$\frac{\begin{array}{c}ASP = \textbf{units } UDS_1\ X : SP \textbf{ given}_\Sigma\ T\ UDS_2 \textbf{ result } UE \\ \vdash^{calc} ASP ::_\perp \square \\ \vdash^{calc} \textbf{units } UDS_1 \textbf{ result } T ::_\perp SP_1 \\ \text{the name } A \text{ and unit names in } UDS' \text{ are distinct from unit names in } UDS_1 \\ r \text{ is a bijective renaming of unit names such that } r(A) = A \\ UDS' \text{ and } T'_1 \text{ are results of applying the renaming } r \text{ to } UDS \text{ and } T_1 \\ \vdash^{calc} \textbf{units } UDS_1\ UDS'\ A : \top_\Sigma \textbf{ given}_\Sigma\ T \textbf{ result } T'_1 ::_\perp SP\end{array}}{\vdash^{calc} \textbf{units } UDS \textbf{ result } \lambda A : SP_1 \bullet T_1 ::_\perp ASP.X}$$

The following proposition shows that the proof calculus defined by the above extension is, indeed, sound and complete, if the original proof calculus was.

Proposition 31. *Assume that* Π *is a property of refinement statements in* $\mathcal{L}$ *and that* $\vdash^{calc}$ *is a sound and complete proof calculus for statically correct refinement statements satisfying* Π *and of the form* $ASP ::_\perp USP$*, i.e., such that* $\vdash ASP ::_\perp USP$ *holds in* $\mathcal{L}$ *iff* $\vdash^{calc} ASP ::_\perp USP$ *is derivable.*

If the specification language $\mathcal{L}$ *has translations and the extension of* $\vdash^{calc}$*, defined by the above rules, is closed under the property* Π*, then, for any statically correct refinement statement* $\mathcal{IT} ::_\perp \mathcal{OT}$ *in* $\mathcal{L}$ *satisfying* Π*,*

$$\mathcal{IT} ::_\perp \mathcal{OT} \text{ in } \mathcal{L}$$

$$\text{iff}$$

$$\vdash^{calc} \mathcal{IT} ::_\perp \mathcal{OT}$$

Proof. A number of cases has to be considered.

If the refinement statement has the form $ASP ::_\perp USP$, then the proposition holds, since it has been assumed that $\vdash^{calc}$ is sound and complete.

If the refinement statement has the form $SP_1 ::_\perp SP_2$, then it holds iff $\mathcal{M}^\perp[SP_1] \subseteq \mathcal{M}^\perp[SP_2]$, i.e., $[\![SP_1]\!] \subseteq [\![SP_2]\!]$, which in turn is equivalent to $\vdash^{calc} SP_1 ::_\perp SP_2$.

If the refinement statement has the form $SP_1 \to SP'_1 ::_\perp SP_2 \to SP'_2$, then it holds iff, for any $F \in [\![SP_1 \to SP'_1]\!]_\perp$, we have $F \in [\![SP_2 \to SP'_2]\!]_\perp$. Assume the statement is derivable. Consequently, the premises of the only rule to derive $SP_1 \to SP'_1 ::_\perp SP_2 \to SP'_2$ hold. Take any $F \in [\![SP_1 \to SP'_1]\!]_\perp$. Take any $M \in [\![SP_2]\!]_\perp$. Then $M \in [\![SP_1]\!]_\perp$ and so $M \in \mathrm{dom}(F)$. Also, $F(M) \in [\![SP'_1]\!]_\perp$ and either $F(M) = \perp \in [\![SP'_2]\!]_\perp$ or $F(M)|_{sig[SP_1]} = M \in [\![SP_2]\!]_\perp$; this implies $F(M) \in [\![SP'_2]\!]_\perp$. Hence, $F \in [\![SP_2 \to SP'_2]\!]_\perp$ and so the calculus is sound. Assume the statement holds. Take any $M \in [\![SP_2]\!]$. Let F be a unit constantly equal $\perp$ on its domain $[\![SP_1]\!]_\perp$. Obviously, $F \in [\![SP_1 \to SP'_1]\!]_\perp$ and so, since the refinement holds, $F \in [\![SP_2 \to SP'_2]\!]_\perp$. Thus, we must have $M \in \mathrm{dom}(F)$, i.e., $M \in [\![SP_1]\!]_\perp$. Therefore, $[\![SP_2]\!]_\perp \subseteq [\![SP_1]\!]_\perp$. Now take any $N \in [\![\iota_{sig[SP_1] \subseteq sig[SP'_1]}(SP_2)]\!] \cap [\![SP'_1]\!]$. Let F be a unit constantly equal $\perp$ except that $F(N|_{sig[SP_1]}) = N$. Such a unit is persistent and, since $N \in [\![SP'_1]\!]$, we have $F \in [\![SP_1 \to SP'_1]\!]_\perp$. Since the refinement holds, $F \in [\![SP_2 \to SP'_2]\!]_\perp$. Notice that $N|_{sig[SP_1]} \in [\![SP_2]\!]$. Hence, $F(N|_{sig[SP_1]}) \in [\![SP'_2]\!]_\perp$, which implies $N \in [\![SP'_2]\!]$. This implies that $[\![\iota_{sig[SP_1] \subseteq sig[SP'_1]}(SP_2)]\!] \cap [\![SP'_1]\!] \subseteq [\![SP'_2]\!]$ and so $\vdash^{calc} SP_1 \to SP'_1 ::_\perp SP_2 \to SP'_2$ is derivable.

Observe that, even in this simple case, $\vdash^{calc}$ would *not* have been complete in the non-partial model semantics.

Note that, clearly, if $ASP ::_{\perp} \Box$ satisfies Π, then ASP is partially correct iff $\vdash^{calc} ASP ::_{\perp} \Box$ is derivable. With this in mind, it is easy to see that in the remaining three cases, i.e., for refinement statements of the form $\mathcal{IT} ::_{\perp} ASP.X$, where X is declared without imports, and of the forms $USP ::_{\perp} ASP.X$ and $ASP' ::_{\perp} ASP.X$, where X is declared with imports, the thesis also holds. $\Box$

14 Verification Assuming Fully Static Correctness

In this section, a proof calculus $\vdash^{f}$ for generative and fully statically correct refinement statements is constructed. The specification language $\mathcal{L}$ is also required to have certain properties. First, it has to have translations, as in the previous section. Second, for the fully static correctness to be meaningful, it has to have amalgamation.

In Section 12, it has been concluded that languages with amalgamation, together with fully statically correct architectural specifications, should be considered the optimal framework for program development. Therefore, the task of finding a proof calculus $\vdash^{f}$ sound and complete in this framework is of fundamental importance. The existence of such a calculus shows that this choice of framework was right and that the framework may be used in practice.

The calculus $\vdash^{f}$ is very similar to the fully static semantics. One difference is that the correctness of amalgamations and of the λ-abstraction does not have to be checked any more, since it is guaranteed by the architectural specification being fully statically correct; on the other hand, in this calculus it is the applications that have to be checked for correctness. Another rather obvious modification is that information on unit specifications is recorded; for this reason, a new component μ has to be added to the context description, and a component $\mathcal{SP}$ has to be added to the results Γ, t of analyzing a term.

In the fully static semantics, the main idea was that the possible values of a term were described as reducts $\_|_t$ of a model of a "global" signature $\Gamma(\star)$. In the calculus $\vdash^{f}$, they are descibed as reducts $\_|_t$ of a model of a "global" specification $\mathcal{SP}$ (more precisely, set of specifications) over the signature $\Gamma(\star)$. In order to construct this set of specifications, it is necessary to have translations in the specification language $\mathcal{L}$.

Note that the statements found in the headers of rules for particular syntactic categories are, in fact, important inductive propositions. They will be intensively used in the proof of soundness and completeness of the calculus.

A *specification context description* is a triple ζ, χ, μ such that ζ, χ is a context description and μ takes any parametric X in $\mathrm{dom}(\chi)$ to a parametric unit specification $\mu(X)$ such that $sig[\mu(X)] = \chi(X)$ and any non-parametric X in $\mathrm{dom}(\chi)$ to a set $\mu(X)$ of specifications over the signature $\mathrm{dom}(\chi(X))$. Moreover, it is required that, if $X \in \mathrm{dom}(\zeta)$ and $Y \in \mathrm{dom}(\zeta(X))$, then $[\![\mu(X)]\!] \subseteq [\![\zeta(X)(Y \to \star)(\mu(Y))]\!]$. Informally, this requirement means that the "bigger" global specification $\mu(X)$ cannot loosen the requirements imposed by the "smaller" global specification $\mu(Y)$.

A full context Γ and a set $\mathcal{SP}$ of $\Gamma(\star)$-specifications are said to *fit* a specification context description ζ, χ, μ, if Γ fits the context description ζ, χ and, for all non-parametric X in $\mathrm{dom}(\chi)$, $[\![\mathcal{SP}]\!] \subseteq [\![\Gamma(X \to \star)(\mu(X))]\!]$. Again, this requirement informally means that constructed global specification $\mathcal{SP}$ cannot loosen the requirements imposed by the global specification $\mu(X)$ that was used to construct it.

The following rules are best read in parallel with the rules of the fully static semantics, presented on pages 63–65.

$$\boxed{\vdash^f ASP ::_\perp USP}$$

ASP must be statically and fully statically correct. Then $ASP ::_\perp USP$ is statically correct.

$$\frac{\begin{array}{c}\vdash^f UDS ::_\perp \zeta, \chi, \mu \\ \zeta, \chi, \mu \vdash^f UE ::_\perp USP\end{array}}{\vdash^f \textbf{units}\ UDS\ \textbf{result}\ UE ::_\perp USP}$$

$$\boxed{\zeta, \chi, \mu \vdash^f UE ::_\perp USP}$$

ζ, χ, μ must be a specification context description, $[UE]_{C_s(\chi)}$ must be defined and $\zeta, \chi \vdash UE \rhd\!\!\rhd \Box$ must hold. Then $sig[USP] = [UE]_{C_s(\chi)}$.

$$\frac{\begin{array}{c}\zeta, \chi, \mu \vdash^f T ::_\perp \Gamma,\ t,\ \mathcal{SP} \\ sig[SP] = \mathrm{dom}(t) \text{ and } [\![\mathcal{SP}]\!] \subseteq [\![t(SP)]\!]\end{array}}{\zeta, \chi, \mu \vdash^f T ::_\perp SP}$$

$$\frac{\begin{array}{c}\zeta[\{\star : \mathit{sig}[SP]\}/A], \chi[\mathrm{id}_{\mathit{sig}[SP]}/A], \mu[\{SP\}/A] \vdash^f T ::_\perp \Gamma,\ t,\ \mathcal{SP} \\ \mathit{sig}[SP_1] = \mathit{sig}[SP] \text{ and } [\![SP_1]\!] \subseteq [\![SP]\!] \\ \mathit{sig}[SP_2] = \mathrm{dom}(t) \text{ and } [\![\mathcal{SP}]\!] \subseteq [\![t(SP_2)]\!]\end{array}}{\zeta, \chi, \mu \vdash^f \lambda A : SP \bullet T ::_\perp SP_1 \to SP_2}$$

$$\boxed{\vdash^f\ UDS ::_\perp \zeta, \chi, \mu}$$

$[UDS]$ must be defined and $\vdash UDS \rhd\!\!\rhd \zeta, \chi$ must hold. Then ζ, χ, μ is a specification context description.

$$\frac{}{\vdash^f\ \epsilon ::_\perp \emptyset, \emptyset, \emptyset}$$

$$\frac{\vdash^f\ UDS ::_\perp \zeta, \chi, \mu}{\vdash^f\ UDS\ X : SP ::_\perp \zeta[\{\star : \mathit{sig}[SP]\}/X], \chi[\mathrm{id}_{\mathit{sig}[SP]}/X], \mu[\{SP\}/A]}$$

$$\frac{\vdash^f\ UDS ::_\perp \zeta, \chi, \mu}{\vdash^f\ UDS\ X : SP \to SP' ::_\perp \zeta, \chi[\mathit{sig}[SP] \to \mathit{sig}[SP']/X], \mu[SP \to SP'/X]}$$

$$\frac{\begin{array}{c}\vdash^f\ UDS ::_\perp \zeta, \chi, \mu \\ \zeta, \chi, \mu \vdash^f T ::_\perp \Gamma,\ t,\ \mathcal{SP} \\ (j : \Gamma(\star) \to \Sigma', \tau : \mathit{sig}[SP] \to \Sigma') \text{ is a pushout of } (t, \iota_{\Delta \subseteq \mathit{sig}[SP]}) \text{ and } j \in \mathcal{J}\end{array}}{\vdash^f\ UDS\ X : SP\ \mathbf{given}_\Delta\ T ::_\perp \zeta[(\Gamma; j)/X], \chi[\tau/X], \mu[j(\mathcal{SP}) \cup \{\tau(SP)\}/X]}$$

$$\boxed{\zeta, \chi, \mu \vdash^f T ::_\perp \Gamma,\ t,\ \mathcal{SP}}$$

ζ, χ, μ must be a specification context description, $[T]_{C_s(\chi)}$ must be defined and $\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma, t$ must hold. Then $\mathcal{SP}$ is a set of $\Gamma(\star)$-specifications and Γ and $\mathcal{SP}$ fit ζ, χ, μ.

$$\frac{}{\zeta, \chi, \mu \vdash^f A ::_\perp \zeta(A) + \{A : \zeta(A)(\star), \mathrm{id}_{\zeta(A)(\star)} : A \to \star\},\ \chi(A),\ \mu(A)}$$

$$\frac{\begin{array}{c}\zeta, \chi, \mu \vdash^f T ::_\perp \Gamma_0,\ t_0,\ \mathcal{SP}_0 \\ \mu(A) = SP \to SP' \\ [\![\mathcal{SP}_0]\!] \subseteq [\![t_0(SP)]\!] \\ (j : \Gamma_0(\star) \to \Sigma,\ t : \mathit{sig}[SP'] \to \Sigma) \text{ is a pushout of } (t_0, \iota_{\mathit{sig}[SP] \subseteq \mathit{sig}[SP']}) \text{ and } j \in \mathcal{J}\end{array}}{\zeta, \chi, \mu \vdash^f A(T) ::_\perp (\Gamma_0; j),\ t,\ j(\mathcal{SP}_0) \cup \{t(SP')\}}$$

$$
\frac{\begin{array}{c}
\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I} \\
\zeta, \chi, \mu \vdash^f T_i ::_\perp \Gamma_i,\, t_i,\, \mathcal{SP}_i \text{ for all } i \in I \\
\Gamma = \bigcup_{i \in I} \Gamma_i \text{ and the relevant transformation is } \{\tau_i : \Gamma_i(\star) \to \Gamma(\star)\}_{i \in I} \\
\textbf{either} \\
D : O \to Sig \text{ is a witness of } \sigma\text{'s regularity} \\
t : \Delta \to \Gamma(\star) \text{ is universal for the colimit of } D \text{ defined by } \sigma \\
\text{and the cocone over } D \text{ defined by } \{t_i; \tau_i : \Delta_i \to \Gamma(\star)\}_{i \in I} \\
\textbf{or} \\
\{\gamma : \Delta \to \Sigma', \iota : \Gamma(\star) \to \Sigma'\} \text{ define a colimit} \\
\text{over } \{\sigma_i : \Delta_i \to \Delta, (t_i; \tau_i) : \Delta_i \to \Gamma(\star)\}_{i \in I} \\
t = \gamma; \iota^{-1}
\end{array}}{\zeta, \chi, \mu \vdash^f \bigcup_{i \in I} \sigma_i(T_i) ::_\perp \Gamma,\ t,\ \bigcup_{i \in I} \tau_i(\mathcal{SP}_i)}
$$

$$
\frac{\zeta, \chi, \mu \vdash^f T ::_\perp \Gamma_0,\, t_0,\, \mathcal{SP}}{\zeta, \chi, \mu \vdash^f T|_{\sigma:\Delta_0 \to \Delta} ::_\perp \Gamma_0,\, (\sigma; t_0),\ \mathcal{SP}}
$$

The soundness and completeness of the calculus $\vdash^f$ is proved in two stages. First, it is proved for the calculus presented above, i.e., only for refinement statements of the form $ASP ::_\perp USP$. Then, in Theorem 35, it is proved for the extension to all forms of refinement statements.

Proposition 32. *If $\mathcal{L}$ is language with amalgamation and translations and a $\mathcal{J}$-cocomplete signature category, ASP is statically and fully statically correct in $(\mathcal{L})_s$ and $\mathcal{J}$ and is generative, then*

$$ASP ::_\perp USP \textit{ holds in } \mathcal{L}$$

$$\textit{iff}$$

$$\vdash^f ASP ::_\perp USP$$

Proof. Here, the proof is presented without proofs of two fundamental lemmas. These proofs may be found in Appendix 4. Lemma 33 is only used in the proof of Lemma 34; nevertheless, we state it here because of its importance.

Assume ζ, χ, μ is a specification context description. An environment η is said to *satisfy* ζ, μ, if it satisfies ζ and, for any X in $\mathrm{dom}(\zeta)$, $\eta(X) \in [\![\mu(X)]\!]_\perp$. An environment ϱ is said to be (χ, μ)*-represented* by η, if ϱ is χ-represented by η and, for any parametric X in $\mathrm{dom}(\chi)$, $\varrho(X) \in [\![\mu(X)]\!]_\perp$. Note that, for any η, an environment ϱ (χ, μ)-represented by η exists; an example is the minimal environment χ-represented by η.

Lemma 33. *Let $\mathcal{L}$ be a language with amalgamation and translations and a $\mathcal{J}$-cocomplete signature category.*

Assume T is generative, ζ, χ, μ is a specification context description such that $[T]_{C_s(\chi)}$ is defined in $\mathcal{L}$ and $\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma$, t holds in $(\mathcal{L})_s$ and $\mathcal{J}$.

The following conditions are equivalent:

(PC) $\zeta, \chi, \mu \vdash^f T ::_\perp \Gamma, t, \mathcal{SP}$,

(PM) *for any environment ϱ (χ, μ)-represented by an environment η satisfying ζ, μ, $[\![T]\!]^\perp_\varrho$ is defined in $\mathcal{L}$.*

Moreover, if **(PC)** *holds and η satisfies ζ, μ, then:*

$$\{[\![T]\!]^\perp_\varrho \mid \varrho \textit{ is } (\chi,\mu)\textit{−represented by } \eta\} \setminus \{\perp\}$$
$$= \{M|_t \mid M \in [\![\mathcal{SP}]\!]_\perp\ \Gamma\textit{−represents } \eta|_{\mathrm{dom}(\Gamma)}\}$$

□

A specification context description ζ, χ, μ and a set ν of parametric unit names in $\mathrm{dom}(\chi)$ *describe* a context C partially fitting $C_s = C_s(\chi)$, if the sets of all $\varrho|_{np(C_s)\cup\nu}$, for $\varrho \in C$ and, respectively, for ϱ (χ, μ)-represented by any environment η satisfying ζ, μ, coincide.

Lemma 34. *Let $\mathcal{L}$ be a language with amalgamation and translations and a $\mathcal{J}$-cocomplete signature category.*

Assume UDS is generative, $[UDS]$ is defined in $\mathcal{L}$ and $\vdash UDS \rhd\!\!\rhd \zeta, \chi$ holds in $(\mathcal{L})_s$ and $\mathcal{J}$. The following conditions are equivalent:

(PC) $\vdash^f UDS ::_\perp \zeta, \chi, \mu$,

(PM) *the context $[\![UDS]\!]^\perp$ is defined in $\mathcal{L}$.*

Moreover, if **(PC)** *holds, then ζ, χ, μ and $\overline{p}(UDS)$ describe $[\![UDS]\!]^\perp$.* □

Assume $\mathcal{L}$ has amalgamation and translations and a $\mathcal{J}$-cocomplete signature category, that ASP is statically and fully statically correct in $(\mathcal{L})_s$ and $\mathcal{J}$ and that it is generative.

Assume $ASP =$ **units** UDS **result** T, $USP = SP$ and $sig[SP] = [T]_{[UDS]}$; the latter two assumptions are clearly equivalent to $ASP ::_\perp USP$ being statically correct. It is obvious that $ASP ::_\perp USP$ partially holds iff $UDS' = UDS\ F : SP \to SP\ X : \top_\Sigma$ $\mathbf{given}_\Sigma$ $F(T)$ has a denotation $[\![UDS']\!]^\perp$ in $\mathcal{L}$, where F and X are new unit names and $\Sigma = sig[SP]$. By Lemma 34, $[\![UDS']\!]^\perp$ is defined iff $\vdash^f UDS' ::_\perp \zeta', \chi', \mu'$ holds, for some ζ', χ', μ'; this is because UDS' is generative and $\vdash UDS' \rhd\!\!\rhd \zeta', \chi'$ holds. It can be easily checked that this is true iff $\vdash^f ASP ::_\perp SP$. This proves the required equivalence for non-parametric ASP.

Assume $ASP = \textbf{units } UDS \textbf{ result } \lambda A : SP \bullet T$, $USP = SP_1 \to SP_2$, $sig[SP_1] = sig[SP]$ and $sig[SP_2] = [T]_{[UDS]}$; again, the latter three assumptions are equivalent to $ASP ::_\perp USP$ being statically correct. The refinement statement $ASP ::_\perp USP$ partially holds iff $[\![SP_1]\!] \subseteq [\![SP]\!]$ and $UDS' = UDS\ A : SP\ F : SP_2 \to SP_2\ X : \top_\Sigma\ \textbf{given}_\Sigma\ F(T)$ has a denotation $[\![UDS']\!]^\perp$ in $\mathcal{L}$, where F and X are new unit names and $\Sigma = sig[SP_2]$. By Lemma 34, $[\![UDS']\!]^\perp$ is defined iff $\vdash^f UDS' ::_\perp \zeta', \chi', \mu'$ holds, for some ζ', χ', μ'; this is because UDS' is generative and $\vdash UDS' \rhd\!\!\rhd \zeta', \chi'$ holds. It can be easily checked that this condition together with $[\![SP_1]\!] \subseteq [\![SP]\!]$ is equivalent to $\vdash^f ASP ::_\perp SP_1 \to SP_2$. This ends the proof. □

A straightforward application of Propositions 31 and 32 gives the final soundness and completeness theorem for the proof calculus $\vdash^f$. This solves the problem of verification for languages with amalgamation and generative fully statically correct refinement statements.

Theorem 35. *Assume $\mathcal{L}$ is language with amalgamation and translations and a $\mathcal{J}$-cocomplete signature category. For any refinement statement $\mathcal{IT} ::_\perp \mathcal{OT}$ which is generative, statically correct and fully statically correct in $(\mathcal{L})_s$ and $\mathcal{J}$,*

$$\mathcal{IT} ::_\perp \mathcal{OT} \textit{ holds in } \mathcal{L}$$

$$\textit{iff}$$

$$\vdash^f \mathcal{IT} ::_\perp \mathcal{OT}$$

□

A natural question is how many proof obligations are generated in a derivation in the calculus $\vdash^f$ and how big the sets $\mathcal{SP}$ on the left side of these obligations can get. The answer is rather simple and reassuring:

Proposition 36. *Let p be the number of proof obligations generated in a derivation of $\vdash^f ASP ::_\perp USP$ and n the number of applications of parametric units in ASP. Then:*

$$n + 1 \leq p \leq n + 2$$

If all amalgamations in ASP are along finite sinks and $[\![\mathcal{SP}]\!] \subseteq [\![SP]\!]$ is a generated proof obligation, then $\mathcal{SP}$ is finite. □

15 Verification Assuming Extended Static Correctness

As has been shown in Section 3, the specification language SubCasl does not have amalgamation. In the previous section, it has been assumed that the refinement statements or architectural specifications of interest are fully statically correct and only then could the verification process proceed via the $\vdash^f$ proof calculus. If the language $\mathcal{L}$ does not have amalgamation, then the assumption of fully static correctness has to be replaced by some other statical form of correctness. This is because fully static correctness is expected to ensure that all amalgamations are correct irrespective of the specifications used in the architectural specification. This is true if the language considered has amalgamation, but otherwise fully static correctness is too weak a property to guarantee this. For example, the SubCasl architectural specification:

units
  $A : \{\textbf{sorts}\ a, b, c, d;\}$
  $F : \{\textbf{sorts}\ a, b, c, d;\} \to \{\textbf{sorts}\ a < b, c < d;\}$
  $G : \{\textbf{sorts}\ a, b, c, d;\} \to \{\textbf{sorts}\ c < a, d < b;\}$
result
  $F(A) + G(A)$

is fully statically correct, since the amalgamation in the result is regular and the signature morphism equalities required by the **either** clause in the rule for amalgamation indeed hold. But the amalgamation is not correct in the partial model semantics, since the embeddings $c < a < b$ and $c < d < b$ need not commute in the unit to be built.

For this reason, we replace the requirement of fully static correctness by the requirement of extended static correctness. An architectural specification ASP is said to be *extended statically correct* in $\mathcal{L}$ if $|ASP|$ is partially correct in $\mathcal{L}$. Note that this notion is still static in the sense that it does not depend on the specifications involved in ASP; however, at the same time it is in a sense less static than fully static correctness, since it does depend on more than just the static language $(\mathcal{L})_s$. The following proposition shows that extended static correctness is a proper generalization of the notion of fully static correctness:

Proposition 37. *If a generative architectural specification ASP is statically correct and fully statically correct in $\mathcal{L}_s$ and $\mathcal{J}$, and if $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ has amalgamation, then ASP is extended statically correct.*

If $\mathcal{L}$ has a cocomplete signature category Sig, amalgamation, iso-reflection and consistent signatures, then a generative, statically correct architectural specification ASP is fully statically correct in $(\mathcal{L})_s$ and Sig iff ASP is extended statically correct.

Proof. The first claim is a direct consequence of the (generalized in Theorem 28) downward implication of Proposition 20.

The second claim is a consequence of Theorem 29 and the fact that a cocomplete signature category *Sig* is always *Sig*-cocomplete. ☐

One might ask whether, in view of the arguments presented in Section 12, it is worthwhile to look for verification methods for specification languages lacking amalgamation. Maybe, one could argue, such languages should be given up altogether? This, of course, would exclude from our consideration the language SubCasl, one of the most prominent specification languages we want to address. The answer to this question must be that, even taking into account the difficulties this causes, one should not give up languages without amalgamation. The reason is that SubCasl can be embedded in the specification language EnrichedCasl, which has amalgamation and is thus a perfectly valid specification language. Therefore, an architectural specification in SubCasl may be, in fact, treated as a specification in EnrichedCasl and correctness on the model level may be transferred to correctness on the level of programs, if programming languages which have amalgamation over EnrichedCasl are considered. These arguments could lead one to the idea that, perhaps, it would simply be better to use EnrichedCasl instead of SubCasl. While this is an option, it also has certain drawbacks. Subsorting in SubCasl has the, quite natural, property that any subsort is embedded in its supersort in exactly one way. This allows one to omit most sort qualifications in the syntax of terms, since they can be inferred automatically by a static analysis. In EnrichedCasl, this would not be possible, at least not in a straightforward manner. Thus, SubCasl may be seen as a light-weight version of EnrichedCasl, meaning that the use of SubCasl is understood equivalent to a syntactically constrained use of EnrichedCasl. As a result, if one wants to consider programming languages for SubCasl, it would be reasonable to consider only ones that are "reducts" of programming languages compatible with EnrichedCasl and with amalgamation.

Before the details of the proposed verification methods are presented, one more remark is necessary. The idea of verification assuming extended static correctness presupposes that extended static correctness itself can be checked. But, in contrast to fully static correctness, which was defined in terms of a signature-level calculus, extended static correctness is defined on the level of models and its checking can be, potentially, a difficult task. Methods for solving this problem turn out to be very similar to those solving the problem of verification. Therefore, below they are presented in parallel.

Both checking extended static correctness, as well as the verification afterwards, are based on an embedding of the specification language $\mathcal{L}$ without amalgamation, such as SUBCASL, in some language $\mathcal{L}'$ which has better properties. This embedding is called a *representation* of $\mathcal{L}$ in $\mathcal{L}'$ and it allows us to transfer properties of architectural specifications and refinement statements in the language $\mathcal{L}$ to properties of similar objects in the language $\mathcal{L}'$.

Assume two specification languages $\mathcal{L}$ and $\mathcal{L}'$ are given, with $\mathcal{L} = (\mathit{Sig}, \mathcal{I}, \mathbf{Mod}, \mathit{Spec}, \mathit{sig}[\_], [\![\_]\!])$ and $\mathcal{L}' = (\mathit{Sig}', \mathcal{I}', \mathbf{Mod}', \mathit{Spec}', \mathit{sig}'[\_], [\![\_]\!]')$. A *model representation* $R : \mathcal{L} \to \mathcal{L}'$ of $\mathcal{L}$ in $\mathcal{L}'$ consists of:

- a functor $\alpha^! : \mathit{Sig} \to \mathit{Sig}'$, which takes inclusions in $\mathcal{I}$ to inclusions in $\mathcal{I}'$,
- a function α taking any signature Σ in Sig to a specification $\alpha(\Sigma) \in \mathit{Spec}'(\alpha^!(\Sigma))$ and which is functorial in the sense that, for any signature morphism $\sigma : \Sigma \to \Delta$ in Sig and any model $M \in \mathbf{Mod}'(\alpha^!(\Delta))$, if $M \in [\![\alpha(\Delta)]\!]'$, then $M|_{\alpha^!(\sigma)} \in [\![\alpha(\Sigma)]\!]'$,
- a natural isomorphism $\beta : \alpha; [\![\_]\!]' \to \mathbf{Mod}$.

Note that the "functoriality" of α really means that α is a functor into the category of specifications and specification morphisms. This guarantees that $\alpha; [\![\_]\!]' : \mathit{Sig}^{op} \to \mathbf{Class}$ is a well-defined (contravariant) functor, mapping any signature morphism $\sigma : \Sigma \to \Delta$ to $\_|_{\alpha^!(\sigma)} : [\![\alpha(\Delta)]\!]' \to [\![\alpha(\Sigma)]\!]'$.

A *representation* $R : \mathcal{L} \to \mathcal{L}'$ consists of the components of a model representation plus a function γ taking, for any Σ in Sig, any specification $\mathit{SP} \in \mathit{Spec}(\Sigma)$ to a specification $\gamma(\mathit{SP}) \in \mathit{Spec}'(\alpha^!(\Sigma))$ and such that the following *representation condition* is satisfied:

- for any specification $\mathit{SP} \in \mathit{Spec}(\Sigma)$,
$$[\![\gamma(\mathit{SP})]\!]' \subseteq [\![\alpha(\Sigma)]\!]'$$
and
- for any model $M \in [\![\alpha(\Sigma)]\!]'$, the following equivalence holds:
$$\beta_\Sigma(M) \in [\![\mathit{SP}]\!] \iff M \in [\![\gamma(\mathit{SP})]\!]'$$

A model representation or representation is called *total* if $\alpha(\Sigma) = \top_{\alpha^!(\Sigma)}$ for all signatures Σ in Sig. In this case, the component α is not really needed. Also, the first part of the representation condition is then void. Such representations are of special value, since they translate signatures in $\mathcal{L}$ to signatures in $\mathcal{L}'$ and so preserve the static nature of properties. On the other hand, it is often easier to find a non-total representation than a total one.

Two concrete representations of the specification language SubCasl are presented in Section 17 and in Sections 18–20. In the current section, verification via total representations is considered, while in the next section verification via non-total representations is tackled.

Any total representation $R : \mathcal{L} \to \mathcal{L}'$ induces a translation of unit specifications, architectural specifications and tasks in $\mathcal{L}$ to ones in $\mathcal{L}'$. The same is true of non-total representations, but then we need to use a new generalized amalgamation construct.

The *generalized amalgamation* construct takes the form:

$$\bigcup_{i \in I} \sigma_i(T_i) \ \mathbf{with}\ SP$$

where $sig[SP] = \Delta$, with $\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I}$. The static denotation of such a term in the static context C_s is Δ, if, for all $i \in I$, $[T_i]_{C_s} = \Delta_i$. For any environment ϱ partially fitting C_s and such that $[\![T_i]\!]^{\perp}_{\varrho}$ is defined for all $i \in I$,

$$[\![\bigcup_{i \in I} \sigma_i(T_i) \ \mathbf{with}\ SP]\!]^{\perp}_{\varrho} = \perp$$

if $[\![T_i]\!]^{\perp}_{\varrho} = \perp$ for some $i \in I$, and otherwise

$$[\![\bigcup_{i \in I} \sigma_i(T_i) \ \mathbf{with}\ SP]\!]^{\perp}_{\varrho}$$

is the unique Δ-model M in $[\![SP]\!]$ such that $M|_{\sigma_i} = [\![T_i]\!]^{\perp}_{\varrho}$ for all $i \in I$, or is undefined if such an M does not exist. Note that even if M exists, there may be other Δ-models N satisfying $N|_{\sigma_i} = [\![T_i]\!]^{\perp}_{\varrho}$ for all $i \in I$; however, these models N must come from outside of $[\![SP]\!]$.

The translation induced by $R : \mathcal{L} \to \mathcal{L}'$ on unit specifications is defined by:

- $R(SP) = \gamma(SP)$,
- $R(SP \to SP') = R(SP) \to R(SP')$.

Note that, for any parametric unit specification USP, $R(USP)$ is a unit specification as well, since $\alpha^!$ is required to preserve the property of being an inclusion. The adequacy of the translation is shown by the following trivial proposition. For any Σ-model M in $\mathcal{L}$, let $R(M)$ be the model $\beta_\Sigma^{-1}(M)$; of course, $R(M)$ is an $\alpha(\Sigma)$-model in $\mathcal{L}'$; also, let $R(\perp) = \perp$. For any parametric unit $F \in \mathbf{Unit}_\perp(\Sigma \to \Delta)$ in $\mathcal{L}$, let $R(F)$ be a unit with domain $R(\mathrm{dom}(F))$ defined by the rule $R(F)(R(M)) = R(F(M))$ for all $M \in \mathrm{dom}(F)$.

Proposition 38. *For any Σ-specification SP, the map $R|_{[\![SP]\!]_\perp} : [\![SP]\!]_\perp \to [\![R(SP)]\!]'_\perp$ is a bijection.*

For any parametric unit specification $SP \to SP'$, the map $R|_{[\![SP \to SP']\!]_\perp} : [\![SP \to SP']\!]_\perp \to [\![R(SP \to SP')]\!]'_\perp$ is a bijection.

For any unit specification USP and unit $F \in \mathbf{Unit}_\perp(sig[USP])$, $F \in [\![USP]\!]_\perp$ iff $R(F) \in [\![R(USP)]\!]'_\perp$. □

The translation of statically correct architectural specifications (without the generalized form of amalgamation) induced by R is defined by induction over the structure of the architectural specification:

- $R(\mathbf{units}\ UDS\ \mathbf{result}\ UE) = \mathbf{units}\ R(UDS)\ \mathbf{result}\ R(UE)$,
- for declarations:
  - $R(\epsilon) = \epsilon$,
  - $R(UDS\ X : USP) = R(UDS)\ X : R(USP)$,
  - $R(UDS\ X : SP\ \mathbf{given}_\Sigma\ T) = R(UDS)\ X : R(SP)\ \mathbf{given}_{\alpha^!(\Sigma)}\ R(T)$,
- $R(\lambda A : SP \bullet T) = \lambda A : R(SP) \bullet R(T)$,
- for terms:
  - $R(X) = X$,
  - $R(X(T)) = X(R(T))$,
  - $R(\bigcup_{i \in I} \sigma_i(T_i)) = \bigcup_{i \in I} \alpha^!(\sigma_i)(R(T_i))$ **with** $\alpha(\Delta)$, where $\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I}$,
  - $R(T|_{\sigma:\Sigma\to\Delta}) = R(T)|_{\alpha^!(\sigma):\alpha^!(\Sigma)\to\alpha^!(\Delta)}$.

It is easy to check that this translation preserves static correctness and that the static denotation of $R(ASP)$ is simply a translation along R of the static denotation of ASP. If the representation R is total, then one may, instead of the generalized amalgamation construct, use normal amalgamation, since in this case $\alpha(\Delta)$ is the total specification anyway. The adequacy of the translation is then shown by:

Proposition 39. *Assume $R : \mathcal{L} \to \mathcal{L}'$ is a representation. If $ASP ::_\perp USP$ is a statically correct refinement statement, then $ASP ::_\perp USP$ holds in $\mathcal{L}$ iff $R(ASP) ::_\perp R(USP)$ holds in $\mathcal{L}'$.*

Proof. For any environment ϱ in $\mathcal{L}$, by $R(\varrho)$ denote the corresponding environment in $\mathcal{L}'$, i.e., one taking any X in $\mathrm{dom}(\varrho)$ to $R(\varrho(X))$. Recall that an equality is said to hold in the strong sense if both sides are undefined or both sides are defined and equal. The following lemma may be proved by an easy induction:

Lemma 40. *If UE is an expression with $[UE]_{C_s}$ defined, then, for any environment ϱ partially fitting C_s in $\mathcal{L}$, $R([\![UE]\!]^\perp_\varrho) = [\![R(UE)]\!]^\perp_{R(\varrho)}$ in the strong sense.*

If UDS is a declaration with $[UDS] = C_s$, then $R([\![UDS]\!]^\perp) = [\![R(UDS)]\!]^\perp$ in the strong sense. □

Suppose $ASP = \textbf{units } UDS \textbf{ result } UE$ is statically correct.

If $ASP ::_\perp USP$ holds in $\mathcal{L}$, then $[\![UDS]\!]^\perp$ must be defined and, for any $\varrho \in [\![UDS]\!]^\perp$, $[\![UE]\!]^\perp_\varrho$ is defined and an element of $[\![USP]\!]_\perp$. By the lemma, $[\![R(UDS)]\!]^\perp$ is defined as well and coincides with $R([\![UDS]\!]^\perp)$. Take any $\varrho' \in [\![R(UDS)]\!]^\perp$. There exists an environment $\varrho \in [\![UDS]\!]^\perp$ such that $R(\varrho) = \varrho'$. By assumption $[\![UE]\!]^\perp_\varrho$ is defined. By the lemma, $[\![R(UE)]\!]^\perp_{R(\varrho)} = [\![R(UE)]\!]^\perp_{\varrho'}$ is defined as well and equal to $R([\![UE]\!]^\perp_\varrho)$. Therefore $R(ASP)$ is partially correct. Moreover, $R(ASP) ::_\perp R(USP)$ holds, since $[\![UE]\!]^\perp_\varrho \in [\![USP]\!]_\perp$ implies $[\![R(UE)]\!]^\perp_{\varrho'} = R([\![UE]\!]^\perp_\varrho) \in [\![R(USP)]\!]_\perp$.

The opposite implication is proved similarily. □

Note that, in particular, an architectural specification is partially correct in $\mathcal{L}$ iff its translation is partially correct in $\mathcal{L}'$. Observe also that the translation preserves generativity and, if R is total, also the property that ASP is "stripped", i.e., that $|ASP| = ASP$.

The translation of tasks along R is already defined, since they come in the forms USP, ASP or $ASP.X$. In the latter case, the translation $R(ASP.X)$ is simply $R(ASP).X$.

The translation induced by $R : \mathcal{L} \to \mathcal{L}'$ thus transfers problems defined in $\mathcal{L}$ to problems in $\mathcal{L}'$. If we know how these problems can be solved in the latter specification language, this gives a method of solving them in the former. Of course, this method generates proof obligations in $\mathcal{L}'$, *not* in $\mathcal{L}$. This idea leads to the following two theorems that solve the problems of checking extended static correctness and of verification for extended statically correct refinement statements, respectively.

Theorem 41. *Assume $R : \mathcal{L} \to \mathcal{L}'$ is a total representation and $\mathcal{L}'$ has a cocomplete signature category Sig', amalgamation, iso-reflection and consistent signatures. If ASP is a generative architectural specification statically correct in $\mathcal{L}$, then:*

$$ASP \textit{ is extended statically correct in } \mathcal{L}$$

$$\textit{iff}$$

$$R(|ASP|) \textit{ is fully statically correct in } (\mathcal{L}')_s \textit{ and } Sig'$$

Proof. By definition, ASP is extended statically correct in $\mathcal{L}$ iff $|ASP|$ is partially correct in $\mathcal{L}$. By Proposition 39, the latter is equivalent to $R(|ASP|)$ being partially correct in $\mathcal{L}'$. Note that, by the totality of R, $|R(|ASP|)| = R(|ASP|)$; also, it is generative. By Theorem 29, $|R(|ASP|)|$ is partially correct in $\mathcal{L}'$ iff $R(|ASP|)$ is fully statically correct in $(\mathcal{L}')_s$ and Sig', which completes the proof.

□

Theorem 42. *Assume $R : \mathcal{L} \to \mathcal{L}'$ is a total representation and that $\mathcal{L}'$ has a cocomplete signature category, amalgamation, iso-reflection, consistent signatures and translations. If $\mathcal{IT} ::_\perp \mathcal{OT}$ is a generative refinement statement statically correct and extended statically correct in $\mathcal{L}$, then:*

$$\mathcal{IT} ::_\perp \mathcal{OT} \textit{ holds in } \mathcal{L}$$

$$\textit{iff}$$

$$\vdash^f R(\mathcal{IT}) ::_\perp R(\mathcal{OT})$$

Proof. We prove the theorem only for refinement statements of the form $ASP ::_\perp USP$, since for other statements it is either obvious or may be reduced to a statement of this form, as demonstrated in Section 13.

Let Sig' be the signature category of $\mathcal{L}'$.

By Theorem 41, since ASP is extended statically correct in $\mathcal{L}$, $R(|ASP|)$ must be fully statically correct in $(\mathcal{L}')_s$ and Sig'. But $|R(ASP)| = R(|ASP|)$, since R is total, and so $R(ASP)$ is fully statically correct in $(\mathcal{L}')_s$ and Sig'.

By Proposition 39, $ASP ::_\perp USP$ holds in $\mathcal{L}$ iff $R(ASP) ::_\perp R(USP)$ holds in $\mathcal{L}'$. Since $R(ASP)$ is fully statically correct in $(\mathcal{L}')_s$ and Sig', it follows that, by Theorem 32, the latter is equivalent to $\vdash^f R(ASP) ::_\perp R(USP)$.

□

By the latter theorem, the proof calculus $\vdash^e$ defined by the rule

$$\vdash^e \mathcal{IT} ::_\perp \mathcal{OT}$$

iff

$$\vdash^f R(\mathcal{IT}) ::_\perp R(\mathcal{OT})$$

is sound and complete for generative refinement statements which are statically and extended statically correct, provided that $R : \mathcal{L} \to \mathcal{L}'$ is a total representation and $\mathcal{L}'$ has a cocomplete signature category, amalgamation, iso-reflection, consistent signatures and translations.

16 Verification without Additional Static Assumptions

In the previous section, verification of architectural specifications in a language $\mathcal{L}$ which is totally represented in a language $\mathcal{L}'$ with amalgamation, iso-reflection and consistent signatures was investigated. If the representation is not total, then Theorem 41 cannot even be hoped to be applicable, since the translation $R(|ASP|)$ may contain a generalized amalgamation construct and, as a consequence, depend on specifications, not only signatures.

For the above reason, with non-total representations checking extended static correctness is not a static matter. It is thus very similar to checking partial correctness. As a consequence, the main reason for requiring extended static correctness, namely simplification of the verification process, disappears. Therefore, in this section, we focus on the problem of checking partial correctness even without the assumption that the architectural specification considered is extended statically correct. Of course, if, for some reason, it should be of interest to check extended static correctness of ASP, the methods presented below may easily be used: one needs only to apply them to the architectural specification $|ASP|$.

To perform verification without static assumptions other than static correctness, one needs certain other assumptions, in particular ones that facilitate encoding the correctness of amalgamations using proof obligations. Of course, various assumptions satisfying our requirements may be chosen. The goal must be to find a reasonably weak set of assumptions satisfying our requirements; in particular, these assumptions should hold for the the specification language SubCasl. The assumptions presented below form such a set.

A *morphism equality of type* $\Sigma \to \Delta$ is an expression $\alpha = \beta$, where $\alpha, \beta : \Sigma \to \Delta$ in *Sig*. The equality is said to be *expressible*, if there exists a Δ-specification SP such that, for any Δ-specification SP_0,

$$\llbracket SP_0 \rrbracket \subseteq \llbracket SP \rrbracket$$

$$\textit{iff}$$

$$\textit{for all } M \in \llbracket SP_0 \rrbracket,\ M|_\alpha = M|_\beta$$

Then SP is said to *express* $\alpha = \beta$; this fact is written as

$$\text{ExpressEquality}(\alpha = \beta) = SP$$

In other words, SP expresses $\alpha = \beta$ if and only if $\llbracket SP \rrbracket$ is the greatest class of models M such that $M|_\alpha = M|_\beta$, among all classes which are of the form $\llbracket SP_0 \rrbracket$ for some SP_0. Note that, in general, there may be models $M \notin \llbracket SP \rrbracket$ such that $M|_\alpha = M|_\beta$.

If $\delta : \Sigma \to \Sigma'$ is a signature morphism and M a Σ-model, then M is said to be *δ-extendible* if a Σ'-model N with $N|_\delta = M$ exists; then N is a *δ-extension* of M.

Let $\delta : \Sigma \to \Sigma'$ be a morphism in *Sig* and SP a Σ'-specification. It is said that *(δ, SP)-extendability is expressible* if there exist morphism equalities $\alpha_k = \beta_k$ of type $\Sigma_k \to \Sigma$ for $1 \leq k \leq K$, and a Σ'-specification SP' such that, for any Σ-model M,

1. M is δ-extendible iff, for all $1 \leq k \leq K$, $M|_{\alpha_k} = M|_{\beta_k}$,
2. if M has a unique δ-extension satisfying SP, then all its δ-extensions satisfy SP',
3. if M has a δ-extension satisfying SP', then it has a unique δ-extension satisfying SP.

One then writes

$$\text{ExpressExtendability}(\delta, SP) = (SP', \{\alpha_k = \beta_k\}_{1 \leq k \leq K})$$

A specification language $\mathcal{L}$ is said to have *(finite) conjunction* if, for any Σ-specifications SP_1 and SP_2, a Σ-specification $SP_1 \wedge SP_2$ exists such that, for any Σ-model M, $M \in \llbracket SP_1 \wedge SP_2 \rrbracket$ iff $M \in \llbracket SP_1 \rrbracket$ and $M \in \llbracket SP_2 \rrbracket$. Though this is, actually, only binary conjunction, it easily follows that in $\mathcal{L}$ all finite conjunctions may be constructed, including the empty one (given as $\top_\Sigma$). Such conjunctions are denoted by $\bigwedge \mathcal{SP}$, where $\mathcal{SP}$ is a finite set of Σ-specifications.

The proof calculus $\vdash_0^g$ is defined for a language $\mathcal{L}$ with translations and a finitely cocomplete signature category *Sig*. It is also assumed that all amalgamations are performed along finite sinks. Such amalgamations are called *finite*. The calculus has the same rules as $\vdash^f$, except that the rule for λ-abstraction is changed and the rule for amalgamation is replaced by a rule for the generalized amalgamation construct. Note that, in the rules inherited from $\vdash^f$, it is assumed that the category $\mathcal{J}$ is equal *Sig*. Also, in the headers, the assumptions regarding the fully static correctness of syntactic constructs should be replaced by assumptions regarding their static correctness, and an assumption regarding the finiteness of all the used sets of specifications $\mathcal{SP}$ should be added.

An important feature is that the proof calculus generates, for parametric architectural specifications, a new type of proof obligation, namely:

$$[\![\mathcal{SP}]\!] = \emptyset$$

This is not as novel as it seems, since if the specification language *has falsity*, i.e., if there is, for any signature Σ, a Σ-specification $\bot_\Sigma$ such that $[\![\bot_\Sigma]\!] = \emptyset$, then the "new" proof obligation may be replaced by:

$$[\![\mathcal{SP}]\!] \subseteq [\![\bot]\!]$$

The changed rule for λ-abstraction looks as follows:

$$\frac{\begin{array}{c}\zeta[\{\star : sig[SP]\}/A], \chi[\mathrm{id}_{sig[SP]}/A], \mu[\{SP\}/A] \vdash_0^g T ::_\bot \Gamma,\ t,\ \mathcal{SP} \\ sig[SP_1] = sig[SP] \text{ and } [\![SP_1]\!] \subseteq [\![SP]\!] \\ sig[SP_2] = \mathrm{dom}(t) \\ \textbf{either} \\ [\![\mathcal{SP}]\!] = \emptyset \\ \textbf{or} \\ A \in \mathrm{dom}(\Gamma) \\ \mathrm{ExpressEquality}(\Gamma(A \to \star) = \iota_{sig[SP] \subseteq \mathrm{dom}(t)}; t) = SP_* \\ [\![\mathcal{SP}]\!] \subseteq [\![SP_*]\!] \\ [\![\mathcal{SP} \cup \{\Gamma(A \to \star)(SP_1)\}]\!] \subseteq [\![t(SP_2)]\!]\end{array}}{\zeta, \chi, \mu \vdash_0^g \lambda A : SP \bullet T ::_\bot SP_1 \to SP_2}$$

Note that the **or** clause extends over four premises. In this rule, after computing $\Gamma, t, \mathcal{SP}$, two possibilities are considered. If $\mathcal{SP}$ is inconsistent, then the term T must, in any environment, have the value $\bot$, and in this case there is nothing more to check. If $\mathcal{SP}$ is consistent, then one has to check the persistence of the defined unit and that it returns $\bot$ when fed

$\perp$. The latter requirement forces A to be an element of $\mathrm{dom}(\Gamma)$ and the former requirement is expressed by premises 2 and 3 in the **or** clause: the "maximal" specification SP_* which guarantees persistency is found and then it is checked that $\mathcal{SP}$ implies SP_*. Finally, the last premise ensures that the result unit really satisfies the unit specification $SP_1 \to SP_2$.

The new rule for generalized amalgamation, replacing the old rule for amalgamation, is presented below. We now informally explain the shape of this, admittedly rather complex, rule. As for computing the sum and relevant transformation, this is done just as in the other calculi. The coproduct $\{\iota_i : \Delta_i \to \Delta_*\}_{i \in I}$ allows us to transform checking the existence of a generalized σ-amalgamation satisfying SP of a model family to checking (δ, SP)-extendability of a Δ_*-model. The problem of (δ, SP)-extendability gives rise to morphism equalities $\alpha_k = \beta_k$ and a Δ-specification SP'. It is then checked, via specifications SP_k which express the morphism equalities, whether these equalities hold, which is equivalent to the considered model being always δ-extendible. Finally, in the last premise points (2) and (3) of the definition of expressibility of (δ, SP)-extendability are used to check whether the required unique Δ-model satisfying SP exists. A formal proof of the soundness and completeness of this rule is given in the proof of Lemma 44 below.

$$\frac{\begin{array}{c}
\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I} \\
\zeta, \chi, \mu \vdash_0^g T_i ::_\perp \Gamma_i,\ t_i,\ \mathcal{SP}_i \text{ for all } i \in I \\
\iota_* = \{\iota_i : \Delta_i \to \Delta_*\}_{i \in I} \text{ is a coproduct sink} \\
\Gamma_* = \bigcup_{i \in I} \Gamma_i \text{ and } \{\tau_i : \Gamma_i(\star) \to \Gamma_*(\star)\}_{i \in I} \text{ is the relevant transformation} \\
\gamma : \Delta_* \to \Gamma_*(\star) \text{ is the universal morphism from} \\
\qquad \text{the coproduct } \iota_* \text{ to the cocone } \{(t_i; \tau_i) : \Delta_i \to \Gamma_*(\star)\}_{i \in I} \\
\delta : \Delta_* \to \Delta \text{ is the universal morphism from the coproduct } \iota_* \text{ to the cocone } \sigma \\
\mathrm{ExpressExtendability}(\delta, SP) = (SP', \{\alpha_k = \beta_k\}_{1 \le k \le K}) \\
\mathrm{ExpressEquality}((\alpha_k; \gamma) = (\beta_k; \gamma)) = SP_k \text{ for all } 1 \le k \le K \\
[\![\bigcup_{i \in I} \tau_i(\mathcal{SP}_i)]\!] \subseteq [\![SP_k]\!] \text{ for all } 1 \le k \le K \\
(t : \Delta \to \Sigma', \iota : \Gamma_*(\star) \to \Sigma') \text{ defines a colimit} \\
\qquad \text{over } \{\sigma_i : \Delta_i \to \Delta\}_{i \in I} \cup \{(t_i; \tau_i) : \Delta_i \to \Gamma_*(\star)\}_{i \in I} \\
[\![\bigcup_{i \in I} \iota(\tau_i(\mathcal{SP}_i))]\!] \subseteq [\![t(SP')]\!]
\end{array}}{\zeta, \chi, \mu \vdash_0^g \bigcup_{i \in I} \sigma_i(T_i) \textbf{ with } SP ::_\perp (\Gamma_*; \iota),\ t,\ \bigcup_{i \in I} \iota(\tau_i(\mathcal{SP}_i)) \cup \{t(SP)\}}$$

A (generalized) amalgamation construct $\bigcup_{i \in I} \sigma_i(T_i)$ **with** SP in an architectural specification ASP is said to be *expressible* if (δ, SP)-extendability is expressible, where $\delta : \Delta_* \to \Delta$ is the universal morphism from the coproduct $\{\iota_i : \Delta_i \to \Delta_*\}_{i\in I}$ to the sink $\{\sigma_i : \Delta_i \to \Delta\}_{i\in I}$. An architectural specification has *expressible amalgamations* if all its amalgamations are expressible.

The following proposition and the, slightly more general, theorem show that the calculus $\vdash^g_0$ is sound and complete for verification without static assumptions other than static correctness. Of course, the calculus $\vdash^g_0$ considered in the theorem is the one extended via the procedure presented in Section 13. However, keep in mind that, here, it is still assumed that $\mathcal{L}$ has amalgamation.

Proposition 43. *Assume $\mathcal{L}$ has a finitely cocomplete signature category, amalgamation, conjunction and translations and that all morphism equalities are expressible. If ASP is a generative architectural specification with expressible and finite amalgamations and ASP $::_\perp$ USP is statically correct, then*

$$ASP ::_\perp USP \text{ holds in } \mathcal{L}$$

$$\text{iff}$$

$$\vdash^g_0 ASP ::_\perp USP$$

Proof. The proof is very similar to the proof of Proposition 32. In particular, all the definitions used there are needed here as well. Also, two lemmas analogous to those proved there are needed:

Lemma 44. *Let $\mathcal{L}$ be a language with a finitely cocomplete signature category, amalgamation, translations and expressible morphism equalities.*

Assume T is generative and uses only expressible and finite amalgamations, and ζ, χ, μ is a specification context description such that $[T]_{C_s(\chi)}$ is defined in $\mathcal{L}$.

The following conditions are equivalent:

(PC) $\zeta, \chi, \mu \vdash^g_0 T ::_\perp \Gamma,\, t,\, \mathcal{SP}$,

(PM) *for any environment ϱ (χ, μ)-represented by an environment η satisfying ζ, μ, $[\![T]\!]^\perp_\varrho$ is defined in $\mathcal{L}$.*

Moreover, if **(PC)** *holds and η satisfies ζ, μ, then the following condition holds:*

$$\{[\![T]\!]^\perp_\varrho \mid \varrho \text{ is } (\chi,\mu)\text{−represented by } \eta\} \setminus \{\perp\}$$
$$= \{M|_t \mid M \in [\![\mathcal{SP}]\!]_\perp\ \Gamma\text{−represents } \eta|_{\mathrm{dom}(\Gamma)}\} \quad (*)$$

Proof. The proof is by induction over the structure of T. The only case in which the proof differs from the proof of Lemma 33 (see Appendix 4) is the case of a generalized amalgamation. So, assume $T = \bigcup_{i \in I} \sigma_i(T_i)$ **with** SP, where $\{\sigma_i : \Delta_i \to \Delta\}_{i \in I}$, SP is a Δ-specification and $[T_i]_{C_s(\chi)} = \Delta_i$ for all $i \in I$.

Assume **(PC)** holds. We then have $\zeta, \chi, \mu \vdash_0^g T_i ::_\perp \Gamma_i, t_i, \mathcal{SP}_i$, for all $i \in I$, and all the other premises of the rule for amalgamation hold as well.

Take any environment η satisfying ζ, μ and any ϱ (χ, μ)-represented by η. By the induction hypothesis, $[\![T_i]\!]_\varrho^\perp$ is defined and the appropriate condition $(*_i)$ holds, in particular, if $[\![T_i]\!]_\varrho^\perp \neq \perp$, then there are models $N_i \in [\![\mathcal{SP}_i]\!]$ Γ_i-representing $\eta|_{\mathrm{dom}(\Gamma_i)}$ and such that $N_i|_{t_i} = [\![T_i]\!]_\varrho^\perp$.

Let $N_* = \bigoplus_{i \in I} \iota_i(N_i|_{t_i})$; the Δ_*-model N_* is defined, since it is an amalgamation along a coproduct sink and $\mathcal{L}$ has amalgamation. Let $N = \bigoplus_{i \in I} \tau_i(N_i)$; the $\Gamma_*(\star)$-model N is defined for the same reasons as explained in the proof of Lemma 33, i.e., it is an amalgamation along a colimit, $\mathcal{L}$ has amalgamation, and the equality $N_{i_1}|_{\Gamma_{i_1}(X \to \star)} = N_{i_2}|_{\Gamma_{i_2}(X \to \star)}$ holds for all $X \in \mathrm{dom}(\Gamma_{i_1}) \cap \mathrm{dom}(\Gamma_{i_2})$.

Note that $N|_{\tau_i} = N_i \in [\![\mathcal{SP}_i]\!]$. Hence, $N \in [\![\tau_i(\mathcal{SP}_i)]\!]$ for all $i \in I$ and so, by the appropriate premise of the rule for generalized amalgamation, $N \in [\![SP_k]\!]$ for all $1 \leq k \leq K$. This implies that $N|_{\alpha_k;\gamma} = N'|_{\beta_k;\gamma}$ and so $N_*|_{\alpha_k} = N_*|_{\beta_k}$ for all $1 \leq k \leq K$. Therefore, by point (1) of the definition of (δ, SP)-extendability being expressible, N_* is δ-extendible, i.e., there exists a model Δ-model M_0 such that $M_0|_\delta = N_*$.

Let $M' = t(M_0) \oplus \iota(N)$; the Σ'-model M' is defined, since it is an amalgamation along a colimit, $\mathcal{L}$ has amalgamation and, for all $i \in I$, $M_0|_{\sigma_i} = M_0|_{\iota_i;\delta} = N_*|_{\iota_i} = N|_{t_i;\tau_i}$. Note that $M'|_{\tau_i;\iota} = N|_{\tau_i} = N_i \in [\![\mathcal{SP}_i]\!]$, i.e., $M' \in [\![\iota(\tau_i(\mathcal{SP}_i))]\!]$ for all $i \in I$. By the appropriate premise of the rule for generalized amalgamation, $M' \in [\![t(SP')]\!]$, that is, $M_0 = M'|_t \in [\![SP']\!]$. By point (3) of the definition, there exists a unique model $M \in [\![SP]\!]$ such that $M|_\delta = N_*$.

Thus $M \in [\![SP]\!]$, $M|_{\sigma_i} = (M|_\delta)|_{\iota_i} = N_*|_{\iota_i} = N_i = [\![T_i]\!]_\varrho^\perp$ for all $i \in I$. Moreover, M is unique with these properties, since if M_1 also has these properties, then $M_1|_\delta = N_*$, because N_* was the unique model with $N_*|_{\iota_i} = N_i$, and so $M_1 = M$. Therefore $[\![T]\!]_\varrho^\perp$ is defined and equal M. This proves **(PM)**.

As for the inclusion "$\subseteq$" in the condition (*), let $M'' = t(M) \oplus \iota(N)$; as in the case of M' above, the Σ'-model M'' is defined. Also, $M''|_t = M$, $M''|_{\tau_i;\iota} = N_i \in [\![\mathcal{SP}_i]\!]$ for all $i \in I$ and $M''|_t \in [\![SP]\!]$. Finally, M'' $(\Gamma_*; \iota)$-represents $\eta|_{\mathrm{dom}(\Gamma_*)}$, since $M''|_\iota = N$ and N Γ_*-represents $\eta|_{\mathrm{dom}(\Gamma_*)}$. The latter fact holds, since $N|_{\tau_i} = N_i$ Γ_i-represents $\eta|_{\mathrm{dom}(\Gamma_i)}$ for all $i \in I$.

The inclusion "$\supseteq$" in condition (*) will be proved below.

Now assume **(PM)** holds. By the induction hypothesis, $\zeta, \chi, \mu \vdash_0^g T_i ::_\perp \Gamma_i, t_i, \mathcal{SP}_i$ holds for some $\Gamma_i, t_i, \mathcal{SP}_i$, as does the condition $(*_i)$. The coproduct sink $\{\iota_i : \Delta_i \to \Delta_*\}_{i \in I}$ and the sum Γ_* and relevant transformation exist, since the signature category is finitely cocomplete. The same is true for the colimit (t, ι). The universal morphisms γ and δ exist by definition of coproduct. The pair $(SP', \{\alpha_k = \beta_k\}_{1 \le k \le K})$ and the specifications SP_k for $1 \le k \le K$ exist, since T uses only expressible amalgamations and all morphism equalities are expressible in $\mathcal{L}$.

Take any $N \in [\![\bigcup_{i \in I} \tau_i(\mathcal{SP}_i)]\!]$. By a standard argument, there exists an environment η satisfying ζ, μ such that N $(\Gamma_*; \iota)$-represents $\eta|_{\mathrm{dom}(\Gamma_*)}$; also by a standard argument and by $(*_i)$, there exists an environment ϱ (χ, μ)-represented by η and such that $[\![T_i]\!]_\varrho^\perp = N|_{t_i;\tau_i}$.

By **(PM)**, there exists a unique model $M \in [\![SP]\!]$ such that $M|_{\sigma_i} = N|_{t_i;\tau_i}$ for all $i \in I$. Note that, for all $i \in I$, $(M|_\delta)|_{\iota_i} = M|_{\sigma_i} = N|_{t_i;\tau_i} = N|_{\iota_i;\gamma}$. Since $\mathcal{L}$ has amalgamation and the ι_i form a coproduct sink, this implies that $N|_\gamma = M|_\delta$.

Thus, M is a δ-extension of $N|_\gamma$ and so, by point (1) of the definition of expressability of (δ, SP)-extendability, $(N|_\gamma)|_{\alpha_k} = (N|_\gamma)|_{\beta_k}$ for all $1 \le k \le K$, that is, $N|_{\alpha_k;\gamma} = N|_{\beta_k;\gamma}$. In other words, we have shown that, for any $N \in [\![\bigwedge\{\tau_i(\mathcal{SP}_i) \mid i \in I\}]\!]$, the equality $N|_{\alpha_k;\gamma} = N|_{\beta_k;\gamma}$ holds. Therefore, since SP_k expresses $(\alpha_k; \gamma) = (\beta_k; \gamma)$, we have $N \in [\![SP_k]\!]$. This proves that $[\![\bigcup_{i \in I} \tau_i(\mathcal{SP}_i)]\!] \subseteq [\![SP_k]\!]$ for all $1 \le k \le K$.

Take any $N' \in [\![\bigcup_{i \in I} \iota(\tau_i(\mathcal{SP}_i))]\!]$. Let $N = N'|_\iota \in [\![\bigcup_{i \in I} \tau_i(\mathcal{SP}_i)]\!]$. As above, there exists a (unique) δ-extension M of $N|_\gamma$ that satisfies SP. Thus, by point (2) of the definition of expressability of (δ, SP)-extendability, all δ-extensions of $N|_\gamma$ satisfy SP'. In particular, $N'|_t$ is such an extension; to see this, observe that $(N'|_t)|_\delta = (N'|_\iota)|_\gamma = N|_\gamma$. Hence, $N'|_t \in [\![SP']\!]$, i.e., $N' \in [\![t(SP')]\!]$. This proves that $[\![\bigcup_{i \in I} \iota(\tau_i(\mathcal{SP}_i))]\!] \subseteq [\![t(SP')]\!]$, as required. This proves that **(PC)** holds.

As for the inclusion "$\supseteq$" in condition $(*)$, assume that N' additionally is an element of $[\![t(SP)]\!]$. Then $N'|_t$ is, as shown above, an extension of $N|_\gamma = M|_\delta$. Moreover, $N'|_t \in [\![SP]\!]$. Therefore, $N'|_t = M$. In other words, $[\![T]\!]_\varrho^\perp = M = N'|_t$, where ϱ is (χ, μ)-represented by η satisfying ζ, μ. This ends the proof. ☐

The proof of the following lemma is virtually the same as the proof of Lemma 34, which may be found in Appendix 4.

Lemma 45. *Let $\mathcal{L}$ be a language with a finitely cocomplete signature category, amalgamation, conjunction and translations and one in which all morphism equalities are expressible.*

Assume UDS is generative and uses only expressible and finite amalgamations and that $[UDS]$ is defined. The following conditions are equivalent:

(PC) $\vdash_0^g UDS ::_\perp \zeta, \chi, \mu,$

(PM) *the context* $[\![UDS]\!]^\perp$ *is defined in* $\mathcal{L}$.

Moreover, if **(PC)** *holds, then* ζ, χ, μ *and* $\overline{p}(UDS)$ *describe* $[\![UDS]\!]^\perp$. □

The proof of Proposition 43 is fairly standard now, therefore only its more interesting parts are discussed. Concerning the case when the architectural specification ASP is parametric, then in the rule for λ-abstraction two cases are considered. If the **either** clause holds, then $\mathcal{SP}$ appearing in the rule for λ-abstraction is inconsistent, and so the defined unit is constantly equal $\perp$ and therefore certainly correct. If the **or** clause holds, then $A \in \mathrm{dom}(\Gamma)$ guarantees that the defined unit returns $\perp$ when fed $\perp$, and $[\![\mathcal{SP}]\!] \subseteq [\![SP_*]\!]$ guarantees that the unit is persistent. In the opposite direction "$\Downarrow$", if the defined unit returns $\perp$ when fed $\perp$ and if $\mathcal{SP}$ is consistent, then A must appear in T and so $A \in \mathrm{dom}(\Gamma)$; otherwise one could construct an environment ϱ taking A to $\perp$ and such that $[\![T]\!]_\varrho^\perp$ is not $\perp$, which would contradict the partial correctness of ASP. Since the returned unit is persistent, any model in $\mathcal{SP}$ satisfies the appropriate morphism equality. Since $\mathcal{SP}$ is finite and $\mathcal{L}$ has finite conjunctions, this implies that $[\![\mathcal{SP}]\!] \subseteq [\![SP_*]\!]$ holds. □

Theorem 46. *Assume $\mathcal{L}$ has a finitely cocomplete signature category, amalgamation, conjunction and translations and that all morphism equalities are expressible.*

If $\mathcal{IT} ::_\perp \mathcal{OT}$ is statically correct, generative and has expressible and finite amalgamations, then

$$\mathcal{IT} ::_\perp \mathcal{OT} \text{ holds in } \mathcal{L}$$

$$\text{iff}$$

$$\vdash_0^g \mathcal{IT} ::_\perp \mathcal{OT}$$

Proof. The theorem follows by an application of Propositions 31 and 43. □

Let $\mathcal{S}$ be a class of sinks in the signature category of $\mathcal{L}$ and $R : \mathcal{L} \to \mathcal{L}'$ be a representation. One says that $\mathcal{S}$-sinks are *R-expressible* in $\mathcal{L}'$ if, for any sink $\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I}$ in $\mathcal{S}$, for any coproduct sink $\iota = \{\iota_i : \alpha^!(\Delta_i) \to \Delta_*\}_{i \in I}$ in the signature category of $\mathcal{L}'$, if the universal morphism from ι to $\alpha^!(\sigma)$ is denoted by $\delta : \Delta_* \to \alpha^!(\Delta)$, then $(\delta, \alpha(\Delta))$-extendability is expressible in $\mathcal{L}'$.

Let $R : \mathcal{L} \to \mathcal{L}'$ be a representation and $\mathcal{L}'$ have translations. Define the proof calculus $\vdash^g$ by the rule

$$\frac{R(\mathcal{IT}) \vdash_0^g R(\mathcal{OT})}{\mathcal{IT} \vdash^g \mathcal{OT}}$$

The following theorem shows that the proof calculus $\vdash^g$ is then sound and complete, though under certain, quite restrictive, assumptions:

Theorem 47. *Assume $R : \mathcal{L} \to \mathcal{L}'$ is a representation and $\mathcal{L}'$ has a finitely cocomplete signature category, amalgamation, conjunction and translations and that all morphism equalities are expressible in $\mathcal{L}'$.*

Assume also that $\mathcal{S}$ is a class of finite signature sinks in $\mathcal{L}$ and that all $\mathcal{S}$-sinks are R-expressible in $\mathcal{L}'$.

Then, for any statically correct generative refinement statement using only non-generalized amalgamations along sinks from $\mathcal{S}$,

$$\mathcal{IT} ::_\bot \mathcal{OT} \text{ holds in } \mathcal{L}$$

$$\textit{iff}$$

$$\vdash^g \mathcal{IT} ::_\bot \mathcal{OT}$$

Proof. By definition, $\vdash^g \mathcal{IT} ::_\bot \mathcal{OT}$ iff $\vdash_0^g R(\mathcal{IT}) ::_\bot R(\mathcal{OT})$. Note that in the latter judgement all amalgamations are along sinks which are finite and expressible; this is a consequence of the fact that sinks in $\mathcal{IT}$ and $\mathcal{OT}$ come from $\mathcal{S}$ and all $\mathcal{S}$-sinks are R-expressible. By Theorem 46, $\vdash_0^g R(\mathcal{IT}) ::_\bot R(\mathcal{OT})$ is thus derivable iff $R(\mathcal{IT}) ::_\bot R(\mathcal{OT})$ holds in $\mathcal{L}'$. By Proposition 39, the latter holds iff $\mathcal{IT} ::_\bot \mathcal{OT}$ holds in $\mathcal{L}$. □

17 A Non-total Representation of CASL

In order to apply Theorems 41, 42 and 47 of the previous two sections to architectural specifications in the specification language SUBCASL, appropriate representations of SUBCASL have to be provided. In this section, a non-total representation of SUBCASL in the specification language MULTICASL is defined and it is shown that this representation has the properties required by Theorem 47. A total representation of SUBCASL satisfying the requirements of Theorems 41 and 42 is defined and analyzed in Sections 18–20.

It might seem that, since, as shown later, an appropriate total representation of SUBCASL exists, there is no reason for defining a non-total one. However, a non-total representation is much simpler to define, since it is, in principle, a by-product of the definition of SUBCASL itself. Also, it represents SUBCASL in a fairly standard specification language, namely MULTICASL, which is very similar to standard multisorted first-order logic. This is not the case for the total representation, which represents SubCasl in a logic devised only for this purpose, namely EnrichedCasl. Moreover, EnrichedCasl has certain rather special properties, for example, the calculation of colimits in it is undecidable [KHT+01]. Finally, a non-total representation is needed if one wants to verify an architectural specification in SubCasl without any assumptions about its static properties, such as extended static correctness.

The current section is heavily based on the definitions of specification languages presented in Chapter I, Section 2. The specification languages used in this section are also (implicitly) considered only in their finite versions, again defined in Chapter I. The reason is that a non-total representation in a language which admits infinite sets of sentences would not be interesting or useful in practice.

What is first needed is a representation of BasicSubCasl in BasicMultiCasl. This turns out to be quite simple, since most of the components have already been defined in Section 2. Let R_0 : BasicSubCasl $\to$ BasicMultiCasl consist of:

- the functor $(\_)^{\#} : \mathit{Sig}_{\leq} \to \mathit{Sig}$,
- the function $(\_)^{\#\#}$ taking any signature Σ in $\mathit{Sig}_{\leq}$ to the $\Sigma^{\#}$-specification $\Sigma^{\#\#}$ in BasicMultiCasl,
- the natural isomorphism $\{\mathrm{id}_{\mathbf{Mod}_{\leq}(\Sigma)}\}_{\Sigma\in \mathit{Sig}_{\leq}} : ((\_)^{\#\#})^{op}; \mathbf{Mod} \to \mathbf{Mod}_{\leq}$,
- the function taking any specification (Σ, Φ) in BasicSubCasl to the specification $(\Sigma^{\#}, \Phi \cup \Psi)$, where $\Sigma^{\#\#} = (\Sigma^{\#}, \Psi)$, in BasicMultiCasl.

Note that, by definition, $\mathbf{Mod}_{\leq} = ((\_)^{\#\#})^{op}; \mathbf{Mod}$ and so the natural isomorphism is correctly defined. Recall that, by definition, if $[\![\_]\!]$ denotes the semantics of specifications in BasicSubCasl and $[\![\_]\!]'$ in BasicMultiCasl, then the equivalence $M \in [\![(\Sigma, \Phi)]\!]$ iff $M \in [\![(\Sigma^{\#}, \Phi)]\!]'$ holds for any model $M \in \mathbf{Mod}_{\leq}(\Sigma)$. The representation condition holds, since, in the notation used above, $[\![(\Sigma^{\#}, \Phi \cup \Psi)]\!]' \subseteq [\![(\Sigma^{\#}, \Psi)]\!]' = [\![\Sigma^{\#\#}]\!]'$ and, for any model M in $[\![\Sigma^{\#\#}]\!]'$, $\mathrm{id}_{\Sigma}(M) \in [\![(\Sigma, \Phi)]\!]$ iff $M \in [\![(\Sigma, \Phi)]\!]$, which holds iff $M \in [\![(\Sigma^{\#}, \Phi \cup \Psi)]\!]'$. Thus, R_0 is a representation.

In the second step, it is shown that a representation of "basic" specification languages, such as R_0 : BasicSubCasl $\to$ BasicMultiCasl, may be lifted to a representation R : SubCasl $\to$ MultiCasl of specification languages with structuring operations. This is stated by the following proposition, in which the only non-obvious case is lifting the "*SP* **with** σ" construct:

Proposition 48. *Assume $\mathcal{D}$ is a class of signature morphisms in a specification language $\mathcal{L}$ and $R = (\alpha^!, \alpha, \beta, \gamma) : \mathcal{L} \to \mathcal{L}'$ is a representation. Then $R' = (\alpha^!, \alpha, \beta, \gamma') : \mathrm{Str}_{\mathcal{D}}(\mathcal{L}) \to \mathrm{Str}_{\mathcal{D}'}(\mathcal{L}')$, where $\mathcal{D}' = \{\alpha^!(\sigma) \mid \sigma \in \mathcal{D}\}$, is a representation as well, with γ' defined as follows by structural induction:*

- $\gamma'(SP) = \gamma(SP)$, *if SP is a (basic) specification in* $\mathcal{L}$,
- $\gamma'(SP\ \mathbf{with}\ \sigma : \Sigma \to \Delta) = \big(\gamma'(SP)\ \mathbf{with}\ \alpha^!(\sigma) : \alpha^!(\Sigma) \to \alpha^!(\Delta)\big) \cup \alpha(\Delta)$,
- $\gamma'(SP_1 \cup SP_2) = \gamma'(SP_1) \cup \gamma'(SP_2)$,
- $\gamma'(SP|_{\sigma:\Sigma\to\Delta}) = \gamma'(SP)|_{\alpha^!(\sigma):\alpha^!(\Sigma)\to\alpha^!(\Delta)}$,
- $\gamma'(\mathbf{free}\ SP\ \mathbf{along}\ \sigma : \Delta \to \Sigma) = \mathbf{free}\ \gamma'(SP)\ \mathbf{along}\ \alpha^!(\sigma) : \alpha^!(\Delta) \to \alpha^!(\Sigma)$. □

Recall that SubCasl = $\mathrm{Str}_{\mathcal{SI}}$(BasicSubCasl) and that MultiCasl = $\mathrm{Str}_{\mathcal{MI}}$(BasicMultiCasl). The *standard representation* R : SubCasl → MultiCasl is obtained from R_0 and the class of morphisms $\mathcal{SI}$ via Proposition 48. This is possible, since, for any signature morphism σ in $\mathcal{SI}$, the signature morphism $\alpha^!(\sigma) = \sigma^\#$ is an element of $\mathcal{MI}$.

In order to prove that the standard representation R satisfies the assumptions of Theorem 47, the following auxiliary fact is need. The standard proof is omitted. Recall that the specification language MultiCasl has been defined as $\mathrm{Str}_{\mathcal{MI}}$(BasicMultiCasl), where the class of morphisms $\mathcal{MI}$, along which reductions are permitted, contained only inclusions.

Proposition 49. *In the enriched specification language* MultiCasl, *isomorphic models are elementarily equivalent, i.e., they satisfy precisely the same specifications.* □

Recall that, for any function g, the notation g^* is used to denote the function $g \times \cdots \times g$. As an extension of this notation, by a variable x^* a tuple of appropriate length of variables is understood. In both cases, the length of the product or tuple should be clear form the context.

Proposition 50. *In* MultiCasl, *all morphism equalities are expressible.*

Proof. Let $\alpha = \beta$ be a morphism equality of type $\Sigma \to \Delta$. If there exists a sort s in Σ such that $\alpha^S(s) \neq \beta^S(s)$, then let *SP* be a multisorted Casl specification over the signature Δ containing the single sentence "*false*". Otherwise, let *SP* be a multisorted Casl specification over the signature Δ containing sentences $\phi(f_{w,s})$ and $\phi(b_w)$ for all function symbols $f_{w,s}$ and predicate symbols b_w in Σ, where these sentences are defined as follows. The sentence $\phi(f_{w,s})$ is:

$$\forall x^* : (\alpha^S)^*(w) \cdot \alpha^F_{w,s}(f)_{(\alpha^S)^*(w),\alpha^S(s)}(x^*) = \beta^F_{w,s}(f)_{(\alpha^S)^*(w),\alpha^S(s)}(x^*)$$

and the sentence $\phi(b_w)$ is

$$\forall x^* : (\alpha^S)^*(w) \cdot \alpha^P_w(b)_{(\alpha^S)^*(w)}(x^*) \iff \beta^P_w(b)_{(\alpha^S)^*(w)}(x^*)$$

We claim that SP expresses $\alpha = \beta$, that is, for any Δ-specification SP_0 in MultiCasl: $[\![SP_0]\!] \subseteq [\![SP]\!]$ iff, for all $M \in [\![SP_0]\!]$, we have $M|_\alpha = M|_\beta$.

"$\Rightarrow$": Take any $M \in [\![SP_0]\!]$. Then $M \in [\![SP]\!]$. For any sort s in Σ, the equality $\alpha^S(s) = \beta^S(s)$ must hold, since otherwise SP would contain the sentence *false* and, hence, be inconsistent. Thus $M^S(\alpha^S(s)) = M^S(\beta^S(s))$. It is easy to see that the sentences $\phi(f_{w,s})$ and $\phi(b_w)$ force the requested equality to hold for function and predicate symbols as well.

"$\Leftarrow$": Take any MultiCasl Δ-specification SP_0 such that, for all $M \in [\![SP_0]\!]$, $M|_\alpha = M|_\beta$. Take any model $M \in [\![SP_0]\!]$.

Because, by definition of model in MultiCasl, in M carriers of all sorts are non-empty, there exists a model N with all carriers distinct and which is isomorphic to M. By Proposition 49, M and N satisfy precisely the same Δ-specifications. Thus $N \in [\![SP_0]\!]$ and, therefore, $N|_\alpha = N|_\beta$. In particular, for any sort s in Σ, the equality $N^S(\alpha^S(s)) = N^S(\beta^S(s))$ holds. By distinctness of carriers, this implies that $\alpha^S(s) = \beta^S(s)$.

As a consequence, SP is a specification containing the sentences $\phi(f_{w,s})$ and $\phi(b_w)$. What remains to be proved is that M satisfies all these sentences, since then $M \in [\![SP]\!]$. But this is obvious, since $M|_\alpha = M|_\beta$, so, for any $f_{w,s}$ in Σ, we must have

$$M^F(\alpha^F_{w,s}(f)_{(\alpha^S)^*(w),\alpha^S(s)}) = M^F(\beta^F_{w,s}(f)_{(\beta^S)^*(w),\beta^S(s)})$$

which implies that M satisfies $\phi(f_{w,s})$. A similar argument applies to $\phi(b_w)$. $\square$

In order to apply Theorem 47, an appropriate class $\mathcal{S}$ of finite signature sinks in SubCasl has to be chosen. Obviously, the bigger this class, the more general a verification method will emerge. As a natural choice, let $\mathcal{S}$ be the sinks which may appear as amalgamation sinks in real Casl architectural specifications, as they are defined in the Reference Manual [CoF04b] for the specification language SubCasl. To make this more precise, let $\mathcal{E}$ be the class of morphisms $\eta : \Sigma \to \Delta$ in SubCasl such that η^S, η^F and η^P are surjective functions and the subsort preorder in Δ is the least preorder generated by the η^S-image of the subsort preorder of Σ. Then sinks in $\mathcal{S}$ are sinks which arise either in

$$\bigcup_{i\in\{1\}} \sigma_i(T_i)$$

where $\sigma_1 : \Delta_1 \to \Delta$ is in $\mathcal{E}$, or in

$$T_1 + T_2$$

Note that the latter form, i.e., consisting of the inclusions $\iota_{\Delta_1 \subseteq \Delta_1 \cup \Delta_2}$ and $\iota_{\Delta_2 \subseteq \Delta_1 \cup \Delta_2}$, is the only form of amalgamation in CASL architectural specifications, and the former form is equivalent to the special CASL construct T_1 **with** σ_1. In this construct, it is always required that $\sigma_1 \in \mathcal{E}$. In other words, the sinks in $\mathcal{S}$ cover all the cases of amalgamation legal in CASL architectural specifications.

Note that $\eta \in \mathcal{E}$ does not imply that $\alpha^!(\eta)$ is an epimorphism in the signature category of MULTICASL. The reason is that the status of embeddings in SUBCASL and MULTICASL differs.

The second type of sink in $\mathcal{S}$ can be shown to actually reduce to the first, in the following sense (below, $\sqcup$ denotes a coproduct):

Proposition 51. *Assume Σ_1 and Σ_2 are compatible signatures in* SUBCASL. *If $\sigma : \Sigma_1 \sqcup \Sigma_2 \to \Sigma_1 \cup \Sigma_2$ is the universal morphism, then $\sigma \in \mathcal{E}$.* □

The choice of $\mathcal{S}$-sinks turns out to not have been too generous, so that the fundamental property required from them does, indeed, hold:

Proposition 52. *All sinks in $\mathcal{S}$ are finite and R-expressible in* MULTICASL, *where R :* SUBCASL $\to$ MULTICASL *is the standard representation.*

Proof. It is obvious that sinks in $\mathcal{S}$ are finite. Below, it will be shown that if $\eta : \Sigma \to \Sigma'$ is in $\mathcal{E}$, then $(\alpha^!(\eta), \alpha(\Sigma'))$-extendability is expressible in MULTICASL. This shows that all sinks in $\mathcal{S}$ are R-expressible. To see this implication, consider two cases. In the first case, the sink consists of one morphism $\sigma_1 : \Delta_1 \to \Delta$, where $\sigma_1 \in \mathcal{E}$. If $\iota_1 : \Delta_1 \to \Delta_*$ is a coproduct of Δ_1, then it is an isomorphism. Thus, the universal morphism $\eta : \Delta_* \to \Delta$ is also in $\mathcal{E}$, since $\eta = \iota_1^{-1}; \sigma_1$ and so, by the argument below, $(\alpha^!(\eta), \alpha(\Delta))$-extendability is expressible in MULTICASL. In the second case, the sink consists of two inclusions $\sigma_1 : \Delta_1 \to \Delta$ and $\sigma_2 : \Delta_2 \to \Delta$, where $\Delta = \Delta_1 \cup \Delta_2$. If $\iota_1 : \Delta_1 \to \Delta_*$ and $\iota_2 : \Delta_2 \to \Delta_*$ is a coproduct of Δ_1 and Δ_2, and $\eta : \Delta_* \to \Delta$ is the universal morphism, then, by Proposition 51, $\eta \in \mathcal{E}$. Again, by the argument below, $(\alpha^!(\eta), \alpha(\Delta))$-extendability is expressible in MULTICASL.

So, assume that $\eta : \Sigma \to \Sigma'$ is in $\mathcal{E}$ and let $\Delta = \alpha^!(\Sigma)$, $\Delta' = \alpha^!(\Sigma')$ and $\theta = \alpha^!(\eta)$. Assume $\Delta = (S, TF, PF, P)$ and $\Delta' = (S', TF', PF', P')$. Also, let $\leq$ be the subsort preorder of Σ and $\leq'$ that of Σ'. We prove that $(\theta, \alpha(\Sigma'))$-extendability is expressible in MultiCasl. Note that the reduct in MultiCasl is written $\_|_\sigma$, as usual, even though this could lead to confusion with the reduct in SubCasl.

To prove condition (1) of the definition of the expressability of $(\theta, \alpha(\Sigma'))$-extendability, consider the following three groups of equalities $\alpha = \beta$ of kind $\Theta \to \Delta$:

- for any sorts s, s' in Δ such that $\theta^S(s) = \theta^S(s')$, let Θ be a signature with only one sort t and let α take t to s and β take t to s',

- for any function symbols $f_{w,s}$ and $f'_{w',s'}$ in Δ, where $w = s_1 \ldots s_n$ and $w' = s'_1 \ldots s'_n$, such that $\theta^F_{w,s_0}(f) = \theta^F_{w',s'_0}(f')$, let Θ be a signature with $n+1$ sorts $t_1, \ldots, t_n, t$ and one function symbol $f : t_1 \times \cdots \times t_n \to t$ and let α take t_i to s_i, t to s and $f_{t_1 \ldots t_n, t}$ to $f_{w,s}$ and let β take t_i to s'_i, t to s' and $f_{t_1 \ldots t_n, t}$ to $f'_{w',s'}$,

- for any predicate symbols b_w and $b'_{w'}$ in Δ, where $w = s_1 \ldots s_n$ and $w' = s'_1 \ldots s'_n$, such that $\theta^P_w(b) = \theta^P_{w'}(b')$, let Θ be a signature with n sorts $t_1, \ldots, t_n$ and one predicate symbol $b : t_1 \times \cdots \times t_n$ and let α take t_i to s_i and $b_{t_1 \ldots t_n}$ to b_w and let β take t_i to s'_i and $b_{t_1 \ldots t_n}$ to $b'_{w'}$.

We claim that a Δ-model M is θ-extendible iff, for all of the above-defined equalities $\alpha = \beta$, we have $M|_\alpha = M|_\beta$.

The "only if" part is fairly obvious: if N is a Δ'-model with $N|_\theta = M$ and $\alpha = \beta$ is one of the above equalties, then, since $\alpha; \theta = \beta; \theta$, we have $M|_\alpha = (N|_\theta)|_\alpha = N|_{\alpha;\theta} = N|_{\beta;\theta} = (N|_\theta)|_\beta = M|_\beta$.

For the "if" part, suppose that a Δ-model M indeed satisfies all of the required equalities. Define a Δ'-model N as follows:

- for any sort s' in Δ', let $N^S(s') = M^S(s)$, for any sort s in Δ with $s' = \theta^S(s)$ (since $\eta \in \mathcal{E}$, such an s must exist),

- for any function symbol $f'_{w',s'}$ in Δ', let $N^F(f'_{w',s'})$ is:

  - $M^F(f_{w,s})$, if $w' = (\theta^S)^*(w)$, $s' = \theta^S(s)$ and $f' = \theta^F_{w,s}(f)$ for some function symbol $f_{w,s}$ in Δ,
  - any function of the appropriate type, otherwise,

- for any predicate symbol $b'_{w'}$ in Δ', $N^P(b'_{w'})$ is:

- $M^P(b_w)$ if $w' = (\theta^S)^*(w)$ and $b' = \theta^P_w(b)$ for some predicate symbol b_w in Δ,
- the empty relation, otherwise.

This definition is correct precisely because, for any of the considered equalities $\alpha = \beta$, we have $M|_\alpha = M|_\beta$. Directly from the definition it follows that $M = N|_\theta$. This proves condition (1) of the definition. Thus, we move to condition (2).

For any sorts s, t in Δ', an (s,t)-*path* q is a sequence $s_1 \leq \cdots \leq s_n$ $(n > 1)$ of sorts in Δ such that $s = \theta^S(s_1)$, $t = \theta^S(s_n)$ and that $s_i = s_j$ and $1 \leq i < j \leq n$ imply $i = 1$ and $j = n$. Observe that in the path no loops can occur except for the whole path being a loop. Then, for any Δ'-term T of sort s, by $q(T)$ we denote the "composition-of-embeddings" term $em_{\theta^S(s_{n-1}) \leq \theta^S(s_n)}(\ldots em_{\theta^S(s_1) \leq \theta^S(s_2)}(T)\ldots)$. These notions are further extended to tuples w of sorts and q^* of sequences (the sequence at each coordinate may be of different length). Observe that since $\eta \in \mathcal{E}$, for any sorts s and t in Δ' such that $s \leq' t$, there exists an (s,t)-path. Also, the set of all paths is finite.

As the specification SP' from condition (2), take a MultiCasl Δ'-specification $\alpha(\Delta') \cup SP''$, where SP'' contains the following sentences:

(A) for any (s,t)-paths q_1 and q_2, the sentence

$$\forall x : s \cdot q_1(x) = q_2(x)$$

(B) for any $f_{w_1,s_1} \sim'_F f_{w_2,s_2}$ in Σ', any (w,w_1)-paths q_1^*, (w,w_2)-paths q_2^*, (s_1,s)-path q_1 and (s_2,s)-path q_2, the sentence

$$\forall x^* : w \cdot q_1(f_{w_1,s_1}(q_1^*(x^*))) = q_2(f_{w_2,s_2}(q_2^*(x^*)))$$

(C) for any $b_{w_1} \sim'_P b_{w_2}$ in Σ', any (w,w_1)-paths q_1^* and (w,w_2)-paths q_2^*, the sentence

$$\forall x^* : w \cdot b_{w_1}(q_1^*(x^*)) \iff b_{w_2}(q_2^*(x^*))$$

Let N be a Δ-model. We will prove that:

1. if N has a θ-extension satisfying $\alpha(\Sigma')$, then any of its θ-extensions satisfies SP',
2. if N has a θ-extension satisfying SP', then it has a unique θ-extension satisfying $\alpha(\Sigma')$.

Observe that, by definition of SP', for Δ'-models P, Q, if $P|_\theta = Q|_\theta$, then $P \in [\![SP']\!]$ iff $Q \in [\![SP']\!]$. Also, $[\![\alpha(\Sigma')]\!] \subseteq [\![SP']\!]$. Now, if $N' \in [\![\alpha(\Sigma')]\!]$ and $N = N'|_\theta = N''|_\theta$, then $N' \in [\![SP']\!]$ and $N'|_\theta = N''|_\theta$, hence $N'' \in [\![SP']\!]$. This proves point (1).

As for point (2), take any model N and its θ-extension N' satisfying SP'. Below, by $\theta(\Delta)$ the "image" of Δ along θ is denoted, i.e., the set of symbols which are images of symbols in Δ; by $\Delta' \setminus \theta(\Delta)$ all other symbols in Δ' are denoted; note that $\theta(\Delta)$ need *not* be a signature, because two embeddings in Δ which did not compose may be mapped by θ to embeddings in Δ' which do compose. Define a Δ'-model M' as follows:

- carriers, function symbols and predicate symbols (including embeddings, etc.) in $\theta(\Delta)$ are interpreted in M' as in N',
- for any embedding $em_{s \leq t}$ in $\Delta' \setminus \theta(\Delta)$, set

  $$(M')^F(em_{s\leq t}) = (N')^F(em_{\theta^S(s_1) \leq \theta^S(s_2)}); \dots ; (N')^F(em_{\theta^S(s_{n-1}) \leq \theta^S(s_n)})$$

  where $q = s_1 \leq \cdots \leq s_n$ is an (s,t)-path,
- for any projection $prj_{t \geq s}$ in $\Delta' \setminus \theta(\Delta)$, set $(M')^F(prj_{t\geq s})$ to be the least right-sided inverse of $(M')^F(em_{s\leq t})$,
- for any predicate $in_{s,t}$ in $\Delta' \setminus \theta(\Delta)$, set

  $$(M')^P(in_{s,t}) = \{x \in (M')^S(s) \mid (M')^F(prj_{s \geq t})(x) \text{ is defined}\}$$

The fact that N' satisfies the sentences from point (A) ensures that this definition is correct. It is also clear that M' is a θ-extension of N. We now check that $M' \in [\![\alpha(\Sigma')]\!] = [\![(\Sigma')^{\#\#}]\!]$.

Defining sentence no. 1 (the notion of defining sentences has been introduced in Section 2) holds, since all self-embeddings are in $\theta(\Delta)$. Defining sentences 3, 4 and 5 hold by definition of M'. Sentence 2 is a consequence of 3 and of the original embeddings being injective in N. Sentences 6 and 7 are consequences of the sentences from points (B) and (C).

Since it is obvious that any θ-extension of N satisfying defining sentences 1-7 is forced to be defined as above, this completes the proof. $\square$

The results of this section may now be summed up. The theorem below shows that all of the data needed to perform verification using the proof calculus $\vdash^g$ has been constructed. Thus, Theorem 47 may be applied to the specification language SubCasl.

Theorem 53. *The standard representation R is a representation of (finite)* SubCasl *in (finite)* MultiCasl. *The specification language of finite* MultiCasl *has a finitely cocomplete signature category, amalgamation, conjunction and translations and all morphism equalities are expressible in it.*

The class $\mathcal{S}$ is a class of finite signature sinks in SubCasl *and all $\mathcal{S}$-sinks are R-expressible in* MultiCasl.

Proof. That MultiCasl has a cocomplete signature category and amalgamation has already been stated in Section 3. It is clear that the colimit of a finite diagram of finite signatures is a finite signature, too, and so finite MultiCasl has a finitely cocomplete signature category. The structuring constructs $\cup$ and **with** on specifications guarantee that MultiCasl has conjunction and translations, respectively. Finally, Proposition 50 says that in MultiCasl all morphism equalities are expressible, and Proposition 52 says that $\mathcal{S}$ contains only sinks which are finite and R-expressible. $\square$

Corollary 54. *For any statically correct generative refinement statement in* SubCasl *using only amalgamation along sinks from $\mathcal{S}$:*

$$\mathcal{IT} ::_{\perp} \mathcal{OT} \text{ holds in SubCasl}$$

$$\textit{iff}$$

$$\vdash^{g} \mathcal{IT} ::_{\perp} \mathcal{OT}$$

$\square$

Recall that sinks from $\mathcal{S}$ are those sinks that may arise in SubCasl by using the **with** construct or the **and** construct; in this thesis, both of these constructs are treated as cases of amalgamation.

18 EnrichedCasl

In this section, a total representation of SubCasl is presented, which translates SubCasl to a new specification language, EnrichedCasl. The specification language EnrichedCasl is, essentially, a "completed" version of SubCasl, namely one with amalgamation and iso-reflection. The language is obtained from SubCasl via two fundamental modifications:

- the subsort preorder is replaced by a right-cancellative subsort category,
- classes of function and predicate symbols are closed with respect to the subsort embeddings.

The first of these changes is needed in order for EnrichedCasl to have amalgamation and the second one in order for it to have iso-reflection. EnrichedCasl has been first defined in [SMT+05].

The main reason for constructing a representation of SubCasl in EnrichedCasl is to apply Theorems 41 and 42. Since EnrichedCasl is not in any way an established specification language, there is no support whatsoever for theorem-proving in it, etc. Even worse, there has been no research on problems such as syntax for EnrichedCasl. For Theorem 41, i.e., for checking extended static correctness, all this is not a problem, since this theorem actually relies only on the model part of the languages and on a model representation. Moreover, questions concerning computing colimits in EnrichedCasl have indeed been tackled, see [KHT+01]. But for real verification, in particular in order to apply Theorem 42 in practice, lack of fixed syntax, proof calculi, etc., is a fundamental problem. Since solving this problem is related rather to specification languages than to formalizing program development, it is out of the scope of this thesis. Therefore we concentrate on providing only a model representation of SubCasl in EnrichedCasl, and abstract away from the syntax of sentences and specifications in EnrichedCasl. Choosing an appropriate precise definition of specifications in EnrichedCasl is a difficult matter due to the number of details that have to be taken into account, and due to the fact that this syntax has to be as friendly to users and theorem-provers as possible. However, a general idea of what EnrichedCasl specifications should look like is easy to come up with by analogy with SubCasl specifications. Thus, in basic specifications, the main difference with respect to SubCasl is that subsort embeddings in terms are not uniquely identified by the sorts they relate; the main problem with defining EnrichedCasl specifications is thus developing rules on when these embeddings may be omitted, etc. In structured specifications, the standard structuring constructs used in this thesis should be used. It is clear that in this way the presented below model representation in the model part of EnrichedCasl can be extended to a "full" representation in "full" EnrichedCasl, whatever its exact definition turns out to be.

Before EnrichedCasl is defined, some categorical notation has to be introduced. If $\varphi : F \to G$ and $\psi : G \to H$ are natural transformations between functors with domain S, then $\varphi * \psi : F \to H$ denotes their composition, given by $(\varphi * \psi)_s = \varphi_s; \psi_s$ for all objects s in S. If $F : S \to S'$ is a functor and $l : G \to H$ is a natural transformation between functors $G : S' \to C$ and $H : S' \to C$, then $F \cdot l : (F; G) \to (F; H)$ denotes the composition of F and l. A category is called *small* if the class of all its morphisms is a set. By $\mathbf{Set_{1-1}}$ the category of sets and injective functions is denoted; similarly, by $\mathbf{Class_{1-1}}$ the category of classes and injective functions. A category is said to be *right-cancellative* if it consists solely of monomorphisms. A *subfunctor* F_0 of a functor $F : C \to \mathbf{Class}$ is a functor $F_0 : C \to \mathbf{Class}$ such that $F_0(o) \subseteq F(o)$ for all objects o in C and $F_0(m : o \to o')$ is the restriction of $F(m : o \to o')$ to $F_0(o)$, for all morphisms $m : o \to o'$ in C. Given a functor $F : C \to D$, the functor $F^{op} : C^{op} \to D^{op}$ is "the same" as the functor F, but considered between the opposite categories.

A more general notational remark is that composition in categories which are treated as elements of signatures, etc., is denoted simply by juxtaposition, without a semicolon (but still in diagrammatical order; hence, categories of monomorphisms are right-, not left-cancellative).

An *enriched presignature* consists of:

- a category S,
- functors $G_n : S^{op} \times \cdots \times S^{op} \times S \to \mathbf{Class}$ (n copies of S^{op}), for all $n \in \mathbb{N}$; we write $(e_1, \ldots, e_n) \cdot x \cdot e$ for $G_n(e_1, \ldots, e_n, e)(x)$, where $e_1, \ldots, e_n, e$ are morphisms in S and x is in $G_n(\mathrm{cod}(e_1), \ldots, \mathrm{cod}(e_n), \mathrm{dom}(e))$,
- subfunctors T_n of G_n, for all $n \in \mathbb{N}$,
- functors $P_n : S^{op} \times \cdots \times S^{op} \to \mathbf{Class}$, for all $n \in \mathbb{N}$; we write $(e_1, \ldots, e_n) \cdot x$ for $P_n(e_1, \ldots, e_n)(x)$, where $e_1, \ldots, e_n$ are morphisms in S and x is in $P_n(\mathrm{cod}(e_1), \ldots, \mathrm{cod}(e_n))$.

For the presignature to be an *enriched signature*, it is additionally required that:

- S is a right-cancellative category,
- for all $n \in \mathbb{N}$, $G_n(\mathrm{id}_{s_1}, \ldots, \mathrm{id}_{s_n}, \_) : S \to \mathbf{Class_{1-1}}$,
- for all $n \in \mathbb{N}$, for $x \in G_n(s_1, \ldots, s_n, s)$ and $e : s \to s'$, if

  $$(\mathrm{id}_{s_1}, \ldots, \mathrm{id}_{s_n}) \cdot x \cdot e$$

  is in $T_n(s_1, \ldots, s_n, s')$, then x is in $T_n(s_1, \ldots, s_n, s)$.

Objects of S are called *sorts* and morphisms of S are called *embeddings*. For sorts $s_1, \ldots, s_n, s$, elements of $G_n(s_1, \ldots, s_n, s)$ are called *function symbols with profile* $s_1 \times \cdots \times s_n \to s$ and elements of $P_n(s_1, \ldots, s_n)$ *predicate symbols with profile* $s_1 \times \cdots \times s_n$. A function symbol in $T_n(s_1, \ldots, s_n, s)$ is said to be *total*. Non-total function symbols are considered *partial*.

A morphism σ between enriched presignatures from an enriched presignature $\Sigma = (S, \{G_n\}_{n\in\mathbb{N}}, \{P_n\}_{n\in\mathbb{N}})$ to $\Sigma' = (S', \{G'_n\}_{n\in\mathbb{N}}, \{P'_n\}_{n\in\mathbb{N}})$ consists of a functor $F : S \to S'$, a family of natural transformations $\{g_n :$

$G_n \to (F^{op} \times \cdots \times F^{op} \times F); G'_n\}_{n\in\mathbb{N}}$ and a family of natural transformations $\{p_n : P_n \to (F^{op} \times \cdots \times F^{op}); P'_n\}_{n\in\mathbb{N}}$; additionally, it is required that the natural transformations g_n preserve totality of function symbols. Composition with a morphism $\sigma' : \Sigma' \to \Sigma''$, where $\Sigma'' = (S'', \{G''_n\}_{n\in\mathbb{N}}, \{P''_n\}_{n\in\mathbb{N}})$, consisting of a functor F' and natural transformations $\{g'_n : G'_n \to ((F')^{op} \times \cdots \times (F')^{op} \times F'); G''_n\}_{n\in\mathbb{N}}$ and $\{p'_n : P'_n \to ((F')^{op} \times \cdots \times (F')^{op}); P''_n\}_{n\in\mathbb{N}}$ consists of the functor $F; F'$ and the natural transformations $\{g_n * ((F^{op} \times \cdots \times F^{op} \times F) \cdot g'_n) : G_n \to ((F;F')^{op} \times \cdots \times (F;F')^{op} \times (F;F')); G''_n\}_{n\in\mathbb{N}}$ and $\{p_n * ((F^{op} \times \cdots \times F^{op}) \cdot p'_n) : P_n \to ((F;F')^{op} \times \cdots \times (F;F')^{op}); P''_n\}_{n\in\mathbb{N}}$.

A signature morphism between enriched presignatures is considered an inclusion if the functor F is an inclusion of categories, i.e., it is a set-theoretic inclusion on objects and on morphisms, and the natural transformations g_n and p_n, for $n \in \mathbb{N}$, all consist of inclusions in **Class**.

Below, by a partial function from X to Y we understand a triple $f = (X, f_0, Y)$ where $f_0 : X \to Y$ is partial. The function is total if $\mathrm{dom}(f_0) = X$.

Let $\mathbf{Set_p}$ be the following enriched signature:

- the sort category is $\mathbf{Set_{1-1}}$ with the empty set removed,
- for any non-empty sets $X_1, \ldots, X_n$ and X, G_n takes them to the class of all partial functions from $X_1 \times \cdots \times X_n$ to X; for any $e_1 : Y_1 \to X_1, \ldots, e_n : Y_n \to X_n$ and $e : X \to Y$ in $\mathbf{Set_{1-1}}$ and any partial function $f = (X_1 \times \cdots \times X_n, f_0, X)$, we define $(e_1, \ldots, e_n) \cdot f \cdot e = (Y_1 \times \cdots \times Y_n, (e_1 \times \cdots \times e_n); f_0; e, Y)$,
- for any non-empty sets $X_1, \ldots, X_n$, P_n takes them to the set of all subsets of $X_1 \times \cdots \times X_n$; for any $e_1 : X_1 \to Y_1, \ldots, e_n : X_n \to Y_n$ in $\mathbf{Set_{1-1}}$ and any $p \subseteq Y_1 \times \cdots \times Y_n$, we define $(e_1, \ldots, e_n) \cdot p = \{(x_1, \ldots, x_n) \in X_1 \times \cdots \times X_n \mid (e_1(x_1), \ldots, e_n(x_n)) \in p\}$.

It is straightforward to verify that the above data form an enriched signature.

The class of signatures of the language EnrichedCasl is defined to consist of all small enriched signatures, whereby a signature is small if its sort category is small and the functors involved only produce sets (not proper classes).

For any small enriched signature Σ, the class of Σ-models in EnrichedCasl is defined to consist of all signature morphism $M : \Sigma \to \mathbf{Set_p}$. This mapping easily extends to a model functor, if, for any signature morphism $\sigma : \Sigma \to \Delta$ of small enriched signatures and any Δ-model $M : \Delta \to \mathbf{Set_p}$, the reduct $M|_\sigma$ is defined to be $\sigma; M$.

As explained in the introduction to this section, problems regarding the specification level of EnrichedCasl are not tackled. Therefore, one may describe EnrichedCasl as the specification language with the above-defined signatures, inclusions and models and with only total specifications; the latter defines *Spec*, $sig[\_]$ and $[\![\_]\!]$. It is clear that this defines a specification language.

A model representation $R = (\alpha^!, \alpha, \beta)$ of SubCasl in EnrichedCasl may be defined as follows. Assume $\Sigma = (S, TF, PF, P, \leq)$ is a subsorted signature. Then $\alpha^!(\Sigma)$ consists of:

- a category with objects S and exactly one morphism $em_{s \leq s'} : s \to s'$ for all $s \leq s'$; composition is defined by $em_{s \leq s'} em_{s' \leq s''} = em_{s \leq s''}$ and identity by $\mathrm{id}_s = em_{s,s}$,

- for any $n \in \mathbb{N}$, $G_n(w, s)$ is the set of all f such that $f \in TF_{w',s'} \cup PF_{w',s'}$ for some $w \leq^* w'$ and $s' \leq s$; for $w'' \leq^* w$ and $s \leq s''$, we define $em^*_{w'' \leq^* w} \cdot f \cdot em_{s \leq s''} = f$; this clearly defines a functor,

- for any $n \in \mathbb{N}$, T_n is defined exactly like G_n, except that $f \in TF_{w',s'}$ is required; of course, T_n is a subfunctor of G_n,

- for any $n \in \mathbb{N}$, $P_n(w)$ is the set of all b such that $b \in P_{w'}$ for some $w \leq^* w'$; for $w'' \leq^* w$, we define $em^*_{w'' \leq^* w} \cdot b = b$; again, this is clearly a functor.

The enriched presignature $\alpha^!(\Sigma)$ is, in fact, a small enriched signature. It is obviously small; also, its sort category is right-cancellative, since it is a preorder. Of course, $G_n(\mathrm{id}_{s_1}, \ldots, \mathrm{id}_{s_n}, em_{s \leq s'})$ is an injective function, since it is a set-theoretic inclusion. Also, if $f : s_1 \times \cdots \times s_n \to s$ is a function symbol and $(\mathrm{id}_{s_1}, \ldots, \mathrm{id}_{s_n}) \cdot f \cdot em_{s \leq s'}$ is total, i.e., an element of $T_n(s_1, \ldots, s_n, s')$, then, by definition of T_n, f is total as well, i.e., an element of $T_n(s_1, \ldots, s_n, s)$.

Of course, since R is supposed to be a total model representation, $\alpha(\Sigma)$ is simply defined to be $\top_{\alpha^!(\Sigma)}$. Finally, for any subsorted signature Σ, the isomorphism β_Σ^{-1} takes any subsorted Σ-model M to a signature morphism $\alpha^!(\Sigma) \to \mathbf{Set_p}$ defined as follows:

- the functor F takes any sort s to the non-empty set $M^S(s)$ and any morphism $em_{s \leq s'}$ to the injective function $M^F(em_{s \leq s'})$,

- for any $n \in \mathbb{N}$, if f is an element of the domain $G_n(w, s)$ of the natural transformation g_n at component (w, s), then there must exist $w \leq^* w'$ and $s' \leq s$ such that $f \in TF_{w',s'} \cup PF_{w',s'}$; then f is taken, at that component, to the function $(M^F)^*(em^*_{w \leq^* w'}); M^F(f_{w',s'}); M^F(em_{s' \leq s})$,

- for any $n \in \mathbb{N}$, if b is an element of the domain $P_n(w)$ of the natural transformation p_n at component w, then there must exist $w \leq^* w'$ such that $b \in P_{w'}$; then b is taken, at that component, to the set of all $x^* \in (M^S)^*(w)$ such that $(M^F)^*(em^*_{w \leq^* w'}(x^*)) \in M^P(b_{w'})$.

The functoriality of F is a consequence of defining sentences no. 1 and 3 (defining sentences have been introduced in Section 2). The correctness of the definitions of g_n and p_n is a consequence of defining sentences no. 6 and 7, i.e., is related to the overloading relations; these transformations are natural by definition. Both the projections and the membership predicates have been removed, since they bring no added value to the model theory. It is quite clear that the above-defined β_Σ^{-1} may be inverted and is thus an isomorphism. Thus:

Proposition 55. *The tuple* $(\alpha^!, \alpha, \beta)$ *defined above is a total representation* $R : \text{SubCasl} \to \text{EnrichedCasl}$. □

Most of the nice properties of the representation R, which are prerequisites for applying Theorems 41 and 42, may be proved rather easily:

Theorem 56. *The representation* $R : \text{SubCasl} \to \text{EnrichedCasl}$ *is total and* EnrichedCasl *has a cocomplete signature category, amalgamation and translations.*

Proof. In the statement of the theorem, listing "having translations" is somewhat misleading, since any specification language which has only total specifications has translations and the property is rather meaningless without a non-trivial specification component.

A standard argument shows that the category of small enriched presignatures is cocomplete.

Notice that the forgetful functor from small enriched signatures to small enriched presignatures has a left adjoint. The free enriched signature may be obtained by performing identifications of embeddings to ensure right-cancellativity, and identifications of function symbols to ensure that the functor $G_n(\mathrm{id}_{s_1}, \ldots, \mathrm{id}_{s_n}, \_)$ is a functor into $\mathbf{Set_{1-1}}$, and by inductively growing the set of total function symbols.

Therefore, by first taking, in the category of presignatures, the colimit $\sigma = \{\sigma_i : D(i) \to \Sigma_0\}_{i \in I}$ of a diagram D with shape I of enriched signatures, and by then taking the free object $\eta : \Sigma_0 \to \Sigma$, where Σ is an enriched signature, we obtain a colimit $\{\sigma_i; \eta : D(i) \to \Sigma\}_{i \in I}$. For, if $\tau = \{\tau_i : D(i) \to \Delta\}_{i \in I}$ is a cocone for enriched signatures, then it also is one for presignatures. Thus, there is a unique morphism $\gamma : \Sigma_0 \to \Delta$ factorizing τ through σ. By freeness, there exists a unique morphism $\delta : \Sigma \to \Delta$ such that $\eta; \delta = \gamma$. Thus $(\sigma_i; \eta); \delta = \sigma_i; \gamma = \tau_i$, that is, τ factorizes through our claimed colimit. Also, if we have $(\sigma_i; \eta); \delta' = \tau_i$ for some δ' and all $i \in I$, then $\sigma_i; (\eta; \delta') = \tau_i$ for all $i \in I$ and so, because σ is a colimit, $\eta; \delta' = \gamma$. But δ was unique with such a property, so $\delta = \delta'$.

Thus, small enriched signatures form a cocomplete category.

As for amalgamation, assume D is a diagram of small enriched signatures with shape I and $\sigma = \{\sigma_i : D(i) \to \Sigma\}_{i \in I}$ its colimit. Let $\{M_i\}_{i \in I}$ be any family of models consistent with D. Then $\{M_i\}_{i \in I}$ is, in fact, a cocone in the category of enriched signatures. Certainly, the images of M_i are all small and so the cocone may be treated as a cocone in the category of all small enriched signatures. Therefore there exists a unique morphism $\eta : \Sigma \to \mathbf{Set_p}$ such that $\sigma_i; \eta = M_i$ for all $i \in I$, that is, a unique Σ-model η such that $\eta|_{\sigma_i} = M_i$. This means precisely that η is a σ-amalgamation of $\{M_i\}_{i \in I}$. Hence, EnrichedCasl has amalgamation. □

The two properties of EnrichedCasl not mentioned above are iso-reflection and having consistent signatures. The following theorem holds:

Theorem 57. EnrichedCasl *has iso-reflection and consistent signatures.*

However, in order to prove this theorem, some amalgamation theory for multicategories has to be introduced, which is done in the next section. Then, in Section 20, with the help of the introduced theory, Theorem 57 is proved.

19 Iso-reflection for Multicategories

In order to prove that EnrichedCasl has iso-reflection, which is done in the next section, some fairly advanced amalgamation theory for monoids has to be introduced and generalized to multicategories. In particular, in this section the renowned Zigzag Theorem is recalled and a certain multicategorical generalization is proved. For information on amalgamation theory for monoids and semigroups, see [How96, Hig92]; I am not aware of any work on amalgamation theory for categories, operads or multicategories. At the end of this section, certain consequences of the Zigzag Theorem are stated. Among them, the fact that the functor taking any small multicategory to the class of algebras over it enjoys a weak form of the iso-reflection property.

Recall that a *multicategory*, also called *colored operad*, is a generalization of the notion of category and is defined as follows. Just like a category, it has objects and morphisms. However, the domain of a morphism m is a tuple of objects, so that $m : o_1 \times \cdots \times o_n \to o$, for $n \geq 0$. Composition with m is then defined for $m_i : o^i_1 \times \cdots \times o^i_{k_i} \to o_i$ for $1 \leq i \leq n$, and is written as

$$(m_1, \ldots, m_n); m : o^1_1 \times \cdots \times o^1_{k_1} \times \cdots \times o^n_1 \times \cdots \times o^n_{k_n} \to o$$

The associativity law for composition says that

$$((m^1_1, \ldots, m^{j_1}_1); m_1, \ldots, (m^1_n, \ldots, m^{j_n}_n); m_n); m$$

coincides with

$$(m_1^1, \ldots, m_1^{j_1}, \ldots, m_n^1, \ldots, m_n^{j_n}); ((m_1, \ldots, m_n); m)$$

for appropriate morphisms m_i^j. Identitities $\mathrm{id}_o : o \to o$ are also required to exist and satisfy $(\mathrm{id}_{o_1}, \ldots, \mathrm{id}_{o_n}); m = m = (m); \mathrm{id}_o$.

An *operad* is a multicategory with only one object. In an operad, the notation $\mathrm{dom}(m) = n$, where m is a morphism in the operad and n is a natural number, is used; then n is the *arity* of m. Clearly, a category is a multicategory in which all morphisms have a one-object domain; the awkward-looking composition $(m); m'$ may then be written as $m; m'$. A monoid is an operad which is a category. In small multicategories, operads, etc., we will simply write $(m_1, \ldots, m_n)m$ for the composition, omitting the symbol ";". Any category with products gives rise to a natural multicategory; in the case of the categories **Set** and **Class**, the multicategories defined in this way are denoted by **Set** and **Class** as well.

If C and D are multicategories, then a *multifunctor* $F : C \to D$ maps objects of C to objects of D and morphisms of C to morphisms of D in such a way that it preserves domains and codomains, identities, and composition. Formally, F takes any morphism $m : o_1 \times \cdots \times o_n \to o$ in C to a morphism $F(m) : F(o_1) \times \cdots \times F(o_n) \to F(o)$ in D and satisfies $F(\mathrm{id}_o) = \mathrm{id}_{F(o)}$ and $F((m_1, \ldots, m_n); m) = (F(m_1), \ldots, F(m_n)); F(m)$ for all appropriate objects and morphisms. If C and D are categories, then this notion coincides with that of a functor, and if they are monoids, with that of a monoid homomorphism. By D^C the class of all multifunctors from C to D is denoted (natural transformations between multifunctors will not be used). For more information on multicategories and operads, see the book [Lei04].

An object of the form $[U \subseteq_{i,j} S, T]$, where U, S and T are small multicategories and $i : U \to S$ and $j : U \to T$ are monomorphic multifunctors, is called an *amalgam*. Note that a multifunctor is a monomorphism iff it is injective on objects and faithful; thus, monomorphic multifunctors are, up to isomorphism, inclusions of multicategories. An amalgam of the form $[U \subseteq_{i,i} S, S]$ is called a *special amalgam*.

An amalgam $[U \subseteq_{i,j} S, T]$ is *weakly embeddable* if there exist monomorphic multifunctors $\mu : S \to V$ and $\nu : T \to V$ such that $i; \mu = j; \nu$; that is, an amalgam is weakly embeddable if there exists a monomorphic cocone over it. The amalgam is *embeddable*, if the monomorphic multifunctors additionally satisfy $\mu(S) \cap \nu(T) = \mu(i(U))$ $(= \nu(j(U)))$. Note that, if we assume that $S \cap T = U$ and i and j are just inclusions, then embeddability means that there is a multicategory V such that $S, T \subseteq V$.

Let s and s' be morphisms in S and t and t' morphisms in T. If, for any $\mu : S \to V$ and $\nu : T \to V$ such that $i; \mu = j; \nu$,

- the equality $\mu(s) = \mu(s')$ holds, then $s \parallel_1 s'$,
- the equality $\mu(s) = \nu(t)$ holds, then $s \parallel t$,
- the equality $\nu(t) = \nu(t')$ holds, then $t \parallel_2 t'$.

Clearly, $\parallel_1$ and $\parallel_2$ are equivalence relations and $\parallel$ contains $\{(i(u), j(u)) \mid u \in U\}$.

By a standard argument one can show that the category of all small multicategories is cocomplete. Let $p_1 : S \to P$ and $p_2 : T \to P$ be the pushout of $i : U \to S$ and $j : U \to T$. In fact, the relations $\parallel$, $\parallel_1$ and $\parallel_2$ are defined by the behavior on the pushout or on **Set**-valued multifunctors:

Proposition 58. *For any $s, s' \in S$, the following are equivalent:*

1. *$s \parallel_1 s'$,*
2. *$p_1(s) = p_1(s')$,*
3. *for any multifunctors $M : S \to \mathbf{Set}$ and $N : T \to \mathbf{Set}$ with $i; M = j; N$, we have $M(s) = M(s')$.*

A similar equivalence holds for any $t, t' \in T$. Also, for any $s \in S$ and $t \in T$, the following are equivalent:

1. *$s \parallel t$,*
2. *$p_1(s) = p_2(t)$,*
3. *for any multifunctors $M : S \to \mathbf{Set}$ and $N : T \to \mathbf{Set}$ with $i; M = j; N$, we have $M(s) = N(t)$.*

Proof. We only prove the proposition for the relation $\parallel$; the proofs for $\parallel_1$ and $\parallel_2$ are essentially the same.

The implications (1) $\Rightarrow$ (2) and (1) $\Rightarrow$ (3) are obvious. The implication (2) $\Rightarrow$ (1) is a direct consequence of the pushout property: if $\mu : S \to V$ and $\nu : T \to V$ form a cocone, then there is a unique multifunctor $\eta : P \to V$ with $p_1; \eta = \mu$ and $p_2; \eta = \nu$. Therefore $p_1(s) = p_2(t)$ implies $\mu(s) = \nu(t)$.

As for the implication (3) $\Rightarrow$ (1), it is a consequence of a Cayley-like representation theorem for multicategories. For any multicategory V, there exists a multifunctor $I_V : V \to \mathbf{Set}$ which is an embedding. It is defined by setting $I_V(o)$ to contain all morphisms with codomain o, for all objects o in V, and $I_V(m : o_1 \times \cdots \times o_n \to o)$ to be postcomposition with m. Because of the existence of identities, I_V is injective on objects and faithful (i.e., it is a monomorphism). Now, if $\mu : S \to V$ and $\nu : T \to V$ form a cocone, then so do $\mu; I_V : S \to \mathbf{Set}$ and $\nu; I_V : T \to \mathbf{Set}$. Since I_V is a monomorphism, (3) implies that $\mu(s) = \nu(t)$, that is, (1) holds. $\square$

It turns out that there is a close connection between the relations $\|_1$, $\|_2$ and $\|$ and the embeddability of an amalgam. This connection is established in the following two easy consequences of Proposition 58.

Proposition 59. *For any amalgam $[U \subseteq_{i,j} S, T]$, the following are equivalent:*

1. *the amalgam is weakly embeddable,*
2. *$\|_1 \ = \mathrm{id}_S$ and $\|_2 \ = \mathrm{id}_T$,*
3. *the pushout injections $p_1 : S \to P$ and $p_2 : T \to P$ are monomorphisms.* □

Proposition 60. *An amalgam $[U \subseteq_{i,j} S, T]$ is embeddable iff it is weakly embeddable and $\| \ = \{(i(u), j(u)) \mid u \in U\}$. If it is embeddable, then it is embeddable in the pushout $p_1 : S \to P$, $p_2 : T \to P$.* □

In this section, the main focus is on special amalgams, since, as will be seen, they give a good description of a monomorphic multifunctor $i : U \to S$. First, two trivial facts can be stated:

Proposition 61. *If $[U \subseteq_{i,j} S, T]$ is an amalgam and $\sigma : S \to T$ a multifunctor with $i; \sigma = j$, then:*

- *for any $s \in S$ and $t \in T$, $s \parallel t$ implies $\sigma(s) = t$,*
- *$\|_2 \ = \mathrm{id}_T$.*

Proof. The sink (σ, id_T) is a cocone over (i, j), since $i; \sigma = j = j; \mathrm{id}_T$. Hence, $s \parallel t$ implies, by definition, $\sigma(s) = \mathrm{id}_T(t) = t$. Also, $t \|_2 t'$ implies $\mathrm{id}_T(t) = \mathrm{id}_T(t')$ and so $t = t'$. □

Corollary 62. *If $\sigma : S \to T$ and $\tau : T \to S$ are multifunctors satisfying $i; \sigma = j$ and $j; \tau = i$, then the amalgam $[U \subseteq_{i,j} S, T]$ is weakly embeddable.*

Proof. By Proposition 61, $\|_1 \ = \mathrm{id}_S$ and $\|_2 \ = \mathrm{id}_T$. By Proposition 59, this implies that the amalgam is weakly embeddable. □

In the case of special amalgams, instead of the relations $\|_1$, $\|_2$ and $\|$, it is sufficient to look at a subset of S, introduced by Isbell [Isb66, HI67] and called the dominion. If $[U \subseteq_{i,i} S, S]$ is a special amalgam, then the *dominion of U in S*, written $\mathrm{Dom}_S(U)$, is the set of all $s \in S$ such that, for any multifunctors $\mu, \nu : S \to V$ with $i; \mu = i; \nu$, the equality $\mu(s) = \nu(s)$ holds. Trivially, the following proposition holds:

Proposition 63. *For any $s \in S$, the following are equivalent:*

1. $s \in \mathrm{Dom}_S(U)$,
2. *for any multifunctors $M, N : S \to \mathbf{Set}$ with $i; M = i; N$, the equality $M(s) = N(s)$ holds,*
3. $s \parallel s$. □

Also, we have $i(U) \subseteq \mathrm{Dom}_S(U) \subseteq S$.

That the dominion makes the relations $\parallel_1$, $\parallel_2$ and $\parallel$ superfluous is stated in the following proposition:

Proposition 64. *A special amalgam $[U \subseteq_{i,i} S, S]$ is always weakly embeddable. It is embeddable iff $\mathrm{Dom}_S(U) = i(U)$.*

Proof. By Corollary 62 applied to the special amalgam with $\sigma = \tau = \mathrm{id}_S$, the amalgam is weakly embeddable. By Propositon 61, if $s \parallel t$ then $\mathrm{id}_S(s) = t$, i.e., $s = t$. Thus, $\parallel \; \subseteq \mathrm{id}_S$. By Proposition 60, the amalgam is embeddable iff $s \parallel s$ is equivalent to $s \in i(U)$. But this is equivalent to $\mathrm{Dom}_S(U) = i(U)$, as required. □

Up to now, no connection between dominions, embeddability, etc., and iso-reflection has been made explicit. Consider a model functor taking any multicategory C, which we view as a signature, to the class $\mathbf{Set}^C$ of $\mathbf{Set}$-valued multifunctors with domain C, viewed as models; the reduct with respect to multifunctors (viewed as signature morphisms) for such models is defined naturally by precomposition. Recall that the problem of iso-reflection concerns the influence of the reduct $i; \_ : \mathbf{Set}^S \to \mathbf{Set}^U$, where $i : U \to S$ is a multifunctor, being a bijection, on the properties of i itself. This problem may be naturally split into questions regarding the influence of this reduct being an injection and it being a surjection. The following proposition shows the connection between dominions and the first of these questions:

Proposition 65. *Assume $i : U \to S$ is a monomorphic multifunctor. The following are then equivalent:*

1. *the map $i; \_ : \mathbf{Set}^S \to \mathbf{Set}^U$ is injective,*
2. *i is an epimorphism,*
3. $\mathrm{Dom}_S(U) = S$.

Proof. Assume that (2) or (3) holds. Take any $M, N : S \to \mathbf{Set}$ and assume $i; M = i; N$. If (2) holds, then, by definition of epimorphism, $M = N$, which proves (1). If (3) holds, then, for any $s \in S$, $s \in \mathrm{Dom}_S(U)$. Hence, $M(s) = N(s)$. But this implies $M = N$ and so (1) holds.

Assume (1) holds. Suppose $\mu, \nu : S \to V$ are such that $i; \mu = i; \nu$. Let $I_V : V \to \mathbf{Set}$ be the standard Cayley-representation of V in $\mathbf{Set}$, defined in the proof of Proposition 58. Then $i; \mu; I_V = i; \nu; I_V$. By (1), $\mu; I_V = \nu; I_V$. Because I_V is a monomorphism, this implies $\mu = \nu$. Hence, i is an epimorphism, i.e., (2) holds. In particular, for any $s \in S$, $\mu(s) = \nu(s)$. In other words, $s \in \mathrm{Dom}_S(U)$. This means that $S \subseteq \mathrm{Dom}_S(U)$, which proves (3). $\square$

It is easy to see that, even in the category of all monoids, not all epimorphisms are retractions or are surjective (think of the inclusion of the natural numbers $\mathbb{N}$ in the integers $\mathbb{Z}$). In the category of small categories, there are trivial examples of such a situation:

Example 5. *Let U consist of a single non-identity morphism $f : a \to b$ (a and b distinct) and let, i being an inclusion, S add to that a morphism $g : b \to a$ with $fg = \mathrm{id}_a$ and $gf = \mathrm{id}_b$. Then i is an epimorphism, is not surjective and not a retraction. Note, however, that the map $i;\_ : \mathbf{Set}^S \to \mathbf{Set}^U$, even though it is injective, is not bijective.*

In the analysis of the relations $\|_1$, $\|_2$ and $\|$, and, hence, also in the analysis of dominions, the notion of *zigzag* is the fundamental tool. It has been first introduced by Isbell in his paper on dominions, and the names "type I zigzag", "type II(a) zigzag", etc., come from [How68] and [Hal78]. Because it is incovenient to use the monomorphisms $i : U \to S$ and $j : U \to T$ explicitly, from now on it is always assumed that, in an amalgam $[U \subseteq_{i,j} S, T]$, these monomorphisms are inclusions and, moreover, that $S \cap T = U$. As for special amalgams, one may simply treat amalgams of the form $[U \subseteq_{i^+, i^-} S^+, S^-]$, where the inclusions $i^+ : U \to S^+$ and $i^- : U \to S^-$ are isomorphic, as special.

If $U \subseteq S \cap T$ are multicategories, then a *U-triple in S, T* is a triple (s, u, t) of unary symbols such that $s \in S$, $u \in U$, $t \in T$ and the compositions su and ut may be formed in S and T, respectively. A triple (s, u, t) is connected with a triple (s', u', t') by a *zig* if $su = s'u'$ and $t = t'$ and by a *zag* if $s = s'$ and $ut = u't'$. They are connected by a *U-zigzag* if one may be obtained from the other by a finite number of zigs and zags (this number may be 0). It is assumed that the zigs and zags are interleaved, since two consecutive zigs may be replaced by one (similarily for zags). The first of the above triples is said to be *upper* if $su \in U$; its *value* is then $(su)t$. The triple is *lower* if

$ut \in U$; its *value* is then $s(ut)$. The *head* of a zigzag is its first triple and the *tail* is its last triple.

Zigzags of three types are used:

- a *type I U-zigzag from s to t in S, T* is a U-zigzag in S, T with a lower head with value $s \in S$ and an upper tail with value $t \in T$,

- a *type II(a) U-zigzag from s to s' in S, T* is a U-zigzag in S, T with a lower head with value $s \in S$ and a lower tail with value $s' \in S$,

- a *type II(b) U-zigzag from t to t' in S, T* is a U-zigzag in S, T with an upper head with value $t \in T$ and an upper tail with value $t' \in T$.

Note that, if $U = S = T$, then all triples in a U-zigzag in S, T are lower and upper at the same time, and their value is always the same. In particular, if the zigzag is a type I, II(a) or II(b) zigzag from s to s', then $s = s'$. Now, if $[U \subseteq_{i,j} S, T]$ is an amalgam and $\mu : S \to V$ and $\nu : T \to V$ coincide on U, then any U-zigzag in S, T can be mapped by μ and ν to a V-zigzag in V, V. By the above observation, the V-zigzag will have a constant value. If the zigzag was of type I from s to t, then this implies that $\mu(s) = \nu(t)$; if it was of type II(a) from s to s', then implies that $\mu(s) = \mu(s')$ (similarly for type II(b)). This observation may be summed up as follows:

Proposition 66. *Let $[U \subseteq_{i,j} S, T]$ be an amalgam. Then, for any unary $s, s' \in S$ and $t, t' \in T$:*

- *if there is a type I zigzag from s to t in S, T or one from t to s in T, S, then $s \parallel t$,*

- *if there is a type II(a) zigzag from s to s' in S, T or a type II(b) one in T, S, then $s \parallel_1 s'$,*

- *if there is a type II(b) zigzag from t to t' in S, T or a type II(a) one in T, S, then $t \parallel_2 t'$.* $\square$

Note that, in general, the converse does not hold. However, the situation is different for special amalgams. For them, as stated above, type II zigzags are not interesting, since $\parallel_1 \;=\; \parallel_2 \;=\; \mathrm{id}_S$. But what is crucial is that, for type I zigzags, the converse of Proposition 66 holds. This fact is well-known for the monoid case and called *Isbell's Zigzag Theorem.* Isbell's Zigzag Theorem, first stated by Isbell [Isb66], proofs of which may be found, e.g., in [How96, Phi74] (an elegant, but unfortunately incorrect, proof by Higgins is [Hig90]) is proved in a generalized version below. The classical version is then stated as a corollary. Our proof is an attempt of using the ideas employed in [Hig90] to obtain an intuitive, elementary proof. Note that, in general, the Zigzag Theorem does *not* hold for non-special amalgams, unless additional assumptions are made, e.g., that the monoids concerned are commutative [How68]. The theorem also holds for semigroups [How96].

The *tree operad*, denoted by Θ, is defined as follows. A morphism in the tree operad is either the empty tree, or a node labeled by a natural number n with attached n (sub)trees; we differentiate between the empty tree and the tree consisting of a single node labeled 0: the former is used to represent a variable "x", while the latter represents a constant. The arity of a tree is the number of empty trees it is built of; composition of trees is defined in the obvious way and is associative; the empty tree serves as identity. The empty tree is written as ϵ and the tree consisting of a node labeled n and of n empty trees attached is written $\Theta(n)$.

Let G be a set generating a multicategory S and let G_1 be the set of all unary elements of G. The multicategory S is said to be *G-structure-preserving*, if there exists a multifunctor F from S to the tree operad Θ such that:

- $F(g_1) = \epsilon$ for all $g_1 \in G_1$, and
- $F(g) = \Theta(n)$ for all $g \in G \setminus G_1$ of arity n.

Since G generates S, the above properties define the value of F on all morphisms in S, so that if F exists, then it is unique. A multicategory S is *structure-preserving* if it is structure preserving for some set of generators G. Of course, any category S is structure-preserving, since it suffices to take $G = S$ and consider the map constantly equal ϵ.

Theorem 67 (Generalized Zigzag Theorem). *Assume U is a submulticategory of a G-structure-preserving multicategory S and that G_1 is the set of all unary elements of G.*

Then, for any $g_1 \in G_1$, $g_1 \in \mathrm{Dom}_S(U)$ iff there is a type I U-zigzag from g_1 to g_1.

Moreover, if $G_1 \subseteq U$, then, for any $g \in G$, $g \in \mathrm{Dom}_S(U)$ implies $g \in U$.

Proof. The implication "$\Leftarrow$" of the equivalence is a direct consequence of Proposition 66.

Let X be the multicategory obtained by freely adjoining to S morphisms of the form $x_o : o \to o$, for all objects o in S, satisfying the laws:

- $(x_o)x_o = \mathrm{id}_o$, and
- $(x_{o_1}, \ldots, x_{o_n})u = ux_o$, for all $u : o_1 \times \cdots \times o_n \to o$ from U.

The first of the above rules is called *deletion* or *creation*, depending on the direction in which it is used; the second of the rules is called *shift.* The rule inherited from the composition in S is called *refactorization.* In the sequel, we simply write "x", omitting the index o.

Now consider multifunctors $\mu, \nu : S \to X$ defined by $\mu(s) = s$ and $\nu(s) = ((x, \ldots, x)s)x$ for all $s \in S$. Then μ and ν coincide on U.

By assumption, there is a multifunctor $F : S \to \Theta$ witnessing the fact that S is G-structure-preserving. Elements of X may be represented as trees built of elements of S and of the x; two such trees represent the same element if there exists a sequence of refactorizations, deletions, creations and shifts from one to the other. If an element of X is represented by some tree t built of elements of S and of x, then one may compute, for any s in the tree t, its "shape" $F(s)$. The shape of the whole tree t may then be obtained by putting together the shapes of the particular $F(s)$, and by forgetting the elements x. The obtained tree in Θ is equal $F(J([t]))$, where $J : X \to S$ is a multifunctor defined by $J(x_o) = \mathrm{id}_o$ and $J(s) = s$ for any object o in S and morphism s in S, and where $[t]$ is the equivalence class of t. This implies that if there exists a sequence of refactorizations, deletions, creations or shifts from a tree t to a tree t', and if, at some point in this sequence, the tree t_0 appears, then the shape of t_0 must be the same as the shapes of t and t'. In particular, if the shape of t is the empty tree ϵ, then this must also be the shape of all the trees that appear in the sequence, which implies that all trees in the sequence are built solely of unary morphisms. If the shape of t is $\Theta(n)$, with $n \neq 1$, then again all trees in the sequence must be built of unary morphisms and exactly one morphism of arity n.

Take any $g \in \mathrm{Dom}_S(U)$ (not necessarily unary). It is clear that $\mu(g) = \nu(g)$, that is, $g = ((x, \ldots, x)g)x$ in X, or, in other words, $(g)x = (x, \ldots, x)g$ in X. Thus, there exists a sequence of refactorizations, deletions, creations and shifts from $(g)x$ to $(x, \ldots, x)g$.

So, if g is unary, i.e., if $g = g_1 \in G_1$, then there exists a sequence of refactorizations, deletions, creations and shifts from $(g_1)x$ to $(x)g_1$ and, moreover, in this sequence only unary morphisms appear.

Below, we show that the existence of such a sequence implies that there is a type I U-zigzag from g_1 to g_1 in S, S; note that, in the proof, we may treat S as if it were a category, since non-unary morphisms cannot appear in the zigzag. But before we go on, we prove part two of the theorem. Assume $G_1 \subseteq U$ and that g has arity $n \neq 1$. There exists a sequence of refactorizations, deletions, creations and shifts from $(g)x$ to $(x, \ldots, x)g$ such that, in each tree in the sequence, only unary morphisms and exactly one n-ary morphism appear. But, by assumption, all the unary morphisms come from U. Thus, in each tree in the sequence, there is at most one morphism

s from S of arity n and unary morphisms from U. But this implies that the x at the top of $(g)x$ can only be pushed downwards provided that g is an element of U. Hence, we have $g \in U$, which proves part two of the theorem.

We now return to the sequence of refactorizations, deletions, creations and shifts from g_1x to xg_1 (we may omit the parantheses in $(g_1)x$ and $(x)g_1$, since all the considered morphisms are unary). Formally, this is a sequence of words over the alphabet $S \cup \{x\}$ (we drop the index o for clarity), connected by reductions of the following forms:

- $\langle s, s' \rangle \leftrightarrow \langle ss' \rangle$ or $\langle \mathrm{id}_o \rangle \leftrightarrow \langle \rangle$ (*refactorization*),
- $\langle x, x \rangle \to \langle \rangle$ (*deletion*),
- $\langle \rangle \to \langle x, x \rangle$ (*creation*),
- $\langle x, u \rangle \leftrightarrow \langle u, x \rangle$ (*shift*).

Let us denote the considered sequence from $\langle g_1, x \rangle$ to $\langle x, g_1 \rangle$ by q. We claim that there exists a sequence q' from $\langle g_1, x \rangle$ to $\langle x, g_1 \rangle$ in which deletions and creations do not appear. This will end the proof, since it is easy to see that a sequence in which, at any given point, there is exactly one occurrence of x defines a type I U-zigzag from g_1 to g_1 in S, S.

The crucial idea is to mark some of the occurrences of x in the sequence q by attaching numbers to them. Thus, the first occurrence of x gets number 1 and so, at the start, we have $\langle g_1, x^1 \rangle$. Observe that, in the sequence, one can track the moves of a given occurrence of x, since it simply performs some shifts and, finally, gets deleted. Thus, x^1 either never gets deleted, or it gets deleted by some other occurrence of x, which we call x^{-2}. This x^{-2} must have been created somewhere earlier, and its "pair", with which it has been created, we call x^2. Again, x^2 may never get deleted, or it gets deleted by some x^{-3} which, in turn, has been created together with some x^3. This process may be iterated and, at some moment, must terminate with some x^n ($n \geq 1$) which does not get deleted in the sequence; this implies that the sequence ends with $\langle x^n, g_1 \rangle$.

It is important to see that if, in some sequence q_1, there is some set A of occurrences of x which do not interact in q_1 with other occurrences of x either by being created together with them, or by being deleted together with them, then one may just as well erase all the occurrences of x which are in A from the whole sequence. The obtained sequence q_2 will then be correct after removing any redunduncies, i.e, consecutive repetitions of the same word.

Let A be the set of all occurrences of x which are *not* in the set

$$\{x^1, x^{-2}, x^2, \ldots, x^{-n}, x^n\}$$

The occurrences of x which are in A do not interact with occurrences not in A, and so one may erase them from q, obtaining a sequence q' from $\langle g_1, x^1\rangle$ to $\langle x^n, g_1\rangle$.

Let q_1 be the sequence q' from the start $\langle g_1, x^1\rangle$ to the moment before deleting x^1. Let q_{-2} be the sequence q' from the moment after creating x^{-2} and x^2 to the moment before deleting x^1 and x^{-2}. Let q_2 be the sequence q' from the moment after creating x^{-2} and x^2 to the moment before deleting x^2 and x^{-3}. Again, this process may be iterated, and it terminates with q_n being the sequence q' from the moment after creating x^{-n} and x^n to the end of q', i.e., ending with $\langle x^n, g_1\rangle$.

For any sequence q_*, let q_*^{-1} be q_* reversed. For any integer i and for any sequence q_*, by $q_*|_i$ we denote the sequence q with all occurrences of x except x^i erased and with x^i renamed to x (and with possible redundancies removed); of course, $q_*|_i$ is a correct sequence only if x^i is neither created nor deleted in q_*.

Now consider the sequence obtained by concatenating the sequences $q_1|_1$, $q_{-2}^{-1}|_{-2}$, $q_2|_2$ and so on, up to $q_{-n}^{-1}|_{-n}$, $q_n|_n$. Notice that, in q_i, the occurrence x^i is neither created, nor deleted; the same is true for q_i^{-1}; hence, sequences of the form $q_i|_i$ and $q_{-i}^{-1}|_{-i}$ are correct. Moreover, the concatenation can be performed, since, for positive i, the last element of $q_i|_i$ is equal to the last element of $q_{-(i+1)}|_{-(i+1)}$, that is, to the first element of $q_{-(i+1)}^{-1}|_{-(i+1)}$. Similarily, the last element of $q_{-i}^{-1}|_{-i}$, that is, the first element of $q_{-i}|_{-i}$ is equal to the first element of $q_i|_i$. This makes the concatenated sequence a correct sequence. It is also obvious that in this sequence no creations or deletions occur. Finally, the first element of this sequence is $\langle g_1, x\rangle$ and the last element is $\langle x, g_1\rangle$, which ends the proof. $\square$

From Theorem 67 we obtain the following corollary:

Theorem 68 (Zigzag Theorem). *Assume U is a subcategory of the category S. Then, for any $s \in S$, $s \in \mathrm{Dom}_S(U)$ iff there is a type I U-zigzag from s to s in S, S.* $\square$

The above theorem is just Isbell's Zigzag Theorem generalized from monoids to categories.

Recall that, in Proposition 65, the consequences of the multifunctor $i;\_ : \mathbf{Set}^S \to \mathbf{Set}^U$ being injective were considered, where $i : U \to S$ is a multifunctor. The following proposition shows what the consequences of the surjectivity of $i;\_$ are. Note that, for monoids, it is a weaker version of a theorem from [Hal78], also found in [Hig92].

Proposition 69. *Let $i : U \to S$ be a multifunctor. If the map $i; \_ : \mathbf{Set}^S \to \mathbf{Set}^U$ is surjective, then i is a monomorphism. If, additionally, S is G-structure-preserving, then all unary elements $g_1 \in G$ such that $g_1 \in \mathrm{Dom}_S(U)$ are of the form $i(u)$, where $u \in U$.*

Proof. Let $I_U : U \to \mathbf{Set}$ be the Cayley representation of U in $\mathbf{Set}$. As noted in the proof of Proposition 58, I_U is a monomorphism. By assumption, there is a multifunctor $M : S \to \mathbf{Set}$ such that $i; M = I_U$. Since I_U is a monomorphism, i must also be one (for $\alpha; i = \beta; i$ implies $\alpha; i; M = \beta; i; M$, that is, $\alpha; I_U = \beta; I_U$, which implies $\alpha = \beta$). This proves the first point.

Since i is a monomorphism, we may, without loss of generality, assume that it is an inclusion.

Now define $J : U \to \mathbf{Set}$ as follows. Define, on morphisms in S, syntactic operations $(\_)^+$ and $(\_)^-$ which simply decorate them with "+" or "−" and are such that $u^+ = u^-$ for $u \in U$ and $s^+ = s^-$ for non-unary $s \in S$. Thus, $s^+ = s^-$ iff $s \in U$ or s is not unary. For any object o in U, let $J(o)$ contain s^+ and s^- for all morphisms s in S with codomain o. For any morphism $u \in U$, let $J(u)(s_1^{z_1}, \ldots, s_n^{z_n})$, where n is the arity of u, $s_1, \ldots, s_n \in S$ and $z_1, \ldots, z_n \in \{+, -\}$, be $((s_1, \ldots, s_n)u)^z$, where the sign z is defined as follows: if the result is not unary, the sign z is immaterial; if the result is unary, then there must be exactly one $1 \leq i \leq n$ such that s_i is unary, and all s_j for $1 \leq j \leq n$ with $i \neq j$ must be nullary. In this case the sign z coincides with the sign z_i. Then J is a multifunctor; this is obvious except for the case of the sign of an unary result. So, assume that $J((u_1, \ldots, u_k)u)(s_1^{z_1}, \ldots, s_n^{z_n})$ is unary. Then there must be exactly one $1 \leq i \leq n$ such that s_i is unary, while all other s_j must be nullary; the result will then have the sign z_i. But if we consider the elements $J(u_1)(s_1^{z_1}, \ldots), \ldots, J(u_k)(\ldots, s_n^{z_n})$, then the one in which s_i appears will be unary and have the sign z_i, while all others will be nullary. Hence, the result of applying $J(u)$ to them will, again, have sign z_i, as required.

By assumption, there is a functor $M : S \to \mathbf{Set}$ such that $i; M = J$. Suppose $g_1 \in G \cap \mathrm{Dom}_S(U)$ is unary. By Theorem 67, there is a U-zigzag of type I from g_1 to g_1 in S, S. Thus, there is a U-zigzag between the triples (s_1, u_1, s_1') and (s_r, u_r, s_r'), such that $u_1 s_1' \in U$ and $s_1(u_1 s_1') = g_1$, and $s_r u_r \in U$ and $(s_r u_r) s_r' = g_1$.

Observe that $g_1^+ = M(s_i')((s_i u_i)^+)$ for all $1 \leq i \leq r$. The proof is by induction over i. For $i = 1$ we have $g_1^+ = (s_1(u_1 s_1'))^+ = J(u_1 s_1')(s_1^+) = M(u_1 s_1')(s_1^+) = M(s_1')(M(u_1)(s_1^+)) = M(s_1')(J(u_1)(s_1^+)) = M(s_1')((s_1 u_1)^+)$.

Assume that the claim holds for some $i \geq 1$. Then the next move in the zigzag is either a zig or a zag. In the first case, the claim obviously holds for $i + 1$, since $s_i' = s_{i+1}'$ and the elements to which $M(s_i') = M(s_{i+1}')$ is applied are equal as well. In the second case, $s_i = s_{i+1}$ and $u_i s'_i = u_{i+1} s'_{i+1}$. Then

$$\begin{aligned}
g_1^+ &= M(s'_i)((s_i u_i)^+) \\
&= M(s'_i)(J(u_i)(s_i^+)) \\
&= M(s'_i)(J(u_i)(s_{i+1}^+)) \\
&= M(s'_i)(M(u_i)(s_{i+1}^+)) \\
&= M(u_i s'_i)(s_{i+1}^+) \\
&= M(u_{i+1} s'_{i+1})(s_{i+1}^+) \\
&= M(s'_{i+1})(M(u_{i+1})(s_{i+1}^+)) \\
&= M(s'_{i+1})(J(u_{i+1})(s_{i+1}^+)) \\
&= M(s'_{i+1})((s_{i+1} u_{i+1})^+)
\end{aligned}$$

as required.

An analogous claim may be proved for the decoration "−". By these claims, we have, for the last, rth triple,

$$\begin{aligned}
g_1^+ &= M(s'_r)((s_r u_r)^+) \\
&= M(s'_r)((s_r u_r)^-) \\
&= g_1^-
\end{aligned}$$

since $s_r u_r \in U$. This proves that $g_1^+ = g_1^-$ and so $g_1 \in U$.

□

An important remark regarding the above proposition has to be made here. Notice that the Cayley representation $I_U : U \to \mathbf{Set}$ is actually a representation in the multicategory of non-empty sets. Also, the multifunctor $J : U \to \mathbf{Set}$ constructed in the proof is a multifunctor into the multicategory of non-empty sets. Therefore, the above proposition holds even if the map $i;\_$ is onto the class of all multifunctors from U to the multicategory of non-empty sets (not necessarily onto the class of all **Set**-valued multifunctors from U).

From Propositions 65 and 69, a theorem stating a weak form of iso-reflection for multicategories and multifunctors into **Set** may easily be obtained:

Theorem 70. *For any multifunctor $\sigma : S \to T$ from a multicategory S to a category T, σ is an isomorphism iff the map $\sigma;\_ : \mathbf{Set}^T \to \mathbf{Set}^S$ is bijective.*

Proof. The implication "⇒" is obvious. As for "⇐", if $\sigma;\_$ is surjective, then σ must be a monomorphism, by Proposition 69. It may thus be assumed that it is an inclusion (up to isomorphism). Since T is a category, it is T-structure-preserving. Hence, by the same proposition, $\mathrm{Dom}_T(S) = S$. Since $\sigma; \_$ is injective and σ is a monomorphism, $\mathrm{Dom}_T(S) = T$, by Proposition 65. Thus, $S = T$ and so σ is, up to isomorphism, the identity. Hence, it is an isomorphism. $\square$

20 Iso-reflection for EnrichedCasl

The theory developed in Section 19 may now be applied to EnrichedCasl. In a nutshell, the idea is that any signature corresponds to a multicategory and any model to a multifunctor from that multicategory into **Set**. Among the technical problems that have to be solved for this to work, the most important one is guaranteeing that sort embeddings are interpreted as injective functions.

Note that, throughout this section, predicates are not taken into account. They are added to the picture at the very end, in the proof of Theorem 57 itself.

As can be seen from the previous section, working with multicategories can be very tedious. In this section, most concepts and propositions are presented in detail only in the case of all function symbols being unary, i.e., in the categorical case. The general, multicategorical versions are then described in a less formal manner.

For any enriched signature Σ with only unary function symbols and no predicate symbols, consider all non-empty sequences $\langle a_1, \ldots, a_n \rangle$ such that each a_i is either an embedding or a function symbol together with its profile and such that $\mathrm{cod}(a_i) = \mathrm{dom}(a_{i+1})$ for all $1 \leq i \leq n-1$, whereby $\mathrm{dom}(f : o_1 \to o) = o_1$ and $\mathrm{cod}(f : o_1 \to o) = o$. The sequence itself has a naturally defined domain and codomain.

Let $\sim$ be the least equivalence relation on the considered sequences, such that, for appropriate sequences p and q,

- $p\langle (f : o_1 \to o), e \rangle q \sim p\langle (f : o_1 \to o) \cdot e \rangle q$,
- $p\langle e, e' \rangle q \sim p\langle ee' \rangle q$,
- $p\langle e, (f : o_1 \to o) \rangle q \sim p\langle e \cdot (f : o_1 \to o) \rangle q$.

Clearly, $\sim$ is a congruence with respect to concatenation of sequences. Also, $\sim$ preserves the domain and codomain of sequences.

A *preterm* over Σ is an equivalence class $[p]$ with respect to $\sim$, where p is a sequence. The domain and codomain of a preterm are defined naturally. We write $T_0(\Sigma)$ for the set of all preterms over Σ. Also, composition of preterms $[p]$ and $[q]$ with $\mathrm{cod}(p) = \mathrm{dom}(q)$ may be defined by the rule $[p][q] = [pq]$. It is, obviously, associative and, for any object o in the sort category, the preterm $[\langle \mathrm{id}_o \rangle]$ is an identity for preterms with domain or codomain o.

Note that preterms do not, in general, form a Σ-model. The reason is that $[p][\langle e \rangle] = [q][\langle e \rangle]$ does not imply $[p] = [q]$:

Example 6. *Let Σ contain:*

- *sorts s, s_1, s_2, s_3, s' and s'',*
- *non-identity embeddings $e_1 : s_1 \to s_2$, $e_3 : s_3 \to s_2$ and $e : s' \to s''$,*
- *function symbols:*
  - *$f_1 : s \to s_1$, $f_2 : s \to s_2$ and $f_3 : s \to s_3$, with $f_1 \cdot e_1 = f_2 = f_3 \cdot e_3$,*
  - *$g_1 : s_1 \to s'$ and $g_3 : s_3 \to s'$,*
  - *$g_1' : s_1 \to s''$, $g_2' : s_2 \to s''$ and $g_3' : s_3 \to s''$, with $g_1 \cdot e = g_1'$, $g_3 \cdot e = g_3'$ and $g_1' = e_1 \cdot g_2'$ and $g_3' = e_3 \cdot g_2'$.*

Then we have $\langle f_1, g_1 \rangle \not\sim \langle f_3, g_3 \rangle$, even though $\langle f_1, g_1, e \rangle \sim \langle f_1, g_1' \rangle \sim \langle f_1, e_1, g_2' \rangle \sim \langle f_2, g_2' \rangle \sim \langle f_3, e_3, g_2' \rangle \sim \langle f_3, g_3' \rangle \sim \langle f_3, g_3, e \rangle$.

Let $\equiv$ be the least congruence, with respect to concatenation of sequences, containing $\sim$ and such that, for all sequences p and q with common domains and codomains and for all embeddings e with $\mathrm{dom}(e) = \mathrm{cod}(p)$ $(= \mathrm{cod}(q))$, if $p\langle e \rangle \equiv q\langle e \rangle$ then $p \equiv q$. A *term* over Σ is an equivalence class $\llbracket p \rrbracket$ with respect to $\equiv$, where p is a sequence. We write $T(\Sigma)$ for the set of all terms over Σ. As with preterms, the domains and codomains of preterms, composition and identities may be defined naturally. Also, if $\llbracket p \rrbracket \llbracket \langle e \rangle \rrbracket = \llbracket q \rrbracket \llbracket \langle e \rangle \rrbracket$, then $\llbracket p \rrbracket = \llbracket q \rrbracket$.

The *term Σ-model* $I = (F, \{l_1\}, \emptyset) : \Sigma \to \mathbf{Set_p}$ is defined as follows. The functor F takes any sort o to the set of all Σ-terms with codomain o, and any embedding $e : o \to o'$ to the function taking any term $\llbracket p \rrbracket$ of sort o and returning the term $\llbracket p \rrbracket \llbracket \langle e \rangle \rrbracket$. Clearly, all the carriers of sorts are non-empty, the interpretations of embeddings are injective, the interpretations of identities are identities and $F(ee') = F(e); F(e')$. The natural transformation l_1 takes any function symbol $f : o_1 \to o$ in Σ to the function taking any term $\llbracket p \rrbracket$ with codomain o_1 to the term $\llbracket p \rrbracket \llbracket \langle f : o_1 \to o \rangle \rrbracket$ with codomain o. It is easy to check that l_1 is, indeed, natural.

The term Σ-model has the peculiarity that all function symbols in Σ are interpreted by total functions, even though some of the function symbols may be partial. The *partial term Σ-model* is defined in the same way as the term Σ-model, except that, if $f : o_1 \to o$ is partial, then the function $l_1(f : o_1 \to o)$ is undefined on the term $[\![\langle \mathrm{id}_{o_1} \rangle]\!]$. This defines a Σ-model, since $f \cdot e$ must be partial and, if $f = f_0 \cdot e$ or $f = e \cdot f_0$, then f_0 is partial as well; finally, even though $e \cdot f$ may be total, l_1 is natural, since, if the range of the interpretation of e contains $[\![\langle \mathrm{id}_{o_1} \rangle]\!]$, then e is an isomorphism, which implies that $f = e^{-1} \cdot (e \cdot f)$ and so f is, in fact, total. Note that, in the partial term Σ-model, partial symbols are always interpreted by partial functions.

The above-defined notions of preterm, term, term model and partial term model may be all easily generalized to the case of function symbols with arities other than 1. All that is needed is the use of trees instead of sequences.

For preterms of a very simple form, the following proposition holds:

Proposition 71. *If $\langle e \rangle \sim p$, then $p = \langle e_1, \ldots, e_n \rangle$, where $e_1 \ldots e_n = e$ and $n \geq 1$.*

If $\langle f : o_1 \to o \rangle \sim p$, then $p = \langle e_1, \ldots, e_k, (g : o_1' \to o'), e_1', \ldots, e_n' \rangle$, where $f = e_1 \ldots e_k \cdot (g : o_1' \to o') \cdot e_1' \ldots e_n'$ and $k, n \geq 0$.

Proof. The proof is by induction on the length of a minimal reduction path which proves the equivalence.

If the minimal path has length 0, then the proposition holds.

Suppose the proposition holds for paths of length n and consider a case in which the minimal path has length $n+1$. Let p_0 be the last sequence in the path before p. Clearly, the path up to p_0 must be minimal. Also, its length is n. Thus, for the first point, we have $p_0 = \langle e_1, \ldots, e_n \rangle$, where $e_1 \ldots e_n = e$ and $n \geq 1$. The last transition can only split some e_i into e' and e'' with $e_i = e'e''$ or join some e_i and e_{i+1} into $e_i e_{i+1}$. In both cases, the proposition holds. For the second point, we have $p_0 = \langle e_1, \ldots, e_k, (g : o_1' \to o'), e_1', \ldots, e_n' \rangle$ and the appropriate equation holds. In the last transition, either some of the embeddings may be joined or split, or the function symbol g may be joined or split with some embeddings. In either case, the proposition holds. $\square$

The above proposition effectively means that all preterms with 0 or 1 function symbols may be uniquely represented by a sequence of length 1.

Proposition 72. *For sequences with 0 or 1 function symbols, the relations $\sim$ and $\equiv$ coincide.*

Proof. Assume that p and q have 0 or 1 function symbols and $p\langle e \rangle \sim q\langle e \rangle$ for some embedding e.

If they contain 0 function symbols, then we have $p = \langle e_1, \dots, e_n\rangle$, $q = \langle e'_1, \dots, e'_k\rangle$ and, by Proposition 71, $e_1 \dots e_n e = e'_1 \dots e'_k e$. Since the sort category is right-cancellative, this implies $e_1 \dots e_n = e'_1 \dots e'_k$. Hence, $p \sim q$.

If they contain 1 function symbol, then the proof is analogous, but, instead of right-cancellativity, the property has to be used that the functor $G_1(\mathrm{id}_o, \_) : S \to \mathbf{Class_{1-1}}$ is a functor into the category of classes and injective functions, where o is the domain of both sequences. $\square$

Both of the above propositions continue to hold if other than unary function symbols are considered.

One can define a notion of "shape" of a preterm, describing the points in the preterm where function symbols of arities other than 1 are used. For any non-empty tree built of function symbols and embeddings, one can produce an element of the tree operad Θ by recording the arity of all function symbols which are not unary, and erasing all unary function symbols and all embeddings. Notice that, for trees which are equivalent with respect to $\sim$, the obtained result will be the same. This is because rules defining $\sim$ do not change the arity of symbols, and erase only embeddings, which are all unary. Thus, we can define a function $\theta(\_)$ which, for any preterm $[t]$, returns an element $\theta([t])$ of the tree operad. In fact, $\theta(\_)$ is a multifunctor from the multicategory of preterms with concatenation of trees modulo $\sim$ to the tree operad Θ. If t is a tree containing the single function symbol $f : o_1 \times \cdots \times o_n \to o$, where $n \neq 1$, then $\theta([t]) = \Theta(n)$; if $n = 1$ or if t is a tree containing a single embedding, then $\theta([t]) = \epsilon$. Since preterms built of single function symbols or single embeddings generate all preterms, this shows that the multicategory of preterms is structure-preserving.

Assume $\Sigma = (S, \{G_n\}_{n\in\mathbb{N}}, \emptyset)$ is an enriched signature without predicate symbols. For any non-isomorphism s in S, let $\bar{s}$ be s decorated by a bar and let $\bar{S}$ be the set of all $\bar{s}$, where s is a non-isomorphism in S. Let $\bar{s}$ be a new morphism going in the opposite direction than s and such that $s\bar{s} = \mathrm{id}_{\mathrm{dom}(s)}$. By adjoining the morphisms from $\bar{S}$ to S we have formed a category extending S and such that any morphism s in S has a right inverse. Note that this category is the free category generated by $S \cup \bar{S}$ and by equations saying that s composed with s' in the generated category coincides with s composed with s' in S and that s composed with $\bar{s}$ in the generated category coincides with the identity $\mathrm{id}_{\mathrm{dom}(s)}$, for all appropriate $s, s' \in S$. This is because the new morphisms $\bar{s}$ are all distinct and are distinct from the old morphisms of the form s — the reason is that we have not added right inverses to morphisms which already possessed them, as the following proposition shows:

Proposition 73. *If S is the sort category of an enriched signature, then any retraction or coretraction in S is an isomorphism.*

Proof. Suppose $st = \mathrm{id}_a$ for $s : a \to b$ and $t : b \to a$.

We have $(ts)t = t(st) = t\,\mathrm{id}_a = t = \mathrm{id}_b\,t$. Hence, by right-cancellativity, we get $ts = \mathrm{id}_b$. $\square$

Had we added a right inverse $\bar{s}$ to an isomorphism s, we would have $\bar{s} = s^{-1}s\bar{s} = s^{-1}$.

Consider now the set of "preterms" over Σ equipped with the extended sort category generated by $S \cup \bar{S}$ (formally, these are not preterms, since the signature is not an enriched signature). As before, this set forms a multicategory, written $\hat{\Sigma}$, generated by preterms of the form $[\langle s \rangle]$ for $s \in S$, $[\langle \bar{s} \rangle]$ for non-isomorphic $s \in S$ and $[\langle f : o_1 \times \cdots \times o_n \to o \rangle]$ for function symbols f in Σ and by the appropriate equations. Note that if $\sigma : \Sigma \to \Delta$ is an enriched signature morphism, then this gives rise to the canonical multifunctor $\hat{\sigma} : \hat{\Sigma} \to \hat{\Delta}$. Now let G^{Σ} contain the following preterms:

- $[\langle s \rangle]$ for morphisms s in S,
- $[\langle \bar{s} \rangle]$ for non-isomorphic morphisms s in S,
- $[\langle f : o_1 \times \cdots \times o_n \to o \rangle]$ for function symbols $f : o_1 \times \cdots \times o_n \to o$ in Σ.

We write G_1^{Σ} for the set of unary elements of G^{Σ}. By definition, G^{Σ} generates the multicategory $\hat{\Sigma}$.

What is crucial is that the multicategory $\hat{\Sigma}$ is G^{Σ}-structure-preserving. The multifunctor $\theta(\_)$ defined on preterms over Σ can be extended to a multifunctor $\theta(\_) : \hat{\Sigma} \to \Theta$, which returns the shape of any given morphism in $\hat{\Sigma}$. This is a consequence of the general fact that if a map defined on the generators of $\hat{\Sigma}$ is consistent with the appropriate equations, then it extends to a functor from $\hat{\Sigma}$. The equations in question are those discussed above and concerning composition of preterms built of embeddings or of an embedding and of its right inverse, as well as equations concerning composition of preterms built of a function symbol and of an embedding.

Proposition 74. *If $i : \Sigma \to \Delta$ is an enriched signature morphism between signatures without predicate symbols, and if the map $i;\_ : (\mathbf{Set_p})^{\Delta} \to (\mathbf{Set_p})^{\Sigma}$ is surjective, then i is a monomorphism and reflects totality. Moreover, $\hat{i} : \hat{\Sigma} \to \hat{\Delta}$ is a monomorphism.*

Proof. The proof is only presented for the case of all function symbols in Σ and Δ being unary; the generalization to signatures with symbols of other arities is straightforward.

Consider the term Σ-model $I : \Sigma \to \mathbf{Set_p}$, where $I = (F, \{l_1\}, \emptyset)$. Let J be Δ-model such that $i; J = I$; such a model exists by surjectivity of the reduct map.

Observe that the Σ-model I is a monomorphism. First, F is injective on objects, since the term $[\![\langle \mathrm{id}_o \rangle]\!]$ is an element only of the carrier of o. Second, F is faithful, because, if $e, e' : o \to o'$ are embeddings, then $F(e)$ takes $[\![\langle \mathrm{id}_o \rangle]\!]$ to $[\![\langle e \rangle]\!]$ and $F(e')$ takes it to $[\![\langle e' \rangle]\!]$. Thus $F(e) = F(e')$ implies $[\![\langle e \rangle]\!] = [\![\langle e' \rangle]\!]$ which, by Proposition 72, implies $e = e'$. Third, the transformation l_1 is injective. For if $f, g : o_1 \to o$ are function symbols, then $l_1(f)$ takes $[\![\langle \mathrm{id}_o \rangle]\!]$ to $[\![\langle f : o_1 \to o \rangle]\!]$ and $l_1(g)$ takes it to $[\![\langle g : o_1 \to o \rangle]\!]$. Thus $l_1(f) = l_1(g)$ implies $[\![\langle f \rangle]\!] = [\![\langle g \rangle]\!]$, which implies $f = g$.

Since $I = i; J$ is mono, i must be mono as well.

Now let $I' : \Sigma \to \mathbf{Set_p}$ be the partial term Σ-model. By surjectivity, there exists a Δ-model J' such that $i; J' = I'$. Let $f : o_1 \to o$ be a partial function symbol in Σ. Then I' interprets f as a partial function. The model J' interprets $i(f)$ in the same way and so also as a partial function. Since J' is a Δ-model, this means that $i(f)$ is a partial symbol in Δ. Thus, i preserves partiality, which is equivalent to saying that it reflects totality.

Suppose now that i is, in fact, an inclusion. Note that i being a monomorphism and reflecting totality is equivalent to it being isomorphic to an inclusion.

Suppose that $[p]$ and $[q]$ are distinct morphisms in $\hat{\Sigma}$. Consider the Cayley representation $I_{\hat{\Sigma}} : \hat{\Sigma} \to \mathbf{Set}$. This representation defines a Σ-model $M : \Sigma \to \mathbf{Set_p}$; this model M is in principle a term model, but one in which the right inverses $\bar{s}$ are also used to build elements of the domains of interpretations of embeddings or function symbols. By surjectivity of $i; \_$ there exists a model $N : \Delta \to \mathbf{Set_p}$ such that $i; N = M$. Consider a functor $K : \hat{\Delta} \to \mathbf{Set_p}$ defined as follows: on generators from $\hat{\Sigma}$ the functor K is defined like $I_{\hat{\Sigma}}$; on generators from $\hat{\Delta}$ which are of the form $[\langle s \rangle]$ or $[\langle f : o_1 \to o \rangle]$ the functor K is defined like the model N on s or $f : o_1 \to o$, respectively; on generators from $\hat{\Delta}$ which are not in $\hat{\Sigma}$ and have the form $[\langle \bar{s} \rangle]$, the functor K is defined as any right inverse of the interpretation of $[\langle s \rangle]$ in N (such an inverse exists, since the interpretation in N is injective). As a consequence of the general fact cited above, this definition is correct, since in overlapping cases the functor $I_{\hat{\Sigma}}$ and the model N coincide and since the appropriate equations are satisfied. Also, K extends $I_{\hat{\Sigma}}$; since the latter is, by definition, a monomorphism, we must have $I_{\hat{\Sigma}}([p]) \neq I_{\hat{\Sigma}}([q])$. Hence, $K([p]) \neq K([q])$. But this proves that $[p] \neq [q]$ in $\hat{\Delta}$. In other words, the distinctness of $[p]$ and $[q]$ in $\hat{\Sigma}$ proves their distinctness in $\hat{\Delta}$. Thus $\hat{i} : \hat{\Sigma} \to \hat{\Delta}$ is a monomorphism. $\square$

If the premises of the above proposition hold, then the inclusion of $\hat{\Sigma}$ into $\hat{\Delta}$ is denoted by $\hat{i} : \hat{\Sigma} \to \hat{\Delta}$.

The construction of $\hat{\Sigma}$ allows us to translate iso-reflection for enriched signatures to iso-reflection for multicategories:

Proposition 75. *If $i : \Sigma \to \Delta$ is an inclusion of enriched signatures without predicate symbols, and if $i; \_ : (\mathbf{Set_p})^\Delta \to (\mathbf{Set_p})^\Sigma$ is bijective, then $\Sigma = \Delta$.*

Proof. From Propositions 74 and from the surjectivity of $i; \_$ we may infer that the inclusion $\hat{i} : \hat{\Sigma} \to \hat{\Delta}$ is well-defined.

We first show that the map $\hat{i}; \_ : \mathbf{Set}^{\hat{\Delta}} \to \mathbf{Set}^{\hat{\Sigma}}$ is onto the class of all functors from $\hat{\Sigma}$ to the category of all non-empty sets.

Take any $\hat{M} : \hat{\Sigma} \to \mathbf{Set}$ such that all $M(o)$, where o is an object of $\hat{\Sigma}$, are non-empty. Let M take any embedding e in Σ to $\hat{M}([\langle e \rangle])$ and any function symbol $f : o_1 \times \cdots \times o_n \to o$ in Σ to $\hat{M}([\langle f : o_1 \times \cdots \times o_n \to o \rangle])$. Then M is a signature morphism $M : \Sigma \to \mathbf{Set_p}$, i.e., a Σ-model. This is because interpretations of embeddings are all injective functions (because of the existence of the one-sided inverses $\bar{e}$) and interpretations of all function symbols are total functions.

Thus, by the bijectivity assumption, there is a unique model $N : \Delta \to \mathbf{Set_p}$ such that $i; N = M$. Assume $N = (F^N, \{l_i^N\}_{i \in \mathbb{N}}, \emptyset)$.

Define $\hat{N} : \hat{\Delta} \to \mathbf{Set}$ as follows:

- for any embedding e in Δ, $\hat{N}([\langle e \rangle]) = F^N(e)$,
- for any non-isomorphic embedding e in Σ, $\hat{N}([\langle \bar{e} \rangle]) = \hat{M}([\langle \bar{e} \rangle])$,
- for any non-isomorphic embedding e in Δ and not in Σ, let $\hat{N}([\langle \bar{e} \rangle])$ be any function f such that $F^N(e); f = F^N(\mathrm{id}_{\mathrm{dom}(e)})$; such a function exists, since $F^N(e)$ is injective,
- for any function symbol $f : o_1 \times \cdots \times o_n \to o$ in Δ, $\hat{N}([\langle f : o_1 \times \cdots \times o_n \to o \rangle]) = l_i^N(f : o_1 \times \cdots \times o_n \to o)$.

The above definition is correct, since it covers all generators and satisfies the appropriate equations on generators.

We have $\hat{i}; \hat{N} = \hat{M}$. This implies that $\hat{i}; \_ : \mathbf{Set}^{\hat{\Delta}} \to \mathbf{Set}^{\hat{\Sigma}}$ is onto the class of all functors from $\hat{\Sigma}$ to the category of non-empty sets. Also, $\hat{\Delta}$ is G^Δ-structure-preserving. Hence, by Proposition 69 and by the remark following it, all unary elements g_1 of $G^\Delta \cap \mathrm{Dom}_{\hat{\Delta}}(\hat{\Sigma})$ are elements of $\hat{\Sigma}$ as well.

Moreover, let $K : \hat{\Delta} \to \mathbf{Set}$ be any functor such that $\hat{i}; K = \hat{M}$. Define $K' : \Delta \to \mathbf{Set_p}$ by $K' = (F^{K'}, \{l_i^{K'}\}_{i \in \mathbb{N}}, \emptyset)$, where $F^{K'}(e) = K([\langle e \rangle])$ for any embedding e in Δ and $l_i^{K'}(f : o_1 \times \cdots \times o_n \to o) = K([\langle f : o_1 \times \cdots \times o_n \to o \rangle])$ for any function symbol $f : o_1 \times \cdots \times o_n \to o$ in Δ. This definition is correct, since the existence of elements of the form $[\langle \bar{e} \rangle]$ in $\hat{\Delta}$ and the equation

$[\langle e\rangle][\langle \bar{e}\rangle] = [\langle \mathrm{id}_{\mathrm{dom}(e)}\rangle]$ guarantee that $F^{K'}(e)$ is injective. Since the equation $i; K' = M$ holds, by the uniqueness of N we have $K' = N$.

In other words, for embeddings e or function symbols $f : o_1 \times \cdots \times o_n \to o$ in Δ, $[\langle e\rangle]$ and $[\langle f : o_1 \times \cdots \times o_n \to o\rangle]$ are elements of $\mathrm{Dom}_{\hat{\Delta}}(\hat{\Sigma})$, since the way a functor K from $\hat{\Delta}$ which extends a functor $\hat{M}$ from $\hat{\Sigma}$ works on these elements is unique. But this means that $[\langle e\rangle]$ and $[\langle f : o_1 \to o\rangle]$ are all elements of $\hat{\Sigma}$ (notice that this follows immediately only for unary elements). In other words, $G_1^{\Delta} \subseteq \hat{\Sigma}$. Thus, by Theorem 67, elements of the form $[\langle f : o_1 \times \cdots \times o_n \to o\rangle]$, for $n \neq 1$, are also in $\hat{\Sigma}$; in other words, $G^{\Delta} \subseteq \hat{\Sigma}$, which implies $\hat{\Delta} = \hat{\Sigma}$. It is easy to see that, as a consequence of Proposition 71 (generalized to arbitrary arities), this implies $\Delta = \Sigma$, which ends the proof. $\square$

With the above results, it is now possible to prove the result promised in Section 18, namely Theorem 57.

Theorem 57. EnrichedCasl *has iso-reflection and consistent signatures.*

Proof. As for consistent signatures, it is sufficient to note that, for any enriched signature Σ without predicate symbols, the term Σ-model is an EnrichedCasl model of Σ. If Σ has predicate symbols, then one only needs to interpret the predicates by empty sets.

Assume $\sigma : \Sigma \to \Delta$ is an enriched signature morphism and the reduct map $\_|_{\sigma} = \sigma; \_ : \mathbf{Mod}(\Delta) \to \mathbf{Mod}(\Sigma)$ is a bijection.

Assume first that Σ and Δ do not contain predicate symbols.

By Proposition 74, σ is a monomorphism and reflects totality. It is thus isomorphic to an inclusion and so it may be assumed, without loss of generality, that it is an inclusion.

By Proposition 75, $\Sigma = \Delta$ and so σ is the identity. Any signature morphism isomorphic to the identity is an isomorphism.

Now assume Σ or Δ contain predicate symbols. Let $\sigma_0 : \Sigma_0 \to \Delta_0$ arise by removing predicate symbols from $\sigma : \Sigma \to \Delta$. Since $\_|_{\sigma}$ is a bijection, $\_|_{\sigma_0}$ must be one as well: if M_0 is a Σ_0-model and M is obtained from M_0 by interpreting all predicates by empty sets, then there is exactly one σ-extension M of N; then $N|_{\Delta_0}$ is a σ_0-extension of M_0; moreover, it is a unique σ_0-extension, since if N_0 is a σ_0-extension of M_0, then N_0 with predicates interpreted by empty sets would be a σ-extension of M, and so would be equal N; hence, we would have $N_0 = N|_{\Delta_0}$. Thus, we may assume that σ_0 is an identity. What remains to be proved is that σ is also an identity on the predicate symbols.

Consider the term Σ_0-model M and assume $b : o_1 \times \cdots \times o_n$ is a predicate symbol in Σ. Let the interpretation of any predicate symbol b' be the empty set, if b' has arity different than n, and otherwise if $b' : o'_1 \times \cdots \times o'_n$ let it be the set of all tuples $([\![\langle e_1 \rangle]\!], \ldots, [\![\langle e_n \rangle]\!])$, where $e_1 : o_1 \to o'_1, \ldots, e_n : o_n \to o'_n$ are such that $(e_1, \ldots, e_n) \cdot b' = b$; this extension of M we denote by M_b. Note that M_b is a Σ-model; we check this for unary predicates b_1 and b_2 with $e_0 \cdot b_1 = b_2$. We then have, for any e, that $[\![\langle e \rangle]\!]$ is in the interpretation of b_2 iff $e \cdot b_2 = b$, that is, iff $e \cdot e_0 \cdot b_1 = (ee_0) \cdot b_1 = b$. But this holds iff $[\![\langle ee_0 \rangle]\!]$ is in the interpretation of b_1, as required. Now, it is clear that $(\mathrm{id}, \ldots, \mathrm{id}) \cdot b = b$ and so a tuple of identities is in the interpretation of b in M_b. It is also clear that, for any other predicate symbol b', this tuple cannot be in the interpretation of b', since then $(\mathrm{id}, \ldots, \mathrm{id}) \cdot b' = b$, i.e., $b' = b$. Since b was chosen arbitrary, this implies that σ must be injective on predicate symbols, since otherwise it would be impossible to find a model N with $N|_\sigma = M_b$. Hence, σ may be assumed to be an identity on embeddings and function symbols, and an inclusion overall.

Let M be the term Σ-model with all predicate symbols interpreted by empty sets. There exists a unique Δ-model N with $N|_\sigma = M$. Clearly, if N' is equal N, but with all predicate symbols interpreted by empty sets, then $N'|_\sigma = M$ and so, by uniqueness, $N' = N$, i.e., all predicate symbols in Δ are, in N, interpreted by empty sets. Let b be a predicate symbol in Δ and which is not in Σ. Consider the model N'' defined to be like N on embeddings and function symbols and like M_b on predicate symbols, that is, if b' is a predicate symbol with arity different than that of b, then the interpretation of b' is the empty set and otherwise it is the set of tuples $([\![\langle e_1 \rangle]\!], \ldots, [\![\langle e_n \rangle]\!])$ such that $(e_1, \ldots, e_n) \cdot b' = b$. Note that, σ_0 being an identity, all embedding always are in Σ, so that, if b' comes from Σ, then no appropriate tuple may exist, since otherwise b would come from Σ as well. Therefore the interpretations of predicate symbols from Σ are empty sets. Hence, $N''|_\sigma = M$. By the uniqueness of N, we have $N'' = N$, i.e., the interpretation of b in N'' must also be the empty set. But this is a contradiction, since $(\mathrm{id}, \ldots, \mathrm{id}) \cdot b = b$. Therefore b cannot exist, which implies that all predicate symbols in Σ are in Δ as well, that is, σ is an identity (up to isomorphism). $\square$

21 Verification for a Non-generative Semantics

In Section 12, the conclusion has been reached that a non-generative semantics should not be considered for architectural specifications. The reason was that non-trivial use of non-generativity leads to problems with extending correctness of the development process on the model level to correctness on the program level. Therefore, in the preceding sections, verification of architectural specifications has been considered only with respect to a generative semantics.

Still, the non-generative semantics is, at the moment, the official semantics of CASL architectural specifications. Even from a purely scientific point of view, it is worth seeing what differences and similarities exist for generative and non-generative verification. All this creates a legitimate interest in developing verification techniques for non-generative semantics.

A very informal survey of such techniques is presented in this section of the thesis. More technical information may be found in [Hof01]; note, however, that [Hof01] concentrates on first-order logic and does not deal with the problem of imports.

The general idea for performing non-generative verification is very simple and may be described as follows. Assume an architectural specification ASP declares two parametric units F and G, whereby F is applied n times and G is applied k times in ASP. The problem of verifying the correctness of ASP with respect to a non-generative semantics can easily be reduced to, unfortunately slightly extended, problems of generative verification. Consider any two applications of F in ASP. There are two cases: either the arguments of these applications differ, in which case the non-generativity is irrelevant, or they coincide, in which case the results must coincide as well. This observation leads to taking any equivalence relations $\sim_F$ and $\sim_G$ on the sets of applications of F and G, respectively, in ASP. For any such $\sim_F$ and $\sim_G$, one can produce equations which say that the arguments of equivalent applications are equal and the arguments of non-equivalent applications are not equal. One then transforms ASP to an architectural specification in which non-equivalent applications are renamed, i.e., treated as applications of different parametric units (for instance by using F_1, F_2, etc.), and equivalent applications are forced to give equal results. The latter can be accomplished, e.g., by taking $F(T_1)$ and $F(T_2)$ and replacing them by an earlier declaration $A : \top$ **given** $F(T_1)$ and, twice, the term A. In this way, the problem of non-generative verification can be reduced to a number of problems of generative verification, all augmented with sets of equalities and non-equalities between (unit) terms.

The equalities on terms do not pose a serious problem: after all, this is exactly what had to be dealt with in the case of amalgamation. In fact, an equality $T_1 = T_2$ is simply equivalent to the correctness of an amalgamation $T_1 + T_2$. Thus, standard techniques may be used here. This reduces the problem further to generative verification augmented with sets of non-equalities, i.e., expressions of the form $T_1 \neq T_2$.

The latter problem is more difficult to solve. Non-equalities are not easy to deal with and much depends on the concrete logic used. In particular, the rather subtle problem of non-equalities between carriers, etc., and its influence on satisfaction of sentences, even if usually rather limited, has to be considered. Another problem is the, again, usually limited, influence of non-equalities on amalgamability. These problems have been described in [Hof01] for total first-order logic in a version allowing for empty carriers in models; enforcing non-emptieness of the carriers should make the verification simpler.

Apart from the problem of non-equalities, another trouble is the number of generative verification problems to which the original non-generative verification problem gets reduced. This number is, unfortunately, in general expotential with respect to the number of applications of parametric units, since the number of equivalences on a set is expotential. However, in practice these numbers should not be too big, since the same unit will usually not be applied more than a few times. Moreover, most applications of the same unit will be obviously non-equivalent.

22 Literature and Conclusions

Not much literature may be found on the verification of architectural specifications. Verification assuming extended static correctness has been discussed in detail by myself in [MHAH04]. There are, however, two main differences between the approach there and in this thesis.

First of all, the model semantics used in [MHAH04] is called "partial", but it is not as partial as the one used in this thesis. The difference is that non-parametric units are not allowed to be replaced by the special object $\perp$. This simplifies the proof calculus used for verification and makes it easier to understand. However, as shown in Chapter II of this thesis, the "fully" partial model semantics allows architectural specifications to describe the program development process more closely, in particular guaranteeing that a flattening theorem holds (Theorem 10). Due to this, it is for this semantics that verification techniques have been presented in this thesis, even though this has made them somewhat more complex.

The second difference is that extended static correctness itself, a notion introduced in [SMT$^+$01b] and then used in [Hof03, CoF04b, SMT$^+$05], is usually defined via a calculus, not semantically (though note that a semantic approach has been proposed in [Hof01] under the name of static/strict correctness). The benefit of the calculus approach is that it is then easier to come up with a method for checking extended static correctness. However, thanks to the definition of fully static correctness, which is provided by a calculus, this benefit was of no value here and, for this reason, a more succint, semantical definition of extended static correctness has been used.

It should be noted that methods of checking extended static correctness have been described in [SMT+01b] and, in the case of SubCasl, in [SMT+05]. In the latter paper it is shown that EnrichedCasl is definitionally complete, a property similar to, but not identical with, iso-reflection; the difference is that it requires reflection of isomorphisms, but for the model functor treated as a functor into **CAT**, not **Class**. In general, this difference makes checking extended static correctness to be incomplete; however, because in [SMT+05] amalgamations are legal only along a restricted class of (surjective) sinks, the method for checking extended static correctness presented there is, in fact, complete.

Concerning extended static correctness, it is important to see that, even though it may be checked via Theorem 41 (if the assumptions of that theorem hold), it need not be decidable. This is because fully static correctness is defined using a calculus, and in the rules of that calculus objects such as colimits or universal morphisms in the signature category are computed. It turns out that, in certain specification languages, computing colimits or universal morphisms may be undecidable. In fact, it has been shown in [KHT+01] that checking extended static correctness of architectural specifications over the specification language SubCasl is undecidable. The reason is that colimits, universal morphisms, etc., in EnrichedCasl cannot be computed effectively. In the same paper, algorithms for checking extended static correctness in SubCasl in practical cases are developed.

Apart from the Casl Reference Manual [CoF04b], previous work on verification of architectural specifications includes [Hof01], where non-generative verification for first-order logic is tackled, and [Hof03], on which Section 16 of this thesis is closely based. Verification in situations close to fully static correctness is also discussed in [Tar03] and [MHST03].

The form of representation used in Sections 15 and 16 is tailored specifically for the needs of this thesis. In [CoF04b], a virtually identical notion is called a (simple) theoroidal comorphism (after [GR02]). This is only one form of a morphism between specification languages (institutions) and many more have been considered in the literature, see [Tar96, Tar00, Mos02, GR02, Mos03].

A quick look at verification techniques for various notions of correctness, i.e., for fully static or extended static correctness, for architectural specifications without any static form of correctness and, finally, for a non-generative semantics, only strengthens the conclusion found in Section 12. That is, it makes clear that verification is most natural and easiest if a generative semantics is used, the specification language has amalgamation and fully static correctness is enforced. It also shows that if, due to lack of amalgamation, a representation in a better specification language is needed, then this language should have consistent signatures and iso-reflection.

As for potential future work on verification of architectural specifications, there is an obvious need for implementing at least some of the presented verification methods. The implementation of checking extended static correctness is already under way in the HETS Tool Set [BM04].

Further theoretical work needs to be done on the behavioral [ST87] interpretation of architectural specifications and, especially, on verification with respect to such an interpretation. The crucial difference between the "regular" semantics and the behavioral interpretation is that not only are the specifications used in the latter closed under behavioral equivalence, but parametric units are also all required to have some stability property. Classically [Sch90], stability means that a parametric unit takes behaviorally equivalent arguments to behaviorally equivalent results. For architectural specifications, a stronger notion of stability is needed, see, e.g., the "local stability" of [BST02b]. We conjecture that most of the work presented here can be adapted to the behavioral semantics rather easily.

Summary

The main aim of this thesis is twofold: to analyze the notion of architectural specifications, and to provide sound and complete methods of their verification. Here, verification means checking that the development process described by an architectural specification is, indeed, correct. Both these aims are considered in the context of the CASL specification language.

As a result of our analysis of architectural specifications, we advocate replacing the existing model semantics of architectural specifications by a *partial model semantics*. This change is needed both to ensure that flattening the development process does not influence its correctness (Theorem 10 vs. Proposition 8), and to facilitate the use of a development process on the level of models instead of one on the level of programs (see, e.g., Proposition 18). Consequently, all verification problems in the thesis are considered for a partial model semantics of architectural specifications.

Results considering *fully static correctness* are important for two reasons.

On the one hand, we shown that fully static correctness is precisely the notion needed to make sure that a correct development process on the level of models leads to a correct development process on the level of programs (Theorems 20 and 28). Of course, for this interchangeability of program and model levels to work, certain other properties also have to be enforced, e.g., the partial model semantics has to be used, programming languages must have amalgamation, etc. Also, fully static correctness is only defined for a *generative semantics* of architectural specifications, described in Section 8. In Proposition 30 it is shown that a notion of fully static correctness for a non-generative semantics of architectural specifications cannot be constructed. For this reason, verification for a non-generative semantics is only informally considered in Section 21.

On the other hand, fully static correctness of an architectural specification makes verification easier. Our basic result on verification says that, for specification languages with amalgamation and translations, the proof calculus $\vdash^f$ defined in Section 14 is sound and complete for proving correctness of fully statically correct architectural specifications (Theorem 35); of course, correctness under a generative, partial model semantics is meant here. We

also show that the number of proof obligations generated by the calculus $\vdash^f$ remains reasonable (Proposition 36).

Because the specification language CASL does not have amalgamation, verification methods which do not assume amalgamation have to be developed. Two possibilities are considered here: either one can replace fully static correctness by the more general extended static correctness, or one can give up this kind of static assumption altogether. With both approaches one considers a *representation* of the used specification language, e.g., CASL, in a specification language with better properties, in particular with amalgamation.

In the first approach, a notion of *extended static correctness* is introduced, which properly generalizes fully static correctness (Proposition 37). We then assume that we have a representation of our specification language $\mathcal{L}$ in a language $\mathcal{L}_G$ with good properties, i.e., with a cocomplete signature category, with amalgamation, translations, iso-reflection and consistent signatures. The property of iso-reflection is needed to guarantee the completeness of our results; in Chapter II, it has been shown that iso-reflection gives fully static correctness a particularily intuitive semantical interpretation (Theorem 29). We also assume that the representation of $\mathcal{L}$ is total, that is, does not use the sentences (specifications) of $\mathcal{L}_G$. It is then shown that the problem of checking extended static correctness in $\mathcal{L}$ may be transformed into a problem of checking fully static correctness in $\mathcal{L}_G$ (Theorem 41). Then a proof calculus $\vdash^e$ is defined, which is a sound and complete calculus for proving correctness of extended statically correct architectural specifications (Theorem 42); this is accomplished by translating correctness in $\mathcal{L}$ to correctness in $\mathcal{L}_G$ and using the calculus $\vdash^f$.

The above approach is then applied to the specification language CASL. A "better" specification language ENRICHEDCASL together with a total representation of the model part of CASL in ENRICHEDCASL is defined and it is proved that ENRICHEDCASL has a cocomplete signature category, amalgamation and translations (Theorem 56). Proving that ENRICHEDCASL has consistent signatures and, particularily, iso-reflection is a much more complex task: this fact is stated in Theorem 57, and Sections 19 and 20 are dedicated to proving it. All this gives us the representation needed for reducing the extended static correctness of CASL architectural specifications to fully static correctness of ENRICHEDCASL architectural specifications; the latter correctness is defined by means of a calculus, though it should be kept in mind that some of the premises in the rules of this calculus involve computing colimits of certain signature diagrams. This may cause problems, since in ENRICHEDCASL checking these premises is, in general, undecidable; however, algorithms for checking practically relevant cases have been developed in [KHT+01]. The constructed representation also allows one to apply the sound and complete calculus $\vdash^e$ to proving correctness of extended statically correct CASL architectural specifications; the proof obligations will be generated in ENRICHEDCASL and their exact form will depend on the syntax chosen for ENRICHEDCASL specifications (see remarks at the beginning of Section 18).

In the second approach, a non-total representation of a specification language $\mathcal{L}$ in a language $\mathcal{L}_G$ with amalgamation, translations and a number of additional special features is used to construct a proof calculus $\vdash^g$. It is then shown that the calculus $\vdash^g$ is a sound and complete calculus for proving the correctness of architectural specifications containing amalgamations of a slightly restricted form (Theorem 47). We then construct an appropriate non-total representation of CASL in MULTICASL and show that this representation has all the properties needed to apply Theorem 47. This gives us a sound and complete method for checking the correctness of CASL architectural specifications (Corollary 54).

In a nutshell, the verification methods developed in this thesis and cases to which they should be applied may be described as follows. For specification languages with amalgamation, one can enforce fully static correctness and then use the sound and complete proof calculus $\vdash^f$. For a specification language without amalgamation, one has two options, which we describe on the example of the language CASL. In the first approach, one totally represents the language CASL in a language with a cocomplete signature category, amalgamation, translations, iso-reflection and consistent signatures, such as ENRICHEDCASL. Then one enforces extended static correctness of the architectural specifications and applies the sound and complete proof calculus $\vdash^e$. In the second approach, one represents the language CASL in a language with amalgamation, translations and certain other special features; such a language is, e.g., MULTICASL, that is, CASL without subsorting. Then the sound and complete proof calculus $\vdash^g$ may be applied.

An example of applying the above methods to concrete CASL architectural specifications and refinement statements follows.

Appendices

1 Proofs of Lemmas 6, 7, 9, 12 and Propositions 14 and 16

Proof of Lemma 6

Lemma 6. *Assume that $[UE']_{C_s}$ is defined and that Z and unit names from C_s are not λ-bound in UE. If $[UE]_{C_s[[UE']_{C_s}/Z]}$ is defined, then it is equal $[UE[UE'/Z]]_{C_s}$.*

Assume that the above expressions are defined and that, additionally, ϱ fits C_s and $[\![UE']\!]_\varrho$ is defined. If $[\![UE]\!]_{\varrho[[\![UE']\!]_\varrho/Z]}$ is defined, then it is equal $[\![UE[UE'/Z]]\!]_\varrho$.

Proof. We first prove the lemma for the case where UE' is a term, not a λ-expression. The proof is by induction over the structure of UE.

The **first** case is "$\lambda Y : SP \bullet T_\lambda$". Then

$$\begin{array}{ll} [\lambda Y : SP \bullet T_\lambda]_{C_s[[UE']_{C_s}/Z]} & \\ = sig[SP] \to [T_\lambda]_{C_s[[UE']_{C_s}/Z][sig[SP]/Y]} & \\ = sig[SP] \to [T_\lambda]_{C_s[sig[SP]/Y][[UE']_{C_s}/Z]} & \text{because } Y \neq Z \\ = sig[SP] \to [T_\lambda[UE'/Z]]_{C_s[sig[SP]/Y]} & \text{by the induction hypothesis} \\ = [\lambda Y : SP \bullet T_\lambda[UE'/Z]]_{C_s} & \\ = [(\lambda Y : SP \bullet T_\lambda)[UE'/Z]]_{C_s} & \text{because } Y \notin \text{dom}(C_s) \text{ and } Y \neq Z \end{array}$$

Also,

$$\begin{array}{ll} [\![\lambda Y : SP \bullet T_\lambda]\!]_{\varrho[[\![UE']\!]_\varrho/Z]} & \\ = \lambda M \in [\![SP]\!] \cdot [\![T_\lambda]\!]_{\varrho[[\![UE']\!]_\varrho/Z][M/Y]} & \\ = \lambda M \in [\![SP]\!] \cdot [\![T_\lambda]\!]_{\varrho[M/Y][[\![UE']\!]_\varrho/Z]} & \text{because } Y \neq Z \\ = \lambda M \in [\![SP]\!] \cdot [\![T_\lambda[UE'/Z]]\!]_{\varrho[M/Y]} & \text{by the induction hypothesis} \\ = [\![\lambda Y : SP \bullet T_\lambda[UE'/Z]]\!]_\varrho & \\ = [\![(\lambda Y : SP \bullet T_\lambda)[UE'/Z]]\!]_\varrho & \text{because } Y \notin \text{dom}(C_s) \text{ and } Y \neq Z \end{array}$$

The **second** case is "Y", where $Y \neq Z$. Then

$$\begin{aligned}
&[Y]_{C_s[[UE']_{C_s}/Z]} && \\
&= C_s(Y) && \text{because } Y \neq Z \\
&= [Y]_{C_s} && \\
&= [Y[UE'/Z]]_{C_s} && \text{because } Y \neq Z
\end{aligned}$$

and

$$\begin{aligned}
&[\![Y]\!]_{\varrho[[\![UE']\!]_\varrho/Z]} && \\
&= \varrho(Y) && \text{because } Y \neq Z \\
&= [\![Y]\!]_\varrho && \\
&= [\![Y[UE'/Z]]\!]_\varrho && \text{because } Y \neq Z
\end{aligned}$$

as required.

The **third** case is "Z". Then

$$\begin{aligned}
&[Z]_{C_s[[UE']_{C_s}/Z]} \\
&= [UE']_{C_s} \\
&= [Z[UE'/Z]]_{C_s}
\end{aligned}$$

and

$$\begin{aligned}
&[\![Z]\!]_{\varrho[[\![UE']\!]_\varrho/Z]} \\
&= [\![UE']\!]_\varrho \\
&= [\![Z[UE'/Z]]\!]_\varrho
\end{aligned}$$

The **fourth** case is "$Y(T_0)$", where $Y \neq Z$.

Since $[Y(T_0)]_{C_s[[UE']_{C_s}/Z]} = \pi_2(C_s(Y))$ is defined, $[T_0]_{C_s[[UE']_{C_s}/Z]}$ is also defined and equal $\pi_1(C_s(Y))$. Then, by the induction hypothesis, they are both equal $[T_0[UE'/Z]]_{C_s}$. So, $[Y(T_0[UE'/Z])]_{C_s} = [Y(T_0)[UE'/Z]]_{C_s}$ is defined and equal $\pi_2(C_s(Y))$.

If $[\![Y(T_0)]\!]_{\varrho[[\![UE']\!]_\varrho/Z]} = M$ is defined, then $[\![T_0]\!]_{\varrho[[\![UE']\!]_\varrho/Z]}$ is defined and an element of the domain of $\varrho(Y)$. By the induction hypothesis, it is then equal $[\![T_0[UE'/Z]]\!]_\varrho$ and so

$$\begin{aligned}
M &= \varrho(Y)([\![T_0]\!]_{\varrho[[\![UE']\!]_\varrho/Z]}) \\
&= \varrho(Y)([\![T_0[UE'/Z]]\!]_\varrho \\
&= [\![Y(T_0[UE'/Z])]\!]_\varrho \\
&= [\![Y(T_0)[UE'/Z]]\!]_\varrho
\end{aligned}$$

The **fifth** and **sixth** cases are "$\bigcup_{i \in I} \sigma_i(T_i)$" and "$T_0|_{\sigma:\Delta\to\Delta_0}$". We omit these cases, since their proofs are standard.

The **seventh** case is "$Z(T_0)$". But, since we have assumed that UE' is non-parametric, the value $[Z(T_0)]_{C_s[[UE']_{C_s}/Z]}$ is undefined and so the lemma holds for free.

This ends the proof of the lemma for non-parametric UE'. For parametric UE' the proof is again by induction over the structure of UE. Cases 1, 2 and 4-6, considered above, do not depend in any way on UE' being non-parametric, and may thus be repeated for parametric UE' as well. In case 3, the value $[Z]_{C_s[[UE']_{C_s}/Z]}$ is always undefined and so the lemma holds for free.

The only case left is case 7, where UE is "$Z(T_0)$".

If $[Z(T_0)]_{C_s[[UE']_{C_s}/Z]} = \pi_2([UE']_{C_s})$ is defined, then UE' must have the form

$$\lambda X : SP \bullet T_\lambda$$

with

$$[UE']_{C_s} = sig[SP] \to [T_\lambda]_{C_s[sig[SP]/X]}$$

Also,

$$[T_0]_{C_s[[UE']_{C_s}/Z]} = sig[SP]$$

must be defined as well. Then, by the induction hypothesis, we get $[UE'']_{C_s} = sig[SP]$, for $UE'' = T_0[UE'/Z]$. Thus,

$$\begin{array}{ll}
[Z(T_0)[UE'/Z]]_{C_s} & \\
= [T_\lambda[T_0[UE'/Z]/X]]_{C_s} & \\
= [T_\lambda[UE''/X]]_{C_s} & \\
= [T_\lambda]_{C_s[[UE'']_{C_s}/X]} & \text{by the non-parametric version of the lemma} \\
= \pi_2([UE']_{C_s}) &
\end{array}$$

where the lemma may be applied, since $[UE'']_{C_s}$ and $[T_\lambda]_{C_s[[UE'']_{C_s}/X]}$ are defined.

Observe that $[\![UE']\!]_\varrho = \lambda M \in [\![SP]\!] \cdot [\![T_\lambda]\!]_{\varrho[M/X]}$. Now, if $[\![Z(T_0)]\!]_{\varrho[[\![UE']\!]_\varrho/Z]}$ is defined, then it is equal $[\![UE']\!]_\varrho([\![T_0]\!]_{\varrho[[\![UE']\!]_\varrho/Z]}) = [\![T_\lambda]\!]_{\varrho[[\![T_0]\!]_{\varrho[[\![UE']\!]_\varrho/Z]}/X]}$. This is equal $[\![T_\lambda]\!]_{\varrho[[\![T_0[UE'/Z]]\!]_\varrho/X]}$, because the induction hypothesis gives us $[\![T_0]\!]_{\varrho[[\![UE']\!]_\varrho/Z]} = [\![T_0[UE'/Z]]\!]_\varrho$, since the former is defined. But we may use the, already proved, non-parametric version of the lemma and so

$$\begin{array}{ll}
[\![T_\lambda]\!]_{\varrho[UE''/X]} & \\
= [\![T_\lambda[UE''/X]]\!]_\varrho & \text{by the non-parametric version of the lemma} \\
= [\![T_\lambda[T_0[UE'/Z]/X]]\!]_\varrho & \\
= [\![Z(T_0)[UE'/Z]]\!]_\varrho &
\end{array}$$

as required. $\square$

Proofs of Lemmas 7, 9 and 12

Lemma 7. *Assume that* $ASP = \textbf{units}\ UDS\ X : USP\ Y_1 : USP_1 \ldots Y_n : USP_n\ \textbf{result}\ UE$ *is correct,* $ASP' = \textbf{units}\ UDS'\ \textbf{result}\ UE'$ *is correct and that* $ASP' :: ASP.X$ *holds.*

Let $ASP'' = \textbf{units}\ UDS\ UDS'\ Y_1 : USP_1 \ldots Y_n : USP_n\ \textbf{result}\, UE[UE'/X]$*. Then* ASP'' *is correct and, moreover:*

1. $\pi_2([ASP'']) = \pi_2([ASP])$,
2. *for any* $\varrho' \in \mathrm{dom}([\![ASP'']\!])$*, there exists an environment* ϱ *such that* $[\![ASP'']\!](\varrho') = [\![ASP]\!](\varrho)$.

Proof. In the proof, we tacitly use the following trivial lemma regarding extension of static contexts and environments (its proof is omitted).

Lemma 76. *If* $[UE]_{C_s}$ *and* C'_s *extends* C_s *and does not contain a variable* λ*-bound in* UE*, then* $[UE]_{C'_s} = [UE]_{C_s}$.

If, additionally, ϱ' *fits* C'_s *and* $\varrho = \varrho'|_{\mathrm{dom}(C_s)}$*, then, if* $[\![UE]\!]_\varrho$ *is defined, then it is equal* $[\![UE]\!]_{\varrho'}$. $\square$

Let $C_s = \pi_1([ASP])$, $C'_s = [UDS']$ and $C''_s = [UDS\ UDS'\ Y_1 : USP_1 \ldots \ldots Y_n : USP_n]$. Then $\pi_2([ASP'']) = [UE[UE'/X]]_{C''_s}$. This is, by Lemma 6, equal $[UE]_{C''_s[[UE']_{C''_s}/X]}$, because: first, $[UE']_{C''_s} = [UE']_{C'_s} = \pi_2([ASP']) = sig[USP]$ is defined, since C''_s is an extension of C'_s and it cannot contain the λ-bound variable of UE' (because of distinctness of variables in the dag considered), and since $ASP' :: ASP.X$ is statically correct; second, $[UE]_{C''_s[[UE']_{C''_s}/X]}$ itself is defined, because it equals $[UE]_{C''_s[sig[USP]/X]} = [UE]_{C_s} = \pi_2([ASP])$, since, again, $C''_s[sig[USP]/X]$ extends C_s and it cannot interfere with λ-bound variables. This proves point (1).

Let $C = \mathrm{dom}([\![ASP]\!])$, $C' = \mathrm{dom}([\![ASP']\!])$ and $C'' = [\![UDS\ UDS'\ Y_1 : USP_1 \ldots Y_n : USP_n]\!]$. Take any $\varrho' \in C''$ and consider $[\![UE[UE'/X]]\!]_{\varrho'}$. The unit $[\![UE']\!]_{\varrho'}$ is defined, since $\varrho'|_{\mathrm{dom}(C'_s)} \in C'$ and $[\![ASP']\!]$ is correct. The unit $[\![UE]\!]_{\varrho'[[\![UE']\!]_{\varrho'}/X]}$ is defined as well, since $\varrho = \varrho'[[\![UE']\!]_{\varrho'}/X]|_{\mathrm{dom}(C_s)}$ is in C and $[\![ASP]\!]$ is correct. The reason why ϱ is in C is that $ASP' :: ASP.X$ and so $[\![UE']\!]_{\varrho'} \in \mathcal{M}[ASP'] \subseteq \mathcal{M}[ASP.X] = [\![USP]\!]$. Thus, the considered unit is equal, by Lemma 6, $[\![UE]\!]_{\varrho'[[\![UE']\!]_{\varrho'}/X]} = [\![ASP]\!](\varrho)$, as required. $\square$

Lemma 9. *Assume* $ASP = \textbf{units}\ UDS\ X : SP\ \textbf{given}_\Sigma\ T\ Y_1 : USP_1 \ldots Y_n : USP_n\ \textbf{result}\ UE$ *is correct, that* $ASP' = \textbf{units}\ UDS'\ \textbf{result}\ \lambda Y : SP_I \bullet T_\lambda$ *is correct and that* $ASP' :: ASP.X$ *holds.*

The architectural specification $ASP'' =$ **units** $UDS\ UDS'\ Y_1 : USP_1 \ldots$ $\ldots Y_n : USP_n$ **result** $UE[T_\lambda[T/Y]/X]$ *is then correct and, moreover:*

1. $\pi_2([ASP'']) = \pi_2([ASP])$,
2. *for any* $\varrho' \in \mathrm{dom}([\![ASP'']\!])$, *there exists an environment* ϱ *such that* $[\![ASP'']\!](\varrho') = [\![ASP]\!](\varrho)$.

Proof. Let $C_s = \pi_1([ASP])$, $C'_s = [UDS']$ and $C''_s = [UDS\ UDS'\ Y_1 : USP_1 \ldots Y_n : USP_n]$.

Let $T' = T_\lambda[T/Y]$. Then $[T']_{C''_s} = [T_\lambda[T/Y]]_{C''_s}$ equals, by Lemma 6, $[T_\lambda]_{C''_s[[T]_{C''_s}/Y]}$, because:

- $[T]_{C''_s}$ is defined and, since C''_s extends $[UDS]$, equals $[T]_{[UDS]} = sig[SP_I]$, because $ASP :: ASP.X$ is statically correct,
- $[T_\lambda]_{C''_s[[T]_{C''_s}/Y]}$ itself is defined, since it equals $[T_\lambda]_{C''_s[sig[SP_I]/Y]}$, which in turn equals the target signature of $[\lambda Y : SP_I \bullet T_\lambda]_{C''_s} = \pi_2([ASP'])$, again because C''_s extends C'_s.

Thus, $[T']_{C''_s} = sig[SP]$, since, the refinement statement $ASP' :: ASP'.X$ being statically correct, $sig[SP]$ is the target signature of $\pi_2([ASP'])$.

Then $\pi_2([ASP'']) = [UE[T'/X]]_{C''_s}$. This is, by Lemma 6, equal to the signature $[UE]_{C''_s[[T']_{C''_s}/X]}$, because:

- $[T']_{C''_s} = sig[SP]$ is defined, and
- $[UE]_{C''_s[[T']_{C''_s}/X]}$ itself is defined, because it equals $[UE]_{C''_s[sig[SP]/X]} = [UE]_{C_s} = \pi_2([ASP])$, since, again, $C''_s[sig[SP]/X]$ extends C_s and it cannot interfere with λ-bound variables.

Let $C = \mathrm{dom}([\![ASP]\!])$, $C' = \mathrm{dom}([\![ASP']\!])$ and $C'' = [\![UDS\ UDS'\ Y_1 : USP_1 \ldots Y_n : USP_n]\!]$. Take any $\varrho' \in C''$. First, let $F = [\![ASP']\!](\varrho')$. Then F is defined, since $\varrho'|_{\mathrm{dom}([UDS'])} \in C'$ and ASP' is correct. Now, $F = [\![\lambda Y : SP_I \bullet T_\lambda]\!]_{\varrho'} = \lambda M \in [\![SP_I]\!] \cdot [\![T_\lambda]\!]_{\varrho'[M/Y]}$. Because $ASP' :: ASP.X$ holds, $F \in \mathcal{M}[ASP.X]$. Hence, $[\![T]\!]_{\varrho'} \in \mathrm{dom}(F) = [\![SP_I]\!]$, since $[\![T]\!]_{\varrho'} = [\![T]\!]_{\varrho'|_{\mathrm{dom}([UDS])}}$. Also, $F([\![T]\!]_{\varrho'}) \in [\![SP]\!]$ and F is persistent.

Now consider $[\![T']\!]_{\varrho'} = [\![T_\lambda[T/Y]]\!]_{\varrho'}$. By Lemma 6, it equals $[\![T_\lambda]\!]_{\varrho'[[\![T]\!]_{\varrho'}/Y]}$, since $[\![T]\!]_{\varrho'}$ is defined and $[\![T_\lambda]\!]_{\varrho'[[\![T]\!]_{\varrho'}/Y]}$ itself is also defined, since it equals $F([\![T]\!]_{\varrho'})$.

Finally, consider $[\![ASP'']\!](\varrho') = [\![UE[T'/X]]\!]_{\varrho'}$. Let $\varrho = \varrho'[[\![T']\!]_{\varrho'}/X]|_{\mathrm{dom}(C_s)}$. Then $[\![UE[T'/X]]\!]_{\varrho'}$ is, by Lemma 6, further equal to $[\![UE]\!]_{\varrho'[[\![T']\!]_{\varrho'}/X]} = [\![UE]\!]_{\varrho}$, because $[\![T']\!]_{\varrho'}$ is defined and equals $F([\![T]\!]_{\varrho'})$, and $[\![UE]\!]_{\varrho}$ itself is defined, since $\varrho \in C$. Indeed, $\varrho \in C$, because $\varrho|_{\mathrm{dom}([UDS])} \in [\![UDS]\!]$, $\varrho(X) = [\![T']\!]_{\varrho'} = F([\![T]\!]_{\varrho'})$ and we have $F([\![T]\!]_{\varrho'}) \in [\![SP]\!]$ and $F([\![T]\!]_{\varrho'})|_{\Sigma} = [\![T]\!]_{\varrho'} = [\![T]\!]_{\varrho}$, since F is persistent, and, finally, $\varrho(Y_i) \in [\![USP_i]\!]$ for all $1 \leq i \leq n$.

Thus, to sum up, we have $[\![ASP'']\!](\varrho') = [\![UE]\!]_{\varrho} = [\![ASP]\!](\varrho)$ and $\varrho \in C$. □

Lemma 12. *Assume* $ASP =$ **units** $UDS_1\ X : USP\ UDS_2$ **result** UE *is partially correct, that* $ASP' =$ **units** UDS' **result** UE' *is partially correct and that* $ASP' ::_{\perp} ASP.X$ *holds.*

Then $ASP'' =$ **units** $UDS_1\ UDS'\ UDS_2[UE'/X]$ **result** $UE[UE'/X]$ *is partially correct and, moreover:*

1. $\pi_2([ASP'']) = \pi_2([ASP])$,
2. *for any* $\varrho' \in \mathrm{dom}([\![ASP'']\!])$*, there exists an environment* ϱ *such that* $[\![ASP'']\!](\varrho') = [\![ASP]\!](\varrho)$.

Proof. We omit the proof that ASP'' is statically correct and that point (1) holds.

In the proof of point (2), the following lemma, extending Lemma 76, is used (its proof is omitted):

Lemma 77. *If* $[UE]_{C_s}$ *is defined,* C'_s *extends* C_s *and does not contain any variable* λ*-bound in* UE*,* ϱ' *partially fits* C'_s *and, for* $\varrho = \varrho'|_{\mathrm{dom}(C_s)}$*,* $[\![UE]\!]^{\perp}_{\varrho}$ *is defined, then* $[\![UE]\!]^{\perp}_{\varrho'} = [\![UE]\!]^{\perp}_{\varrho}$. □

Let UDS_2^i be the first i declarations of UDS_2, for $0 \leq i \leq n$, where n is the number of declarations in UDS_2. Let $C_s^i = [UDS_1\ X : USP\ UDS_2^i]$, $C_i = [\![UDS_1\ X : USP\ UDS_2^i]\!]^{\perp}$ and $C''_i = [\![UDS_1\ UDS'\ UDS_2^i[UE'/X]]\!]^{\perp}$ for $0 \leq i \leq n$; note that all C_i are defined, since ASP is partially correct, and that C''_0 is defined, since ASP and ASP' are partially correct. Note also that, for any $\varrho' \in C''_i$, the unit $[\![UE']\!]^{\perp}_{\varrho'}$ is defined, again since ASP' is partially correct.

By induction over i we prove that C''_i is defined and, for any $\varrho' \in C''_i$, $\varrho \in C_i$, where $\varrho = \varrho'[[\![UE']\!]^{\perp}_{\varrho'}/X]|_{\mathrm{dom}(C_s^i)}$.

For $i = 0$, as already stated, C''_0 is defined. Because of $ASP' ::_{\perp} ASP.X$, we have $\varrho(X) = [\![UE']\!]^{\perp}_{\varrho'} \in \mathcal{M}^{\perp}[ASP'] \subseteq \mathcal{M}^{\perp}[ASP.X] = [\![USP]\!]_{\perp}$ and so $\varrho \in C_0$.

Assume the claim holds for some $i < n$ and consider it for $i + 1$.

First, assume that UDS_2^{i+1} is $UDS_2^i\ Y : USP'$. By the induction hypothesis, C''_i is defined and so C''_{i+1} is defined as well. Also, if $\varrho' \in C''_{i+1}$, then let $\varrho'_i = \varrho'|_{\mathrm{dom}(C_s^i)}$. Clearly, $\varrho'_i \in C''_i$ and $\varrho'(Y) \in [\![USP']\!]_{\perp}$. By the induction hypothesis, $\varrho_i \in C_i$ for $\varrho_i = \varrho'_i[[\![UE']\!]^{\perp}_{\varrho'_i}/X]|_{\mathrm{dom}(C^i_s)}$. Thus, $\varrho = \varrho_i[\varrho'(Y)/Y] = \varrho'[[\![UE']\!]^{\perp}_{\varrho'}/X]|_{\mathrm{dom}(C^{i+1}_s)}$ is an element of C_{i+1}.

Now assume UDS^{i+1}_2 is $UDS^i_2\ Y : SP$ **given** T. By the induction hypothesis, C''_i is defined and $\varrho_i \in C_i$ for $\varrho_i = \varrho'_i[[\![UE']\!]^{\perp}_{\varrho'_i}/X]|_{\mathrm{dom}(C^i_s)}$, where $\varrho'_i = \varrho'|_{\mathrm{dom}(C^i_s)}$. The model $[\![T]\!]^{\perp}_{\varrho_i}$ is defined, since ASP is partially correct. But this model is equal $[\![T]\!]^{\perp}_{\varrho'_i[[\![UE']\!]^{\perp}_{\varrho'_i}/X]}$. Hence, by Lemma 11, $[\![T[UE'/X]]\!]^{\perp}_{\varrho'_i}$ is defined as well and equal $[\![T]\!]^{\perp}_{\varrho_i}$. For $\varrho = \varrho_i[\varrho'(Y)/Y] = \varrho'[[\![UE']\!]^{\perp}_{\varrho'}/X]|_{\mathrm{dom}(C^{i+1}_s)}$ we then have $\varrho \in C_{i+1}$, because $\varrho_i \in C_i$, $\varrho(Y) = \varrho'(Y)$, $\varrho'(Y) \in [\![SP]\!]_{\perp}$ and, if $\varrho'(Y) \neq \perp$, then $\varrho'(Y)|_{\Sigma} = [\![T[UE'/X]]\!]^{\perp}_{\varrho'_i}$, which, by Lemma 11, is equal to $[\![T]\!]^{\perp}_{\varrho'_i[[\![UE']\!]^{\perp}_{\varrho'_i}/X]} = [\![T]\!]^{\perp}_{\varrho_i}$.

This ends the proof of the claim. Thus, $[\![ASP'']\!]^{\perp}$ is defined and equal $C'' = C''_n$. Let $C_s = C^n_s$ and $C = C_n$ and take any $\varrho' \in C''$. Then $\varrho \in C$ for $\varrho = \varrho'[[\![UE']\!]^{\perp}_{\varrho'}/X]|_{\mathrm{dom}(C_s)}$. Because $\varrho \in C$ and ASP is partially correct, the unit $[\![UE]\!]^{\perp}_{\varrho}$ is defined. But $[\![UE]\!]^{\perp}_{\varrho} = [\![UE]\!]^{\perp}_{\varrho'[[\![UE']\!]^{\perp}_{\varrho'}/X]}$. Hence, by Lemma 11, $[\![UE[UE'/X]]\!]^{\perp}_{\varrho'}$ is defined as well and it is equal to $[\![UE]\!]^{\perp}_{\varrho} = [\![ASP]\!]^{\perp}(\varrho)$, as required. $\square$

Proof of Proposition 14

Proposition 14. *If ASP is partially correct, then it is correct and either it contains an inconsistent declaration, or*

$$\mathrm{dom}([\![ASP]\!]) = C$$

where $C = \{\varrho \text{ fits } C_s \mid \perp(\varrho) \in \mathrm{dom}([\![ASP]\!]^{\perp})\}$ and $C_s = \pi_1([ASP])$, and, for all $\varrho \in C$, if ASP is non-parametric, then

$$[\![ASP]\!](\varrho) = [\![ASP]\!]^{\perp}(\perp(\varrho))$$

and if it is parametric, then it uses its λ-argument and

$$[\![ASP]\!](\varrho)[\perp/\perp] = [\![ASP]\!]^{\perp}(\perp(\varrho))$$

Proof.

Lemma 78. *If $[T]_{C_s}$ is defined and ϱ fits C_s, then:*

$$[\![T]\!]_{\varrho} \textit{ is defined} \iff [\![T]\!]^{\perp}_{\perp(\varrho)} \textit{ is defined}$$

and, if both are defined, then they are equal.

Proof. The proof is by induction over the structure of T.

For "X" both sides are always defined and $[\![X]\!]_\varrho = \varrho(X) = \bot(\varrho)(X) = [\![X]\!]^\bot_{\bot(\varrho)}$.

For "$X(T_0)$" the left side is defined iff $[\![T_0]\!]_\varrho$ is defined and in the domain of $\varrho(X)$, and the right side iff $[\![T_0]\!]^\bot_{\bot(\varrho)}$ is defined and in the domain of $\bot(\varrho)(X)$. If the left side is defined, then, by the induction hypothesis, $[\![T_0]\!]^\bot_{\bot(\varrho)}$ is defined as well and equals $[\![T_0]\!]_\varrho \in \mathrm{dom}(\varrho(X)) \subseteq \mathrm{dom}(\varrho(X)) \cup \{\bot\} = \mathrm{dom}(\bot(\varrho)(X))$, and so the right side is then defined as well. Also, we then have $[\![X(T_0)]\!]_\varrho = \varrho(X)([\![T_0]\!]_\varrho) = \bot(\varrho)(X)([\![T_0]\!]^\bot_{\bot(\varrho)}) = [\![X(T_0)]\!]^\bot_{\bot(\varrho)}$, which proves the stated equation.

If the right side is defined, then, by the induction hypothesis, $[\![T_0]\!]_\varrho$ is defined as well and equals $[\![T_0]\!]^\bot_{\bot(\varrho)}$. Therefore the latter may not be $\bot$, and so if it is an element of $\mathrm{dom}(\bot(\varrho)(X))$, then it must also be an element of $\mathrm{dom}(\varrho(X))$. This means that the left side is defined.

For "$\bigcup_{i\in I} T_i$", where $\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i\in I}$, the left side is defined iff all models $[\![T_i]\!]_\varrho$ are defined and σ-amalgamable, and the right side is defined iff all $[\![T_i]\!]^\bot_{\bot(\varrho)}$ are defined and, unless at least one of them is equal $\bot$, σ-amalgamable. If the left side is defined, then, by the induction hypothesis, $[\![T_i]\!]^\bot_{\bot(\varrho)}$ is defined and equal $[\![T_i]\!]_\varrho$, for all $i \in I$. Then the models $[\![T_i]\!]^\bot_{\bot(\varrho)} = [\![T_i]\!]_\varrho$ are σ-amalgamable and so the right side is defined and $[\![\bigoplus_{i\in I} \sigma_i(T_i)]\!]^\bot_{\bot(\varrho)} = [\![\bigoplus_{i\in I} \sigma_i(T_i)]\!]_\varrho$, as required.

If the right side is defined, then, by the induction hypothesis, $[\![T_i]\!]_\varrho$ is defined and equal $[\![T_i]\!]^\bot_{\bot(\varrho)}$ for all $i \in I$. Thus, none of $[\![T_i]\!]^\bot_{\bot(\varrho)}$ can be equal $\bot$ and they must be σ-amalgamable, which implies that the left side is defined.

For "$T_0|_{\sigma:\Sigma\to\Delta}$" the left side is defined iff $[\![T_0]\!]_\varrho$ is defined, and the right side iff $[\![T_0]\!]^\bot_{\bot(\varrho)}$ is. By the induction hypothesis, the definedness of both sides is equivalent and if both are defined, then $[\![T_0]\!]_\varrho = [\![T_0]\!]^\bot_{\bot(\varrho)}$. We have $[\![T_0|_{\sigma:\Sigma\to\Delta}]\!]_\varrho = [\![T_0]\!]_\varrho|_\sigma = [\![T_0]\!]^\bot_{\bot(\varrho)}|_\sigma = [\![T_0]\!]^\bot_{\bot(\varrho)}|^\bot_\sigma = [\![T_0|_{\sigma:\Sigma\to\Delta}]\!]^\bot_{\bot(\varrho)}$. ☐

Lemma 79. *If $[UDS] = C_s$ and $[\![UDS]\!]^\bot$ is defined, then $[\![UDS]\!]$ is defined as well. If, additionally, $[\![UDS]\!] \neq \emptyset$, then, for any ϱ fitting C_s:*

$$\bot(\varrho) \in [\![UDS]\!]^\bot \iff \varrho \in [\![UDS]\!]$$

Proof. We proceed by induction over the structure of UDS.

For $UDS = \epsilon$, the proposition holds, since $[\![\epsilon]\!] = \{\emptyset\} = [\![\epsilon]\!]^\bot$ and $\bot(\emptyset) = \emptyset$.

If $UDS = UDS_0\ X : USP$ and $[\![UDS]\!]^\bot$ is defined, then $[\![UDS_0]\!]^\bot$ is defined as well and so, by the induction hypothesis, $[\![UDS_0]\!]$ is defined. Hence, $[\![UDS]\!]$ is defined. Assume $[\![UDS]\!] \neq \emptyset$. Then $[\![UDS_0]\!]$ must be non-empty as well.

Assume $\varrho \in [\![UDS]\!]$. Then $\varrho = \varrho_0[U/X]$, where $\varrho_0 \in [\![UDS_0]\!]$ and $U \in [\![USP]\!]$. By the induction hypothesis, $\bot(\varrho_0) \in [\![UDS_0]\!]^{\bot}$. Let $U_\bot = U$ if USP is non-parametric and $U_\bot = U[\bot/\bot]$ otherwise. Then $U_\bot \in [\![USP]\!]_\bot$. Thus, $\bot(\varrho_0)[U_\bot/X] \in [\![UDS]\!]^{\bot}$. Hence, $\bot(\varrho) = \bot(\varrho_0[U/X]) = \bot(\varrho_0)[U_\bot/X] \in [\![UDS]\!]^{\bot}$.

Assume $\bot(\varrho) \in [\![UDS]\!]^{\bot}$. Then $\varrho = \varrho_0[U/X]$, where $\varrho_0 = \varrho|_{\mathrm{dom}(C_s)\setminus\{X\}}$ is an appropriate restriction and $U = \varrho(X)$. Also, $\bot(\varrho) = \bot(\varrho_0)[U_\bot/X]$, where $U_\bot$ is defined as above. Hence, $\bot(\varrho_0) \in [\![UDS_0]\!]^{\bot}$ and, as above, $U_\bot \in [\![USP]\!]_\bot$. By the induction hypothesis, $\varrho_0 \in [\![UDS_0]\!]$. Also, $U \in [\![USP]\!]$. Thus $\varrho = \varrho_0[U/X] \in [\![UDS]\!]$.

If $UDS = UDS_0\ X : SP\ \mathbf{given}_\Sigma\ T$ and $[\![UDS]\!]^{\bot}$ is defined, then $[\![UDS_0]\!]^{\bot}$ is defined as well and, for any $\varrho_0' \in [\![UDS_0]\!]^{\bot}$, the model $[\![T]\!]^{\bot}_{\varrho_0'}$ must be defined. By the induction hypothesis, $[\![UDS_0]\!]$ is defined as well. If $[\![UDS_0]\!] = \emptyset$, then $[\![UDS]\!]$ is defined and empty as well. So, assume $[\![UDS_0]\!] \neq \emptyset$.

Assume $\varrho \in [\![UDS]\!]$. Then $\varrho = \varrho_0[M/X]$, where $\varrho_0 \in [\![UDS_0]\!]$, $M \in [\![SP]\!]$ and $[\![T]\!]_{\varrho_0}$ is defined and equal $M|_\Sigma$. By the induction hypothesis, $\bot(\varrho_0) \in [\![UDS_0]\!]^{\bot}$. By Lemma 78, $[\![T]\!]^{\bot}_{\bot(\varrho_0)} = [\![T]\!]_{\varrho_0}$, which implies that it is defined and equal $M|_\Sigma$. Thus, $\bot(\varrho) = \bot(\varrho_0[M/X]) = \bot(\varrho_0)[M/X] \in [\![UDS]\!]^{\bot}$.

Assume $\bot(\varrho) \in [\![UDS]\!]^{\bot}$. Then $\varrho = \varrho_0[M/X]$, where $\varrho_0 = \varrho|_{\mathrm{dom}(C_s)\setminus\{X\}}$ is the appropriate restriction and $M = \varrho(X)$; observe that $M \neq \bot$, since ϱ fits $[UDS]$. Clearly, $\bot(\varrho) = \bot(\varrho_0)[M/X]$. Thus, $\bot(\varrho_0) \in [\![UDS_0]\!]^{\bot}$, $M \in [\![SP]\!]$ and $[\![T]\!]^{\bot}_{\bot(\varrho_0)}$ is defined and equal $M|_\Sigma$. By the induction hypothesis, $\varrho_0 \in [\![UDS_0]\!]$. By Lemma 78, $[\![T]\!]_{\varrho_0} = [\![T]\!]^{\bot}_{\bot(\varrho_0)}$ is defined and equal $M|_\Sigma$. Also, we have assumed $[\![UDS]\!] \neq \emptyset$, so, for any $\eta_0 \in [\![UDS_0]\!]$, there exists a model $N \in [\![SP]\!]$ such that $[\![T]\!]_{\eta_0} = N|_\Sigma$. Therefore, $\varrho = \varrho_0[M/X] \in [\![UDS]\!]$. $\square$

Lemma 80. *Assume UE is a λ-expression, $[UE]_{C_s}$ is defined and ϱ fits C_s. If $[\![UE]\!]^{\bot}_{\bot(\varrho)}$ is defined, then UE uses its λ-argument, $[\![UE]\!]_\varrho$ is defined and*

$$[\![UE]\!]_\varrho[\bot/\bot] = [\![UE]\!]^{\bot}_{\bot(\varrho)}$$

Proof. Assume UE takes the form $\lambda X : SP \bullet T$.

Take any $M \in [\![SP]\!]$. By assumption, $[\![T]\!]^{\bot}_{\bot(\varrho)[M/X]}$ is defined. But this equals $[\![T]\!]^{\bot}_{\bot(\varrho[M/X])}$ and so, by Lemma 78, $[\![T]\!]_{\varrho[M/X]}$. Because the latter is certainly not $\bot$, we have, by assumption, $[\![T]\!]_{\varrho[M/X]}|_{sig[SP]} = M$. Hence, $[\![UE]\!]_\varrho$ is defined.

Also, $\mathrm{dom}\,[\![UE]\!]^{\bot}_{\bot(\varrho)} = [\![SP]\!]_\bot = \mathrm{dom}\,[\![T]\!]_\varrho \cup \{\bot\}$, $[\![UE]\!]^{\bot}_{\bot(\varrho)}(\bot) = \bot$ and, for any $M \in [\![SP]\!]$, $[\![UE]\!]^{\bot}_{\bot(\varrho)}(M) = [\![T]\!]^{\bot}_{\bot(\varrho)[M/X]} = [\![T]\!]^{\bot}_{\bot(\varrho[M/X])} = [\![T]\!]_{\varrho[M/X]} = [\![UE]\!]_\varrho(M)$. Hence, the equality $[\![UE]\!]_\varrho[\bot/\bot] = [\![UE]\!]^{\bot}_{\bot(\varrho)}$ holds.

Finally, since $[\![UE]\!]^{\bot}_{\bot(\varrho)}$ is defined and so, in particular, $[\![T]\!]^{\bot}_{\bot(\varrho)[\bot/X]}|_{C_s(X)} = \bot$, we have $[\![T]\!]^{\bot}_{\bot(\varrho)[\bot/X]} = \bot$. If $X \notin np(T)$, then we have $[\![T]\!]^{\bot}_{\bot(\varrho)[\bot/X]} =$

$[\![T]\!]^{\perp}_{\perp(\varrho)} = [\![T]\!]_{\varrho} \neq \perp$, where the penultimate equation holds because a unit not referenced in T cannot influence its value (cf. Lemma 77), and the ultimate one by Lemma 78. Thus, $X \in np(T)$, that is, UE uses its λ-argument. $\square$

We prove Proposition 14 using Lemmas 78–80.

Suppose that ASP = **units** UDS **result** UE is partially correct. Then $[\![UDS]\!]^{\perp}$ is defined, which, by Lemma 79, implies that $[\![UDS]\!]$ is defined as well. If $[\![UDS]\!] = \emptyset$, then $[\![ASP]\!]$ is correct and contains an inconsistent declaration. So, assume $[\![UDS]\!] \neq \emptyset$. By Lemma 79, $[\![UDS]\!] = \{\varrho \text{ fits } C_s \mid \perp(\varrho) \in [\![UDS]\!]^{\perp}\} = C$. Also, for any $\varrho \in [\![UDS]\!]$, $[\![UE]\!]^{\perp}_{\perp(\varrho)}$ is defined, since $\perp(\varrho) \in [\![UDS]\!]^{\perp}$ and ASP is partially correct. Hence, $[\![UE]\!]_{\varrho}$ is defined and, for non-parametric UE, we have $[\![UE]\!]_{\varrho} = [\![UE]\!]^{\perp}_{\perp(\varrho)}$, by Lemma 78, and, for parametric UE, $[\![UE]\!]_{\varrho}[\perp/\perp] = [\![UE]\!]^{\perp}_{\perp(\varrho)}$ and UE uses its λ-argument, by Lemma 80. Thus, ASP is correct, $\mathrm{dom}([\![ASP]\!]) = C$ and, for any $\varrho \in C$, $[\![ASP]\!](\varrho) = [\![ASP]\!]^{\perp}(\perp(\varrho))$ in the non-parametric case and $[\![ASP]\!](\varrho)[\perp/\perp] = [\![ASP]\!]^{\perp}(\perp(\varrho))$ and ASP uses its λ-argument for parametric ASP. $\square$

Proof of Proposition 16

Proposition 16. *If an architectural specification is correct, contains no inconsistent specification and, if parametric, uses its λ-argument, then it is partially correct.*

Proof. An environment ϱ partially fitting C_s is a $\perp$*-restriction* of an environment ϱ' fitting C_s, if:

- for any non-parametric unit name X, if $\varrho(X) \neq \perp$ then $\varrho(X) = \varrho'(X)$, and
- for any parametric unit name X, $\mathrm{dom}(\varrho(X)) \supseteq \mathrm{dom}(\varrho'(X))$ and, for any $M \in \mathrm{dom}(\varrho(X))$, if $\varrho(X)(M) \neq \perp$, then $M \in \mathrm{dom}(\varrho'(X))$ and $\varrho(X)(M) = \varrho'(X)(M)$.

Less formally, ϱ is a $\perp$-restriction of ϱ' if it is essentially the same, except that it can produce $\perp$ in random places, i.e., both in places where ϱ' was defined, and in those where it was not.

Lemma 81. *If $[T]_{C_s}$ is defined, ϱ partially fitting C_s is a $\perp$-restriction of ϱ' fitting C_s and $[\![T]\!]_{\varrho'}$ is defined, then $[\![T]\!]^{\perp}_{\varrho}$ is defined and equals $\perp$ or $[\![T]\!]_{\varrho'}$.*

Proof. The proof is by induction over the structure of T.

For "X", the model $[\![T]\!]_{\varrho'}$ is always defined and, if $[\![T]\!]^{\perp}_{\varrho} = \varrho(X) \neq \perp$, then $[\![T]\!]_{\varrho'} = \varrho'(X) = \varrho(X) = [\![T]\!]^{\perp}_{\varrho}$, since ϱ is a $\perp$-restriction of ϱ'.

For "$X(T_0)$", if $[\![T]\!]_{\varrho'}$ is defined, then $[\![T_0]\!]_{\varrho'}$ is defined and an element of $\mathrm{dom}(\varrho'(X))$. By the induction hypothesis, $[\![T_0]\!]^{\perp}_{\varrho}$ is then defined. If it equals $\perp$, then $[\![X(T_0)]\!]^{\perp}_{\varrho}$ equals $\perp$ as well. If it equals $[\![T_0]\!]_{\varrho'}$, then it is also an element of $\mathrm{dom}(\varrho'(X)) \subseteq \mathrm{dom}(\varrho(X))$, and so $[\![X(T_0)]\!]^{\perp}_{\varrho}$ is defined and equal to $\varrho(X)([\![T_0]\!]^{\perp}_{\varrho}) = \varrho(X)([\![T_0]\!]_{\varrho'})$, which either equals $\perp$ or $\varrho'(X)([\![T_0]\!]_{\varrho'}) = [\![X(T_0)]\!]_{\varrho'}$.

For "$\bigcup_{i \in I} \sigma_i(T_i)$", if $[\![T]\!]_{\varrho'}$ is defined, then $[\![T_i]\!]_{\varrho'}$ must be defined for all $i \in I$, and the models $[\![T_i]\!]_{\varrho'}$ must be σ-amalgamable, where $\sigma = \{\sigma_i : \Delta_i \to \Delta\}_{i \in I}$. By the induction hypothesis, for all $i \in I$, $[\![T_i]\!]^{\perp}_{\varrho}$ is then defined and if it not equal to $\perp$ then it equals $[\![T_i]\!]_{\varrho'}$. If, for some $i \in I$, $[\![T_i]\!]^{\perp}_{\varrho} = \perp$, then $[\![T]\!]^{\perp}_{\varrho} = \perp$, which ends the proof of this case. If, for all $i \in I$, $[\![T_i]\!]^{\perp}_{\varrho}$ is not $\perp$, then it must be equal to $[\![T_i]\!]_{\varrho}$, and since these models are σ-amalgamable, we also have $[\![T]\!]^{\perp}_{\varrho} = \bigoplus_{i \in I} [\![T_i]\!]^{\perp}_{\varrho} = \bigoplus_{i \in I} [\![T_i]\!]_{\varrho'} = [\![T]\!]_{\varrho'}$, as required.

For "$T_0|_{\sigma:\Sigma\to\Delta}$", if $[\![T]\!]_{\varrho'}$ is defined, then so is $[\![T_0]\!]_{\varrho'}$. By the induction hypothesis, $[\![T_0]\!]^{\perp}_{\varrho}$ is then defined and it is either equal to $\perp$ or it is equal to $[\![T_0]\!]_{\varrho'}$. In the first case, $[\![T_0|_{\sigma:\Sigma\to\Delta}]\!]^{\perp}_{\varrho}$ is equal to $\perp$ as well. In the second case, $[\![T_0|_{\sigma:\Sigma\to\Delta}]\!]^{\perp}_{\varrho} = [\![T_0]\!]^{\perp}_{\varrho}|^{\perp}_{\sigma} = [\![T_0]\!]_{\varrho'}|_{\sigma} = [\![T_0|_{\sigma:\Sigma\to\Delta}]\!]_{\varrho'}$. $\square$

Lemma 82. *If $[UDS]$ is defined and $[\![UDS]\!]$ is defined and non-empty, then $[\![UDS]\!]^{\perp}$ is defined as well and any $\varrho \in [\![UDS]\!]^{\perp}$ is a $\perp$-restriction of some $\varrho' \in [\![UDS]\!]$.*

Proof. The proof is by induction over the structure of UDS.

If $UDS = \epsilon$, then $[\![UDS]\!]^{\perp} = \{\emptyset\} = [\![UDS]\!]$ and $\emptyset$ is $\perp$-restriction of $\emptyset$.

If $UDS = UDS_0\ X : USP$ and $[\![UDS]\!]$ is defined and non-empty, then $[\![UDS_0]\!]$ is defined and non-empty and $[\![USP]\!] \neq \emptyset$. By the induction hypothesis, $[\![UDS_0]\!]^{\perp}$ is then defined. Hence, $[\![UDS]\!]^{\perp}$ is defined as well.

Assume $\varrho \in [\![UDS]\!]^{\perp}$. Then $\varrho = \varrho_0[\varrho(X)/X]$, where $\varrho_0 \in [\![UDS_0]\!]^{\perp}$, and $\varrho(X) \in [\![USP]\!]_{\perp}$. By the induction hypothesis, $\varrho_0 \in [\![UDS_0]\!]^{\perp}$ implies that there exists some $\varrho'_0 \in [\![UDS_0]\!]$ such that ϱ_0 is a $\perp$-restriction of ϱ'_0. Let F be any element of $[\![USP]\!]$; such an element must exist. Let $\varrho' = \varrho'_0[U/X]$, where:

- if USP is non-parametric, then $U = F$,
- if $USP = SP \to SP'$, then

$$U = \lambda M \in [\![SP]\!] \cdot \text{ if } \varrho(X)(M) \neq \perp \text{ then } \varrho(X)(M) \text{ else } F(M)$$

Note that in the parametric case U is well-defined, since for $M \in [\![SP]\!]$ we have $M \in [\![SP]\!]_{\perp} \subseteq \mathrm{dom}(\varrho(X))$ and $M \in [\![SP]\!] \subseteq \mathrm{dom}(F)$. Then $U \in$

$[\![USP]\!]$. In the non-parametric case, this is clear. In the parametric case, U is persistent, since $\varrho(X)$ and F were. Also, $\mathrm{dom}(U) = [\![SP]\!]$ and $U(M)$ is always an element of $[\![SP']\!]$, since $\varrho(X)(M)$ is an element of $[\![SP']\!] \cup \{\perp\}$ and $F \in [\![SP \to SP']\!]$. Hence, $\varrho' \in [\![UDS]\!]$. Also, ϱ is a $\perp$-restriction of ϱ', since ϱ_0 was a $\perp$-restriction of ϱ'_0, $\varrho(X) = U = \varrho'(X)$ in the non-parametric case, and, in the parametric case, $\mathrm{dom}(\varrho(X)) \supseteq [\![SP]\!]_\perp \supseteq [\![SP]\!] = \mathrm{dom}(U)$ and, for any $M \in \mathrm{dom}(\varrho(X))$, if $\varrho(X)(M) \neq \perp$, then $M \in [\![SP]\!] = \mathrm{dom}(U)$, since $M \neq \perp$, and $\varrho(X)(M) = U(M)$.

If $UDS = UDS_0\ X : SP\ \mathbf{given}_\Sigma\ T$ and $[\![UDS]\!]$ is defined and non-empty, then $[\![UDS_0]\!]$ is defined and non-empty and, for any $\varrho'_0 \in [\![UDS_0]\!]$, $[\![T]\!]_{\varrho'_0}$ is defined and there exists a model $N \in [\![SP]\!]$ such that $N|_\Sigma = [\![T]\!]_{\varrho'_0}$. By the induction hypothesis, $[\![UDS_0]\!]^\perp$ is then defined.

Take any $\varrho_0 \in [\![UDS_0]\!]^\perp$. Then, again by the induction hypothesis, there exists an environment $\varrho'_0 \in [\![UDS_0]\!]$ such that ϱ_0 is a $\perp$-restriction of ϱ'_0. Since we know that $[\![T]\!]_{\varrho'_0}$ is defined, $[\![T]\!]^\perp_{\varrho_0}$ must be, by Lemma 81, defined as well. Thus, $[\![UDS]\!]^\perp$ is defined.

Now take any $\varrho \in [\![UDS]\!]^\perp$. Then $\varrho = \varrho_0[M/X]$, where $\varrho_0 \in [\![UDS_0]\!]^\perp$, $M \in [\![SP]\!]_\perp$ and $M \neq \perp$ implies $M|_\Sigma = [\![T]\!]^\perp_{\varrho_0}$. Take the ϱ'_0 discussed above. Let $N \in [\![SP]\!]$ be any model such that $N|_\Sigma = [\![T]\!]_{\varrho'_0}$; by assumption, such a model exists. If $M = \perp$, let $\varrho' = \varrho'_0[N/X]$. Then $\varrho' \in [\![UDS]\!]$, since $\varrho'_0 \in [\![UDS_0]\!]$, $N \in [\![SP]\!]$ and $N|_\Sigma = [\![T]\!]_{\varrho'_0}$. Also, ϱ is a $\perp$-restriction of ϱ', since ϱ_0 is a $\perp$-restriction of ϱ'_0 and $\varrho(X) = M = \perp$. If $M \neq \perp$, let $\varrho' = \varrho'_0[M/X]$. Then $\varrho' \in [\![UDS]\!]$, since $\varrho'_0 \in [\![UDS_0]\!]$, $M \in [\![SP]\!]$ and $M|_\Sigma = [\![T]\!]^\perp_{\varrho_0} = [\![T]\!]_{\varrho'_0}$, where the last equation holds, by Lemma 81, since $[\![T]\!]^\perp_{\varrho_0}$ is defined and not equal to $\perp$. Also, ϱ is a $\perp$-restriction of ϱ', since ϱ_0 is a $\perp$-restriction of ϱ'_0 and $\varrho(X) = M = \varrho'(X)$. □

Suppose $ASP = \mathbf{units}\ UDS\ \mathbf{result}\ UE$ is correct, does not contain inconsistent specifications and, if parametric, uses it λ-argument.

Then $[UDS]$ is defined and $[\![UDS]\!]$ is defined and non-empty. Thus, by Lemma 82, $[\![UDS]\!]^\perp$ is defined and, for any $\varrho \in [\![UDS]\!]^\perp$, there exists an environment $\varrho' \in [\![UDS]\!]$ such that ϱ is a $\perp$-restriction of ϱ'.

Assume that $UE = T$. Take any $\varrho \in [\![UDS]\!]^\perp$. Since ASP is correct, $[\![T]\!]_{\varrho'}$ is defined. Because ϱ is a $\perp$-restriction of ϱ', we may, by Lemma 81, conclude that $[\![T]\!]^\perp_\varrho$ is defined as well and so ASP is partially correct.

Now assume that $UE = \lambda X : SP \bullet T$. Take any $\varrho \in [\![UDS]\!]^\perp$ and any $M \in [\![SP]\!]_\perp$. We need to prove that $[\![T]\!]^\perp_{\varrho[M/X]}$ is defined and that if it is not equal to $\perp$, then $[\![T]\!]^\perp_{\varrho[M/X]}|_{sig[SP]} = M$. If $M \neq \perp$, let $N = M$, and otherwise let N be some element of $[\![SP]\!]$; note that, by assumption, such an element must exist. In any case, $N \in [\![SP]\!]$, so, ASP being correct, $[\![T]\!]_{\varrho'[N/X]}$ is defined. Clearly, $\varrho[M/X]$ is a $\perp$-restriction of $\varrho'[N/X]$. Thus, by Lemma 81, $[\![T]\!]^{\perp}_{\varrho[M/X]}$ is indeed defined and, if it is not equal to $\perp$, then it is equal to $[\![T]\!]_{\varrho'[N/X]}$. If $M = \perp$, then $[\![T]\!]^{\perp}_{\varrho[M/X]}$ is equal to $\perp$ as well, since *UE* uses its λ-argument, which means that X appears freely in T, and since, in terms, $\perp$ gets propagated. Thus, for $M = \perp$ we arrive at the needed conclusion. So, assume $M \neq \perp$. Then $M = N$, which implies that $[\![T]\!]^{\perp}_{\varrho[M/X]}|_{sig[SP]} = [\![T]\!]_{\varrho'[N/X]}|_{sig[SP]} = N = M$, the penultimate equation holding since *ASP* is correct. □

2 Proof of Proposition 21

Below, proofs of lemmas used in the proof, as well as a proof of the proposition itself, may be found. They all use notions introduced in the sketch of the proof, in Section 9, page 67.

Lemma 22. *Let ϱ partially fit C_s in* $\Pr(\mathcal{L}, \mathcal{P})$. *Then:*

- *ϱ has a model iff ϱ is domain-stable,*
- *ϱ has an exact model iff ϱ has a model,*
- *ϱ has exactly one exact model iff ϱ is stable.*

Proof. It suffices to consider, in the proof, only the question of a parametric unit $F \in \mathbf{Unit}_{\perp}(\Sigma \to \Delta)$ and its models, denoted F'. This is because, on non-parametric units, models are always uniquely defined.

For the first point, assume first that F has a model, F', and that $[\![P]\!] = [\![Q]\!]$ and $P \in \mathrm{dom}(F)$. Then $P \in \mathrm{dom}(F)$ implies $[\![P]\!] \in \mathrm{dom}(F')$, i.e., $[\![Q]\!] \in \mathrm{dom}(F')$, which implies $Q \in \mathrm{dom}(F)$. This proves that F is domain-stable. Now assume F is domain-stable. Denote the map $P \mapsto [\![P]\!]$ for all $P \in \mathrm{dom}(F)$ by Ψ. Of course, Ψ is onto $\Psi(\mathrm{dom}(F))$. Hence, by the axiom of choice, there is a map Ψ^{-1} such that $\Psi^{-1}; \Psi = \mathrm{id}_{\Psi(\mathrm{dom}(F))}$. Let F' have as domain $\Psi(\mathrm{dom}(F))$ and be defined by $F'(M) = [\![F(\Psi^{-1}(M))]\!]$. Then F' is a model of F. To see this, take any M in its domain and let $P = \Psi^{-1}(M)$. Then $F'(M)|_{\Sigma} = F'([\![P]\!])|_{\Sigma} = [\![F(P)]\!]|_{\Sigma} = [\![F(P)|_{\Sigma}]\!] = [\![P]\!] = M$ and so F' is persistent. Also, $F'(\perp) = \perp$, since the only "program" with semantics $\perp$ is $\perp$ itself and $F(\perp) = \perp$. Domain-stability means precisely that a set of all $[\![P]\!]$ such that $P \in \mathrm{dom}(F)$ is well defined. But this set is exactly $\Psi(\mathrm{dom}(F))$, that is, $\mathrm{dom}(F')$; clearly, $[\![P]\!]$ is an element of this set, i.e., of $\mathrm{dom}(F')$, iff $P \in \mathrm{dom}(F)$. Finally, if $P \in \mathrm{dom}(F)$, then $F'([\![P]\!]) = [\![F(\Psi^{-1}(\Psi(P)))]\!] = [\![F(Q)]\!]$, where $Q = \Psi^{-1}(\Psi(P))$, whereby $Q \in \mathrm{dom}(F)$ and $[\![P]\!] = \Psi(P) = \Psi(\Psi^{-1}(\Psi(P))) = \Psi(Q) = [\![Q]\!]$. This ends the proof of the first point.

As for the second point, obviously, if F has an exact model, then it has a model as well. If, on the other hand, it has a model, then, in fact, one may pick the model F' defined above. The model F' is exact, since, for any $M \in \mathrm{dom}(F')$, we have $M = \Psi(\Psi^{-1}(M)) = [\![P]\!]$, where $P = \Psi^{-1}(M)$.

As for the third point, assume there is exactly one exact model F' of F. This means that F' is the model constructed in point one, irrespective of which Ψ^{-1} was chosen. Take any $P, Q \in \mathrm{dom}(F)$ such that $[\![P]\!] = [\![Q]\!]$, i.e., $\Psi(P) = \Psi(Q) = M$, for some model $M \in \mathrm{dom}(F')$. Of course, there exist left inverses Ψ_1 and Ψ_2 of Ψ such that $\Psi_1(M) = P$ and $\Psi_2(M) = Q$. Both Ψ_1 and Ψ_2 must generate the same exact model F', which implies that $[\![F(P)]\!] = [\![F(\Psi_1(M))]\!] = F'(M) = [\![F(\Psi_2(M))]\!] = [\![F(Q)]\!]$. Thus, F is stable. Now assume that F is stable and that F'_1 and F'_2 are both exact models of F. Clearly, they both have the same domains, since the domain of any exact model is the set of all $[\![P]\!]$ such that $P \in \mathrm{dom}(F)$. Take any $P \in \mathrm{dom}(F)$. There exist Q_1 and Q_2 in $\mathrm{dom}(F)$ such that $[\![Q_1]\!] = [\![P]\!] = [\![Q_2]\!]$, $F'_1([\![P]\!]) = [\![F(Q_1)]\!]$ and $F'_2([\![P]\!]) = [\![F(Q_2)]\!]$. Then, by stability of F, $F'_1([\![P]\!]) = F'_2([\![P]\!])$. This implies that F'_1 and F'_2 are equal and so there is exactly one exact model of F. $\square$

Lemma 23. *Assume that ν is a set of parametric unit names in C_s and that $[T]_{C_s}$ is defined. Assume further that ϱ^P partially fits C_s in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ and that the context E partially fits C_s in $\mathcal{L}$.*

If

1. *T is generative,*
2. *$p(T) \subseteq \nu$,*
3. *ϱ^P is domain-stable,*
4. *$[\![T]\!]^{\perp}_{\top(\varrho^P)}$ is defined,*
5. *$[\![T]\!]^{\perp}_{\eta}$ is defined for all $\eta \in E$, and*
6. *for any model ϱ of ϱ^P, there exists a ν-equivalent model $\eta \in E$,*

then

(*) $[\![T]\!]^{\perp}_{\varrho^P}$ is defined and

$$[\![T]\!]^{\perp}_{\varrho^P} = [\![T]\!]^{\perp}_{\top(\varrho^P)}$$

(**) for any model ϱ of ϱ^P, there exists a $(\nu \setminus p(T))$-equivalent model $\eta' \in E$ such that, for any subterm T' of T,

$$[\![[\![T']\!]^{\perp}_{\varrho^P}]\!] = [\![T']\!]^{\perp}_{\eta'}$$

Proof. In the proof, a simple lemma is needed.

Lemma 83. *Assume ϱ_1 and ϱ_2 are ν-equivalent models of an environment ϱ^P partially fitting C_s, $[T]_{C_s}$ is defined and $p(T) \subseteq \nu$. If $[\![T']\!]^{\perp}_{\varrho_1}$ is defined and implementable for all subterms T' of T, then $[\![T]\!]^{\perp}_{\varrho_2}$ is defined and equal to $[\![T]\!]^{\perp}_{\varrho_1}$.*

Proof. The proof is by induction over T.

If $T =$"A", then $[\![T]\!]^{\perp}_{\varrho_1} = \varrho_1(A) = [\![\varrho^P(A)]\!] = \varrho_2(A) = [\![T]\!]^{\perp}_{\varrho_2}$.

If $T =$"$A(T_0)$", then, by the induction hypothesis, $[\![T_0]\!]^{\perp}_{\varrho_1} = [\![T_0]\!]^{\perp}_{\varrho_2}$. Moreover, this model is implementable. Therefore $\varrho_1(A)$ and $\varrho_2(A)$ coincide on it, thus proving $[\![T]\!]^{\perp}_{\varrho_1} = \varrho_1(A)([\![T_0]\!]^{\perp}_{\varrho_1}) = \varrho_2(A)([\![T_0]\!]^{\perp}_{\varrho_2}) = [\![T]\!]^{\perp}_{\varrho_2}$.

If $T =$"$\bigcup_{i\in I} \sigma_i(T_i)$", then, by the induction hypothesis, $[\![T_i]\!]^{\perp}_{\varrho_1} = [\![T_i]\!]^{\perp}_{\varrho_2}$ for all $i \in I$. Thus $[\![T]\!]^{\perp}_{\varrho_1} = \bigoplus_{i\in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho_1}) = \bigoplus_{i\in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho_2}) = [\![T]\!]^{\perp}_{\varrho_2}$.

If $T =$"$T_0|_{\sigma:\Delta_0\to\Delta}$", then, by the induction hypothesis, $[\![T_0]\!]^{\perp}_{\varrho_1} = [\![T_0]\!]^{\perp}_{\varrho_2}$ and so $[\![T]\!]^{\perp}_{\varrho_1} = [\![T_0]\!]^{\perp}_{\varrho_1}|^{\perp}_{\sigma} = [\![T_0]\!]^{\perp}_{\varrho_2}|^{\perp}_{\sigma} = [\![T]\!]^{\perp}_{\varrho_2}$. $\square$

The proof is by induction over the structure of T.

The **first** case is $T = A$. Then $[\![T]\!]^{\perp}_{\varrho^P} = \varrho^P(A) = \top(\varrho^P)(A) = [\![T]\!]^{\perp}_{\top(\varrho^P)}$, which proves (*). Assume ϱ is a model of ϱ^P. By (6), there exists a ν-equivalent model $\eta \in E$. Let $\eta' = \eta$. Then η' is a model of ϱ^P, η' and ϱ are $(\nu \setminus p(T))$-equivalent and $\eta' \in E$. Also, $[\![[\![T]\!]^{\perp}_{\varrho^P}]\!] = [\![\varrho^P(A)]\!] = \eta'(A) = [\![T]\!]^{\perp}_{\eta'}$ and T does not have any proper subterms. This proves (**).

Before we consider the remaining cases, observe that if, for T, ν, ϱ^P and E, conditions (1)–(6) hold and T' is a subterm of T, then the same conditions hold for T', ν, ϱ^P and E. Therefore, one can always apply the lemma inductively to subterms.

The **second** case is $T = A(T_0)$. By the above remark, the lemma may be applied inductively to T_0, ν, ϱ^P and E. Then:

($*_0$) $[\![T_0]\!]^{\perp}_{\varrho^P}$ is defined and equal to $[\![T_0]\!]^{\perp}_{\top(\varrho^P)}$,

($**_0$) for any model ϱ of ϱ^P, there exists a $(\nu\setminus p(T_0))$-equivalent model $\eta'_0 \in E$ such that, for any subterm T'_0 of T, $[\![[\![T'_0]\!]^{\perp}_{\varrho^P}]\!] = [\![T'_0]\!]^{\perp}_{\eta'_0}$.

By (3) and Lemma 22, there exists some model ϱ of ϱ^P. Consider the corresponding model $\eta'_0 \in E$, which exists by ($**_0$). By (5), $[\![A(T_0)]\!]^{\perp}_{\eta'_0}$ is defined and so $[\![T_0]\!]^{\perp}_{\eta'_0} \in \mathrm{dom}(\eta'_0(A))$, which, by ($**_0$), implies $[\![[\![T_0]\!]^{\perp}_{\varrho^P}]\!] = [\![T_0]\!]^{\perp}_{\eta'_0} \in \mathrm{dom}(\eta'_0(A))$. Since η'_0 is a model of ϱ^P, this implies that $[\![T_0]\!]^{\perp}_{\varrho^P} \in \mathrm{dom}(\varrho^P(A))$.

For this reason, $[\![A(T_0)]\!]^{\perp}_{\varrho^P}$ is defined and coincides with $\varrho^P(A)([\![T_0]\!]^{\perp}_{\varrho^P}) = \top(\varrho^P)(A)([\![T_0]\!]^{\perp}_{\varrho^P})$, which, by ($*_0$), coincides with $\top(\varrho^P)(A)([\![T_0]\!]^{\perp}_{\top(\varrho^P)}) = [\![A(T_0)]\!]^{\perp}_{\top(\varrho^P)}$, which proves ($*$).

As for ($**$), take any model ϱ if ϱ^P. Let $P_0 = [\![T_0]\!]^{\perp}_{\varrho^P}$ and $P = [\![A(T_0)]\!]^{\perp}_{\varrho^P}$. Let $F = \varrho(A)[[\![P]\!]/[\![P_0]\!]]$ and $\varrho' = \varrho[F/A]$. Then F is a correct unit, since $P_0 = \perp$ implies $P = \perp$ and $P_0 \neq \perp$ implies $P|_\Sigma = P_0$, where $C_s(A) = \Sigma \to \Sigma'$. Also, ϱ' is a model of ϱ^P, since ϱ is a model of ϱ^P, $P_0 \in \mathrm{dom}(\varrho^P(A))$ and $\varrho'(A)([\![P_0]\!]) = [\![P]\!] = [\![\varrho^P(A)(P_0)]\!]$. Clearly, ϱ' and ϱ are $(\nu \setminus p(T))$-equivalent, since $A \in p(T)$. Consider the model $\eta'_0 \in E$ corresponding to ϱ', which exists by ($**_0$). Then η'_0 and ϱ are $(\nu \setminus p(T))$-equivalent, because η'_0 and ϱ' are and ϱ' and ϱ are.

Take any subterm T' of T. If $T' = T$, then $[\![[\![T']\!]^{\perp}_{\varrho^P}]\!] = [\![[\![A(T_0)]\!]^{\perp}_{\varrho^P}]\!] = [\![P]\!] = F([\![P_0]\!]) = F([\![[\![T_0]\!]^{\perp}_{\varrho^P}]\!]) = F([\![T_0]\!]^{\perp}_{\eta'_0})$, since T_0 is a subterm of itself. This is further equal to $\eta'_0(A)([\![T_0]\!]^{\perp}_{\eta'_0})$, since, first, by (1) and (2), $A \in \nu \setminus p(T_0)$, and, second, η'_0 and ϱ' are $(\nu \setminus p(T_0))$-equivalent and, third, $[\![T_0]\!]^{\perp}_{\eta'_0}$ is certainly implementable, since it is equal to $[\![[\![T_0]\!]^{\perp}_{\varrho^P}]\!]$. Further, this is equal to $[\![A(T_0)]\!]^{\perp}_{\eta'_0}$, which ends the proof of ($**$) for $T' = T$.

If T' is a subterm of T_0, then, by definition of η'_0, $[\![[\![T']\!]^{\perp}_{\varrho^P}]\!] = [\![T']\!]^{\perp}_{\eta'_0}$. This ends the proof for the second case.

The **third** case is $T = \bigcup_{i \in I} \sigma_i(T_i)$. Then, by the earlier remark, the lemma may be applied inductively to T_i, ν, ϱ^P and E, obtaining, for any $i \in I$,

($*_i$) $[\![T_i]\!]^{\perp}_{\varrho^P}$ is defined and equal to $[\![T_i]\!]^{\perp}_{\top(\varrho^P)}$,

($**_i$) for any model ϱ of ϱ^P, there exists a $(\nu \setminus p(T_i))$-equivalent model $\eta'_i \in E$ such that, for any subterm T' of T_i, $[\![[\![T']\!]^{\perp}_{\varrho^P}]\!] = [\![T']\!]^{\perp}_{\eta'_i}$.

By (4), the amalgamation $[\![T]\!]^{\perp}_{\top(\varrho^P)} = \bigoplus_{i \in I} [\![T_i]\!]^{\perp}_{\top(\varrho^P)}$ exists. Since, by ($*_i$), $[\![T_i]\!]^{\perp}_{\top(\varrho^P)} = [\![T_i]\!]^{\perp}_{\varrho^P}$, this implies that $[\![T]\!]^{\perp}_{\varrho^P} = \bigoplus_{i \in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho^P})$ is defined and equal to $[\![T]\!]^{\perp}_{\top(\varrho^P)}$. This proves ($*$).

As for ($**$), take any model ϱ of ϱ^P. For all $i \in I$, take the corresponding η'_i, which exists by ($**_i$). Let ϱ' partially fitting C_s be defined as follows, for all X in C_s:

- if $X \notin p(T)$, then $\varrho'(X) = \varrho(X)$,
- if $X \in p(T_i)$, then $\varrho'(X) = \eta'_i(X)$.

The above definition is correct, since, if one of the cases holds, then the others may not hold; this is because T is, by (1), generative. Now, ϱ' is a model of

ϱ^P. This is clear, since being a model is a pointwise property and ϱ and all η'_i are models of ϱ^P. Also, ϱ' and ϱ are $(\nu \setminus p(T))$-equivalent, since they coincide on non-parametric unit names and, for any parametric $X \in \nu \setminus p(T)$, we have $\varrho'(X) = \varrho(X)$. By (6), there exists a model $\eta' \in E$ of ϱ^P such that η' and ϱ' are ν-equivalent. Of course, this implies that they are $(\nu \setminus p(T))$-equivalent as well and so η' and ϱ are $(\nu \setminus p(T))$-equivalent.

Observe that, for any proper subterm T' of T, the equality $[\![[\![T']\!]^{\perp}_{\varrho^P}]\!] = [\![T']\!]^{\perp}_{\eta'}$ holds. To see this, note that T' must be a subterm of some T_i. Let T'' be any subterm of T'. Note that ϱ' and η'_i coincide on all non-parametric unit names (because they are both models of ϱ^P) and on all parametric unit names from $p(T_i) \supseteq p(T'')$. Additionally, $[\![T'']\!]^{\perp}_{\eta'_i}$ is defined and equal to $[\![[\![T'']\!]^{\perp}_{\varrho^P}]\!]$. Hence $[\![T'']\!]^{\perp}_{\varrho'}$ must be defined and equal $[\![[\![T'']\!]^{\perp}_{\varrho^P}]\!]$ and so implementable. Note also that η' and ϱ' are ν-equivalent models of ϱ^P and that $p(T'') \subseteq p(T) \subseteq \nu$. Hence, by Lemma 83, $[\![T']\!]^{\perp}_{\eta'}$ is defined and equal to $[\![T']\!]^{\perp}_{\varrho'} = [\![[\![T']\!]^{\perp}_{\varrho^P}]\!]$, as claimed.

Take any subterm T' of T. If T' is a proper subterm of T, then the required equality $[\![[\![T']\!]^{\perp}_{\varrho^P}]\!] = [\![T']\!]^{\perp}_{\eta'}$ holds, as has just been proven. So, assume $T' = T$. Observe that the proved equality for proper subterms T' can be specialized to $[\![[\![T_i]\!]^{\perp}_{\varrho^P}]\!] = [\![T_i]\!]^{\perp}_{\eta'}$.

Then $[\![[\![T]\!]^{\perp}_{\varrho^P}]\!] = [\![\bigoplus_{i\in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho^P})]\!] = \bigoplus_{i\in I} \sigma_i([\![[\![T_i]\!]^{\perp}_{\varrho^P}]\!])$, because:

- $([\![\bigoplus_{i\in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho^P})]\!])|_{\sigma_i} = [\![(\bigoplus_{i\in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho^P}))|_{\sigma_i}]\!] = [\![[\![T_i]\!]^{\perp}_{\varrho^P}]\!]$, and
- such an object is unique, because the amalgamation $\bigoplus_{i\in I} \sigma_i([\![[\![T_i]\!]^{\perp}_{\varrho^P}]\!])$ is defined, since it is equal to $\bigoplus_{i\in I} \sigma_i([\![T_i]\!]^{\perp}_{\eta'}) = [\![T]\!]^{\perp}_{\eta'}$, which is defined, since $\eta' \in E$ and (5) holds.

Further $\bigoplus_{i\in I} \sigma_i([\![[\![T_i]\!]^{\perp}_{\varrho^P}]\!]) = \bigoplus_{i\in I} \sigma_i([\![T_i]\!]^{\perp}_{\eta'}) = [\![T]\!]^{\perp}_{\eta'}$. This proves (**).

The **fourth** case is $T = T_0|_{\sigma:\Delta_0\to\Delta}$. We then get, by the induction hypothesis for T_0, ν, ϱ^P and E, exactly the same conditions ($*_0$) and ($**_0$) as in the case "$A(T_0)$". Then, by ($*_0$), $[\![T]\!]^{\perp}_{\varrho^P}$ is, of course, defined and equals $[\![T_0]\!]^{\perp}_{\varrho^P}|^{\perp}_{\sigma} = [\![T_0]\!]^{\perp}_{\top(\varrho^P)}|^{\perp}_{\sigma} = [\![T]\!]^{\perp}_{\top(\varrho^P)}$, which proves (*).

As for (**), take any model ϱ of ϱ^P. Take $\eta' = \eta'_0$, where η'_0 is the model corresponding to ϱ, which exists by ($**_0$). Since $p(T_0) = p(T)$, the environments η' and ϱ are $(\nu \setminus p(T))$-equivalent. Take any subterm T' of T. If T' is a proper subterm, then (**) holds, since it is equivalent to ($**_0$). So, assume $T' = T$. Since $[\![T_0]\!]^{\perp}_{\eta'_0} = [\![[\![T_0]\!]^{\perp}_{\varrho^P}]\!]$, we also have $[\![T]\!]^{\perp}_{\eta'_0} = [\![[\![T]\!]^{\perp}_{\varrho^P}]\!]$, which proves (**). □

Lemma 24. *Assume UDS is generative, $[UDS]$ is defined, $C = [\![|UDS|]\!]^{\perp}$ is defined in $\Pr(\mathcal{L}, \mathcal{P})$ and $E = [\![UDS]\!]^{\perp}$ is defined in $\mathcal{L}$.*

Then $C^P = \llbracket UDS \rrbracket^\perp$ *is defined in* $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$.
Assume, additionally, that $\varrho^P \in C^P$. *Then:*

1. ϱ^P *is domain-stable,*

2. $\top(\varrho^P) \in C$,

3. *for any model* ϱ *of* ϱ^P, *there exists a* $\overline{p}(UDS)$*-equivalent model* $\eta \in E$.

Proof. The proof is by induction over the structure of UDS.

The **first** case is $UDS = \epsilon$. Then $C_s = \emptyset$, $C = E = C^P = \{\emptyset\}$. Also $\emptyset$ is domain-stable, $\top(\emptyset) = \emptyset$ and as the model η one may take $\varrho = \emptyset \in E$.

The **second** case is $UDS = UDS_0\ X : USP$. Then UDS_0 is generative and:

- $[UDS_0]$ is defined and $[UDS] = [UDS_0][sig[USP]/X]$,

- $C_0 = \llbracket |UDS_0| \rrbracket^\perp$ is defined in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ and $C = C_0[\mathcal{U}/X]$, where $\mathcal{U}$ is the class of all units $\mathbf{Unit}_\perp(sig[USP])$ interpreted in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$,

- $E_0 = \llbracket UDS_0 \rrbracket^\perp$ is defined in $\mathcal{L}$ and $E = E_0[\llbracket USP \rrbracket_\perp / X]$,

By the induction hypothesis, $C_0^P = \llbracket UDS_0 \rrbracket^\perp$ is defined in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$ and, if $\varrho_0^P \in C_0^P$, then:

1′ ϱ_0^P is domain-stable,

2′ $\top(\varrho_0^P) \in C_0$,

3′ for any model ϱ_0 of ϱ_0^P, there exists a $\overline{p}(UDS_0)$-equivalent model $\eta_0 \in E_0$.

Of course, $C^P = \llbracket UDS \rrbracket^\perp$ is defined in $\mathrm{Pr}(\mathcal{L}, \mathcal{P})$, since $\llbracket UDS_0 \rrbracket^\perp$ is.

Assume $\varrho^P \in C^P$. Then $\varrho^P = \varrho_0^P[F/X]$, where $\varrho_0^P \in C_0^P$ and $F \in \llbracket USP \rrbracket_\perp$. Then, ϱ^P is domain-stable, since ϱ_0^P is and, if X is parametric, then F is domain-stable (its domain is of the form $\llbracket SP \rrbracket_\perp$ for some specification SP). This proves (1). Consider $\top(\varrho^P)$. Since (2′) holds and, if X is parametric, the "totalized" version of F is in $\llbracket \top \to \top \rrbracket_\perp$, we have $\top(\varrho^P) \in C$. This proves (2). Now take any model $\varrho = \varrho_0[G/X]$ of ϱ^P. Then ϱ_0 is a model of ϱ_0^P. Let η_0 be the model corresponding to ϱ_0 via (3′). Let $\eta = \eta_0[G'/X]$, where $G' = G$ if X is non-parametric and otherwise G' is the restriction of G to $\perp$ and implementable models. Then η is a model of ϱ^P. Since, if X is parametric, $\eta(X)$ and $\varrho(X)$ agree on implementable models and $\overline{p}(UDS) = \overline{p}(UDS_0) \cup \{X\}$, or, if X is non-parametric, $\overline{p}(UDS) = \overline{p}(UDS_0)$, η and ϱ are $\overline{p}(UDS)$-equivalent. If X is non-parametric, then $G' = G = [\![F]\!] \in [\![USP]\!]_\perp$, since $F \in [\![USP]\!]_\perp$. If X is parametric and $USP = SP \to SP'$, then $M \in \mathrm{dom}(G')$ iff $M = \perp$ or $M = [\![P]\!]$ for some program $P \in \mathrm{dom}(F)$, since ϱ is a model of ϱ^P. Hence, $\mathrm{dom}(F) = [\![SP]\!]_\perp$ and $\mathrm{dom}(G') = [\![SP]\!]_\perp$. Also, $G'([\![P]\!]) = G([\![P]\!]) = [\![F(Q)]\!]$ for some program $Q \in \mathrm{dom}(F)$. Then $F(Q) \in [\![SP']\!]_\perp$, which implies $G'([\![P]\!]) \in [\![SP]\!]_\perp$. This means that, for parametric X, we also have $G' \in [\![USP]\!]_\perp$. Because $\eta_0 \in E$ and $\eta(X) = G' \in [\![USP]\!]_\perp$, we may conclude that $\eta \in E$. This proves (3).

The **third** case is $UDS = UDS_0\ X : SP$ $\mathbf{given}_\Sigma$ T. Then UDS_0 is generative and:

- $[UDS_0]$ is defined and $[UDS] = [UDS_0][sig[SP]/X]$,
- $C_0 = [\![|UDS_0|]\!]^\perp$ is defined in $\Pr(\mathcal{L}, \mathcal{P})$ and $C = \{\varrho_0[P/X] \mid \varrho_0 \in C_0,\ P \in [\![\top_{sig[SP]}]\!]_\perp,\ P \neq \perp \text{ implies } P|_\Sigma = [\![T]\!]^\perp_{\varrho_0}\}$,
- $E_0 = [\![UDS_0]\!]^\perp$ is defined in $\mathcal{L}$ and $E = \{\varrho_0[M/X] \mid \varrho_0 \in E_0,\ M \in [\![SP]\!]_\perp,\ M \neq \perp \text{ implies } M|_\Sigma = [\![T]\!]^\perp_{\varrho_0}\}$.

By the induction hypothesis, $C_0^P = [\![UDS_0]\!]^\perp$ is defined in $\Pr(\mathcal{L}, \mathcal{P})$ and, if $\varrho_0^P \in C_0^P$, then:

1′ ϱ_0^P is domain-stable,

2′ $\top(\varrho_0^P) \in C_0$,

3′ for any model ϱ_0 of ϱ_0^P, there exists a $\overline{p}(UDS_0)$-equivalent model $\eta_0 \in E_0$.

Observe also, that:

- T is generative, since UDS is,
- $p(T) \subseteq \overline{p}(UDS_0)$, since UDS is generative,
- $[\![T]\!]^\perp_{\top(\varrho_0^P)}$ is defined, since (2′) holds and C is defined,
- $[\![T]\!]^\perp_{\eta_0}$ is defined for all $\eta_0 \in E_0$, since E is defined.

Thus, we may apply Lemma 23 to T, $\overline{p}(UDS_0)$, ϱ_0^P and E_0. Then:

(*) $[\![T]\!]^\perp_{\top(\varrho_0^P)}$ is defined and equal to $[\![T]\!]^\perp_{\varrho_0^P}$,

(**) for any model ϱ_0 of ϱ_0^P, there exists a $(\overline{p}(UDS_0) \setminus p(T))$-equivalent model $\eta_0 \in E_0$ such that, for any subterm T' of T, $[\![[\![T']\!]^\perp_{\varrho_0^P}]\!] = [\![T']\!]^\perp_{\eta_0}$.

By (*), $C^P = [\![UDS]\!]^\perp$ is defined; moreover, $C^P = \{\varrho_0^P[P/X] \mid \varrho_0^P \in C_0^P,\ P \in [\![SP]\!]_\perp,\ P \neq \perp$ implies $P|_\Sigma = [\![T]\!]^\perp_{\varrho_0^P}\}$.

Assume $\varrho^P \in C^P$. Then $\varrho^P = \varrho_0^P[P/X]$, where ϱ_0^P and P are as above. Since X is non-parametric and (1′) holds, ϱ^P is domain-stable, which proves (1). Because (2′) holds, X is non-parametric and the equality of (*) holds, $\top(\varrho^P) \in C$, which proves (2). Now let $\varrho = \varrho_0[M/X]$ be any model of ϱ^P. Then ϱ_0 is a model of ϱ_0^P and $M = [\![\varrho^P(X)]\!]$. Let η_0 be the model corresponding to ϱ_0 via (**). Then $\eta = \eta_0[M/X]$ is a model of ϱ^P. Observe that $\overline{p}(UDS_0) \setminus p(T) = \overline{p}(UDS)$, so η_0 and ϱ_0 are $\overline{p}(UDS)$-equivalent; since $\eta(X) = M = \varrho(X)$, this implies that η and ϱ are $\overline{p}(UDS)$-equivalent as well. Finally, $\eta \in E$, since $\eta_0 \in E_0$, $\eta(X) = M = [\![\varrho^P(X)]\!] \in [\![SP]\!]_\perp$, because $\varrho^P(X) \in [\![SP]\!]_\perp$, and $\eta(X) \neq \perp$ implies $\varrho^P(X) \neq \perp$, which implies $\varrho^P(X)|_\Sigma = [\![T]\!]^\perp_{\varrho_0^P}$, which, in turn, implies $\eta(X)|_\Sigma = M|_\Sigma = [\![\varrho^P(X)]\!]|_\Sigma = [\![\varrho^P(X)|_\Sigma]\!] = [\![[\![T]\!]^\perp_{\varrho_0^P}]\!]$; this, by (**), equals $[\![T]\!]^\perp_{\eta_0}$. This proves (3). $\square$

Finally, the whole proposition may be proved:

Proposition 21. *If $\mathcal{P}$ is a programming language compatible with $\mathcal{L}$, ASP is generative, $|ASP|$ is partially correct in* $\mathrm{Pr}(\mathcal{L},\mathcal{P})$ *and ASP is partially correct in $\mathcal{L}$, then ASP is partially correct in* $\mathrm{Pr}(\mathcal{L},\mathcal{P})$.

Proof. Assume that $\mathcal{P}$ is compatible with $\mathcal{L}$, that ASP is generative, $|ASP|$ is partially correct in $\mathrm{Pr}(\mathcal{L},\mathcal{P})$, ASP is partially correct in $\mathcal{L}$.

Let UDS be the declarations of ASP.

Then UDS is generative, $[UDS]$ is defined, $C = [\![|UDS|]\!]^\perp$ is defined in $\mathrm{Pr}(\mathcal{L},\mathcal{P})$ and $E = [\![UDS]\!]^\perp$ is defined in $\mathcal{L}$. By Lemma 24, we may infer that $C^P = [\![UDS]\!]^\perp$ is defined in $\mathrm{Pr}(\mathcal{L},\mathcal{P})$ and that, for any $\varrho^P \in C^P$, ϱ^P is domain-stable, $\top(\varrho^P) \in C$ and, for any model ϱ of ϱ^P, there exists a $\overline{p}(UDS)$-equivalent model $\eta \in E$.

Suppose that the **result** part of ASP is a non-parametric expression T and take any $\varrho^P \in C^P$.

Then T is generative, $p(T) \subseteq \overline{p}(UDS)$, ϱ^P is domain-stable, $[\![T]\!]^\perp_{\top(\varrho^P)}$ is defined, since $\top(\varrho^P) \in C$ and $|ASP|$ is partially correct in $\mathrm{Pr}(\mathcal{L},\mathcal{P})$, $[\![T]\!]^\perp_\eta$ is defined for all $\eta \in E$, since ASP is partially correct in $\mathcal{L}$, and, for any model ϱ of ϱ^P, there exists a $\overline{p}(UDS)$-equivalent model $\eta \in E$. This, by Lemma 23, implies that $[\![T]\!]^\perp_{\varrho^P}$ is defined.

Hence, ASP is partially correct in $\mathrm{Pr}(\mathcal{L},\mathcal{P})$.

Now suppose that the **result** part of ASP is a parametric expression $\lambda X : SP \bullet T$ and take any $\varrho^P \in C^P$ and $P \in [\![SP]\!]_\perp$. Let $C_s^1 = [UDS][sig[SP]/X]$, $E_1 = E[[\![SP]\!]_\perp/X]$, $C_1 = C[[\![\top_{sig[SP]}]\!]_\perp/X]$ and $\varrho_1^P = \varrho^P[P/X]$.

Then T is generative, $p(T) \subseteq \overline{p}(UDS)$, ϱ_1^P is domain-stable, $[\![T]\!]^\perp_{\top(\varrho_1^P)}$ is defined, since $\top(\varrho_1^P) \in C_1$ and $|ASP|$ is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$, $[\![T]\!]^\perp_{\eta_1}$ is defined for all $\eta_1 \in E_1$, since ASP is partially correct in $\mathcal{L}$, and, for any model ϱ_1 of ϱ_1^P, there exists a $\overline{p}(UDS)$-equivalent model $\eta_1 \in E_1$, since as η_1 we may simply take $\eta[\varrho_1(X)/X]$, where $\eta \in E$ is a $\overline{p}(UDS)$-equivalent model of ϱ^P. This, by Lemma 23, implies that $[\![T]\!]^\perp_{\varrho_1^P}$ is defined. Moreover, it is equal to $[\![T]\!]^\perp_{\top(\varrho_1^P)}$. Thus, if $[\![T]\!]^\perp_{\varrho_1^P} \neq \perp$, then $[\![T]\!]^\perp_{\varrho_1^P}|_{sig[SP]} = [\![T]\!]^\perp_{\top(\varrho_1^P)}|_{sig[SP]} = P$, since $|ASP|$ is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$.

Hence, ASP is partially correct in $\Pr(\mathcal{L}, \mathcal{P})$. $\square$

3 Proof of Proposition 25

As in the previous section, proofs of lemmas used in the proof and a proof of the whole Proposition 25 itself may be found below. Again, they all use notions introduced in the sketch of the proof found in Section 9 on page 70.

The following trivial lemma states that names not used in a term cannot influence its semantics. The proof by an obvious induction is omitted.

Lemma 84. *If $[T]_{C_s}$ is defined, ϱ_1 and ϱ_2 partially fit C_s and coincide on names of non-parametric units and on names from $p(T)$, and if $[\![T]\!]^\perp_{\varrho_1}$ is defined, then $[\![T]\!]^\perp_{\varrho_2}$ is defined as well and they are both equal.* $\square$

Lemma 26. *Let $\mathcal{L}_s$ be a static language.*

Assume T is generative, ζ, χ is a context description such that $[T]_{C_s(\chi)}$ is defined in $\mathcal{L}_s$ and that fully static checking is runnable on T in $\mathcal{L}_s$ and $\mathcal{J}$.

The following conditions are equivalent:

(FS) $\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma, t$ *in $\mathcal{L}_s$ and $\mathcal{J}$,*

(PM) *for any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation, for any total environment ϱ χ-represented by an environment η satisfying ζ, $[\![T]\!]^\perp_\varrho$ is defined in $\mathcal{L}$.*

Moreover, if **(FS)** *holds, $(\mathcal{L})_s = \mathcal{L}_s$, $\mathcal{L}$ has amalgamation and η satisfies ζ, then:*

$$\{[\![T]\!]^\perp_\varrho \mid \varrho \textit{ is total and } \chi\textit{-represented by } \eta\} \setminus \{\perp\}$$
$$= \{M|_t \mid M \ \Gamma\textit{-represents } \eta|_{\mathrm{dom}(\Gamma)}\} \quad (*)$$

If the signature category of $\mathcal{L}_s$ is cocomplete and $\mathcal{L}_0$ is a language with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation and iso-reflection, then **(FS)** *and* **(PM)** *are also equivalent to:*

(EPM) *for any total environment ϱ χ-represented by an environment η satisfying ζ, $[\![T]\!]^{\perp}_{\varrho}$ is defined in $\mathcal{L}_0$.*

Proof. In the proof we denote the signature $[T]_{C_s(\chi)}$ by Δ.

It is obvious that **(PM)** implies **(EPM)**. Therefore we only prove, first, that **(FS)** implies **(PM)** and that condition (*) then holds and, second, that **(PM)** or **(EPM)** implies **(FS)**. The proof is by induction over the structure of the term T.

The **first** case is $T = A$. Then **(FS)**, **(PM)** and **(EPM)** must hold, with $\Gamma = \zeta(A) + \{A : \Delta, \mathrm{id}_\Delta : A \to \star\}$ and $t = \chi(A)$. Now consider (*). Take any appropriate $\mathcal{L}$ and any η satisfying ζ. As for $\subseteq$, take any total environment ϱ χ-represented by η and such that $[\![T]\!]^{\perp}_{\varrho} \neq \perp$, and let $M = \eta(A)$. Then $[\![T]\!]^{\perp}_{\varrho} = \varrho(A) = \eta(A)|^{\perp}_{\chi(A)} = \eta(A)|_{\chi(A)} = M|_t$. Also, M Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$. For take any $X \in \mathrm{dom}(\Gamma)$. Either $X = A$ or $X \in \mathrm{dom}(\zeta(A))$. If $X = A$, then $\eta(A) = \eta(A)|_{\mathrm{id}_\Delta} = M|_{\Gamma(A \to \star)}$. If $X \in \mathrm{dom}(\zeta(A))$, then $\eta(X) = \eta(A)|_{\zeta(A)(X \to \star)}$, because η satisfies ζ, and this is further equal to $M|_{\Gamma(X\to\star)}$. As for $\supseteq$, take any M Γ-representing $\eta|_{\mathrm{dom}(\Gamma)}$. Let ϱ be the minimal total environment χ-represented by η. Then $[\![T]\!]^{\perp}_{\varrho} = \varrho(A) = \eta(A)|^{\perp}_{\chi(A)} = (M|_{\Gamma(A\to\star)})|^{\perp}_{\chi(A)} = (M|_{\mathrm{id}_\Delta})|^{\perp}_{\chi(A)} = M|^{\perp}_{\chi(A)} = M|_{\chi(A)} = M|_t$, since $\eta(A) = M|_{\Gamma(A\to\star)}$, because M Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$.

The **second** case is $T = A(T_0)$. Assume **(FS)** holds, $\mathcal{L}$ is an appropriate language, η satisfies ζ and ϱ is a total environment χ-represented by η. We then must have $\zeta, \chi \vdash T_0 \rhd\!\!\!\rhd \Gamma_0, t_0$, where (j, t) is a pushout for $(t_0, \iota_{\mathrm{dom}(t_0)\subseteq\Delta})$, $j \in \mathcal{J}$ and $\Gamma = \Gamma_0; j$. By the induction hypothesis, $[\![T_0]\!]^{\perp}_{\varrho}$ is defined. Because ϱ is total, $[\![T_0]\!]^{\perp}_{\varrho} \in \mathbf{Mod}_{\perp}(\mathrm{dom}(t_0)) = \mathrm{dom}(\varrho(A))$. Thus $[\![T]\!]^{\perp}_{\varrho}$ is defined, which proves **(PM)**, and it is equal to $\varrho(A)([\![T_0]\!]^{\perp}_{\varrho})$.

As for $\subseteq$ of (*), assume $\varrho(A)([\![T_0]\!]^{\perp}_{\varrho}) \neq \perp$; denote it by N. By the induction hypothesis and by the fact that $[\![T_0]\!]^{\perp}_{\varrho} \neq \perp$, we have that $[\![T_0]\!]^{\perp}_{\varrho} = M_0|_{t_0}$ for some M_0 Γ_0-representing $\eta|_{\mathrm{dom}(\Gamma_0)}$. Let $M = j(M_0) \oplus t(N)$. The amalgamation exists, since $\mathcal{L}$ has amalgamation, (j, t) is a pushout of $(t_0, \iota_{\mathrm{dom}(t_0)\subseteq\Delta})$ and $M_0|_{t_0} = [\![T_0]\!]^{\perp}_{\varrho} = N|_{\iota_{\mathrm{dom}(t_0)\subseteq\Delta}}$, by persistency. Then $M|_t = N = [\![T]\!]^{\perp}_{\varrho}$ and M Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$, since, for any $X \in \mathrm{dom}(\Gamma)$, $X \in \mathrm{dom}(\Gamma_0)$ and $\Gamma(X \to \star) = \Gamma_0(X \to \star); j$ and so $\eta(X) = M_0|_{\Gamma_0(X\to\star)} = (M|_j)|_{\Gamma_0(X\to\star)} = M|_{\Gamma(X\to\star)}$.

As for $\supseteq$ of (*), take any M Γ-representing $\eta|_{\mathrm{dom}(\Gamma)}$. Let $M_0 = M|_j$ and $N = M|_t$. Then M_0 Γ_0-represents $\eta|_{\mathrm{dom}(\Gamma_0)}$. By the induction hypothesis, there exists a total environment ϱ χ-represented by some environment η satisfying ζ, and such that $[\![T_0]\!]^{\perp}_{\varrho} = M_0|_{t_0}$. Let $\varrho' = \varrho[F/A]$, where $F \in \mathbf{Unit}_{\perp}(\chi(A))$ is constantly equal $\perp$ except that $F(M_0|_{t_0}) = N$; such an F is persistent, since $N|_{\iota_{\mathrm{dom}(t_0)\subseteq\Delta}} = M|_{\iota_{\mathrm{dom}(t_0)\subseteq\Delta};t} = M|_{t_0;j} = (M|_j)|_{t_0} = M_0|_{t_0}$. The environment ϱ' is total and it is χ-represented by η, since ϱ was. Finally,

$[\![T]\!]^{\perp}_{\varrho'} = \varrho'(A)([\![T_0]\!]^{\perp}_{\varrho'}) = F([\![T_0]\!]^{\perp}_{\varrho})$, by Lemma 84, since $[\![T_0]\!]^{\perp}_{\varrho}$ is defined and, by generativity, $A \notin p(T_0)$. Further, $F([\![T_0]\!]^{\perp}_{\varrho}) = F(M_0|_{t_0}) = N = M|_t$.

Now assume that **(PM)** or **(EPM)** holds. This implies that, for any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation or for $\mathcal{L}_0$ with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation and iso-reflection, for any environment η satisfying ζ and any total environment ϱ χ-represented by η, $[\![T_0]\!]^{\perp}_{\varrho}$ is defined. By the induction hypothesis we then have $\zeta, \chi \vdash T_0 \rhd\!\!\rhd \Gamma_0, t_0$. Since the signature category is $\mathcal{J}$-cocomplete, we then have $\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma, t$ for some Γ and t, which proves **(FS)**.

The **third** case is $T = \bigcup_{i \in I} \sigma_i(T_i)$. Assume that **(FS)** holds. We then have $\zeta, \chi \vdash T_i \rhd\!\!\rhd \Gamma_i, t_i : \Delta_i \to \Gamma_i(\star)$, for all $i \in I$, $\Gamma = \bigcup_{i \in I} \Gamma_i$ and a relevant transformation $\{\tau_i : \Gamma_i(\star) \to \Gamma(\star)\}_{i \in I}$. Moreover, one of the conditions **either** or **or** holds. Take any appropriate $\mathcal{L}$ with amalgamation and any environment η satisfying ζ. By the induction hypothesis, for any total environment ϱ χ-represented by η, $[\![T_i]\!]^{\perp}_{\varrho}$ is defined and the appropriate condition $(*_i)$ holds.

Suppose that the **either**-clause holds. In particular, let D be the witness of the regularity of σ and t be the universal morphism.

Take any total environment ϱ χ-represented by η. If $[\![T_i]\!]^{\perp}_{\varrho} = \perp$, for some $i \in I$, then $[\![T]\!]^{\perp}_{\varrho}$ is defined. So assume all $[\![T_i]\!]^{\perp}_{\varrho} \neq \perp$. By $(*_i)$, there exist models N_i which Γ_i-represent $\eta|_{\mathrm{dom}(\Gamma_i)}$ and such that $N_i|_{t_i} = [\![T_i]\!]^{\perp}_{\varrho}$.

Let $L = \{L_o\}_{o \in O}$ be a model family defined by $L_o = N_i|_{D(m);t_i}$, where $m : o \to i$ in D. The above definition is correct: first, for any $o \in O$ an appropriate m must exist by definition of witness of regular sink; second, since $\mathcal{L}$ has amalgamation, $\{\tau_i : \Gamma_i(\star) \to \Gamma(\star)\}_{i \in I}$ is a colimit over $\{\Gamma_i(X \to \star)\}_{i \in I, X \in \mathrm{dom}(\Gamma_i)}$ and $N_{i_1}|_{\Gamma_{i_1}(X \to \star)} = \eta(X) = N_{i_2}|_{\Gamma_{i_2}(X \to \star)}$ for $X \in \mathrm{dom}(\Gamma_{i_1}) \cap \mathrm{dom}(\Gamma_{i_2})$, so there exists a unique $\Gamma(\star)$-model N such that $N|_{\tau_i} = N_i$ for all $i \in I$; thus if $m_1 : o \to i_1$ and $m_2 : o \to i_2$ in D, then $N_{i_1}|_{D(m_1);t_{i_1}} = N|_{D(m_1);t_{i_1};\tau_{i_1}} = N|_{D(m_2);t_{i_2};\tau_{i_2}} = N_{i_2}|_{D(m_2);t_{i_2}}$.

Moreover, $L_i = N_i|_{t_i} = [\![T_i]\!]^{\perp}_{\varrho}$ and L is consistent with D, since for $m : o \to o'$ we have $L_{o'}|_{D(m)} = (N_i|_{D(m');t_i})|_{D(m)}$, where $m' : o' \to i$, which in turn equals $N_i|_{D(m;m');t_i} = L_o$. Because $\mathcal{L}$ has amalgamation, L is consistent with D and σ defines a colimit over D, there exists a unique Δ-model Q such that $L_i = Q|_{\sigma_i}$ for all $i \in I$. Thus $[\![T]\!]^{\perp}_{\varrho}$ is defined, which proves **(PM)**, and equals Q.

The inclusion $\subseteq$ in condition $(*)$ holds, since, if $[\![T]\!]^{\perp}_{\varrho} \neq \perp$, then $[\![T]\!]^{\perp}_{\varrho} = Q = N|_t$ and N Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$, since $N|_{\tau_i} = N_i$ and N_i Γ_i-represents $\eta|_{\mathrm{dom}(\Gamma_i)}$; the equality $Q = N|_t$ holds, since Q is the unique σ-amalgamation of L and $(N|_t)|_{\sigma_i} = N|_{\sigma_i;t} = N|_{t_i;\tau_i} = N_i|_{t_i} = L_i$.

As for the inclusion $\supseteq$ in $(*)$, take any N Γ-representing $\eta|_{\mathrm{dom}(\Gamma)}$ and define $N_i = N|_{\tau_i}$; then N_i Γ_i-represents $\eta|_{\mathrm{dom}(\Gamma_i)}$. By conditions $(*_i)$, there exist total ϱ_i χ-represented by η and such that $[\![T_i]\!]^{\perp}_{\varrho_i} = N_i|_{t_i}$. Let ϱ be defined so that, for non-parametric X, we have $\varrho(X) = \eta(X)|^{\perp}_{\chi(X)}$ and, for any parametric unit X, if $X \in p(T_i)$ then $\varrho(X) = \varrho_i(X)$, and if $X \notin p(T)$, then $\varrho(X)$ is the unit constantly equal $\perp$. This definition is correct, since, by the generativity of T, no X may appear in more than one T_i. Also, ϱ is total and χ-represented by η. For any $i \in I$, we have, by Lemma 84, $[\![T_i]\!]^{\perp}_{\varrho} = [\![T_i]\!]^{\perp}_{\varrho_i}$, because ϱ and ϱ_i differ only on unit names which do not appear in T_i. Define the family $\{L_o\}_{o \in O}$ as above; as before, L is well-defined and consistent with D, and $L_i = [\![T_i]\!]^{\perp}_{\varrho}$ for all $i \in I$. Thus, as before, there exists a unique Q such that $Q|_{\sigma_i} = L_i$ for all $i \in I$. Then $Q = N|_t$ must hold, since $(N|_t)|_{\sigma_i} = N|_{\sigma_i;t} = N|_{t_i;\tau_i} = N_i|_{t_i} = L_i$ and Q is unique. Thus $[\![T]\!]^{\perp}_{\varrho} = \bigoplus_{i \in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho}) = \bigoplus_{i \in I} \sigma_i(L_i) = Q = N|_t$, as required by (*).

Suppose that the **or**-clause holds.

Thus $\gamma : \Delta \to \Sigma'$, $\iota : \Gamma(\star) \to \Sigma'$ form a colimit over $\{\sigma_i : \Delta_i \to \Delta, (t_i;\tau_i) : \Delta_i \to \Gamma(\star)\}_{i \in I}$, ι is an isomorphism and $t = \gamma;\iota^{-1}$. Take, as in the **either**-case, any total ϱ χ-represented by η and find the appropriate N_i. As in the **either**-case, there exists a unique N such that $N|_{\tau_i} = N_i$ for all $i \in I$. Let $N' = N|_t$. Then $N'|_{\sigma_i} = (N|_t)|_{\sigma_i} = N|_{\sigma_i;t} = N_{\sigma_i;\gamma;\iota^{-1}} = N|_{t_i;\tau_i;\iota;\iota^{-1}} = N|_{t_i;\tau_i} = N_i|_{t_i} = [\![T_i]\!]^{\perp}_{\varrho}$. Moreover, if $N''|_{\sigma_i} = [\![T_i]\!]^{\perp}_{\varrho}$ for all $i \in I$, then, because $N|_{t_i;\tau_i} = [\![T_i]\!]^{\perp}_{\varrho}$ as well, $\mathcal{L}$ has amalgamation and (γ, ι) is a colimit, there would exist a unique N''' such that $N'''|_{\gamma} = N''$ and $N'''|_{\iota} = N$. But the latter implies $N''' = N|_{\iota^{-1}}$, which gives $N'' = N'''|_{\gamma} = N|_{\gamma;\iota^{-1}} = N'$, proving that N' is unique. Hence, $\bigoplus_{i \in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho}) = N'$. Thus, $[\![T]\!]^{\perp}_{\varrho}$ is defined, which proves **(PM)**, and equals N'.

The inclusion $\subseteq$ in (*) holds, since $[\![T]\!]^{\perp}_{\varrho} = N' = N|_t$ and N Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$.

As for $\supseteq$, we proceed as in the proof of $\supseteq$ in the **either**-case and define ϱ as is defined there. Then ϱ is total, χ-represented by η and $[\![T]\!]^{\perp}_{\varrho} = \bigoplus_{i \in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho}) = \bigoplus_{i \in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho_i}) = \bigoplus_{i \in I} \sigma_i(N_i|_{t_i}) = N|_t$, where the last equation holds since $(N|_t)|_{\sigma_i} = N|_{t_i;\tau_i} = N_i|_{t_i}$, which in turn may be inferred as follows: since (γ, ι) form a cocone, we have $\sigma_i;\gamma = t_i;\tau_i;\iota$; since $t = \gamma;\iota^{-1}$, this implies $\sigma_i;t = t_i;\tau_i$, as required.

Now assume **(PM)** or **(EPM)**. Thus, for any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation or for $\mathcal{L}_0$ with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation and isoreflection, for any environment η satisfying ζ, for any total environment ϱ χ-represented by η, $[\![T_i]\!]^{\perp}_{\varrho}$ is defined, for all $i \in I$, and either $[\![T_i]\!]^{\perp}_{\varrho} = \perp$ for some $i \in I$, or the amalgamation $\bigoplus \sigma_i([\![T_i]\!]^{\perp}_{\varrho})$ exists.

By the induction hypothesis, for all $i \in I$ we have $\zeta, \chi \vdash T_i \rhd\!\!\!\rhd \Gamma_i, t_i$ and the condition $(*_i)$ holds: $\{[\![T_i]\!]^{\perp}_{\varrho} \mid \varrho \text{ is total and } \chi\text{-represented by } \eta\} \setminus \{\perp\} =$

$\{N|_{t_i} \mid N\ \Gamma_i\text{-represents } \eta|_{\mathrm{dom}(\Gamma_i)}\}$. Let $\Gamma = \bigcup_{i\in I} \Gamma_i$ and $\{\tau_i : \Gamma_i(\star) \to \Gamma(\star)\}_{i\in I}$ be the relevant transformation. The sum Γ exists, since the signature category is $\mathcal{J}$-cocomplete.

First consider the case where **(PM)** holds.

Let $\mathcal{L}$ be a language defined by $\mathbf{Mod}(\Sigma_0) = \{k \mid k : \Sigma_0 \to \Gamma(\star)\}$, $k|_l = l; k$, and $\mathit{Spec}(\Sigma_0) = \{\top_{\Sigma_0}\}$, $[\![\top_{\Sigma_0}]\!] = \mathbf{Mod}(\Sigma_0)$. Then $\mathcal{L}$ has amalgamation: for any diagram D with colimit Θ and any family consistent with D, this family is a cocone over D. Thus, there exists a unique morphism from Θ to $\Gamma(\star)$, and this morphism is the amalgamation of the considered family.

Define $\eta(X) = \Gamma(X \to \star)$ for all $X \in \mathrm{dom}(\Gamma)$ and $\eta(X) = \perp$ for $X \in \mathrm{dom}(\zeta) \setminus \mathrm{dom}(\Gamma)$. Then η is an environment in $\mathcal{L}$ and it satisfies ζ, since, for any $X \in \mathrm{dom}(\eta)$ with $\eta(X) \neq \perp$, for any $Y \in \mathrm{dom}(\zeta(X))$, we have $\eta(Y) = \Gamma(Y \to \star) = \Gamma_i(Y \to \star); \tau_i$, where i is such that $X \in \mathrm{dom}(\Gamma_i)$, which, because Γ_i fits ζ, χ, equals $\zeta(X)(Y \to \star); \Gamma_i(X \to \star); \tau_i = \zeta(X)(Y \to \star); \Gamma(X \to \star) = \zeta(X)(Y \to \star); \eta(X) = \eta(X)|_{\zeta(X)(Y\to\star)}$. The same calculation shows that $\eta|_{\mathrm{dom}(\Gamma_i)}$ is Γ_i-represented by τ_i, for all $i \in I$.

Because $(*_i)$ holds, for all $i \in I$ there exist total ϱ_i χ-represented by η and such that $[\![T_i]\!]^{\perp}_{\varrho_i} = \tau_i|_{t_i} = t_i; \tau_i$. Let ϱ be defined so that, for non-parametric X in χ, $\varrho(X) = \eta(X)$ and, for any parametric unit X in χ, if $X \in p(T_i)$ then $\varrho(X) = \varrho_i(X)$ and if $X \notin p(T)$ then $\varrho(X)$ is the unit constantly equal $\perp$. By generativity of T, this definition is correct. Also, ϱ is total and χ-represented by η. For any $i \in I$, we have, by Lemma 84, $[\![T_i]\!]^{\perp}_{\varrho} = [\![T_i]\!]^{\perp}_{\varrho_i}$. Since we assumed **(PM)**, $[\![T]\!]^{\perp}_{\varrho}$ must be defined. Denote it by N.

By runnability, either the considered amalgamation is regular, or the signature category is cocomplete.

Suppose the amalgamation is regular. Then $N = \bigoplus_{i\in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho}) = \bigoplus_{i\in I} \sigma_i(t_i; \tau_i)$, therefore $D(m_1); t_{i_1}; \tau_{i_1} = D(m_1); N|_{\sigma_{i_1}} = D(m_1); \sigma_{i_1}; N = D(m_2); \sigma_{i_2}; N = D(m_2); N|_{\sigma_{i_2}} = D(m_2); t_{i_2}; \tau_{i_2}$, which proves that the **either** condition holds.

Suppose now that the signature category is cocomplete. Then $N : \Delta \to \Gamma(\star)$ is the unique morphism such that $N|_{\sigma_i} = t_i; \tau_i$ for all $i \in I$. Thus $(N, \mathrm{id}_{\Gamma(\star)})$ is a colimit over $\{\sigma_i : \Delta_i \to \Delta,\ (t_i; \tau_i) : \Delta_i \to \Gamma(\star)\}_{i\in I}$ and $\mathrm{id}_{\Gamma(\star)}$ is an isomorphism, which proves that the **or** condition holds.

Thus, **(PM)** implies **(FS)**.

Next, consider the case where **(EPM)** holds. This case is analogous to the one considered above. Let $(\gamma : \Delta \to \Sigma', \iota : \Gamma(\star) \to \Sigma')$ be a colimit of $\{\sigma_i, (t_i; \tau_i)\}_{i\in I}$. The colimit exists, since the signature category is cocomplete.

Take any $\Gamma(\star)$-model N and let $\eta(X) = N|_{\Gamma(X\to\star)}$ for all $X \in \mathrm{dom}(\Gamma)$ and $\eta(X) = \perp$ for $X \in \mathrm{dom}(\zeta) \setminus \mathrm{dom}(\Gamma)$.

Let $N_i = N|_{\tau_i}$. Note that N_i Γ_i-represents $\eta|_{\mathrm{dom}(\Gamma_i)}$, since $\eta(X) = N|_{\Gamma(X\to\star)} = N|_{\Gamma_i(X\to\star);\tau_i} = N_i|_{\Gamma_i(X\to\star)}$, for all $X \in \mathrm{dom}(\Gamma_i)$. Because $(*_i)$

holds, for all $i \in I$ there exist total environments ϱ_i χ-represented by η and such that $[\![T_i]\!]^{\perp}_{\varrho_i} = N_i|_{t_i} = N|_{t_i;\tau_i}$. Let ϱ be defined so that, for non-parametric X in χ, $\varrho(X) = \eta(X)$ and, for any parametric unit X in χ, if $X \in p(T_i)$ then $\varrho(X) = \varrho_i(X)$ and if $X \notin p(T)$ then $\varrho(X)$ is the unit constantly equal $\perp$. By generativity of T, this definition is correct. Also, ϱ is total and χ-represented by η. For any $i \in I$, we have, by Lemma 84, $[\![T_i]\!]^{\perp}_{\varrho} = [\![T_i]\!]^{\perp}_{\varrho_i}$. Since we assumed **(EPM)**, $[\![T]\!]^{\perp}_{\varrho}$ must be defined. Denote it by Q.

Let $M = \gamma(Q) \oplus \iota(N)$. The amalgamation exists, since (γ, ι) is a colimit over $\{\sigma_i : \Delta_i \to \Delta,\ (t_i;\tau_i) : \Delta_i \to \Gamma(\star)\}_{i \in I}$, $\mathcal{L}_0$ has amalgamation and $Q|_{\sigma_i} = \left(\bigoplus_{i \in I} \sigma_i([\![T_i]\!]^{\perp}_{\varrho_i})\right)|_{\sigma_i} = [\![T_i]\!]^{\perp}_{\varrho_i} = N|_{t_i;\tau_i}$. Then $M|_{\iota} = N$. Moreover, M is unique with this property. For if $M'|_{\iota} = N$, then $(M'|_{\gamma})|_{\sigma_i} = M'|_{\sigma_i;\gamma} = M'|_{t_i;\tau_i;\iota} = (M'|_{\iota})|_{t_i;\tau_i} = N|_{t_i;\tau_i} = [\![T_i]\!]^{\perp}_{\varrho_i}$, since (γ, ι) is a cocone. But this would imply that $M'|_{\gamma} = Q$, by the uniqueness of amalgamation. But $M'|_{\gamma} = Q$ and $M'|_{\iota} = N$ implies $M = M'$.

Thus, it has been proved that, for any $\Gamma(\star)$-model N, there exists exactly one Σ'-model M such that $M|_{\iota} = N$. In other words, $\_|_{\iota}$ is a bijection. Since $\mathcal{L}_0$ has iso-reflection, this implies that ι is an isomorphism. This proves that the **or**-condition holds.

Thus **(EPM)** implies **(FS)**.

The **fourth** case, namely $T = T_0|_{\sigma:\Delta_0 \to \Delta}$, is relatively simple and may therefore be omitted. □

Lemma 27. *Let $\mathcal{L}_s$ be a static language.*

If UDS is generative, $[UDS]$ is defined in $\mathcal{L}_s$ and fully static checking is runnable on UDS in $\mathcal{L}_s$ and $\mathcal{J}$, then the following conditions are equivalent:

(FS) $\vdash$ *UDS* $\rhd\!\!\!\rhd \zeta, \chi$ *in $\mathcal{L}_s$ and $\mathcal{J}$,*

(PM) *for any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation, $C = [\![|UDS|]\!]^{\perp}$ is defined in $\mathcal{L}$.*

Moreover, if **(FS)** *holds and $\mathcal{L}$ and C are as in* **(PM)***, then C is total and $\overline{p}(UDS)$-volatile and ζ, χ describes C.*

If the signature category of $\mathcal{L}_s$ is cocomplete and $\mathcal{L}_0$ is a language with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation and iso-reflection, then **(FS)** *and* **(PM)** *are equivalent to:*

(EPM) *the context $C = [\![|UDS|]\!]^{\perp}$ is defined in $\mathcal{L}_0$.*

Proof. Obviously, **(PM)** implies **(EPM)**. We prove, first, that **(FS)** implies **(PM)** and that the "moreover" statement then holds and, second, that **(EPM)** or **(PM)** implies **(FS)**. The proof is by induction over the structure of *UDS*.

The **first** case is $UDS = \epsilon$. Then **(FS)**, **(PM)** and **(EPM)** all hold, with $\zeta = \chi = \emptyset$ and $C = \{\emptyset\}$. Of course, C is total. Also, C is $\overline{p}(UDS)$-volatile, since $\overline{p}(UDS) = \emptyset$. Note that $np(C) = \{\emptyset\}$. At the same time, the only η satisfying ζ is the empty function, as is the only environment ϱ χ-represented by η; hence, the set of all $np(\varrho)$ for such environments ϱ is equal $\{\emptyset\}$ as well, which proves the "moreover" part of the lemma.

The **second** case is $UDS = UDS_0\ X : USP$. Assume that **(FS)** holds and take any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation. Then we have $\vdash UDS_0 \rhd\!\!\rhd \zeta_0, \chi_0$, whereby, if USP is non-parametric, then $\zeta = \zeta_0[\{\star : sig[USP]\}/X]$ and $\chi = \chi_0[\mathrm{id}_{sig[USP]}/X]$, and if it is parametric, then $\zeta = \zeta_0$ and $\chi = \chi_0[sig[USP]/X]$.

By the induction hypothesis, $C_0 = [\![| UDS_0 |]\!]^{\perp}$ is defined in $\mathcal{L}$, total, $\overline{p}(UDS_0)$-volatile and described by ζ_0, χ_0. Hence, $C = [\![| UDS |]\!]^{\perp}$ is defined as well, which proves **(PM)**, and it is equal $C_0[[\![\top_{sig[SP]}]\!]_{\perp}/X]$ if $USP = SP$ and $C_0[[\![\top_{sig[SP]} \to \top_{sig[SP']}]\!]_{\perp}/X]$ if $USP = SP \to SP'$.

Obviously, C is total, since C_0 is and, if USP is parametric, then the unit added under the name X is total. The context C is also $\overline{p}(UDS)$-volatile. If X is non-parametric, then this follows from $\overline{p}(UDS) = \overline{p}(UDS_0)$ and C_0 being $\overline{p}(UDS_0)$-volatile. If X is parametric, the matter is slightly more complicated. Take any $Y \in \overline{p}(UDS)$, any $\varrho' \in C$ and any total $F \in \mathbf{Unit}_{\perp}(sig[USP])$. Let ϱ'_0 be ϱ' with X removed from its domain; of course, $\varrho'_0 \in C_0$. Now, either $Y \in \overline{p}(UDS_0)$ or $Y = X$. In the former case, since C_0 is $\overline{p}(UDS_0)$-volatile, $\varrho'_0[F/Y] \in C_0$; hence, $\varrho'[F/Y] \in C$. In the latter case, $F \in [\![\top_{sig[SP]} \to \top_{sig[SP']}]\!]_{\perp}$, where $USP = SP \to SP'$, and so we have $\varrho' = \varrho'_0[F/Y] \in C$.

Finally, we prove that ζ, χ describes C.

Take any $\varrho' \in C$. Then $\varrho' = \varrho'_0[F/X]$, where $\varrho'_0 \in C_0$ and $F = \varrho'(X)$. By the induction hypothesis, there is an environment ϱ_0 χ_0-represented by some environment η_0 satisfying ζ_0, and such that $np(\varrho_0) = np(\varrho'_0)$. We have two cases. If X is parametric, then take $\eta = \eta_0$ and $\varrho = \varrho_0[F/X]$. Since $\zeta = \zeta_0$, η satisfies ζ. Also, ϱ is χ-represented by η, since ϱ_0 was χ_0-represented by η and X is parametric. Of course, $np(\varrho) = np(\varrho_0) = np(\varrho'_0) = np(\varrho')$, since X is parametric. If, on the other hand, X is non-parametric, then take $\eta = \eta_0[F/X]$ and $\varrho = \varrho_0[F/X]$. We have $\zeta = \zeta_0[[\{\star : sig[USP]\}/X]$. Note that X cannot appear in any $\zeta(Y)$ for $Y \neq X$, since UDS is statically correct. Hence, η satisfies ζ, since the context $\zeta(X)$ contains no edges. Also, ϱ is χ-represented by η, since ϱ_0 is χ_0-represented by η_0 and $\varrho(X) = F = \eta(X) = \eta(X)|_{\mathrm{id}_{sig[SP]}} = \eta(X)|^{\perp}_{\chi(X)}$. So, $np(\varrho) = np(\varrho_0)[F/X] = np(\varrho'_0)[F/X] = np(\varrho')$, as required.

Now take any η satisfying ζ and let ϱ be an environment χ-represented by η. Let η_0 be equal η with X removed from the domain if X is non-parametric (if X is parametric it is not in the domain anyway). Then η_0 satisfies ζ_0. Let ϱ_0 be equal ϱ with X removed from the domain. Then ϱ_0 is χ_0-represented by η_0. By the induction hypothesis, there is an environment $\varrho_0' \in C_0$ such that $np(\varrho_0) = np(\varrho_0')$. If X is parametric, let $\varrho' = \varrho_0'[F/X]$, where F is the total unit constantly equal $\perp$; then $\varrho' \in C$ and $np(\varrho) = np(\varrho_0) = np(\varrho_0') = np(\varrho')$. If X is non-parametric, then let $\varrho' = \varrho_0'[\eta(X)/X]$. Again, $\varrho' \in C$ and, in the first case, $np(\varrho) = np(\varrho_0[\varrho(X)/X]) = np(\varrho_0)[\eta(X)|^{\perp}_{\chi(X)}/X] = np(\varrho_0')[\eta(X)/X] = np(\varrho')$. This ends the proof of the "moreover" part.

Assume now that **(PM)** or **(EPM)** holds. Then, by the induction hypothesis, $\vdash UDS_0 \rhd\!\!\!\rhd \zeta_0, \chi_0$ holds and so $\vdash UDS \rhd\!\!\!\rhd \zeta, \chi$ must hold as well, since the rules contain no additional premises.

The **third** case is $UDS = UDS_0\ X : SP$ **given**$_\Delta$ T. Assume **(FS)** holds and take any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation. Then $\vdash UDS_0 \rhd\!\!\!\rhd \zeta_0, \chi_0$ and $\zeta_0, \chi_0 \vdash T \rhd\!\!\!\rhd \Gamma, t$ and $(j : \Gamma(\star) \to \Sigma', \tau : sig[SP] \to \Sigma')$ is a pushout of $(t, \iota_{\Delta \subseteq sig[SP]})$; finally, $\zeta = \zeta_0[(\Gamma; j)/X]$ and $\chi = \chi_0[\tau/X]$.

By the induction hypothesis, $C_0 = [\![|UDS_0|]\!]^{\perp}$ is defined in $\mathcal{L}$, total, $\overline{p}(UDS_0)$-volatile and described by ζ_0, χ_0.

Take any $\varrho_0' \in C_0$. Of course, ϱ_0' is total. Since ζ_0, χ_0 describe C_0, there exists an environment ϱ_0 χ_0-represented by some η_0 satisfying ζ_0, and such that $np(\varrho_0) = np(\varrho_0')$. Note that this implies that η_0 χ_0-represents ϱ_0' as well, since the property of being χ_0-represented by η_0 depends only on the non-parametric part of an environment. By Lemma 26, $[\![T]\!]^{\perp}_{\varrho_0'}$ is defined, which proves **(PM)**, and we have

$$\{[\![T]\!]^{\perp}_{\varrho_0'} \mid \varrho_0' \text{ is total and } \chi_0\text{-represented by } \eta_0\} \setminus \{\perp\} \\ = \{M_0|_t \mid M_0\ \Gamma\text{-represents } \eta_0|_{\mathrm{dom}(\Gamma)}\}$$

Clearly, C is total, since C_0 is. We now show that C is $\overline{p}(UDS)$-volatile.

Take any $Z \in \overline{p}(UDS)$, any $\varrho' \in C$ and any total $F \in \mathbf{Unit}_{\perp}(C_s(Z))$; we will prove that $\varrho'' = \varrho'[F/Z] \in C$. We have $\varrho' = \varrho_0'[N/X]$, where $\varrho_0' \in C_0$ and $N = \varrho'(X)$. Then $\varrho'' = (\varrho_0'[F/Z])[N/X]$; by $\overline{p}(UDS_0)$-volatility of C_0 and the fact that $Z \in \overline{p}(UDS) \subseteq \overline{p}(UDS_0)$, $\varrho_0'[F/Z] \in C_0$ and so what remains to be proved is that $N \neq \perp$ implies $N|_\Delta = [\![T]\!]^{\perp}_{\varrho_0'[F/Z]}$. This in fact holds, by Lemma 84, since $N \neq \perp$ implies $N|_\Delta = [\![T]\!]^{\perp}_{\varrho_0'}$ and $Z \notin p(T)$.

Finally, we prove that C is described by ζ, χ.

Take any $\varrho' \in C$. Then $\varrho' = \varrho_0'[N/X]$, where $\varrho_0' \in C_0$, $N = \varrho'(X)$ and $N \neq \perp$ implies $N|_\Delta = [\![T]\!]^{\perp}_{\varrho_0'}$. Let η_0 and ϱ_0 be as considered above. If $N = \perp$, let $\eta = \eta_0[\perp/X]$ and $\varrho = \varrho_0[\perp/X]$. Then η satisfies ζ, χ-represents ϱ and $np(\varrho) = np(\varrho_0)[\perp/X] = np(\varrho_0')[\perp/X] = np(\varrho')$. So, assume $N \neq \perp$; then $[\![T]\!]^{\perp}_{\varrho_0'} \neq \perp$ as well. As stated earlier, there then is a model M_0 Γ-representing

$\eta_0|_{\mathrm{dom}(\Gamma)}$ and such that $[\![T]\!]^{\perp}_{\varrho'_0} = M_0|_t$. Let $\eta = \eta_0[j(M_0) \oplus \tau(N)/X]$; note that the amalgamation is defined, since $\mathcal{L}$ has amalgamation, it is performed along a pushout and $M_0|_t = [\![T]\!]^{\perp}_{\varrho'_0} = N|_{\iota_{\Delta \subseteq sig[SP]}}$. Then η satisfies ζ, because η_0 satisfies ζ_0 and $j(M_0) \oplus \tau(N)$ $\zeta(X)$-represents $\eta_0|_{\mathrm{dom}(\zeta(X))}$, since M_0 Γ-represents $\eta_0|_{\mathrm{dom}(\Gamma)}$ and $\zeta(X) = \Gamma; j$. Also, if ϱ is an environment χ-represented by η, then $\varrho = \varrho_0[\eta(X)^{\perp}_{\chi(X)}/X] = \varrho_0[(j(M_0) \oplus \tau(N))|_{\tau}/X] = \varrho_0[N/X]$. Hence, $np(\varrho) = np(\varrho_0)[N/X] = np(\varrho'_0)[N/X] = np(\varrho')$.

Now take any η satisfying ζ and any environment ϱ χ-represented by η. Let η_0 be η with X removed from the domain and ϱ_0 be ϱ with X removed from the domain. Then η_0 satisfies ζ_0 and ϱ_0 is χ_0-represented by η_0.

We now consider two cases, depending on whether $\varrho(X) = \perp$. Note that $\varrho(X) = \perp$ iff $\eta(X) = \perp$.

Assume $\varrho(X) = \perp$. By the induction hypothesis, there exists an environment $\varrho'_0 \in C_0$ such that $np(\varrho_0) = np(\varrho'_0)$. Let $\varrho' = \varrho'_0[\varrho(X)/X]$. Of course, $\varrho' \in C$. Also, $np(\varrho) = np(\varrho_0)[\varrho(X)/X] = np(\varrho'_0)[\varrho'(X)/X] = np(\varrho')$, as required.

Now assume $\varrho(X) \neq \perp$, which implies $\eta(X) \neq \perp$. Since η satisfies ζ and $\eta(X) \neq \perp$, the model $\eta(X)$ must $\zeta(X)$-represent $\eta|_{\mathrm{dom}(\zeta(X))}$. But $\zeta(X) = \Gamma; j$ and $X \notin \mathrm{dom}(\zeta(X))$. Thus $\eta(X)$ $(\Gamma; j)$-represents $\eta_0|_{\mathrm{dom}(\Gamma;j)}$. Hence, $\eta(X)|_j$ Γ-represents $\eta_0|_{\mathrm{dom}(\Gamma)}$. As has been already stated, there then exists a total environment ϱ_{00} χ_0-represented by η_0 and such that $[\![T]\!]^{\perp}_{\varrho_{00}} = (\eta(X)|_j)|_t$, which is further equal $\eta(X)|_{t;j} = \eta(X)|_{\iota_{\Delta \subseteq sig[SP]};\tau} = (\eta(X)|_{\chi(X)})|_{\Delta} = \varrho(X)|_{\Delta}$. Note that, in fact, $np(\varrho_{00}) = np(\varrho_0)$, since both environments are χ_0-represented by the same environment η_0.

By the induction hypothesis, there exists an environment $\varrho'_{00} \in C_0$ such that $np(\varrho'_{00}) = np(\varrho_{00})$, which is equal $np(\varrho_0)$. Assume $p(T)$ contains the n elements $X_1, \ldots, X_n$. Let $\varrho''_{00} = \varrho'_{00}[\varrho_{00}(X_1)/X_1] \ldots [\varrho_{00}(X_n)/X_n]$. Observe that, for any $1 \leq i \leq n$, $X_i \in \overline{p}(UDS_0)$, since UDS is generative, and that $\varrho_{00}(X_i)$ is a total unit, since ϱ_{00} is total. Hence, by $\overline{p}(UDS_0)$-volatility of C_0, we have $\varrho''_{00} \in C_0$. Also, by Lemma 84, $[\![T]\!]^{\perp}_{\varrho''_{00}} = [\![T]\!]^{\perp}_{\varrho_{00}}$, since both environments coincide on non-parametric unit names and on names from $p(T)$. Let $\varrho' = \varrho''_{00}[\varrho(X)/X]$. Then $\varrho'(X)|_{\Delta} = \varrho(X)|_{\Delta} = [\![T]\!]^{\perp}_{\varrho_{00}} = [\![T]\!]^{\perp}_{\varrho''_{00}}$. Hence, $\varrho' \in C$. It is clear that $np(\varrho) = np(\varrho_0)[\varrho(X)/X] = np(\varrho_{00})[\varrho(X)/X] = np(\varrho''_{00})[\varrho(X)/X] = np(\varrho')$, as required.

This ends the proof of the "moreover" part of the lemma.

Assume now that **(PM)** or **(EPM)** holds. Thus, in any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation or in the language $\mathcal{L}_0$ with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation and iso-reflection, the context $C = [\![|UDS|]\!]^{\perp}$ is defined This implies that $C_0 = [\![|UDS_0|]\!]^{\perp}$ is defined as well and for any $\varrho'_0 \in C_0$, $[\![T]\!]^{\perp}_{\varrho'_0}$ is defined and that $C = \{\varrho'_0[N/X] \mid \varrho'_0 \in C_0,\ N \neq \perp \text{ implies } N|_{\Delta} = [\![T]\!]^{\perp}_{\varrho'_0}\}$.

By the induction hypothesis we then know that $\vdash UDS_0 \rhd\!\!\rhd \zeta_0, \chi_0$ and C_0 is total, $\overline{p}(UDS_0)$-volatile and described by ζ_0, χ_0.

Take any total environment ϱ_0 χ_0-represented by an environment η_0 satisfying ζ_0. Because ζ_0, χ_0 describe C_0, there exists $\varrho_0' \in C_0$ such that $np(\varrho_0) = np(\varrho_0')$. Assume $p(T)$ has n elements $X_1, \ldots, X_n$ and let $\varrho_0'' = \varrho_0'[\varrho_0(X_1)/X_1] \ldots \ldots [\varrho_0(X_n)/X_n]$. Because C_0 is $\overline{p}(UDS_0)$-volatile, ϱ_0' is total and $X_1, \ldots, X_n \in \overline{p}(UDS_0)$, the declarations UDS being generative, we have $\varrho_0'' \in C_0$. Thus $[\![T]\!]^{\perp}_{\varrho_0''}$ is defined. This means that $[\![T]\!]^{\perp}_{\varrho_0}$ is defined as well, by Lemma 84, because ϱ_0 and ϱ_0'' coincide on non-parametric units and on $p(T)$. Thus, we may apply Lemma 26. By that lemma, we have $\zeta_0, \chi_0 \vdash T \rhd\!\!\rhd \Gamma, t$ for some Γ and t. Thanks to $\mathcal{J}$-cocompleteness, the required pushout (j, τ) then exists and so **(FS)** indeed holds. $\square$

Proposition 25. *Assume ASP is generative and statically correct in a static language $\mathcal{L}_s$ and that fully static checking is runnable on ASP in $\mathcal{L}_s$ and $\mathcal{J}$. Then*

$$ASP \text{ is fully statically correct in } \mathcal{L}_s \text{ and } \mathcal{J}$$

$$\textit{iff}$$

$$\text{for any language } \mathcal{L} \text{ with } (\mathcal{L})_s = \mathcal{L}_s \text{ and amalgamation,}$$

$$|ASP| \text{ is partially correct in } \mathcal{L}$$

Moreover, if the signature category of $\mathcal{L}_s$ is cocomplete and $\mathcal{L}_0$ is a specification language with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation, iso-reflection and consistent signatures, then the above statements both hold iff $|ASP|$ is partially correct in $\mathcal{L}_0$.

Proof. Assume that $\mathcal{L}_s$ is a static language, ASP is statically correct in $\mathcal{L}_s$ and generative and that fully static checking is runnable on ASP in $\mathcal{L}_s$ and $\mathcal{J}$.

Assume the declarations of ASP are UDS. Note that it suffices to prove that the top statement implies the middle one and that the middle or the bottom one imply the top one, whereby by "bottom" we mean the statement concerning languages with iso-reflection.

"$\Downarrow$": Assume ASP is fully statically correct. Then $\vdash UDS \rhd\!\!\rhd \zeta, \chi$ for some ζ, χ. By Lemma 27, the context $C = [\![|UDS|]\!]^{\perp}$ is then defined, total and described by ζ, χ.

Take any $\varrho \in C$. There exists an environment η satisfying ζ and such that ϱ is χ-represented by η. If the **result** unit of ASP is a term T, then $\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma, t$ for some Γ and t and so, by Lemma 26, $[\![T]\!]^{\perp}_{\varrho}$ is defined. Thus, $|ASP|$ is partially correct in $\mathcal{L}$. If the **result** unit is an expression $\lambda A : SP \bullet T$, then take any $N \in \mathbf{Mod}_{\perp}(sig[SP])$. Let $\eta' = \eta[N/A]$. Then

η' satisfies $\zeta[\{\star : sig[SP]\}/X]$. Also, η' $\chi[\mathrm{id}_{sig[SP]}/X]$-represents $\varrho[N/X]$. Because $\zeta[\{\star : sig[SP]\}/X], \chi[\mathrm{id}_{sig[SP]}/X] \vdash T \rhd\!\!\rhd \Gamma, t$ for some Γ and t, we have, by Lemma 26, that $[\![T]\!]^{\perp}_{\varrho[N/X]}$ is defined. By the same lemma, we know that, if $[\![T]\!]^{\perp}_{\varrho[N/X]} \neq \perp$, then it is equal $M|_t$ for some model M Γ-representing $\eta|_{\mathrm{dom}(\Gamma)}$. By fully static correctness, $A \in \mathrm{dom}(\Gamma)$ and $\Gamma(A \to \star) = \iota_{sig[SP] \subseteq \mathrm{dom}(t)}; t$. Thus, $[\![T]\!]^{\perp}_{\varrho[N/X]} \neq \perp$ implies $[\![T]\!]^{\perp}_{\varrho[N/X]}|_{\Delta} = (M|_t)|_{\Delta} = M|_{\Gamma(A \to \star)} = \eta(A) = N$. This proves that $|ASP|$ is partially correct in $\mathcal{L}$.

"$\Uparrow$": Assume that $|ASP|$ is partially correct either in any language $\mathcal{L}$ with $(\mathcal{L})_s = \mathcal{L}_s$ and amalgamation, or in some language $\mathcal{L}_0$ with $(\mathcal{L}_0)_s = \mathcal{L}_s$, amalgamation and iso-reflection. We first consider the latter case.

Let $C = [\![|UDS|]\!]^{\perp}$ in $\mathcal{L}_0$. By Lemma 27, $\vdash ASP \rhd\!\!\rhd \zeta, \chi$ and C is total, $\overline{p}(UDS)$-volatile and described by ζ, χ.

Assume the **result** unit of ASP is a term T. Take any total environment ϱ χ-represented by some η satisfying ζ. There exists an environment $\varrho' \in C$ such that $np(\varrho) = np(\varrho')$. Assume $p(T)$ contains n names $X_1, \ldots, X_n$. Let $\varrho'' = \varrho'[\varrho(X_1)/X_1] \ldots [\varrho(X_n)/X_n]$. Then $\varrho'' \in C$, because all $X_i \in \overline{p}(UDS)$, by the generativity of ASP, and C is $\overline{p}(UDS)$-volatile and all $\varrho(X_i)$, for $1 \leq i \leq n$, are total. Of course, $[\![T]\!]^{\perp}_{\varrho''}$ is defined, since $\varrho'' \in C$ and ASP is partially correct. Note that ϱ and ϱ'' coincide on all non-parametric names and on names from $p(T)$. Hence, by Lemma 84, $[\![T]\!]^{\perp}_{\varrho}$ is defined and equal $[\![T]\!]^{\perp}_{\varrho''}$. Thus, by Lemma 26, $\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma, t$ for some Γ and t. Hence, ASP is fully statically correct.

Assume the **result** unit of ASP is an expression $\lambda A : SP \bullet T$. Take any total environment ϱ $\chi[\mathrm{id}_{sig[SP]}/A]$-represented by an environment η satisfying $\zeta[\{\star : sig[SP]\}/A]$. Let ϱ_0 and η_0 be ϱ and η, respectively, with A removed from the domain. Then ϱ_0 is χ-represented by η_0 satisfying ζ. There exists an environment $\varrho'_0 \in C$ such that $np(\varrho_0) = np(\varrho'_0)$. If, as before, $p(T)$ consists of $X_1, \ldots, X_n$, one may again construct the environment $\varrho''_0 = \varrho'_0[\varrho_0(X_1)/X_1] \ldots [\varrho_0(X_n)/X_n]$ such that $\varrho''_0 \in C$ and ϱ''_0 and ϱ_0 coincide on non-parametric unit names and on names from $p(T)$. Because $|ASP|$ is partially correct, $[\![T]\!]^{\perp}_{\varrho''_0[\varrho(A)/A]}$ is defined. But, by Lemma 84, this is equal $[\![T]\!]^{\perp}_{\varrho_0[\varrho(A)/A]} = [\![T]\!]^{\perp}_{\varrho}$, which must thus also be defined. Hence, we may apply Lemma 26 and obtain $\zeta[\{\star : sig[SP]\}/A], \chi[\mathrm{id}_{sig[SP]}/A] \vdash T \rhd\!\!\rhd \Gamma, t$ for some Γ and t.

Note that, for any $\Gamma(\star)$-model M, there exists an environment η_M satisfying $\zeta[\{\star : sig[SP]\}/A]$ and such that $\eta_M|_{\mathrm{dom}(\Gamma)}$ is Γ-represented by M. This environment is defined by the rule $\eta_M(X) = M|_{\Gamma(X \to \star)}$ if $X \in \mathrm{dom}(\Gamma)$ and $\eta(X) = \perp$ if $X \in (\mathrm{dom}(\zeta) \cup \{A\}) \setminus \mathrm{dom}(\Gamma)$; it satisfies $\zeta[\{\star : sig[SP]\}/A]$ since, for any $X \in \mathrm{dom}(\eta_M)$ with $\eta_M(X) \neq \perp$, for any $Y \in \mathrm{dom}(\zeta(X))$, we have $Y \in \mathrm{dom}(\Gamma)$ and $\eta_M(Y) = M|_{\Gamma(Y \to \star)} = M|_{\zeta(X)(Y \to \star);\Gamma(X \to \star)} =$

$\eta_M(X)|_{\zeta(X)(Y\to\star)}$, since Γ fits ζ, χ.

Because $\mathcal{L}_0$ has consistent signatures, there exists a model M over the signature $\Gamma(\star)$. By Lemma 26, there exists a total environment ϱ_M $\chi[\mathrm{id}_{sig[SP]}/A]$-represented by η_M and such that $[\![T]\!]^{\perp}_{\varrho_M} = M|_t$. Since $M|_t \neq \perp$ and $|ASP|$ is partially correct in $\mathcal{L}_0$, this implies that $\perp \neq [\![T]\!]^{\perp}_{\varrho_M}|_{sig[SP]} = \varrho_M(A) = \eta_M(A)|^{\perp}_{\mathrm{id}_{sig[SP]}} = \eta_M(A)$. Thus, by definition of η_M, $A \in \mathrm{dom}(\Gamma)$.

Let ι be a coequalizer of $\Gamma(A \to \star)$ and $\iota_{sig[SP]\subseteq\mathrm{dom}(t)}; t$; a coequalizer exists, since the signature category is cocomplete. Observe that the maps taking any $\Gamma(\star)$-model M to a $sig[SP]$-model $M|_{\Gamma(A\to\star)}$ or $(M|_t)|_{sig[SP]}$ are equal, since the former is equal $\eta_M(A)$ and the latter is equal $[\![T]\!]^{\perp}_{\varrho_M}|_{sig[SP]} = \varrho_M(A) = \eta_M(A)$. Also, because of amalgamation, the map $\_|_\iota$ is their equalizer. Hence, the equalizer is a bijection, i.e., $\_|_\iota$ is a bijection. By iso-reflection, ι is an isomorphism, which implies that both signature morphisms are actually equal. This proves that ASP is fully statically correct.

We now consider the case that $|ASP|$ is partially correct for any appropriate $\mathcal{L}$. The proof is fully analogous to the proof for $\mathcal{L}_0$ with iso-reflection and consistent signatures, up to the point where we show that $A \in \mathrm{dom}(\Gamma)$.

Let the language $\mathcal{L}$ be the same as the one in Lemma 26, that is, let $\mathbf{Mod}(\Sigma_0) = \{k \mid k : \Sigma_0 \to \Gamma(\star)\}$, $k|_l = l; k$, and $Spec(\Sigma_0) = \{\top_{\Sigma_0}\}$, $[\![\top_{\Sigma_0}]\!] = \mathbf{Mod}(\Sigma_0)$. As proved there, $\mathcal{L}$ has amalgamation. Consider the $\Gamma(\star)$-model $M = \mathrm{id}_{\Gamma(\star)}$. The proof for the case of $\mathcal{L}_0$ may now be repeated for this choice of M, showing $A \in \mathrm{dom}(\Gamma)$.

When showing, further, that the required equality holds, we cannot assume the existence of a coequalizer. But still, it is true that the two considered maps are equal. If these maps are applied to $M = \mathrm{id}_{\Gamma(\star)}$, then we obtain $\mathrm{id}_{\Gamma(\star)}|_{\Gamma(A\to\star)} = (\mathrm{id}_{\Gamma(\star)}|_t)|_{sig[SP]}$. But this means precisely that $\Gamma(A \to \star) = \Gamma(A \to \star); \mathrm{id}_{\Gamma(\star)} = (t; \mathrm{id}_{\Gamma(\star)})|_{sig[SP]} = \iota_{sig[SP]\subseteq\mathrm{dom}(t)}; t$, as required. This proves that ASP is fully statically correct. $\square$

4 Proofs of Lemmas 33 and 34

Below, proofs of lemmas used in the proof of Proposition 32 are provided. In them, notions defined in Section 14 on page 93 are used.

Lemma 33. *Let $\mathcal{L}$ be a language with amalgamation and translations and a $\mathcal{J}$-cocomplete signature category.*

Assume T is generative, ζ, χ, μ is a specification context description such that $[T]_{C_s(\chi)}$ is defined in $\mathcal{L}$ and $\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma$, t holds in $(\mathcal{L})_s$ and $\mathcal{J}$.

The following conditions are equivalent:

(PC) $\zeta, \chi, \mu \vdash^f T ::_\perp \Gamma,\ t,\ \mathcal{SP}$,

(PM) *for any environment ϱ (χ,μ)-represented by an environment η satisfying ζ,μ, $[\![T]\!]^{\perp}_{\varrho}$ is defined in $\mathcal{L}$.*

Moreover, if **(PC)** *holds and η satisfies ζ,μ, then:*

$$\{[\![T]\!]^{\perp}_{\varrho} \mid \varrho \textit{ is } (\chi,\mu)\textit{-represented by } \eta\} \setminus \{\perp\} \\ = \{M|_t \mid M \in [\![\mathcal{SP}]\!]_{\perp}\ \Gamma\textit{-represents } \eta|_{\mathrm{dom}(\Gamma)}\} \quad (*)$$

Proof. In the proof we denote the signature $[T]_{C_s(\chi)}$ by Δ.

The **first** case is $T = A$. Then **(PC)** and **(PM)** must hold, with $\Gamma = \zeta(A) + \{A : \Delta, \mathrm{id}_\Delta : A \to \star\}$ and $t = \chi(A)$. Now consider (*). Take any η satisfying ζ,μ. As for "$\subseteq$", take any environment ϱ (χ,μ)-represented by η and such that $[\![T]\!]^{\perp}_{\varrho} \neq \perp$, and let $M = \eta(A)$. Then $[\![T]\!]^{\perp}_{\varrho} = \varrho(A) = \eta(A)|^{\perp}_{\chi(A)} = \eta(A)|_{\chi(A)} = M|_t$. Also, M Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$. For take any $X \in \mathrm{dom}(\Gamma)$. Either $X = A$ or $X \in \mathrm{dom}(\zeta(A))$. If $X = A$, then $\eta(A) = \eta(A)|_{\mathrm{id}_\Delta} = M|_{\Gamma(A\to\star)}$. If $X \in \mathrm{dom}(\zeta(A))$, then $\eta(X) = \eta(A)|_{\zeta(A)(X\to\star)}$, because η satisfies ζ, and this is further equal to $M|_{\Gamma(X\to\star)}$. Finally, $M \in [\![\mathcal{SP}]\!]$, since $M \neq \perp$ and $M = \eta(A) \in [\![\mu(A)]\!]_{\perp} = [\![\mathcal{SP}]\!]_{\perp}$, since η satisfies ζ,μ. As for "$\supseteq$", take any $M \in [\![\mathcal{SP}]\!]$ Γ-representing $\eta|_{\mathrm{dom}(\Gamma)}$. Let ϱ be any environment (χ,μ)-represented by η. Then $[\![T]\!]^{\perp}_{\varrho} = \varrho(A) = \eta(A)|^{\perp}_{\chi(A)} = (M|_{\Gamma(A\to\star)})|^{\perp}_{\chi(A)} = (M|_{\mathrm{id}_\Delta})|^{\perp}_{\chi(A)} = M|^{\perp}_{\chi(A)} = M|_{\chi(A)} = M|_t$, since $\eta(A) = M|_{\Gamma(A\to\star)}$, because M Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$.

The **second** case is $T = A(T_0)$. Assume **(PC)** holds. Then $\zeta,\chi,\mu \vdash T_0 ::_{\perp} \Gamma_0, t_0, \mathcal{SP}_0$, $\mu(A) = SP \to SP'$ and $[\![\mathcal{SP}_0]\!] \subseteq [\![SP]\!]$, where (j,t) is a pushout of $(t_0, \iota_{sig[SP]\subseteq\Delta})$, $j \in \mathcal{J}$, $\Gamma = \Gamma_0; j$ and $\mathcal{SP} = j(\mathcal{SP}_0) \cup \{t(SP')\}$.

Take any η satisfying ζ,μ and any ϱ (χ,μ)-represented by η. By the induction hypothesis, $[\![T_0]\!]^{\perp}_{\varrho}$ is defined and equals either $\perp$ or $M_0|_{t_0}$ for some $M_0 \in [\![\mathcal{SP}_0]\!]$ Γ_0-representing $\eta|_{\mathrm{dom}(\Gamma_0)}$; this implies that $M_0|_{t_0} \in [\![SP]\!]$. Thus $[\![T_0]\!]^{\perp}_{\varrho} = M_0|_{t_0} \in [\![SP]\!] \subseteq \mathrm{dom}(\varrho(A))$, since ϱ is (χ,μ)-represented by some environment and $\mu(A) = SP \to SP'$. Hence, $[\![T]\!]^{\perp}_{\varrho}$ is defined and equals either $\perp = [\![T_0]\!]^{\perp}_{\varrho}$ or $\varrho(A)(M_0|_{t_0})$.

As for "$\subseteq$" in (*), assume $\varrho(A)([\![T_0]\!]^{\perp}_{\varrho}) \neq \perp$; denote it by N. Then $N = \varrho(A)(M_0|_{t_0}) \neq \perp$. Let $M = j(M_0) \oplus t(N)$. The amalgamation exists, since $\mathcal{L}$ has amalgamation, (j,t) is a pushout of $(t_0, \iota_{sig[SP]\subseteq\Delta})$ and $M_0|_{t_0} = [\![T_0]\!]^{\perp}_{\varrho} = N|_{\iota_{sig[SP]\subseteq\Delta}}$, by persistency. Then $M|_t = N = [\![T]\!]^{\perp}_{\varrho}$ and M Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$, since, for any $X \in \mathrm{dom}(\Gamma)$, $X \in \mathrm{dom}(\Gamma_0)$ and $\Gamma(X \to \star) = \Gamma_0(X \to \star); j$ and so $\eta(X) = M_0|_{\Gamma_0(X\to\star)} = (M|_j)|_{\Gamma_0(X\to\star)} = M|_{\Gamma(X\to\star)}$. Also, $M \in [\![\mathcal{SP}]\!]$, since $M|_j = M_0 \in [\![\mathcal{SP}_0]\!]$ and $M|_t = N = \varrho(A)(M_0|_{t_0}) \in [\![SP']\!]$, since ϱ is (χ,μ)-represented by some environment, $\mu(A) = SP \to SP'$ and $N \neq \perp$.

As for "$\supseteq$" of (*), take any $M \in [\![\mathcal{SP}]\!]$ Γ-representing $\eta|_{\mathrm{dom}(\Gamma)}$. Let $M_0 = M|_j$ and $N = M|_t$. Then M_0 Γ_0-represents $\eta|_{\mathrm{dom}(\Gamma_0)}$. Also, $M_0 \in [\![\mathcal{SP}_0]\!]$. By the induction hypothesis, there exists an environment ϱ (χ,μ)-represented by some environment η satisfying ζ,μ, and such that $[\![T_0]\!]^{\perp}_{\varrho} = M_0|_{t_0}$. Let $\varrho' = \varrho[F/A]$, where $F \in \mathbf{Unit}_{\perp}(\chi(A))$ is constantly equal $\perp$ except that $F(M_0|_{t_0}) = N$; such an F is persistent, since $N|_{\iota_{sig[SP]\subseteq\Delta}} = M|_{\iota_{sig[SP]\subseteq\Delta};t} = M|_{t_0;j} = (M|_j)|_{t_0} = M_0|_{t_0}$. The environment ϱ' is (χ,μ)-represented by η, since ϱ was and since $N = M|_t \in [\![SP']\!]$ and $\mu(A) = SP \to SP'$. Finally, $[\![T]\!]^{\perp}_{\varrho'} = \varrho'(A)([\![T_0]\!]^{\perp}_{\varrho'}) = F([\![T_0]\!]^{\perp}_{\varrho})$, by Lemma 84, since $[\![T_0]\!]^{\perp}_{\varrho}$ is defined and, by generativity, $A \notin p(T_0)$. Further, $F([\![T_0]\!]^{\perp}_{\varrho}) = F(M_0|_{t_0}) = N = M|_t$.

Now assume that **(PM)** holds. This implies that, for any environment η satisfying ζ,μ and any environment ϱ (χ,μ)-represented by η, $[\![T_0]\!]^{\perp}_{\varrho}$ is defined. By the induction hypothesis, we then have $\zeta,\chi,\mu \vdash T_0 ::_{\perp} \Gamma_0, t_0, \mathcal{SP}_0$ for some Γ_0, t_0 and $\mathcal{SP}_0$. Assume $\mu(A) = SP \to SP'$. Since the signature category is $\mathcal{J}$-cocomplete, what remains to be proved is that $[\![\mathcal{SP}_0]\!] \subseteq [\![t_0(SP)]\!]$. Take any model $M_0 \in [\![\mathcal{SP}_0]\!]$. Let η_{M_0} be an environment such that M_0 Γ_0-represents $\eta_{M_0}|_{\mathrm{dom}(\Gamma_0)}$ and η_{M_0} satisfies ζ,μ. Such an environment exists, since it may be defined by $\eta_{M_0}(X) = M_0|_{\Gamma_0(X\to\star)}$ for $X \in \mathrm{dom}(\Gamma_0)$ and $\eta_{M_0}(X) = \perp$ for $X \in \mathrm{dom}(\zeta) \setminus \mathrm{dom}(\Gamma_0)$; such an η_{M_0} satisfies ζ, since Γ_0 fits ζ,χ, and it satisfies ζ,μ, since, for any $X \in \mathrm{dom}(\zeta)$, $M_0 \in [\![\mathcal{SP}_0]\!] \subseteq [\![(\Gamma_0(X \to \star))(\mu(X))]\!]$ and so $\eta_{M_0}(X) = M_0|_{\Gamma_0(X\to\star)} \in [\![\mu(X)]\!]$. By the induction hypothesis, there is an environment ϱ (χ,μ)-represented by η and such that $[\![T_0]\!]^{\perp}_{\varrho} = M_0|_{t_0}$. We may assume that $\mathrm{dom}(\varrho(A)) = [\![SP]\!]_{\perp}$, where $\mu(A) = SP \to SP'$. This is because, first, such a ϱ will still satisfy χ,μ and, second, the parametric unit A is never applied in T_0, by generativity, and so removing elements from its domain cannot change the value of $[\![T_0]\!]^{\perp}_{\varrho}$. Since $[\![T]\!]^{\perp}_{\varrho}$ is defined, we must have $M_0|_{t_0} = [\![T_0]\!]^{\perp}_{\varrho} \in \mathrm{dom}(\varrho(A)) = [\![SP]\!]_{\perp}$. Hence, $M_0 \in [\![t_0(SP)]\!]$, as required.

The **third** case is $T = \bigcup_{i\in I} \sigma_i(T_i)$. Assume that **(PC)** holds. We then have $\zeta,\chi,\mu \vdash^f T_i ::_{\perp} \Gamma_i,\, t_i,\, \mathcal{SP}_i$, for all $i \in I$, $\Gamma = \bigcup_{i\in I} \Gamma_i$ and a relevant transformation $\{\tau_i : \Gamma_i(\star) \to \Gamma(\star)\}_{i\in I}$. Moreover, one of the conditions **either** or **or** holds and $\mathcal{SP} = \bigcup_{i\in I} \tau_i(\mathcal{SP}_i)$.

Take any environment η satisfying ζ,μ and any ϱ (χ,μ)-represented by η. By the induction hypothesis, $[\![T_i]\!]^{\perp}_{\varrho}$ is defined and the appropriate condition $(*_i)$ holds, in particular, if $[\![T_i]\!]^{\perp}_{\varrho} \neq \perp$, then there are models $N_i \in [\![\mathcal{SP}_i]\!]$ Γ_i-representing $\eta|_{\mathrm{dom}(\Gamma_i)}$ and such that $N_i|_{t_i} = [\![T_i]\!]^{\perp}_{\varrho}$. By Lemma 26 and the fact that $\zeta,\chi \vdash T_i \rhd\!\!\rhd \Gamma_i,\, t_i$, that the environment η satisfies ζ and that ϱ is χ-represented by η, the term $[\![T]\!]^{\perp}_{\varrho}$ is defined, which proves **(PM)**.

Let $M = \bigoplus_{i\in I} \tau_i(N_i)$. Note that the amalgamation exists, since $\{\tau_i : \Gamma_i(\star) \to \Gamma(\star)\}_{i\in I}$ defines a colimit over a diagram with morphisms $\{\Gamma_i(X \to \star)\}_{i\in I, X\in\mathrm{dom}(\Gamma_i)}$, the language $\mathcal{L}$ has amalgamation and $N_{i_1}|_{\Gamma_{i_1}(X\to\star)} = \eta(X) = N_{i_2}|_{\Gamma(X\to\star)}$, for $X \in \mathrm{dom}(\Gamma_{i_1}) \cap \mathrm{dom}(\Gamma_{i_2})$, because N_i Γ_i-represents η for all

$i \in I$. Then M Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$, since $M|_{\tau_i} = N_i$ Γ_i-represents $\eta|_{\mathrm{dom}(\Gamma_i)}$. Also, $M|_t = [\![T]\!]^{\perp}_{\varrho}$, because $(M|_t)|_{\sigma_i} = (M|_{\tau_i})|_{t_i} = N_i|_{t_i} = [\![T_i]\!]^{\perp}_{\varrho}$ for all $i \in I$ and $[\![T]\!]^{\perp}_{\varrho}$ is the unique model with such a property. Finally, $M \in [\![\mathcal{SP}]\!]$, because, for all $i \in I$, $M|_{\tau_i} = N_i \in [\![\mathcal{SP}_i]\!]$ and so $M \in [\![\tau_i(\mathcal{SP}_i)]\!]$. This proves "$\subseteq$" in (*).

As for "$\supseteq$", take any model $M \in [\![\mathcal{SP}]\!]$ Γ-representing $\eta|_{\mathrm{dom}(\Gamma)}$ and define $N_i = M|_{\tau_i}$; then N_i Γ_i-represents $\eta|_{\mathrm{dom}(\Gamma_i)}$ and $N_i \in [\![\mathcal{SP}_i]\!]$. By conditions ($*_i$), there exist environments ϱ_i (χ, μ)-represented by η and such that $[\![T_i]\!]^{\perp}_{\varrho_i} = N_i|_{t_i}$. Let ϱ be defined so that, for non-parametric X, we have $\varrho(X) = \eta(X)|^{\perp}_{\chi(X)}$ and, for any parametric unit X, if $X \in p(T_i)$ then $\varrho(X) = \varrho_i(X)$, and if $X \notin p(T)$, then $\varrho(X)$ is the unit constantly equal $\perp$. This definition is correct, since by the generativity of T no X may appear in more than one T_i. Also, ϱ is (χ, μ)-represented by η. For any $i \in I$, we have, by Lemma 84, $[\![T_i]\!]^{\perp}_{\varrho} = [\![T_i]\!]^{\perp}_{\varrho_i}$, because ϱ and ϱ_i differ only on unit names which do not appear in T_i. We claim that $[\![T]\!]^{\perp}_{\varrho} = M|_t$. To see this, consider that $[\![T]\!]^{\perp}_{\varrho}|_{\sigma_i} = [\![T_i]\!]^{\perp}_{\varrho} = [\![T_i]\!]^{\perp}_{\varrho_i} = N_i|_{t_i}$ for all $i \in I$ and that $[\![T]\!]^{\perp}_{\varrho}$ is unique with such a property. But $(M|_t)|_{\sigma_i} = (M|_{\tau_i})|_{t_i} = N_i|_{t_i}$ as well. Hence, $[\![T]\!]^{\perp}_{\varrho} = M|_t$.

Now assume **(PM)**. Then, for all $i \in I$, for any environment η satisfying ζ, μ and any ϱ (χ, μ)-represented by η, $[\![T_i]\!]^{\perp}_{\varrho}$ is defined. By the induction hypothesis, we then have $\zeta, \chi, \mu \vdash^{f} T_i ::_{\perp} \Gamma_i, t_i, \mathcal{SP}_i$ for all $i \in I$. Since $\zeta, \chi \vdash T \rhd\!\!\rhd \Gamma, t$, the appropriate sum, transformation and witness or colimit must exist. Hence, **(PC)** holds.

The **fourth** case, $T = T_0|_{\sigma:\Delta_0 \to \Delta}$, is simple and we omit it. $\square$

Lemma 34. *Let $\mathcal{L}$ be a language with amalgamation and translations and a $\mathcal{J}$-cocomplete signature category.*

Assume UDS is generative, $[UDS]$ is defined in $\mathcal{L}$ and $\vdash UDS \rhd\!\!\rhd \zeta, \chi$ holds in $(\mathcal{L})_s$ and $\mathcal{J}$. The following conditions are equivalent:

(PC) $\vdash^{f} UDS ::_{\perp} \zeta, \chi, \mu,$

(PM) *the context $[\![UDS]\!]^{\perp}$ is defined in $\mathcal{L}$.*

Moreover, if **(PC)** *holds, then ζ, χ, μ and $\overline{p}(UDS)$ describe $[\![UDS]\!]^{\perp}$.*

Proof. The proof is by induction over the structure of UDS. Throughout, by C we denote the context $[\![UDS]\!]^{\perp}$.

The **first** case is $UDS = \epsilon$. Then **(PC)** and **(PM)** both hold, with $\zeta = \chi = \mu = \emptyset$ and $C = \{\emptyset\}$. Also, both sets used in the definition of description of a context are obviously equal $\{\emptyset\}$.

The **second** case is $UDS = UDS_0\ X : USP$. Assume that **(PC)** holds. Then $\vdash^{f} UDS_0 ::_{\perp} \zeta_0, \chi_0, \mu_0$, whereby, if USP is non-parametric, then $\zeta =$

$\zeta_0[\{\star : sig[USP]\}/X]$, $\chi = \chi_0[\mathrm{id}_{sig[USP]}/X]$, and $\mu = \mu_0[\{USP\}/X]$, and, if it is parametric, then $\zeta = \zeta_0$, $\chi = \chi_0[sig[USP]/X]$ and $\mu = \mu_0[USP/X]$.

By the induction hypothesis, $C_0 = [\![UDS_0]\!]^\perp$ is defined in $\mathcal{L}$ and described by ζ_0, χ_0, μ_0 and $\overline{p}(UDS_0)$. Hence, $C = [\![UDS]\!]^\perp$ is defined as well, which proves **(PM)**, and it is equal $C_0[[\![USP]\!]_\perp/X]$.

We now prove that ζ, χ, μ and $\overline{p}(UDS)$ describe C.

Take any $\varrho' \in C$. Then $\varrho' = \varrho'_0[F/X]$, where $\varrho'_0 \in C_0$ and $F \in [\![USP]\!]_\perp$. By the induction hypothesis, there exists an environment η_0 satisfying ζ_0, μ_0 and an environment ϱ_0 (χ_0, μ_0)-represented by η_0 such that, when restricted to non-parametric unit names in UDS_0 and unit names in $\overline{p}(UDS_0)$, they are equal. Let $\varrho = \varrho_0[F/X]$ and $\eta = \eta_0$ if X is parametric and $\eta = \eta_0[F/X]$ otherwise. Then η satisfies ζ, μ and (χ, μ)-represents ϱ. Moreover, ϱ' and ϱ coincide on non-parametric unit names: by the induction hypothesis, this is true for unit names declared in UDS_0, and it is obviously true for X, if X is non-parametric. They also coincide on unit names from $\overline{p}(UDS)$. Again, this is true, by the induction hypothesis, for names from $\overline{p}(UDS_0)$ and, if X is parametric, then it is obviously true for X as well.

Now take any ϱ (χ, μ)-represented by an environment η satisfying ζ, μ. Then $\eta = \eta_0$ if X is parametric and $\eta = \eta_0[F/X]$ if X is non-parametric, where η_0 satisfies ζ_0, μ_0 and, in the non-parametric case, $F \in [\![\mu(X)]\!]_\perp$. Also, $\varrho = \varrho_0[F/X]$, where ϱ_0 is (χ_0, μ_0)-represented by η_0 and, in the parametric case, $F \in [\![\mu(X)]\!]_\perp$. By the induction hypothesis, there exists an environment $\varrho'_0 \in C_0$ (χ_0, μ_0)-represented by η_0. Let $\varrho' = \varrho'_0[F/X]$. Since $\varrho'_0 \in C_0$ and, in any case, $F \in [\![\mu(X)]\!]_\perp$, we have $\varrho' \in C$, as required.

Assume now that **(PM)** holds. Then, by the induction hypothesis, we have $\vdash^f UDS_0 ::_\perp \zeta_0, \chi_0, \mu_0$ and so $\vdash^f UDS ::_\perp \zeta, \chi, \mu$ must hold as well, since the rules contain no additional premises.

The **third** case is $UDS = UDS_0\ X : SP\ \mathbf{given}_\Delta\ T$. Assume **(PC)** holds. Then $\vdash^f UDS_0 ::_\perp \zeta_0, \chi_0, \mu_0$ and $\zeta_0, \chi_0, \mu_0 \vdash^f T ::_\perp \Gamma, t, \mathcal{SP}$ and $(j : \Gamma(\star) \to \Sigma', \tau : sig[SP] \to \Sigma')$ is a pushout of $(t, \iota_{\Delta \subseteq sig[SP]})$; finally, $\zeta = \zeta_0[(\Gamma; j)/X]$, $\chi = \chi_0[\tau/X]$ and $\mu = \mu_0[j(\mathcal{SP}) \cup \{\tau(SP)\}/X]$.

By the induction hypothesis, $C_0 = [\![UDS_0]\!]^\perp$ is defined in $\mathcal{L}$ and described by ζ_0, χ_0, μ_0 and $\overline{p}(UDS_0)$.

Take any $\varrho'_0 \in C_0$. There exists an environment ϱ_0 (χ_0, μ_0)-represented by an environment η_0 satisfying ζ_0, μ_0 and such that ϱ'_0 and ϱ_0 coincide on non-parametric unit names and on unit names from $\overline{p}(UDS_0)$. By Lemma 33, $[\![T]\!]^\perp_{\varrho_0}$ is defined and, if it is not equal to $\perp$, then it is equal to $M|_t$ for some $M \in [\![\mathcal{SP}]\!]$ Γ-representing η_0. Since ϱ'_0 and ϱ_0 coincide on non-parametric unit names and on names from $p(T) \subseteq \overline{p}(UDS_0)$, where the inclusion holds because of generativity of UDS, we have, by Lemma 84, that $[\![T]\!]^\perp_{\varrho'_0}$ is defined and, if not equal to $\perp$, then equal to $M|_t$. This proves **(PM)**.

We now prove that ζ, χ, μ and $\overline{p}(UDS_0)$ describe C. Note that C consists of environments of the form $\varrho_0'[N/X]$, where $\varrho_0' \in C_0$ and $N \neq \perp$ implies $N|_\Delta = [\![T]\!]^{\perp}_{\varrho_0'}$.

Take any $\varrho' \in C$; then $\varrho' = \varrho_0'[N/X]$ for some $\varrho_0' \in C_0$ and N such that $N \neq \perp$ implies $N|_\Delta = [\![T]\!]^{\perp}_{\varrho_0'}$. Let η_0 and ϱ_0 be as above. Let $\varrho = \varrho_0[N/X]$ and $\eta = \eta_0[j(M) \oplus \tau(N)/X]$ if $N \neq \perp$ (which implies $[\![T]\!]^{\perp}_{\varrho'} \neq \perp$ and, hence, that M exists) and $\eta = \eta_0[\perp/X]$ if $N = \perp$; note that the amalgamation $j(M) \oplus \tau(M)$ exists, since $\mathcal{L}$ has amalgamation, (j, τ) is a pushout and $M|_t = [\![T]\!]^{\perp}_{\varrho_0'} = N|_\Delta$. Then η satisfies ζ, μ, since η_0 satisfies ζ_0, μ_0 and, if $N \neq \perp$, then $N = M|_t$, where $M \in [\![\mathcal{SP}]\!]$ Γ-represents $\eta|_{\mathrm{dom}(\Gamma)}$; thus, for any $Y \in \mathrm{dom}(\zeta(X))$, we have $\eta(Y) = M|_{\Gamma(Y \to \star)} = ((j(M) \oplus \tau(N))|_j)|_{\Gamma(Y \to \star)} = \eta(X)|_{\Gamma(Y \to \star);j} = \eta(X)|_{(\Gamma;j)(Y \to \star)}$; at the same time, $\eta(X) \in [\![\mu(X)]\!] = [\![j(\mathcal{SP}) \cup \{\tau(SP)\}]\!]$, since $\eta(X)|_j = M \in [\![\mathcal{SP}]\!]$ and $\eta(X)|_\tau = N \in [\![SP]\!]$. Moreover, ϱ is (χ, μ)-represented by η, since ϱ_0 is (χ_0, μ_0)-represented by η_0 and $\eta(X) = \perp$ implies $\varrho(X) = \perp$ and $\eta(X) \neq \perp$ implies $\varrho(X) = N = (j(M) \oplus \tau(N))|_\tau = \eta(X)|_{\chi(X)}$. Finally, it is clear that ϱ' and ϱ coincide on non-parametric unit names and names from $\overline{p}(UDS)$, since ϱ_0' and ϱ_0 coincide on non-parametric unit names and names from $\overline{p}(UDS_0)$ and $\overline{p}(UDS) \subseteq \overline{p}(UDS_0)$ and $\varrho'(X) = N = \varrho(X)$.

Now take any ϱ (χ, μ)-represented by some η satisfying ζ, μ. Let η_0 be η with X removed from the domain and ϱ_0 be ϱ with X removed from the domain. Then η_0 satisfies ζ_0, μ_0 and ϱ_0 is (χ_0, μ_0)-represented by η_0. Let $N = \varrho(X)$.

We now consider two cases, depending on whether $N = \perp$. Note that $N = \perp$ iff $\eta(X) = \perp$.

Assume $N = \perp$. By the induction hypothesis, there exists an environment $\varrho_0' \in C_0$ which coincides with ϱ_0 on non-parametric unit names and names from $\overline{p}(UDS_0)$. Let $\varrho' = \varrho_0'[N/X]$. Of course, $\varrho' \in C$ and ϱ' and ϱ coincide on non-parametric unit names and on names from $\overline{p}(UDS) \subseteq \overline{p}(UDS_0)$.

Now assume $N \neq \perp$, which implies $\eta(X) \neq \perp$. Since η satisfies ζ, μ and $\eta(X) \neq \perp$, the model $\eta(X)$ must $\zeta(X)$-represent $\eta|_{\mathrm{dom}(\zeta(X))}$. But $\zeta(X) = \Gamma; j$ and $X \notin \mathrm{dom}(\zeta(X))$. Thus $\eta(X)$ $(\Gamma; j)$-represents $\eta_0|_{\mathrm{dom}(\Gamma;j)}$. Hence, $\eta(X)|_j$ Γ-represents $\eta_0|_{\mathrm{dom}(\Gamma)}$. Moreover, $\eta(X)|_j \in [\![\mathcal{SP}]\!]$, since $\eta(X) \in [\![\mu(X)]\!] = [\![j(\mathcal{SP}) \cup \{\tau(SP)\}]\!]$, because η satisfies ζ, μ. Thus, there exists an environment ϱ_{00} (χ_0, μ_0)-represented by η_0 and such that $[\![T]\!]^{\perp}_{\varrho_{00}} = (\eta(X)|_j)|_t$, which is further equal $\eta(X)|_{t;j} = \eta(X)|_{\iota_{\Delta \subseteq sig[SP]};\tau} = (\eta(X)|_{\chi(X)})|_\Delta = \varrho(X)|_\Delta$. Note that, in fact, ϱ_{00} and ϱ_0 coincide on non-parametric unit names, since both are (χ_0, μ_0)-represented by the same environment η_0. We may also assume that they coincide on unit names from $\overline{p}(UDS)$, since these names could not have been used in T and so, by Lemma 84, do not influence the semantics of the term T.

By the induction hypothesis, there exists an environment $\varrho'_0 \in C_0$ which coincides on non-parametric unit names and on names from $\overline{p}(UDS_0)$ with ϱ_{00}. Note that, by Lemma 84, we have $[\![T]\!]^{\perp}_{\varrho'_0} = [\![T]\!]^{\perp}_{\varrho_{00}}$, which is, as already stated, equal $[\![T]\!]^{\perp}_{\varrho_0}$. Also, ϱ'_0 and ϱ_0 coincide on non-parametric unit names and on names from $\overline{p}(UDS) \subseteq \overline{p}(UDS_0)$. Let $\varrho' = \varrho'_0[N/X]$. Then $\varrho'(X)|_\Delta = N|_\Delta = \varrho(X)|_\Delta = [\![T]\!]^{\perp}_{\varrho_0} = [\![T]\!]^{\perp}_{\varrho'_0}$. Hence, $\varrho' \in C$. Also, ϱ' and ϱ coincide on non-parametric unit names, including X, and on names from $\overline{p}(UDS)$.

This ends the proof of the "moreover" part of the lemma.

Assume now that **(PM)** holds. Thus, the context $C = [\![UDS]\!]^{\perp}$ is defined. This implies that $C_0 = [\![UDS_0]\!]^{\perp}$ is defined as well and, for any $\varrho'_0 \in C_0$, $[\![T]\!]^{\perp}_{\varrho'_0}$ is defined and that $C = \{\varrho'_0[N/X] \mid \varrho'_0 \in C_0,\ N \neq \perp \text{ implies } N|_\Delta = [\![T]\!]^{\perp}_{\varrho'_0}\}$. By the induction hypothesis, we then know that $\vdash^f UDS_0 ::_\perp \zeta_0, \chi_0, \mu_0$ and C_0 is described by ζ_0, χ_0, μ_0 and $\overline{p}(UDS_0)$.

Take any environment ϱ_0 (χ_0, μ_0)-represented by an environment η_0 satisfying ζ_0, μ_0. Because ζ_0, χ_0, μ_0 and $\overline{p}(UDS_0)$ describe C_0, there exists $\varrho'_0 \in C_0$ such that ϱ'_0 and ϱ_0 coincide on non-parametric unit names and on names from $\overline{p}(UDS_0)$. By assumption, $[\![T]\!]^{\perp}_{\varrho'_0}$ is defined. By Lemma 84 and the fact that $p(T) \subseteq \overline{p}(UDS_0)$, it is equal $[\![T]\!]^{\perp}_{\varrho_0}$, which is thus defined as well. By Lemma 33, this implies that $\zeta_0, \chi_0, \mu_0 \vdash^f T ::_\perp \Gamma,\ t,\ \mathcal{SP}$, for some Γ, t and $\mathcal{SP}$. Since the signature category of $\mathcal{L}$ is $\mathcal{J}$-cocomplete, the required pushout also exists. Hence, **(PC)** holds. □

Bibliography

[ABH$^+$04] David Aspinall, Lennart Beringer, Martin Hofmann, Hans-Wolfgang Loidl, and Alberto Momigliano. A program logic for resource verification. In K. Slind, A. Bunker, and G. Gopalakrishnan, editors, *TPHOLs'04: Theorem Proving in Higher Order Logics, 17th International Conference, Park City, Utah, USA, 2004, Proceedings*, LNCS Vol. 3223, pages 34–49. Springer, 2004.

[ABK$^+$02] Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella, and Andrzej Tarlecki. CASL: The Common Algebraic Specification Language. *Theoretical Computer Science*, 286(2):153–196, 2002.

[ABL96] J.-R. Abrial, E. Börger, and H. Langmaack, editors. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, LNCS Vol. 1165. Springer, 1996.

[AG97] Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, 1997.

[AKKB99] E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors. *Algebraic Foundations of Systems Specification.* IFIP State-of-the-Art Reports. Springer, 1999.

[BCH99] Michel Bidoit, Maria Victoria Cengarle, and Rolf Hennicker. Proof systems for structured specifications and their refinements. In Astesiano et al. [AKKB99], chapter 11.

[BG77] Rod Burstall and Joseph Goguen. Putting theories together to make specifications. In *Proc. 5th Intl. Joint Conference on Artificial Intelligence, Cambridge, Mass. (USA)*, pages 1045–1058, 1977.

[BG80] Rod Burstall and Joseph Goguen. The semantics of Clear, a specification language. In *Proc. Copenhagen Winter School on Abstract Software Specification*, LNCS Vol. 86, pages 292–332. Springer, 1980.

[BM04] Michel Bidoit and Peter D. Mosses. Casl *User Manual: Introduction to Using the Common Algebraic Specification Language.* LNCS Vol. 2900 (IFIP Series). Springer, 2004. With chapters by Till Mossakowski, Donald Sannella, and Andrzej Tarlecki.

[Bor02] Tomasz Borzyszkowski. Logical systems for structured specifications. *Theoretical Computer Science*, 286(2):197–245, 2002.

[BST98] Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Architectural specifications in Casl. In A. M. Haeberer, editor, *Algebraic Methodology and Software Technology, 7th International Conference, AMAST'98, Amazonia, Brazil, January 1999, Proceedings*, LNCS Vol. 1548, pages 341–357. Springer, 1998. An extended and improved version is [BST02a].

[BST02a] Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Architectural specifications in Casl. *Formal Aspects of Computing*, 13:252–273, 2002.

[BST02b] Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Global development via local observational construction steps. In K. Diks and W. Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, Proceedings*, LNCS Vol. 2420, pages 1–24. Springer, 2002.

[CoF04a] CoFI Language Design Group. Casl summary. In Casl Reference Manual [CoF04b], part I. Edited by B. Krieg-Brückner and P. D. Mosses.

[CoF04b] CoFI (The Common Framework Initiative). Casl *Reference Manual: The Complete Documentation of the Common Algebraic Specification Language.* LNCS Vol. 2960 (IFIP Series). Springer, 2004.

[FJ90] John S. Fitzgerald and Cliff B. Jones. Modularizing the formal description of a database system. In D. Bjørner, C. A. R. Hoare, and H. Langmaack, editors, *VDM Europe*, LNCS Vol. 428, pages 189–210. Springer, 1990.

[FL97] José-Luiz Fiadeiro and Antonia Lopes. Semantics of architectural connectors. In M. Bidoit and M. Dauchet, editors, *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, Lille, France, Proceedings*, LNCS Vol. 1214, pages 505–519. Springer, 1997.

[GB92] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39:95–146, 1992.

[GR02] Joseph Goguen and Grigore Rosu. Institution morphisms. *Formal Aspects of Computing*, 13:274–307, 2002.

[Hal78] Tom E. Hall. Representation extension and amalgamation for semigroups. *Quarterly Journal of Mathematics Oxford*, 29(2):309–334, 1978.

[HI67] John M. Howie and John R. Isbell. Epimorphisms and dominions II. *Journal of Algebra*, 6:7–21, 1967.

[Hig90] Peter M. Higgins. A short proof of Isbell's zigzag theorem. *Pacific Journal of Mathematics*, 144(1):47–50, 1990.

[Hig92] Peter M. Higgins. *Techniques of Semigroup Theory.* Oxford University Press, 1992.

[Hof01] Piotr Hoffman. Verifying architectural specifications. In M. Cerioli and G. Reggio, editors, *Recent Trends in Algebraic Development Techniques, 15th International Workshop, WADT 2001, Joint with the CoFI WG Meeting, Genova, Italy, 2001, Selected Papers*, LNCS Vol. 2267, pages 152–175. Springer, 2001.

[Hof03] Piotr Hoffman. Verifying generative CASL architectural specifications. In Wirsing et al. [WPH03], pages 233–252.

[How68] John M. Howie. Commutative semigroup amalgams. *Journal of the Australian Mathematical Society*, 8:609–630, 1968.

[How96] John M. Howie. *Fundamentals of Semigroup Theory.* London Mathematical Society Monographs, New Series, No. 12. Oxford University Press, 1996.

[Isb66] John R. Isbell. Epimorphisms and dominions. In *Conference on Categorical Algebra, La Jolla, California, 1965, Proceedings*, pages 232–246. Springer, 1966.

[KHT+01] Bartek Klin, Piotr Hoffman, Andrzej Tarlecki, Lutz Schröder, and Till Mossakowski. Checking amalgamability conditions for CASL architectural specifications. In J. Sgall, A. Pultr, and P. Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001, Marianske Lazne, Czech Republic, Proceedings*, LNCS Vol. 2136, pages 451–463. Springer, 2001.

[Lei04] Tom Leinster. *Higher Operads, Higher Categories.* London Mathematical Society Lecture Notes Series Vol. 298. Cambridge University Press, 2004.

[Mac71] Saunders MacLane. *Categories for the Working Mathematician.* Springer, 1971.

[MHAH04] Till Mossakowski, Piotr Hoffman, Serge Autexier, and Dieter Hutter. CASL logic. In CASL Reference Manual [CoF04b], part IV. Edited by T. Mossakowski.

[MHST03] Till Mossakowski, Anne Haxthausen, Donald Sannella, and Andrzej Tarlecki. CASL, the Common Algebraic Specification Language: Semantics and proof theory. *Computing and Informatics*, 22:285–321, 2003.

[Mos98] Till Mossakowski. Colimits of order-sorted specifications. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97, Tarquinia, Italy, 1997, Selected Papers*, LNCS Vol. 1376, pages 316–332. Springer, 1998.

[Mos00] Till Mossakowski. Specification in an arbitrary institution with symbols. In D. Bert, C. Choppy, and P. D. Mosses, editors, *Recent Trends in Algebraic Development Techniques, 14th International Workshop, WADT'99, Château de Bonas, France, 1999, Selected Papers*, LNCS Vol. 1827, pages 252–270. Springer, 2000.

[Mos02] Till Mossakowski. Relating CASL with other specification languages: The institution level. *Theoretical Computer Science*, 286:367–475, 2002.

[Mos03] Till Mossakowski. Foundations of heterogeneous specification. In Wirsing et al. [WPH03], pages 359–375.

[MST04] Till Mossakowski, Donald Sannella, and Andrzej Tarlecki. A simple refinement language for CASL. In J.-L. Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques, 17th International Workshop, WADT 2004, Barcelona, Spain, 2004, Revised Selected Papers*, LNCS Vol. 3423, pages 162–185. Springer, 2004. To appear.

[Ore99] Fernando Orejas. Structuring and modularity. In Astesiano et al. [AKKB99], chapter 6.

[Pau96] Lawrence C. Paulson. *ML for the Working Programmer*. Cambridge University Press, second edition, 1996.

[PCW85] David L. Parnas, Paul C. Clements, and David M. Weiss. The modular structure of complex systems. *IEEE Transactions on Software Engineering*, 11(3):259–266, 1985.

[Phi74] J. M. Philip. A proof of Isbell's zigzag theorem. *Journal of Algebra*, 32:328–331, 1974.

[Sch90] Oliver Schoett. Behavioural correctness of data representations. *Science of Computer Programming*, 14(1):43–57, 1990.

[SMT01a] Lutz Schröder, Till Mossakowski, and Andrzej Tarlecki. Amalgamation in CASL via enriched signatures. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, Proceedings*, LNCS Vol. 2076, pages 993–1004. Springer, 2001.

[SMT+01b] Lutz Schröder, Till Mossakowski, Andrzej Tarlecki, Piotr Hoffman, and Bartek Klin. Semantics of architectural specifications in CASL. In H. Hussmann, editor, *Fundamental Approaches to Software Engineering, 4th International Conference, FASE 2001, Genova, Italy, Proceedings*, LNCS Vol. 2029, pages 253–268. Springer, 2001.

[SMT+05] Lutz Schröder, Till Mossakowski, Andrzej Tarlecki, Bartek Klin, and Piotr Hoffman. Amalgamation in the semantics of CASL. *Theoretical Computer Science*, 331(1):215–247, 2005. extends [SMT+01b, SMT01a].

[SST92] Donald Sannella, Stefan Sokołowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Parameterisation revisited. *Acta Informatica*, 29(8):689–736, 1992.

[ST87] Donald Sannella and Andrzej Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Sciences*, 34(2/3):150–178, 1987.

[ST88a] Donald Sannella and Andrzej Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76(2/3):165–210, 1988.

[ST88b] Donald Sannella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica*, 25(3):233–281, 1988.

[ST96] Donald Sannella and Andrzej Tarlecki. Mind the gap! abstract versus concrete models of specifications. In W. Penczek and A. Szałas, editors, *Mathematical Foundations of Computer Science 1996, 21st International Symposium, MFCS'96, Cracow, Poland, September 2-6, 1996, Proceedings*, LNCS Vol. 1113, pages 114–134. Springer, 1996.

[ST97] Donald Sannella and Andrzej Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, 9(3):229–269, 1997.

[Tar96] Andrzej Tarlecki. Moving between logical systems. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specification, 11th Workshop on Specification of Abstract Data Types Joint with the 8th COMPASS Workshop, Oslo, Norway, September 19-23, 1995, Selected Papers*, LNCS Vol. 1130, pages 478–502. Springer, 1996.

[Tar99] Andrzej Tarlecki. Institutions: An abstract framework for formal specifications. In Astesiano et al. [AKKB99], chapter 4.

[Tar00] Andrzej Tarlecki. Towards heterogeneous specifications. In *Frontiers of Combining Systems 2, 1998*, pages 337–360. Research Studies Press, 2000.

[Tar03] Andrzej Tarlecki. Abstract specification theory: An overview. In M. Pizka and M. Broy, editors, *Models, Algebras and Logic* *of Engineering Software*, NATO Science Series: Computer & Systems Sciences Vol. 191, pages 43–79. IOS Press, 2003.

[WPH03] M. Wirsing, D. Pattinson, and R. Hennicker, editors. *Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT 2002, Frauenchiemsee, Germany, 2002, Revised Selected Papers*, LNCS Vol. 2755. Springer, 2003.

Index