University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Grzegorz Marczyński

Specifications of Software Architectures using Diagrams of Constructions

PhD dissertation

Supervisor

prof. dr hab. Andrzej Tarlecki

Institute of Informatics
University of Warsaw

September 2014

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

September 22, 2014	.....
date	*Grzegorz Marczyński*

Supervisor's declaration:
the dissertation is ready to be reviewed

September 22, 2014	.....
date	*prof. dr hab. Andrzej Tarlecki*

Abstract

Formal methods promise the ultimate quality of software artifacts with mathematical proof of their correctness. Algebraic specification is one of such methods, providing formal specifications of system components suitable for verification of correctness of all individual steps in the software development process, and hence of the entire development process and of the resulting program.

In this thesis we propose a new approach to algebraic specifications of software architectures, called diagrams of construction specifications. Constructions, as introduced here, model parameterised modules, with dependency relation captured directly on signature symbols. They give a uniform treatment of first- and higher-order parameterisation, and are equipped with a single sum operation which subsumes the most standard operations on parameterised modules, We introduce specifications for such constructions, study their compositionality properties, and define a notion of refinement for constructor specifications. Diagrams of construction specifications capture design and development of modular software architecture, based on decomposition and refinement of construction specifications.

Throughout the thesis we illustrate new concepts and problems discussed by means of simple examples; a somewhat longer example is also added to summarize our presentation.

To Anne, Ada and Ernest

Acknowledgments

This work would have never been accomplished without great help of other people.

First of all, I would like to thank Andrzej Tarlecki. Thank you, Andrzej, for your patience, guidance and constant support!

I am also grateful to other people from MIMUW, especially Artur Zawłocki. Thank you, Artur, for endless discussions, common work and your insightful explanations. Thank you, Aleksy, Jacek, Patryk, and other participants of SLIWOWICA seminar.

Many thanks go to my family. Thank you, Anne, for your support and understanding! Thank you, Mom and Dad, for believing in me, Thank you, Kasia and Julia, for asking questions.

Finally, I feel indebted to great friends : Ania, Antek, Bartek, Diane, Donald, Ewa, Magda, Małgosia, Marion, Marta, Mateusz, Paweł, Rafał, Regis, Rudy and others.

Thank you all nice people!

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Aims and Objectives 3

1.3 Contributions 4

1.4 Outline 6

2 Related Work **7**

2.1 Introduction 7

2.2 Clear and Parameterised Programming 8

2.3 Modules in ACT TWO 8

2.4 ASL and Parametric Algebras 9

2.5 CASL and Architectural Specifications 9

2.6 Software Architectures 10

3 Technical Preliminaries and Assumptions **13**

3.1 Introduction 13

3.2 Basic Category Theory 13

3.2.1 Categories of Ordered Sets 16

3.3 Institutions and Specifications 17

3.3.1 Universal Algebra and Examples of Institutions 19

3.4 Parameterisation 22

3.5 Assumptions 23

4 Signature Fragments **27**

4.1 Introduction 27

4.2 Basic Approach 28

4.3 Extended Approach 30

4.3.1 Special Fragments 34

4.A Appendix: Proofs 36

5 Signatures with Dependencies **39**
5.1 Introduction . . . 39
5.2 Signatures with Dependencies . . . 39
5.2.1 Dependency Relation . . . 39
5.2.2 Signatures with Dependency Structure . . . 41
5.3 Fragments of Signatures with Dependencies . . . 43
5.A Appendix: Proofs . . . 48

6 Constructions **51**
6.1 Introduction . . . 51
6.2 Construction Signatures . . . 52
6.2.1 Signatures of Modules as Construction Signatures . . . 53
6.3 Construction Models . . . 57
6.3.1 Models of Modules as Construction Models . . . 60
6.4 Construction Specifications . . . 61
6.4.1 Consistency of Construction Specification . . . 67
6.4.2 Cleaning Operator . . . 69
6.4.3 Module Specifications as Construction Specifications . 70
6.5 Construction Fittings and Sum . . . 75
6.5.1 Construction Fittings and Sum of Construction Signatures . . . 75
6.5.2 Sum of Construction Models . . . 77
6.5.3 Sum of Construction Specifications . . . 78
6.5.4 Other Operations as Sum Operation . . . 83
6.A Appendix: Proofs . . . 97

7 Refinements **115**
7.1 Introduction . . . 115
7.2 Construction Signature Refinement Morphisms . . . 117
7.3 Construction Specification Refinements . . . 120
7.4 Refinement Compositionality . . . 124
7.A Appendix: Proofs . . . 128

8 Architectures as Diagrams of Constr. 143
8.1 Introduction . . . 143
8.2 Diagrams of Constructions . . . 143
8.3 Operations as Diagrams of Constructions . . . 151
8.A Appendix: Proofs . . . 153

9 Example 155
9.1 Introduction . . . 155
9.2 Travel Agency System . . . 155
9.3 Further Refinement Steps . . . 160

10 Summary 165
10.1 Future Work . . . 167

Bibliography 169

Chapter 1

Introduction

"It is not enough that we do our best; sometimes we must do what is required." Winston Churchill

1.1 Motivation

There are two main roles of software architectures (cf. [GS94]): to describe software system decomposition into components and their interconnections, and to define the system development process and its evolution. Contemporary software systems are large and complex. They are constructed against no smaller and no less complex functional and quality requirements. Typically, requirements change over time and the system evolves in response to those changes. As a result, the development process rarely matches the waterfall model, where the phases of analysis and design are followed by software development, testing and deployment. Instead, many alternative approaches to development process, like iterative, agile, extreme and prototype-based, prove to be more effective and closer to the everyday practice (cf. [CRS+11]).

Formal methods promise the ultimate quality of software artifacts by providing a mathematical proof of their correctness with respect to formally presented requirements. Since the main factor common to all alternative approaches to software development is its changing nature, support of formal methods for the development process has to offer flexibility throughout entire development life-cycle with the constant emphasis on modularisation and reuse. Unfortunately, the use of formal methods in practical software development is still limited to core components of critical systems (cf. [WLBF09]). The main reason for that situation is the higher cost of formal methods use

when compared with popular quality assurance approaches based on testing and good practice. Wider adoption of formal methods requires simplification and automation.

A formal method that we examine in this thesis is *algebraic specification*. The idea is to provide formal specifications of system components and to prove the correctness of single steps in the development process, thus, by construction, ensuring correctness of the finally composed system.

Parameterised programming (cf. [Gog96]), ACT TWO (cf. [EM90]) and CASL *architectural specifications* (cf. [ST88, Mos04]) are three representative examples of algebraic specification frameworks aiming at formal development of software systems based on modularisation and reuse.

The basic building blocks in the three approaches are (specifications of) *parameterised modules* (called *modules* in ACT TWO and *generic units* in CASL). Their parameterisation is of first-order functional type on the signature level, i.e. module specifications are signature morphisms together with specifications of parameters and specification of the result. A module realisation (implementation), may be represented as $\lambda X : SP_P \,.\, B[X] : SP_R$, where SP_P is a specification of the parameter, SP_R is a specification of the result, X is the formal parameter and B is the body of the module's realisation, which typically extends X.

All three approaches provide basic operations on modules, like composition, instantiation, enrichment and hiding. Every module operation requires additional connection between modules, be it a *view* or a *fitting morphism*. A *module expression* (or *result expression)* combines modules represented in a *module graph* (or *unit declaration and definition list*) into the resulting parameterised module. The explicit sharing resolution is required, thus the interconnections between modules in a module graph usually are non-trivial. All approaches actually define a graph of architectural decomposition of the system and they lack a higher-level specification to express architectural properties upon the graph itself.

The plurality of operations and the need for a module expression to compose a system is a source of potential confusion. The same modules connected via the same views produce different results, depending on the operation

that is prescribed to combine them. This overly complicates the specification process, where, on such a high level, the composition should be a simple operation, without additional unnecessary variation.

In the three frameworks mentioned above, all equipped with functional signature-level parameterisation, partial instantiation of modules either requires additional work or is impossible. It is also not evident whether all parameters are actually needed by a parameterised module, because there is no structure of the fine-grained (in)dependency between result symbols and parameter symbols. Thus one needs to assume that all parameters are needed and consider a system incomplete if some of them are missing, even when the missing symbol is not actually required by the result to be complete. As a consequence, also mutual and reflexive instantiations are problematic.

An extension from first-order to higher-order parameterisation, while increasing considerably the expressiveness of the language, requires a tremendous complication of syntax (cf. [ST12]), which makes such an extended framework unusable from the practical point of view.

We consider the above-presented limitations as potential pitfalls when it comes to real-life use of algebraic specification framework.

1.2 Aims and Objectives

Our goal is to define a formal specification approach to system development that uniformly represents various kinds of decomposition units and their interconnections. Given their realisations, the composition should be automatic. The impact of changes to requirements should be minimised and limited only to dependent components.

Even though we base our work on typical approach to parameterised programming (and architectural specifications) with parameterised module as the main architectural primitive, we would like to challenge the limitations of functional-type parameterisation and provide somehow more fine-grained parameterisation, on the level of symbols.

The formalism should support a top-down approach to software development by stepwise refinement and intuitive representation of software archi-

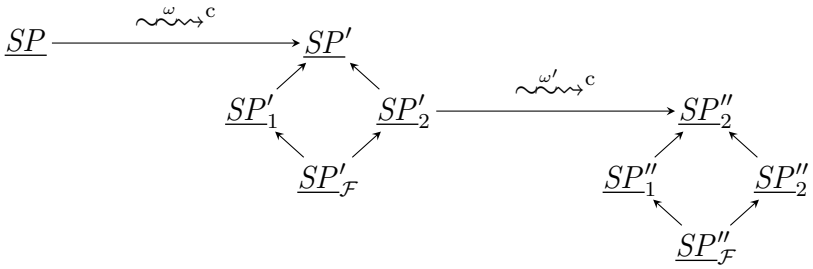


Figure 1.1: Example of a desired diagram of specifications, fittings, sums and refinements

tectures via diagrams of system building blocks, as on the schematic example diagram of module specifications in Fig. 1.1. The nodes are specifications of modules, the squares are pushouts and represent module sums, the horizontal arrows are refinements. In order to get uniform treatment of all stages of construction of the system, it is desirable that all basic composition operations be reduced to one simple sum operation.

1.3 Contributions

In this thesis we define a new approach to software architecture specifications called *diagrams of construction specifications* (or, *diagrams of constructions*, for short). *Constructions* are non-functional symbol-level parameterised modules with dependency relation provided directly on signature symbols. Constructions give uniform treatment of first- and higher-order parameterised modules and a single composition operation suffices to express different composition variants depending solely on the fitting connectives. We also define construction specifications and introduce their refinements.

Technically, construction signatures are fragments of signatures with dependency structure, [Mar12]. A signature fragment consists of a signature with a distinguished set of symbols marked as *defined* by the construction. All other symbols are considered *assumed*, i.e., expected from outside (as parameters). Construction models are simply classes of models, subject to conditions formulated on the symbol dependency from the construction signature. Constructions are inspired by constructors from [ST88] (hence the name). We define the category of construction signatures and their mor-

phisms, which makes the pushout operation applicable to construction signatures linked by a span of morphisms (a pair of morphisms with a common source).

Two constructions connected by a span of morphisms satisfying mild technical conditions may be joined by the *sum operation* that corresponds to what is usually called an application of a parameterised module (e.g. in CASL [Mos04]). However, in contrast to the typical application, the sum is a symmetric operation. Moreover, the sum operation may be used not only to compose a parameterised module and its actual parameter module, but it may also be used to sum two parameterised modules (like composition and union operations in [EM90]). This gives rise to the mutual parameterised module application where two sides are actual parameters for each other.

Construction specifications enrich construction signatures by axioms (or structured specifications build upon them). We define when a construction satisfies a construction specification, introduce sum operation for construction specifications, and show its compatibility with sums of constructions.

Construction specification refinements allow for addition of extra dependencies to symbols, therefore, reducts along such morphisms are suitable for hiding. The target specification may be stronger on defined symbols only.

As envisaged in the previous section, diagrams of constructions are homogenous with construction refinements and sum squares (like in the example from Fig. 1.1). Every diagram has a distinguished set of seed nodes, i.e., the construction specifications that need to be implemented to construct the whole system. The sharing of symbols in a diagram of constructions is explicit and every symbol has exactly one path to its definition.

Unfortunately, nice properties of constructions come at the cost of more complex and perhaps less intuitive semantics than those of parameterised modules. However, as the complications concern mainly the internal mechanics, they should not be visible to the user, hidden by a specification formalism that, by definition, takes into account requirements imposed on construction models, specifications, sums and refinements.

1.4 Outline

The thesis is organised as follows. In Chapter 2 we present related work. Then, in Chapter 3, we recall technical preliminaries and list technical assumptions for further chapters. Chapter 4 presents signature fragments. Chapter 5 defines extension of signatures by dependency relation on their symbols and introduces the concept of signature fragments with dependency structure. Chapter 6 gives formalisation of construction signatures, construction models and construction specifications; additionally, in Sect. 6.5, we introduce construction fittings and the sum operation. Chapter 7 concerns construction specification refinements. In Chapter 8 we take the concepts from the previous chapters, put them together and propose diagrams of constructions as representations of software architecture, possibly capturing also a software development process. Chapter 9 provides an example of a top-down software development process based on diagrams of constructions. Finally, Chapter 10 summarises the thesis and discusses possible future work.

Instead of having one huge appendix with proofs and additional lemmas at the end of the document, we distributed the proofs in appendices throughout the thesis as last sections of individual chapters.

Chapter 2

Related Work

2.1 Introduction

The research on *algebraic specifications* (cf. [BL69]) dated back to late 1960s. First they used single- and later *many-sorted algebras* (see [BL70]) to represent software systems (implementations) and equational logic to abstractly describe them (cf. [GTW78]). Such a rigorous approach promised the ability to mathematically prove the correctness of implementations against formal specifications prepared in advance. The task of describing the complete end-user requirements as monolithic specifications turned out to be unrealistic, thus the idea of *specification structuring and parameterisation* emerged (cf. [BG77, BG80, GB80]). However, for larger systems, specification parameterisation does not suffice, mainly due to the lack of independence of specification units. Module specifications (cf. [Par72, EM90]) and *specifications of parameterised programs* (cf. [Gog84]), also called *generic modules* (cf. [ST88]), allow for decomposition of system specifications into specifications of independent smaller units that can be refined and further decomposed separately. Such decompositions, together with the constructive connections between units are called *software architectures* (cf. [GS94, Gog96]). The typical approach to description of software architectures is to represent them as diagrams; in parameterised programming approach (cf. [Gog96]) they are called *module graphs*; in CASL (cf. [Mos04]) they are called *architectural specifications*.

In this chapter we review different approaches to module specifications and modularisation in general, including parameterisation, architectural specifications and software architectures.

2.2 Clear and Parameterised Programming

Specification language Clear, introduced in [BG77] with semantics presented in [BG80], is considered the first algebraic specification language. In [GB80], the CAT process of system implementation is envisaged as a two-dimensional process with *horizontal dimension* corresponding to the structure of the specification and *vertical dimension* corresponding to the step-wise refinement. The structuring is achieved by parameterisation and specification-building operations (extend, combine, enrich, derive, apply). The successive refinements go from the most abstract specifications on the top to the most concrete implementations at the bottom. (Loose) implementations of parameterised specifications (cf. [SW82]) compose vertically (two subsequent implementation steps may be represented as a one) and, under certain conditions, also horizontally (independently implemented actual parameter specification and parameterised specification compose and the result is an application).

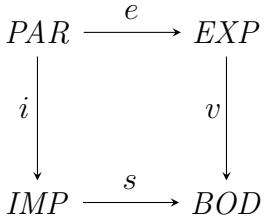
Parameterised programming, introduced in [Gog84], is a technique, inspired by parameterisation in Clear, for reliable reuse of software with *parameterised modules* as basic building blocks. Theories are used to describe formal parameters and module result. During instantiation, a view presents the actual parameter module as a formal parameter theory and, by composition, the new module is created. Parameterised programming uses *module expressions* for combination (like instantiation and sum) and modification (like restrict and renaming) of parameterised modules.

OBJ3 (cf. [GWM+92]), an executable specification language based on Clear, supports *module hierarchies* and parameterised programming.

2.3 Modules in ACT TWO

ACT TWO [EM90] is a modularisation meta-language built upon algebraic specification language ACT ONE (cf. [EM85]) with initial semantics and support for parameterisation and structuring. In ACT TWO a *module specification* consists of four components (algebraic specifications) describing the *body* of the module, its *imports*, *exports* and (shared) *parameters*. The com-

ponents are connected by specification morphisms like in the commuting diagram below.



Construction semantics of module specification is given as a composition of the free functor along s and the forgetful functor along v.

ACT TWO provides several *operations on module specifications*, including basic operations like renaming, composition, union and actualisation, and more advanced, like partial composition and recursion.

2.4 ASL and Parametric Algebras

ASL is a kernel algebraic specification language introduced in [SW83]. It offers basic specification-building operations (form, sum, derive) and behavioral operations (restrict, abstract). Moreover it includes (recursive) parameterised specifications and a flexible notion of implementation supporting both vertical and horizontal composition. Pluss (cf. [BGM89]) is a higher-level specification language based on ASL.

Investigations on the parameterisation presented in [SST92] introduce specifications of *parametric algebras* and their distinctions from parameterised specifications. As an extension to ASL, in [ST91], the authors provide higher-order parameterisation for both cases (parameterised specifications and specifications of parameterised objects).

2.5 CASL and Architectural Specifications

Casl (cf. [Mos04]) is an algebraic specification framework supporting basic algebraic specifications, generic (parameterised) specifications and structured specifications constructed using specification-building operations (translation, reduction, union, extension) or by freeness constraint.

One of the CASL layers are *architectural specifications*, a formalism for defining the composition of the system from reusable components. An architectural specification is a list of (generic) unit specifications with the *unit expression* prescribing the composition of the units to get the resulting unit. Unit expressions are built out of unit terms, translations, amalgamations (sums) and unit applications. All units may be seen as generic, i.e. parameterised (non-parameterised units are represented as parameterised units with empty parameter). A unit specification is a pair of specifications, one for the parameter and one for the result. A unit function is a map between models of the unit parameter specification and models of the unit result specification. A model of an architectural specification is a list of unit functions (for the component units) and the unit function for the result. Unit specifications may themselves be architectural specifications, what enables hierarchal decomposition of the system.

2.6 Software Architectures

Software architectures, an area of software engineering, are about structures of software system decomposition. In [GS94] a concept of *architectural style* is introduced to describe typical characteristics of groups of software architecture instances. An architectural style specifies the types of *components* and *connectors* between them together with *constraints* on the ways of their composition. An interesting formalisation of the notion of *architectural connector* from [AG97], decomposes it to *roles* (specifications) and a *glue specification* that describes how the activities of the roles are *coordinated*.

The approach to software architecture using parameterised programming (cf. Sect. 2.2) given in [Gog96] shows close correspondence of the two concepts. Parameterised modules may serve the purpose of components, views (and other connectives like inheritance, parameterisation, instantiation) represent connectors, finally module expressions define the way of system composition, which may be seen as composition constraints. An architecture as a whole is described by a *module graph* giving description of modules and relationships between them. In presence of the module graph, module expression

defines the result of the system construction. In [FLW03] the formal concept of architectural connector is given in terms known from parameterised programming.

Chapter 3

Technical Preliminaries and Assumptions

3.1 Introduction

In this chapter we present an overview of theoretical notions (mostly taken from [ST12]) used in the rest of our work. We start by presentation of the basics of category theory, further we introduce institutions, algebraic specifications and parameterisation. Finally we list assumptions that provide technical context for the chapters that follow.

3.2 Basic Category Theory

We briefly cover some of the basics of category theory. For gentle introduction for computer scientists see [ST12], for broader coverage see [ML98, AHS90].

A *category* $\mathbf{C}$ consists of a collection[1] $|\mathbf{C}|$ of $\mathbf{C}$-*objects* and for each pair $a, b \in |\mathbf{C}|$, a collection $\mathbf{C}(a, b)$ of $\mathbf{C}$-*morphisms* from a to b. For each object $a \in |\mathbf{C}|$ there exists a unique *identity morphism* $id_a \in \mathbf{C}(a, a)$. For any $a, b, c \in |\mathbf{C}|$, there is the composition operation $\_;\_ : \mathbf{C}(a, b) \to \mathbf{C}(b, c)$ such that identity morphisms are its identity elements, for $a, b \in |\mathbf{C}|$ and $f \in \mathbf{C}(a, b)$, $f;id_b = id_a;f = f$, and it is associative, for $a, b, c, d \in |\mathbf{C}|$ and $f \in \mathbf{C}(a, b)$, $g \in \mathbf{C}(b, c)$, $h \in \mathbf{C}(c, d)$, $(f;g);h = f;(g;h)$. A category is *small* if the collection of its objects and the union of the collections of its morphisms are sets. A category is *discrete* if it has only identities as morphisms.

[1] Like in [ST12], we use a neutral term *collection* to disregard the problems related to set-theoretical foundations for category theory (cf. Sect. 3.1.1.1 in [ST12]).

Notation. When $\mathbf{C}$ is clear from the context, we write *objects* and *morphisms* instead of $\mathbf{C}$-objects and $\mathbf{C}$-morphisms. We use notation $a \in \mathbf{C}$ for $a \in |\mathbf{C}|$. We write $f\colon a \to b \in \mathbf{C}$ or simply $f\colon a \to b$, for $f \in \mathbf{C}(a,b)$, $a, b \in \mathbf{C}$. For any $f\colon a \to b$, f is also called an *arrow*, object a is called the *source* or *domain*, and b the *target* or *codomain* of morphism f.

The *opposite of a category* $\mathbf{C}$, denoted by $\mathbf{C}^{op}$, is obtained by reversing the direction of arrows of $\mathbf{C}$.

A category $\mathbf{C}$ is a *subcategory* of a category $\mathbf{D}$ if $|\mathbf{C}| \subseteq |\mathbf{D}|$ and for all $a, b \in \mathbf{C}$, $\mathbf{C}(a,b) \subseteq \mathbf{D}(a,b)$, with composition and identities in $\mathbf{C}$ the same as in $\mathbf{D}$. Category $\mathbf{C}$ is a *full subcategory* of $\mathbf{D}$ if additionally $\mathbf{C}(a,b) = \mathbf{D}(a,b)$ for all $a, b \in \mathbf{C}$.

A morphism $f\colon a \to b \in \mathbf{C}$ is an *epimorphism* (or *epi*) if for all $g\colon b \to c, h\colon b \to c \in \mathbf{C}$, $f;g = f;h$ implies $g = h$. A morphism $f\colon b \to a \in \mathbf{C}$ is a *monomorphism* (or *mono*, *monic*) if for all $g\colon c \to b, h\colon c \to b \in \mathbf{C}$, $g;f = h;f$ implies $g = h$. A morphism $f\colon a \to b$ is an *isomorphism* (or *iso*) if there is a morphism $f^{-1}\colon b \to a$ such that $f;f^{-1} = id_a$ and $f^{-1};f = id_b$; the morphism $f^{-1} : b \to a$ is then called the *inverse* of f.

A *functor* $\mathbf{F}\colon \mathbf{C} \to \mathbf{D}$ from a category $\mathbf{C}$ to a category $\mathbf{D}$ is a collection of functions: a function on $\mathbf{C}$-objects, $\mathbf{F}_{obj}\colon |\mathbf{C}| \to |\mathbf{D}|$ (later called $\mathbf{F}$), and for any $a, b \in \mathbf{C}$ a function on $\mathbf{C}$-morphisms, $\mathbf{F}_{m(a,b)}\colon \mathbf{C}(a,b) \to \mathbf{D}(\mathbf{F}(a), \mathbf{F}(b))$ (later ambiguously also called $\mathbf{F}$) such that $\mathbf{F}$ preserves the identities, for any $a \in \mathbf{C}$, $\mathbf{F}(id_a) = id_{\mathbf{F}(a)}$ and it preserves the composition, for any $f\colon a \to b, g\colon b \to c \in \mathbf{C}$, $\mathbf{F}(f;g) = \mathbf{F}(f);\mathbf{F}(g)$. A *natural transformation* $\tau\colon \mathbf{F} \to \mathbf{G}$ from a functor $\mathbf{F}\colon \mathbf{C} \to \mathbf{D}$ to a functor with the same target and domain $\mathbf{G}\colon \mathbf{C} \to \mathbf{D}$ is a family of $\mathbf{D}$-morphisms, $\tau_a\colon \mathbf{F}(a) \to \mathbf{G}(a)$ for each $a \in \mathbf{C}$, such that for any $a, b \in \mathbf{C}$ and a $\mathbf{C}$-morphism $f\colon a \to b$, $\mathbf{F}(f);\tau_a = \tau_b;\mathbf{G}(f)$. For each category $\mathbf{C}$, the *identity functor* $\mathbf{Id}_{\mathbf{C}}\colon \mathbf{C} \to \mathbf{C}$ maps objects and morphisms to themselves. A functor $\mathbf{F}\colon \mathbf{C} \to \mathbf{D}$ is *faithful/full* when for any $a, b \in \mathbf{C}$ it is injective/surjective on $\mathbf{C}(a,b)$.

The *comma category* $(\mathbf{F} \downarrow \mathbf{G})$ of two functors $\mathbf{F}\colon \mathbf{C} \to \mathbf{D}$ and $\mathbf{G}\colon \mathbf{C}' \to \mathbf{D}$ has triples $\langle a, a', f\colon \mathbf{F}(a) \to \mathbf{G}(a')\rangle$ as objects, where $a \in \mathbf{C}$, $a' \in \mathbf{C}'$ and $f \in \mathbf{D}$, and, for two $(\mathbf{F} \downarrow \mathbf{G})$-objects, $\langle a, a', f\colon \mathbf{F}(a) \to \mathbf{G}(a')\rangle$ and $\langle b, b', g\colon \mathbf{F}(b) \to \mathbf{G}(b')\rangle$, has pairs $\langle h\colon a \to b, h'\colon a' \to b'\rangle$ as morphisms, where

$h \in \mathbf{C}$, $h' \in \mathbf{C}'$ are such that $\mathbf{F}(h); g = f; \mathbf{G}(h')$. Comma categories of identity functors $(\mathbf{Id_D} \downarrow \mathbf{G})$ or $(\mathbf{F} \downarrow \mathbf{ID_D})$ are denoted by $(\mathbf{D} \downarrow \mathbf{G})$ or $(\mathbf{F} \downarrow \mathbf{D})$, respectively.

A *diagram* of shape $\mathbf{J}$, where $\mathbf{J}$ is a small category, in a category $\mathbf{C}$ is a functor from the category $\mathbf{J}$ to $\mathbf{C}$, denoted by $\mathfrak{D}\colon \mathbf{J} \to \mathbf{C}$. When presenting a diagram we often omit the identity morphisms in $\mathbf{J}$. A diagram of shape $\mathbf{J}$ is *finite* if $\mathbf{J}$ is a finite category. A *morphism of diagrams* of shape $\mathbf{J}$ in a category $\mathbf{C}$ is a natural transformation between respective functors. A *constant diagram* $\Delta(n)\colon \mathbf{J} \to \mathbf{C}$ sends every object of $\mathbf{J}$ to an object $n \in \mathbf{C}$ and every morphism to the identity morphism id_n. A *cone* with vertex $n \in \mathbf{C}$ of a diagram $\mathfrak{D} : \mathbf{J} \to \mathbf{C}$ is a natural transformation from the constant diagram $\Delta(n)$ to $\mathfrak{D}$. A *co-cone* with vertex $n \in \mathbf{C}$ of a diagram $\mathfrak{D} : \mathbf{J} \to \mathbf{C}$ is a natural transformation from $\mathfrak{D}$ to the constant diagram $\Delta(n)$. A *universal cone* $\tau\colon \Delta(n) \to \mathfrak{D}$ is such that all cones of $\mathfrak{D}$ uniquely *factor through* it, i.e. for any cone $\tau'\colon \Delta(n') \to \mathfrak{D}$ there exists a unique $\mathbf{C}$-morphism $h\colon n' \to n$ such that for every $j \in \mathbf{J}$, $\tau'_j = h; \tau_j$. A *universal co-cone* $\tau\colon \mathfrak{D} \to \Delta(n)$ is such that it factors uniquely through all co-cones of $\mathfrak{D}$. The *limit of a diagram* $\mathfrak{D}$ is a universal cone of $\mathfrak{D}$. The *colimit of a diagram* $\mathfrak{D}$ is a universal co-cone of $\mathfrak{D}$.

The important limits and colimits have names. The *final object* is the limit of the empty diagram. The *initial object* is the colimit of the empty diagram. A *product* is a limit of two objects. A *coproduct* is a colimit of two objects. A *pullback* is a limit of a *co-span*, given by two arrows with common target. A *pushout* is a colimit of a *span*, given by two arrows with common source.

A category $\mathbf{C}$ has all (finite) (co)limits if all (finite) diagrams in $\mathbf{C}$ have (co)limits. A category is *(finitely) (co)complete* if it has all (finite) (co)limits. A functor is *(finitely) (co)continuous* if it preserves all (finite) (co)limits.

Category **Set** has sets as objects and functions as arrows. Category **Cat** has categories[2] as objects and functors as morphisms.

A *concrete category* (over **Set**) is a pair $\langle \mathbf{C}, \mathbf{U} \rangle$ where $\mathbf{C}$ is a category and $\mathbf{U}\colon \mathbf{C} \to \mathbf{Set}$ is a faithful functor (called *forgetful or concretisation functor*).

[2]Small categories in fact (cf. [ST12])

A category $\mathbf{C}$ has an *inclusion system* $\langle \mathbf{I_C}, \mathbf{E_C} \rangle$ if $\mathbf{I_C}$ and $\mathbf{E_C}$ are two subcategories of $\mathbf{C}$ with $|\mathbf{I_C}| = |\mathbf{E_C}| = |\mathbf{C}|$ such that $\mathbf{I_C}$ is a partial order, and every $\mathbf{C}$-morphism f can be factored uniquely by $e_f \in \mathbf{E_C}$ and $i_f \in \mathbf{I_C}$ as $f = e_f; i_f$. The morphisms in $\mathbf{E_C}$ are called *abstract surjections*. The morphisms in $\mathbf{I_C}$ are called *inclusions* and they are denoted by $\subseteq$; for $o_1, o_2 \in |\mathbf{C}|$ and an $\mathbf{I_C}$-morphism $i \colon o_1 \to o_2$, we write $o_1 \subseteq o_2$ (cf. [Dia08]). In a *trivial inclusion system* for a category $\mathbf{C}$, category $\mathbf{I_C}$ is a discrete category. An inclusion system *has unions* if $\mathbf{I_C}$ has finite least upper bounds; unions are denoted by $\cup$. The standard inclusion system in **Set** is non-trivial and has unions.

3.2.1 Categories of Ordered Sets

Following [Mar12] we give the formalisation of categories of ordered sets and p-morphisms.

Definition 3.1 (R-sets) *An* R-set *is a pair* $\langle A, R_A \rangle$ *where* $R_A \subseteq A \times A$ *is a* transitive *relation on a set* A*. We sometimes write* A_R *instead of* $\langle A, R_A \rangle$*. We may use the infix notation for* R_A *and for* $a_1, a_2 \in A_R$ *we may also write* $a_1 \, R \, a_2$ *instead of* $a_1 \, R_A \, a_2$*, when decorations are clear from the context.*

Definition 3.2 (Category of R-sets and p-morphisms) Rset$\downarrow$ *has* R-sets *as objects and* pseudo-epimorphisms*, or* p-morphisms for short*, as morphisms. A p-morphism is a function that preserves the relation* R *and weakly reflects* R*-set down-closures, i.e. a p-morphism* $f \colon \langle A, R_A \rangle \to \langle B, R_B \rangle$ *is a function* $f \colon A \to B$ *such that:*

1. *(monotonicity) for all* $a_1, a_2 \in A$*,* $a_1 \, R_A \, a_2$ *implies* $f(a_1) \, R_B \, f(a_2)$*.*
2. *(weakly reflect* R*-down-closures) for all* $a_2 \in A, b_1 \in B$*,* $b_1 \, R_B \, f(a_2)$ *implies that there exists* $a_1 \in A$ *such that* $a_1 \, R_A \, a_2$ *and* $f(a_1) = b_1$*.*

Identities and composition are defined as expected.

The conditions listed above correspond to those of bisimulation (cf. [San09]). In modal logics, R-sets are called transitive Kripke frames (cf. [Seg70]) and p-morphisms are sometimes called bounded morphisms. The category of all

Kripke frames and p-morphisms is the category $\mathbf{Set_P}$ of coalgebras of the powerset functor (cf. [GS01]). This makes the category $\mathbf{Rset}\downarrow$ a full subcategory of $\mathbf{Set_P}$.

Definition 3.3 *We define three full subcategories of* $\mathbf{Rset}\downarrow$ *by limiting the relations:* $\mathbf{Preord}\downarrow$ *(reflexive),* $\mathbf{Soset}\downarrow$ *(irreflexive), and* $\mathbf{Soset_b}\downarrow$ *(irreflexive and bounded).*

Objects of $\mathbf{Soset_b}\downarrow$ are strict orders (irreflexive, transitive, therefore asymmetric) $A_<$ with the length of all descending chains limited by a natural number. Notice that this is a stronger requirement than just requiring that there are no infinite descending chains in $A_<$.

Definition 3.4 (Dependency Bound) *For* $A_< \in \mathbf{Soset_b}\downarrow$*, let the* dependency bound of $A_<$*, denoted by* $db(A_<)$*, be the length (number of elements) of the longest descending chain in* $A_<$*.*

Example 3.5 *We have* $db(\langle\emptyset,\emptyset\rangle) = 0$*,* $db(\langle\{\star\},\emptyset\rangle) = 1$ *and, for* $A_< = \langle\{1,2,3\},\{\langle1,2\rangle,\langle2,3\rangle,\langle1,3\rangle\}\rangle$*,* $db(A_<) = 3$*.*

3.3 Institutions and Specifications

Institutions are abstract formulations of model theory of logical systems. They are used as a basis for logic-independent abstract algebraic specifications. Many different logics have been shown to be institutions. See [GB84, BG92, ST12, Dia08] for introduction, comprehensive overview and examples.

An *institution* $\mathbb{I} = \langle\mathbf{Sig},\mathbf{Mod},\mathbf{Sen},\models\rangle$ consists of: a category $\mathbf{Sig}$ of *signatures*; a functor $\mathbf{Mod}\colon \mathbf{Sig}^{op} \to \mathbf{Cat}$ giving a category of Σ-*models* for each $\Sigma \in |\mathbf{Sig}|$; a functor $\mathbf{Sen}\colon \mathbf{Sig} \to \mathbf{Set}$ giving a set of Σ-*sentences* for each $\Sigma \in |\mathbf{Sig}|$; a family $\{\models_\Sigma\}_{\Sigma\in|\mathbf{Sig}|}$ of *satisfaction relations*, where $\models_\Sigma \subseteq \mathbf{Mod}(\Sigma) \times \mathbf{Sen}(\Sigma)$ for each $\Sigma \in |\mathbf{Sig}|$. The components of $\mathbb{I}$ are subject to the following *satisfaction condition*: for every signature morphism

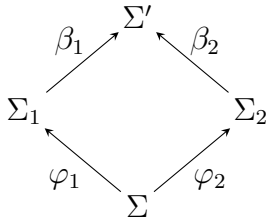
$\sigma : \Sigma \to \Sigma'$, Σ'-model M' and Σ-sentence φ,

$$\mathbf{Mod}(\sigma)(M') \models_\Sigma \varphi \text{ iff } M' \models_{\Sigma'} \mathbf{Sen}(\sigma)(\varphi) \ .$$

For a signature Σ, the class of all Σ-models $|\mathbf{Mod}(\Sigma)|$ is denoted by $[\![\Sigma]\!]$. For a morphism $\sigma : \Sigma \to \Sigma'$ in **Sig**, $\mathbf{Sen}(\sigma)$ is called the *σ-translation* map and $\mathbf{Mod}(\sigma)$ the *σ-reduct* functor. The σ-translation of a sentence $\varphi \in \mathbf{Sen}(\Sigma)$ is denoted by $\sigma(\varphi)$. The σ-reduct of a Σ'-model M' is denoted by $M'|_\sigma$.

For a signature $\Sigma \in \mathbf{Sig}$ and a set of Σ-sentences $\Phi \subseteq \mathbf{Sen}(\Sigma)$, we say that a Σ-model M *satisfies* Φ, $M \models_\Sigma \Phi$, if for each $\varphi \in \Phi$, $M \models_\Sigma \varphi$.

Given an institution $\mathbb{I} = \langle \mathbf{Sig}, \mathbf{Mod}, \mathbf{Sen}, \models \rangle$, the following diagram in **Sig** *admits amalgamation*



if for any two models $M_1 \in \mathbf{Mod}(\Sigma_1)$ and $M_2 \in \mathbf{Mod}(\Sigma_2)$ such that $M_1|_{\varphi_1} = M_2|_{\varphi_1}$, there exists a unique model $M' \in \mathbf{Mod}(\Sigma')$ such that $M'|_{\beta_1} = M_1$ and $M'|_{\beta_2} = M_2$ (then M' is called the *amalgamation* of M_1 and M_2); and for any two model morphisms $f_1 \colon M_1 \to M_1' \in \mathbf{Mod}(\Sigma_1)$ and $f_2 \colon M_2 \to M_2' \in \mathbf{Mod}(\Sigma_2)$ such that $f_1|_{\varphi_1} = f_2|_{\varphi_2}$, there exists a unique model morphism $f' : M' \to M'' \in \mathbf{Mod}(\Sigma')$ such that $f'|_{\beta_1} = f_1$ and $f'|_{\beta_2} = f_2$ (then f' is called the *amalgamation* of f_1 and f_2). Institution $\mathbb{I}$ has the *amalgamation property* if all pushouts in **Sig** exist and every pushout diagram in **Sig** admits amalgamation. Institution $\mathbb{I}$ is *semi-exact* if all pushouts exist in **Sig** and its model functor **Mod** *preserves pushouts*, that is, it maps them to pullbacks in **Cat**. An institution has the amalgamation property iff it is semi-exact.

For an institution $\mathbb{I} = \langle \mathbf{Sig}, \mathbf{Mod}, \mathbf{Sen}, \models \rangle$, *specifications* in $\mathbb{I}$ are abstract objects classified by signatures via operation *Sig* and defining classes of models via operation $[\![\_]\!]$ such that, for every specification SP, $Sig(SP) \in |\mathbf{Sig}|$ and $[\![SP]\!] \subseteq |\mathbf{Mod}(Sig(SP))|$. We assume here that the class of specifications

is closed under the following *specification-building operations*:

- for any $\Sigma \in |\mathbf{Sig}|$ and $\Phi \subseteq \mathbf{Sen}(\Sigma)$, a *presentation* $\langle \Sigma, \Phi \rangle$ is a specification with $Sig(\langle \Sigma, \Phi \rangle) = \Sigma$ and $[\![\langle \Sigma, \Phi \rangle]\!] = \{M \in |\mathbf{Mod}(\Sigma)| \mid M \models_\Sigma \Phi\}$;
- for any signature morphism $\sigma : \Sigma \to \Sigma'$ and a specification SP such that $Sig(SP) = \Sigma$, the *translation of SP along σ* is a specification $\sigma(SP)$ such that $Sig(\sigma(SP)) = \Sigma'$ and $[\![\sigma(SP)]\!] = \{M' \in \mathbf{Mod}(\Sigma') \mid M'|_\sigma \in [\![SP]\!]\}$;
- for any specifications SP_1, SP_2 such that $Sig(SP_1) = Sig(SP_2)$, the *union* $SP_1 \cup SP_2$ is a specification with $Sig(SP_1 \cup SP_2) = Sig(SP_1)$ and $[\![SP_1 \cup SP_2]\!] = [\![SP_1]\!] \cap [\![SP_2]\!]$;
- for any signature morphism $\sigma : \Sigma \to \Sigma'$ and a specification SP' such that $Sig(SP') = \Sigma'$, the *reduct of SP' along σ* is a specification $SP'|_\sigma$ such that $Sig(SP'|_\sigma) = \Sigma$ and $[\![SP'|_\sigma]\!] = \{M'|_\sigma \mid M' \in [\![SP']\!]\}$.

For a signature $\Sigma \in \mathbf{Sig}$, by $\mathbf{Spec}(\Sigma)$ we denote the class of all specifications over Σ (presentations and those obtained by translation, union and reduct operations). A model $M \in \mathbf{Mod}(\Sigma)$ *satisfies* $SP \in \mathbf{Spec}(\Sigma)$, written as $M \models SP$, if $M \in [\![SP]\!]$.

A specification $SP' \in \mathbf{Spec}(\Sigma')$ *refines* SP along a signature morphism $\sigma : \Sigma \to \Sigma'$, written $SP \rightsquigarrow^{\sigma} SP'$, if $[\![SP'|_\sigma]\!] \subseteq [\![SP]\!]$. Specifications in an arbitrary institution form a category of specifications and *specification morphisms*, where a specification morphism $\sigma : SP \to SP'$ is a signature morphism $\sigma : Sig(SP) \to Sig(SP')$ such that it is the refinement $SP \rightsquigarrow^{\sigma} SP'$. A specification morphism $\sigma : SP \to SP'$ is *conservative* if $[\![SP'|_\sigma]\!] = [\![SP]\!]$. The composition of conservative morphisms is also conservative.

3.3.1 Universal Algebra and Examples of Institutions

Below we formalise two logics as institutions (see [ST12] for more examples of institutions). Both are used in examples in further chapters. We start by brief introduction of universal algebra preliminaries.

For a set S, an *S-sorted set* is an S-indexed family of sets $X = \langle X_s \rangle_{s \in S}$. Let X and Y be S-sorted sets, an *S-sorted function* $f \colon X \to Y$ is an S-indexed family of functions $f = \langle f_s \colon X_s \to Y_s \rangle_{s \in S}$. A *many-sorted signature* is a pair $\Sigma = \langle S, \Omega \rangle$, where S is a set of sort names and Ω is an $(S^* \times S)$-sorted set of operation names, where S^* is the set of finite (including empty) sequences of elements of S. We write $sorts(\Sigma)$ for S and $ops(\Sigma)$ for Ω and we say that $f \colon s_1 \times \ldots \times s_n \to s$ of *arity* $s_1 \ldots s_n$ and *result sort* s is in Σ when $s_1 \ldots s_n \in S^*$, $s \in S$ and $f \in \Omega_{s_1 \ldots s_n, s}$. A signature $\Sigma = \langle S, \Omega \rangle$ is *finite* if it has finitely many symbols, i.e. S is a finite set; for any $s_1 \ldots s_n \in S^*$ and $s \in S$, $\Omega_{s_1 \ldots s_n, s}$ is a finite set; and for any $s \in S$ only for finitely many $s_1 \ldots s_n \in S^*$, $\Omega_{s_1 \ldots s_n, s} \neq \emptyset$. A *many-sorted signature morphism* $\sigma \colon \Sigma \to \Sigma'$ is a pair $\sigma = \langle \sigma_{sorts}, \sigma_{ops} \rangle$ with $\sigma_{sorts} \colon S \to S'$ and $\sigma_{ops} \colon \Omega \to \Omega'$ being a family of functions that map operation names respecting their arities and result sorts. The *category* **AlgSig** *of algebraic signatures* has many-sorted signatures as objects and many-sorted signature morphisms as morphisms. In examples we use the keywords **sorts** and **ops** to list the sorts and the operations of a many-sorted signature. For example we write

$$(\textbf{sorts } s, t;\ \textbf{ops } f : s \times s \to t)$$

to describe a signature Σ such that $sorts(\Sigma) = \{s, t\}$ and $ops(\Sigma)_{ss,t} = \{f\}$, and $ops(\Sigma)_o = \emptyset$ for all $o \in S^* \times S$ such that $o \neq (ss, t)$. In examples, while referring to the signature symbols, we sometimes shorten the name of an operation and write f instead of $f \colon s \times s \to t$. When writing presentations over algebraic signatures we sometimes use the keyword **axms** to list axioms, for example we write

$$(\textbf{sorts } s, t;\ \textbf{ops } f : s \times s \to t;\ \textbf{axms } \varphi_1, \varphi_2)$$

to describe the presentation $\langle \Sigma, \Phi \rangle$, where Σ is described above and $\Phi = \{\varphi_1, \varphi_2\}$ and $\varphi_1, \varphi_1 \in \mathbf{Sen}(\Sigma)$.

The *basic dependency* functor $\mathbf{SigSymb}_{\mathbf{AlgSig}} \colon \mathbf{AlgSig} \to \mathbf{Soset}_{\mathbf{b}}\downarrow$ takes signatures to ordered sets of their symbols and makes operation symbols

dependent on all the sorts from their arity and on the result sort. For any signature $\Sigma = (S, \Omega) \in \mathbf{AlgSig}$, $\mathbf{SigSymb}_{\mathbf{AlgSig}}(\Sigma) = (A, <_A)$, where A is the set of all symbols from Σ given as $A = S \cup \bigcup_{\overline{s} \in S^* \times S} \Omega_{\overline{s}} \times \{\overline{s}\}$[3] and the basic dependency of Σ is $<_A = \{\langle s', \langle f, \overline{s}\rangle\rangle \mid f \in \Omega_{\overline{s}}, \overline{s} = \langle s_1, \ldots, s_n, s\rangle$ and $s' = s$ or $s' = s_i$, for $1 \leq i \leq n\}$ (cf. [Mar12]). Then, for any set of symbols $B \subseteq A$ such that B is *closed-down w.r.t.* $<_A$ (i.e. for every $b \in B$ and every $a \in A$, if $a <_A b$ then $a \in B$) there exists the *reconstruction of a* Σ*-subsignature from* B *w.r.t.* $\mathbf{SigSymb}_{\mathbf{AlgSig}}$ containing all symbols from B, denoted by Σ_B, i.e. Σ_B is a signature such that $\Sigma_B \subseteq \Sigma$ and $\mathbf{SigSymb}_{\mathbf{AlgSig}}(\Sigma_B) = \langle B, <_B\rangle$, where $<_B = <_A|_B$.

Let $\Sigma = \langle S, \Omega\rangle$ be a many-sorted signature. A Σ*-algebra* A consists of an S-sorted set $|A|$ of nonempty[4] carrier sets and, for each $f\colon s_1 \times \ldots \times s_n \to s$ in Σ, an operation $(f\colon s_1 \times \ldots \times s_n \to s)_A\colon |A|_{s_1} \times \ldots \times |A|_{s_n} \to |A|_s$. Let A and B be Σ-algebras, a Σ*-homomorphism* $h\colon A \to B$ is an S-sorted function $h\colon |A| \to |B|$ respecting the operations of Σ. For any signature Σ, the *category* $\mathbf{Alg}(\Sigma)$ *of* Σ*-algebras* has Σ-algebras as objects and Σ-homomorphisms as morphisms. By $T_\Sigma(X)$ we denote the Σ-algebra of Σ*-terms* with variables from an S-sorted set X (cf. [ST12] for details). For a Σ-algebra A, an S-sorted function $v\colon X \to |A|$ is called an S*-sorted valuation*. There exists a unique Σ-homomorphism $v^\#\colon T_\Sigma(X) \to A$ extending the valuation of variables to valuation of terms. Let $\mathcal{X}$ be a fixed but arbitrary infinite set of variables, a Σ*-equation* $\forall X \cdot t = t'$ consists of a finite S-sorted set X of variables such that for all $s \in S$, $X_s \subseteq \mathcal{X}$ and two Σ-terms $t, t' \in |T_\Sigma(X)|_s$, for some sort $s \in S$. A Σ-algebra A *satisfies* a Σ-equation $\forall X \cdot t = t'$ if for every S-sorted valuation function $v\colon X \to |A|$, $v^\#(t) = v^\#(t')$.

Definition 3.6 (Institution of Equational Logic) The institution of equational logic*, denoted by* $\mathbb{EQ}$*, is a tuple* $\langle \mathbf{Sig}_{\mathbf{EQ}}, \mathbf{Mod}_{\mathbf{EQ}}, \mathbf{Sen}_{\mathbf{EQ}}, \models_{\mathbf{EQ}}\rangle$*, where* $\mathbf{Sig}_{\mathbf{EQ}} = \mathbf{AlgSig}$*; the model functor* $\mathbf{Mod}_{\mathbf{EQ}}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ *for each signature* $\Sigma \in |\mathbf{AlgSig}|$ *gives the category* $\mathbf{Alg}(\Sigma)$*, and for each signature*

[3] This makes the set of sort names and the sets of operation names disjoint.

[4] The requirement that sort carriers are nonempty follows for instance [Dia08] and differs this definition from the one in [ST12]. It is needed to ensure assumption (7) in Sect. 3.5 below.

morphism $\sigma\colon \Sigma \to \Sigma'$ *gives the reduct functor* $\_|_\sigma\colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$ *which maps* Σ'*-algebras and* Σ'*-homomorphisms to* Σ*-algebras and* Σ*-homomorphisms; the sentence functor* $\mathbf{Sen_{EQ}}\colon \mathbf{AlgSig} \to \mathbf{Set}$ *for each* $\Sigma \in |\mathbf{AlgSig}|$ *gives the set of* Σ*-equations, and for each* $\mathbf{AlgSig}$*-morphism* $\sigma\colon \Sigma \to \Sigma'$ *it gives the* σ*-translation function taking* Σ*-equations to* Σ'*-equations; for each* $\Sigma \in |\mathbf{AlgSig}|$*, the satisfaction relation* $\models_{\mathbf{EQ}}\colon |\mathbf{Alg}(\Sigma)| \times |\mathbf{Sen_{EQ}}(\Sigma)|$ *is the relation of satisfaction of* Σ*-equations by* Σ*-algebras.*

Definition 3.7 The institution of finite equational logic*, denoted by* $\mathbb{EQF}$*, is a variant of* $\mathbb{EQ}$ *obtained by restricting its category of signatures to be* **FAlgSig**.

Definition 3.8 (Institution of First-Order Logic with Equality) The institution of first-order logic with equality*, denoted by* $\mathbb{FOEQ}$*, is defined as a tuple* $\langle \mathbf{Sig_{FOEQ}}, \mathbf{Mod_{FOEQ}}, \mathbf{Sen_{FOEQ}}, \models_{\mathbf{FOEQ}} \rangle$*, where* $\mathbf{Sig_{FOEQ}} = \mathbf{AlgSig}$*; the model functor* $\mathbf{Mod_{FOEQ}} = \mathbf{Mod_{EQ}}$ *(cf. Def. 3.6); the sentence functor* $\mathbf{Sen_{FOEQ}}\colon \mathbf{AlgSig} \to \mathbf{Set}$ *for each* $\Sigma \in |\mathbf{AlgSig}|$ *gives the set of all closed (i.e. without unbound occurrences of variables) first-order formulae built out of atomic formulae using the standard propositional connectives (*$\vee$*,* $\wedge$*,* $\Rightarrow$*,* $\Longleftrightarrow$ *,* $\neg$*) and quantifiers (*$\forall$*,* $\exists$*), where by atomic formulae we mean the logical constants (*true*,* false*) and equalities of the form* $t = t'$ *with* t *and* t' *being* Σ*-terms (possibly with variables) of the same sort; for each first-order signature morphism* $\sigma : \Sigma \to \Sigma'$*, it gives the usual translation of first-order* Σ*-sentences to first-order* Σ'*-sentences; for each* $\Sigma \in |\mathbf{AlgSig}|$*, the satisfaction relation* $\models_{\mathbf{FOEQ}}\colon |\mathbf{Alg}(\Sigma)| \times |\mathbf{Sen_{FOEQ}}(\Sigma)|$ *is the usual relation of satisfaction of first-order formulae by a* Σ*-algebra.*

Definition 3.9 The institution of finite first-order logic with equality*, denoted by* $\mathbb{FOEQF}$*, is a variant of* $\mathbb{FOEQ}$ *obtained by restricting its category of signatures to be* **FAlgSig**.

3.4 Parameterisation

Parameterisation is a mechanism allowing for abstraction of a definition or an expression from its context in a way that such abstraction may be considered

as an independent entity. Dependencies on the context are represented by an interface describing the admissible parameters (cf. [ST12]).

In the literature one considers *parameterised specifications*, which take specifications as parameters and give specifications as results (in the spirit of specification-building operations introduced above) and *specifications of parameterised modules* (or *parameterised module specifications*), where parameterised modules take models as parameters and give models as results (corresponding to parametric program modules e.g. in Standard ML [Mac84]). In this thesis we discuss only the latter.

In a given institution $\mathbb{I} = \langle \mathbf{Sig}, \mathbf{Mod}, \mathbf{Sen}, \models \rangle$, a *parameterised module signature* is a signature morphism $\sigma\colon \Sigma_P \to \Sigma_R \in \mathbf{Sig}$, where Σ_P is a signature of the parameter and Σ_R is a signature of the result. *A parameterised module model* of σ (called also a *constructor* in [ST12]) is a (sometimes partial) function $\kappa : [\![\Sigma_P]\!] \to [\![\Sigma_R]\!]$. A *parameterised module specification* over σ is a triple $\langle \sigma, SP_P, SP_R \rangle$, where SP_P is a specification of the parameter and SP_R is the specification of the result such that $\sigma\colon SP_P \to SP_R$ is a specification morphism. A parameterised module model κ *satisfies* a parameterised module specification $\langle \sigma, SP_P, SP_R \rangle$ if for all $M \in [\![SP_P]\!]$, $\kappa(M) \in [\![SP_R]\!]$. Parameterised module model κ is a *persistent parameterised module model* that satisfies $\langle \sigma, SP_P, SP_R \rangle$ if additionally, for all $M \in [\![SP_P]\!]$, $\kappa(M)|_\sigma = M$.

3.5 Assumptions

In what follows we work in an institution $\mathbb{I} = \langle \mathbf{Sig}, \mathbf{Mod}, \mathbf{Sen}, \models \rangle$, called *base institution*, such that:

1. institution $\mathbb{I}$ is semi-exact;
2. category of signatures **Sig** is a concrete category (via the forgetful functor $\mathbf{SetSymb}\colon \mathbf{Sig} \to \mathbf{Set}$)[5], and **Sig** has all finite colimits; additionally, we require functor **SetSymb** to preserve and reflect finite colimits;

[5]Most of the work in this thesis is done for infinite signatures, however, in some places we are obliged to assume that signatures are finite, i.e. $\Sigma \in |\mathbf{Sig}|$ such that $\mathbf{SetSymb}(\Sigma)$ is a finite set. The finiteness of signatures is used in Sect. 6.4.3, Theorem 6.22, Lemma 6.35, Lemma 7.6, Theorem 7.15, Corollary 7.16, and Def. 8.3 below.

3. category **Sig** comes with an inclusion system with unions (thus non-trivial) and **SetSymb** maps inclusions in **Sig** to standard inclusions in **Set**;

4. there is the *basic dependency* functor $\mathbf{SigSymb}\colon \mathbf{Sig} \to \mathbf{Soset_b}\downarrow$, the functor transforming signatures to bounded sosets (cf. Def. 3.3) of their symbols (see Sect. 5.2 for use of **SigSymb**); the basic dependency functor is compatible with the concretisation functor, i.e. for any signature $\Sigma \in |\mathbf{Sig}|$, $\mathbf{SigSymb}(\Sigma) = \langle \mathbf{SetSymb}(\Sigma), < \rangle$, for some relation $<$;

5. it is possible to reconstruct the subsignature from any closed-down ordered subset of the symbols from the signature, i.e. for a signature $\Sigma \in |\mathbf{Sig}|$, let $\langle A, <_A \rangle = \mathbf{SigSymb}(\Sigma)$; for any set of symbols $B \subseteq A$ such that B is *closed-down w.r.t.* $<_A$ (i.e. for every $b \in B$ and every $a \in A$, if $a <_A b$ then $a \in B$) there exists the *reconstruction of a* Σ*-subsignature from* B *w.r.t.* **SigSymb** containing all symbols from B, denoted by Σ_B, i.e. such that $\Sigma_B \subseteq \Sigma$ and $\mathbf{SigSymb}(\Sigma_B) = \langle B, <_B \rangle$, where $<_B = <_A|_B$;

6. reduct functors for morphisms that are surjective on their symbols are injective on models, i.e. for a signature morphism $\sigma\colon \Sigma_1 \to \Sigma_2$, if $\mathbf{SetSymb}(\sigma)$ is surjective in **Set** then for $M, M' \in [\![\Sigma_2]\!]$, if $M|_\sigma = M'|_\sigma$, then $M = M'$;

7. for a signature morphism $\sigma\colon \Sigma_1 \to \Sigma_2$, if $\mathbf{SetSymb}(\sigma)$ is injective in **Set** then for every model $M_1 \in [\![\Sigma_1]\!]$, there exists a model $M_2 \in [\![\Sigma_2]\!]$ such that $M_2|_\sigma = M_1$;

8. there is a unique model $M_\emptyset \in [\![\Sigma_\emptyset]\!]$, where $\Sigma_\emptyset \in \mathbf{Sig}$ is the initial object of **Sig**.

Assumptions (2), (7) and (8) together imply that for every $\Sigma \in \mathbf{Sig}$ there is $M \in [\![\Sigma]\!]$.

By default examples below are given in the institution of equational logic $\mathbb{EQ}$ (cf. Def. 3.6). In some cases the institution is given explicitly (e.g. in

Chapter 9) and then it is the institution of the first order logic with equality $\mathbb{FOEQ}$ (cf. Def. 3.8) or its finite variant $\mathbb{FOEQF}$ (cf. Def. 3.9). It is easy to check that both $\mathbb{EQ}$ and $\mathbb{FOEQ}$ (and their finite variants $\mathbb{EQF}$ and $\mathbb{FOEQF}$) meet all requirements given above. The assumption that all carriers in algebras are non-empty is required to satisfy assumption (7). The basic dependency functor for many sorted algebraic signatures is $\mathbf{SigSymb}_{\mathbf{AlgSig}}$ from Sect. 3.3.1.

Chapter 4

Signature Fragments

4.1 Introduction

Typically an algebraic signature is a set of symbols formed along some rules into a well-defined structure. In the case of algebraic many-sorted signatures the rule is that, given a signature, if an operation symbol is attributed by a sort symbol as its argument or result type, then this sort symbol must also be part of the signature (cf. Sect. 3.3.1). This "closure" requirement causes that not all sets of symbols constitute signatures. Moreover, not all operations known from the set theory are given for signatures. For example, the subtraction operation is not defined on such signatures. It is impossible to subtract from a signature a set of sort symbols, if any of those symbols is an argument or result sort of a non-subtracted operation symbol.

Why would we need to subtract signatures? In this thesis we focus on the signatures of parameterised modules, typically represented as signature morphisms $\sigma\colon \Sigma_P \to \Sigma_R$ (cf. Sect. 3.4). The morphism σ (usually being an inclusion) simply marks which result symbols come from the parameter and which are defined by (are the result of) the parameterised module. Unfortunately, it is impossible to express this kind of information directly. The subtraction $\Sigma_R \setminus \Sigma_P$ in general does not yield a well formed signature. Here comes the idea of *signature fragments*. Given a signature, we distinguish those elements that are *defined* in the signature, leaving the rest *assumed* (parameter). This approach, even though it may seem only superficially different, gives rise to a direct representation of signature inclusion (with no morphisms used) and enables all set-theoretic operations on defined symbols. Moreover, it does not require the assumed part of a signature to be a signature itself.

In this chapter we introduce a concept of signature fragments independently of the detailed definition of the category of signatures. In the first section we give a basic approach with simple representation of signatures as signature fragments and a completion of fragments into a full signature. The other section presents more elaborate version providing more ways to transform signature fragments into signatures and back. The content of that other section is not to be used in the rest of the thesis. We provide it here to give a somewhat more general view on the concept of fragments.

4.2 Basic Approach

In this section we define signature fragments assuming that the category of signatures **Sig** is a concrete category with the concretisation functor named $\mathbf{SetSymb}\colon \mathbf{Sig} \to \mathbf{Set}$. The results from this section apply to any concrete category, we name it **Sig** to emphasise that a definition of signature fragments is the main purpose of the results.

Definition 4.1 Category of **Sig**-fragments *is a comma category*

$$\mathbf{Sig}^{frag} = (\mathbf{Set} \downarrow \mathbf{SetSymb})$$

Signature fragments are $\mathbf{Sig}^{frag}$-objects, i.e. triples of the form

$$\langle A, \Sigma, f\colon A \to \mathbf{SetSymb}(\Sigma)\rangle$$

where $A \in \mathbf{Set}$, $\Sigma \in \mathbf{Sig}$, and f is the *internal function* of the fragment.

A signature fragment morphisms from $\langle A, \Sigma, f\colon A \to \mathbf{SetSymb}(\Sigma)\rangle$ to $\langle A', \Sigma', f'\colon A' \to \mathbf{SetSymb}(\Sigma')\rangle$ is a pair

$$\langle g\colon A \to A', \sigma\colon \Sigma \to \Sigma'\rangle$$

where g is a function and σ is a **Sig**-morphism such that $g; f' = f; \mathbf{SetSymb}(\sigma)$.

The idea behind the fragment $\langle A, \Sigma, f\colon A \to \mathbf{SetSymb}(\Sigma)\rangle$ is that $f(A)$ marks a distinguished set of Σ symbols. Those distinguished symbols are

called *defined*, whereas the rest of Σ-symbols are called *assumed*. The sets of defined and assumed symbols are disjoint and their union is the set of all symbols in Σ. Neither of them needs to be the set of symbols of a subsignature of Σ.

A signature fragment is a fragment of the given whole. There is a set of defined symbols (a fragment) and a signature (the whole). In that sense the completion of the fragment is its whole.

There are three important functors between **Sig** and $\mathbf{Sig}^{frag}$. The first is the *embedding functor* $\mathbf{Frag}_{\mathbf{Sig}} \colon \mathbf{Sig} \to \mathbf{Sig}^{frag}$ that takes a signature Σ to the signature fragment $\langle \mathbf{SetSymb}(\Sigma), \Sigma, id_{\mathbf{SetSymb}(\Sigma)} \rangle$. The second is the *completion functor* $\mathbf{Compl}_{\mathbf{Sig}} \colon \mathbf{Sig}^{frag} \to \mathbf{Sig}$ defined as a projection that takes $\langle A, \Sigma, f \colon A \to \mathbf{SetSymb}(\Sigma) \rangle$ to Σ. The third is the *emptying functor* $\mathbf{Empt}_{\mathbf{Sig}} \colon \mathbf{Sig} \to \mathbf{Sig}^{frag}$ taking a signature Σ to $\langle \emptyset, \Sigma, \epsilon_\Sigma \colon \emptyset \to \mathbf{SetSymb}(\Sigma) \rangle$ where ϵ_Σ is the unique function from the empty set to $\mathbf{SetSymb}(\Sigma)$. The behavior of the three functors on morphisms is as one expects (cf. the forthcoming Sect. 4.3).

Notation. We omit the index **Sig** and simply write **Compl**, **Frag** and **Empt** in order to make the notation more compact. For the same reason we leave the embedding **Frag** and the projection **Compl** implicit where possible. Signature fragments are underlined, e.g. we typically write

$$\underline{\Sigma} = \langle A, \Sigma, f \colon A \to \mathbf{SetSymb}(\Sigma) \rangle$$

Defined symbols are considered as elements of the signature fragment, i.e. we write $a \in \underline{\Sigma}$ and this means that a is defined in $\underline{\Sigma}$. Assumed symbols are not considered elements of the fragment, but they are elements of the completion of the fragment, i.e. we write $a \in \mathbf{Compl}(\underline{\Sigma})$, $a \notin \underline{\Sigma}$, and this means that a is assumed in $\underline{\Sigma}$. When writing the contents of signature fragments we underline assumed symbols, e.g. for $\underline{\Sigma} = (\mathbf{sort}\ \underline{s};\ \mathbf{ops}\ a : s,\ \underline{b : s})$, we have $a : s \in \underline{\Sigma}$, but $s \notin \underline{\Sigma}$ and $b : s \notin \underline{\Sigma}$. Of course we have $a : s, b : s, s \in \mathbf{Compl}(\underline{\Sigma})$.

Notation. When **Sig** contains symbols that expose some typing information (like operation symbols in algebraic signatures) sometimes we drop the type information of such symbols, of course only when the type is obvious

from the context. For example, we may write $a, b, s \in \mathbf{Compl}(\underline{\Sigma})$ instead of $a : s, b : s, s \in \mathbf{Compl}(\underline{\Sigma})$.

The following theorem shows that the (co)completeness of $\mathbf{Sig}^{frag}$ follows the (co)completeness of **Sig**.

Theorem 4.2 *Given a (finitely) (co)complete concrete category of signatures* **Sig***, the category of its signature fragments* $\mathbf{Sig}^{frag}$ *is (finitely) (co)complete as well.*

The proof is in Appendix 4.A.

The overall idea behind the signature fragments is that the distinguished set simply marks some symbols as defined. However, the definition is more general and does not limit fragments to those with injective internal functions only. This is to keep the definition as simple as possible and to have the (co)completeness of the category of signature fragments. Nevertheless, we recognise such uniform signatures fragments and call them *injective*.

An *injective signature fragment* has an injection as the internal function: $\langle A, \Sigma, f \colon A \to \mathbf{SetSymb}(\Sigma)\rangle$ is injective if f is an injection.

There are two other special kinds of fragments that merit distinction: *empty* and *complete signature fragments*.

An *empty signature fragment* is such that all symbols are assumed: signature fragment $\langle A, \Sigma, f \colon A \to \mathbf{SetSymb}(\Sigma)\rangle$ is empty if $A = \emptyset$.

In a *complete signature fragment* all symbols are uniquely defined: signature fragment $\langle A, \Sigma, f \colon A \to \mathbf{SetSymb}(\Sigma)\rangle$ is complete if f is bijective.

The inclusion system of $\mathbf{Sig}^{frag}$ is built over the inclusion systems of **Set** and **Sig** (cf. diagonal-fill lemma in [Dia08]).

In literature one can find definitions of entities similar to our signature fragments, e.g. signature fragments in CASL (cf. Sect. III:2.1 of [Mos04]).

4.3 Extended Approach

In this section we elaborate the notion of fragments, offering an interesting further insight into this concept. Nevertheless this section may be skipped at first reading, its content is not explicitly used in the chapters that follow.

Here we talk about *fragments*, not necessarily signature fragments, as previously. This is to emphasise the possibility to define fragments of objects in any concrete category, not only signatures. Consequently, when discussing describing the contents of a fragment we talk about elements instead of symbols.

Nevertheless, the definition of fragments is essentially the same as the one of signature fragments in Def. 4.1.

Definition 4.3 (Category of Fragments) *Given a concrete category* $\langle \mathbf{C}, \mathbf{U} \rangle$*, the category of* $\mathbf{C}$-fragments*,* $\langle \mathbf{C}, \mathbf{U} \rangle^{frag}$*, is the comma category* $(\mathbf{Set} \downarrow \mathbf{U})$*.*

Notation. If the forgetful functor $\mathbf{U} \colon \mathbf{C} \to \mathbf{Set}$ is obvious from the context, we omit it and write just $\mathbf{C}$ instead of $\langle \mathbf{C}, \mathbf{U} \rangle$ and $\mathbf{C}^{frag}$ to denote $\langle \mathbf{C}, \mathbf{U} \rangle^{frag}$.

Similarly to the basic approach of Sect. 4.2, we underline fragments, e.g. $\underline{p} = \langle A, c, f \colon A \to \mathbf{U}(c) \rangle$. Set $f(A)$ contains *defined elements* of the fragment, we write $a \in \underline{p}$, for $a \in f(A)$. *Assumed elements* are those elements of $\mathbf{U}(c)$ that are not defined, i.e. not in the range of f. Assumed elements are not considered being "in" the fragment, for $b \in \mathbf{U}(c)$ we write $b \notin \underline{p}$, if $b \notin f(A)$.

As already given in the previous section, there are functors between a category $\langle \mathbf{C}, \mathbf{U} \rangle$ and its category of fragments. Here we show that they are adjoint.

Definition 4.4 *For each category of fragments* $\langle \mathbf{C}, \mathbf{U} \rangle^{frag}$ *there are functors:*

$$\begin{aligned} \textit{(emptying functor)} &: \mathbf{Empt}_{\mathbf{C}} \colon \mathbf{C} \to \mathbf{C}^{frag}, \\ \textit{(completion functor)} &: \mathbf{Compl}_{\mathbf{C}} \colon \mathbf{C}^{frag} \to \mathbf{C}, \\ \textit{(embedding functor)} &: \mathbf{Frag}_{\mathbf{C}} \colon \mathbf{C} \to \mathbf{C}^{frag}, \end{aligned}$$

defined as

$$\begin{aligned}
\mathbf{Empt}_{\mathbf{C}}(c) &= \langle \emptyset, c, \emptyset_{\mathbf{U}(c)} \colon \emptyset \to \mathbf{U}(c) \rangle, \\
\mathbf{Empt}_{\mathbf{C}}(g) &= \langle id_{\emptyset}, g \rangle, \\
\mathbf{Compl}_{\mathbf{C}}(\langle A, c, f \rangle) &= c, \\
\mathbf{Compl}_{\mathbf{C}}(\langle g_1, g_2 \rangle) &= g_2, \\
\mathbf{Frag}_{\mathbf{C}}(c) &= \langle \mathbf{U}(c), c, id_{\mathbf{U}(c)} \rangle, \\
\mathbf{Frag}_{\mathbf{C}}(g) &= \langle \mathbf{U}(g), g \rangle
\end{aligned}$$

for any object $c \in \mathbf{C}$*, morphism* $g \colon c \to d \in \mathbf{C}$*, fragment* $\langle A, c, f \rangle \in \mathbf{C}^{frag}$ *and fragment morphism* $\langle g_1, g_2 \rangle \colon p \to q \in \mathbf{C}^{frag}$*.*

Notation. In order to make the text slightly more readable we omit the index $\mathbf{C}$ and simply write **Frag** instead of $\mathbf{Frag}_{\mathbf{C}}$, etc.

Theorem 4.5 *The three functors defined in Def. 4.3 are adjoint in the following way*

$$\mathbf{Empt} \dashv \mathbf{Compl} \dashv \mathbf{Frag}$$

and the counit of $\mathbf{Compl} \dashv \mathbf{Frag}$ *is the identity. Moreover, category* $\mathbf{C}$ *is fully embeddable into its category of fragments* $\langle \mathbf{C}, \mathbf{U} \rangle^{frag}$ *via* **Frag**.

Proof. It is easy to check that the functors are adjoint as indicated using directly their definitions.

For $\mathbf{Empt} \dashv \mathbf{Compl}$, given $f \colon c \to \mathbf{Compl}(p) \in \mathbf{C}$, for some $p = \langle A, d, i_A \colon A \to \mathbf{U}(d) \rangle \in \mathbf{C}^{frag}$, $f^{\#} \colon \mathbf{Empt}(c) \to p = \langle \emptyset_A, f \rangle$.

For $\mathbf{Compl} \dashv \mathbf{Frag}$, given $\langle f', f \rangle \colon \langle A, d, i_A \colon A \to \mathbf{U}(d) \rangle \to \mathbf{Frag}(c) \in \mathbf{C}^{frag}$, for some $c \in \mathbf{C}$, $\langle f', f \rangle^{\#} \colon d \to c = f$.

Given an object $c \in \mathbf{C}$, the counit of $\mathbf{Compl} \dashv \mathbf{Frag}$ for c is the identity, $\epsilon_c \colon \mathbf{Compl}(\mathbf{Frag}(c)) \to c = id_c$, because $\mathbf{Compl}(\mathbf{Frag}(c)) = c$.

Category $\mathbf{C}$ is fully embeddable into $\langle \mathbf{C}, \mathbf{U} \rangle^{frag}$ via **Frag**, because **Frag** is injective on objects, faithful and full. □

The (co)completeness of the concrete category induces (co)completeness of its category of fragments. Theorem 4.2 applies also here.

Some concrete categories contain enough objects and morphisms to guarantee the existence of the greatest subobject included in the given fragment. In the absence of a better name we call such categories *fragmentable*.

Definition 4.6 *A concrete category* $\langle \mathbf{C}, \mathbf{U} \rangle^{frag}$ *is* fragmentable *if there exists the fourth functor (an addition to the three functors from Def. 4.4)*

$$\textit{(subobject functor)}: \ \mathbf{Sub_C}\colon \mathbf{C}^{frag} \to \mathbf{C}$$

such that it is the right adjoint of $\mathbf{Frag_C}$*, i.e.* $\mathbf{Frag_C} \dashv \mathbf{Sub_C}$*, and the unit of this adjunction is the identity.*

Given a fragmentable category $\mathbf{C}$ and a $\mathbf{C}$-fragment, the subobject functor **Sub** gives the greatest $\mathbf{C}$-object with all elements being defined in the fragment. It is, however, not guaranteed that all elements defined in the fragment are in the resulting $\mathbf{C}$-object. Informally, the missing ones are those dependent on some assumed elements.

The following theorem gives a very close relationship between the fragmentable categories and their categories of fragments.

Theorem 4.7 *Every fragmentable category* $\mathbf{C}$ *is fully embeddable into its category of fragments* $\langle \mathbf{C}, \mathbf{U} \rangle^{frag}$ *via* **Frag** *as a reflective and a coreflective subcategory (cf. Sect. 4 in [AHS90]).*

Proof. By Theorem 4.5 the category $\mathbf{C}$ is fully embeddable into its category of fragments. The functor **Frag** is both left and right adjoint, thus the embedding is reflective and coreflective (cf. Sect. 18.2 in [AHS90]).

□

By the requirement that the counit of $\mathbf{Compl} \dashv \mathbf{Frag}$ (cf. Def. 4.3) and the unit of $\mathbf{Frag} \dashv \mathbf{Sub}$ (cf. Def. 4.6) are identities we easily get the following result for any fragmentable category $\mathbf{C}$.

Corollary 4.8 *For any* $c \in \mathbf{C}$, $\mathbf{Compl}(\mathbf{Frag}(c)) = \mathbf{Sub}(\mathbf{Frag}(c)) = c$.

The above result gives proper grounds to the notation introduced in Sect. 4.2, so that we sometimes leave the embedding **Frag** and the projection **Compl** implicit.

4.3.1 Special Fragments

In this section we rediscover the kinds of fragments that exhibit some nice categorical properties. Essentially they prove to be the same structures as already introduced in Sect. 4.2.

Empty fragments are those without any defined elements.

Definition 4.9 (Empty Fragments) *A fragment* $p \in \mathbf{C}^{frag}$ *is* empty *iff the counit of* $\mathbf{Empt} \dashv \mathbf{Compl}$, $\epsilon_p \colon \mathbf{Empt}(\mathbf{Compl}(p)) \to p$, *is an isomorphism.*

The lemma below shows the correspondence of the above defined empty fragments and those from Sect. 4.2.

Lemma 4.10 *A fragment* $p = \langle A, c, f \rangle \in \mathbf{C}^{frag}$ *is an empty fragment iff* $A = \emptyset$.

Proof. Let the counit morphism of $\mathbf{Empt} \dashv \mathbf{Compl}$ for p be $\epsilon_p = \langle \epsilon_{p1}, \epsilon_{p2} \rangle$ where $\epsilon_{p1} \colon \emptyset \to A$ and $\epsilon_{p2} \colon c \to c$. It is easy to check that $\epsilon_{p2} = id_c$.

($\Rightarrow$) Function ϵ_{p1} is bijective, thus $A = \emptyset$;

($\Leftarrow$) if $A = \emptyset$ then ϵ_{p1} is bijective, ϵ_{p2} is iso, thus ϵ_p is iso. □

Complete fragments have all elements defined, i.e. they are like objects of the base category. The definition below and the lemma that follows show that this intuition is correct.

Definition 4.11 (Complete Fragments) *A fragment* $p \in \mathbf{C}^{frag}$ *is* complete *iff the unit of* $\mathbf{Compl} \dashv \mathbf{Frag}$, $\eta_p \colon p \to \mathbf{Frag}(\mathbf{Compl}(p))$, *is an isomorphism.*

Lemma 4.12 *Given a complete fragment* $\underline{p} \in C^{frag}$, *there exists an object* $o \in C$ *such that* $\mathbf{Frag}(o)$ *is isomorphic to* $\underline{p}$.

Proof. Let $o = \mathbf{Compl}(p)$. Fragment $\underline{p}$ is complete thus, by Def. 4.11, $\mathbf{Frag}(\mathbf{Compl}(\underline{p}))$ is isomorphic to $\underline{p}$. □

The following lemma shows that the complete fragments as defined above and those from Sect. 4.2 coincide in the framework of Sect. 4.2.

Lemma 4.13 *A fragment $p = \langle A, c, f\rangle \in \mathbf{C}^{frag}$ is a complete fragment iff the function f is bijective.*

Proof. The unit morphism of **Compl** $\dashv$ **Frag** for p is $\eta_p = \langle \eta_{p1}, \eta_{p2}\rangle$ where $\eta_{p1}\colon A \to \mathbf{U}(c)$ and $\eta_{p2}\colon c \to c$. It is easy to check that $\eta_{p1} = f$ and $\eta_{p2} = id_{\mathbf{U}(c)}$.

($\Rightarrow$) η_{p1} is bijective, thus f is also bijective;

($\Leftarrow$) f is bijective, so η_p is iso. □

The subobject functor does not alter complete fragments, because the greatest subobject included in a complete fragment is the whole object. The lemma and the corollary below validate this intuition.

Lemma 4.14 *For a fragmentable category $\mathbf{C}$, given a complete fragment $\underline{p} \in \mathbf{C}^{frag}$, the counit of* **Frag** $\dashv$ **Sub**, $\epsilon_p\colon \mathbf{Frag}(\mathbf{Sub}(p)) \to p$, *is an isomorphism.*

Proof. Let $p = \langle A, c, f\rangle$ and $\epsilon_p = \langle \epsilon_{p1}, \epsilon_{p2}\rangle$ where $\epsilon_{p1}\colon \mathbf{U}(\mathbf{Sub}(p)) \to A$ and $\epsilon_{p2}\colon \mathbf{Sub}(p) \to c$. By Lemma 4.8, $\mathbf{Sub}(\langle \mathbf{U}(c), c, id_{\mathbf{U}(c)}\rangle) = c$. It makes $\langle f, id_c\rangle\colon p \to \langle \mathbf{U}(c), c, id_{\mathbf{U}(c)}\rangle$ a $\mathbf{C}^{frag}$-morphism. Therefore, we have $\epsilon_{p2} = \mathbf{Sub}(\langle f, id_c\rangle)$. By Lemma 4.13, f is bijective, thus ϵ_{p2} is also iso and so is ϵ_{p1}, because ϵ_p is a $\mathbf{C}^{frag}$-morphism; therefore, $\epsilon_{p1}; f = id_{\mathbf{U}(\mathbf{Sub}(p))}; \mathbf{U}(\epsilon_{p2})$. □

Corollary 4.15 *For a complete fragment $p \in \mathbf{C}^{frag}$,* **Compl**(p) *is isomorphic to* **Sub**(p).

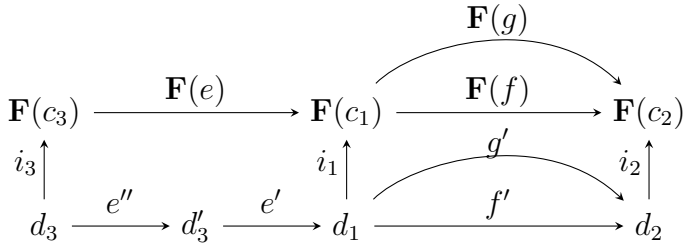
4.A Appendix: Proofs

Proof of Theorem 4.2. The following general lemma does the work. Notice that we do not put any limitations on the functor $\mathbf{F}$, only the properties of two categories are used.

Lemma 4.16 *Given categories* $\mathbf{C}$, $\mathbf{D}$ *and a functor* $\mathbf{F}\colon \mathbf{C} \to \mathbf{D}$. *If both categories are (finitely) (co)complete then the comma category* $(\mathbf{D} \downarrow \mathbf{F})$ *is also (finitely) (co)complete.*

Proof. The first functor of the comma category $(\mathbf{D} \downarrow \mathbf{F})$ is identity, so it is cocontinuous. The cocompleteness of $(\mathbf{D} \downarrow \mathbf{F})$ is a consequence of the well known fact that if both categories are (finitely) cocomplete and the first functor is (finitely) cocontinuous then the comma category is (finitely) cocomplete (cf. [ST12]).

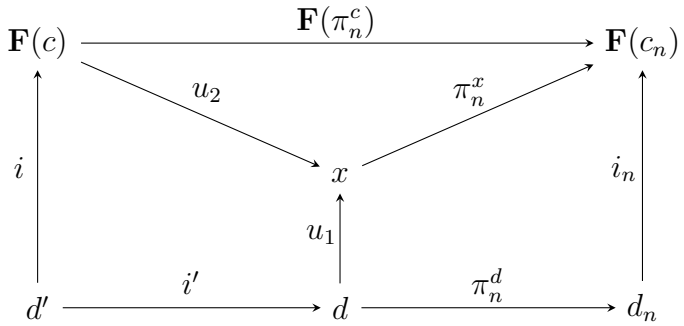
The proof of completeness without the requirement that $\mathbf{F}$ is continuous is less known, we present it here. Let us first show the existence of all equalizers and then the existence of all products. Given two $(\mathbf{D} \downarrow \mathbf{F})$-objects $\langle d_1, c_1, i_1\colon d_1 \to \mathbf{F}(c_1)\rangle$ and $\langle d_2, c_2, i_2\colon d_2 \to \mathbf{F}(c_2)\rangle$ and two morphisms $\langle f', f\rangle, \langle g', g\rangle\colon \langle d_1, c_1, i_1\rangle \to \langle d_2, c_2, i_2\rangle$, their equalizer is a morphism $\langle e''; e', e\rangle\colon \langle d_3, c_3, i_3\rangle \to \langle d_1, c_1, i_1\rangle$, where $e\colon c_3 \to c_1$ is the equalizer of f and g in $\mathbf{C}$, $e'\colon d'_3 \to d_1$ is the equalizer of f' and g' in $\mathbf{D}$, and morphisms $i_3\colon d_3 \to \mathbf{F}(c_3)$ and $e''\colon d_3 \to d'_3$ are the pullback of $\mathbf{F}(e)$ and $e'; i_1$ in $\mathbf{D}$.



The universality of $\langle e''; e', e\rangle$ is a direct consequence of the universality of equalizers e and e' and the pullback i_3, e''.

The proof of the existence of products follows the same idea. Given a (finite) collection $\langle d_n, c_n, i_n\rangle$ of $(\mathbf{D} \downarrow \mathbf{F})$-objects, $n \in N$, let c with projections $\pi^c_n\colon c \to c_n$ for $n \in N$ be the product of c_n in $\mathbf{C}$, let d with projections

$\pi_n^d \colon d \to d_n$ for $n \in N$ be the product of d_n in $\mathbf{D}$ and let x with projections $\pi_n^x \colon x \to \mathbf{F}(c_n)$ for $n \in N$ be the product of $\mathbf{F}(c_n)$ in $\mathbf{D}$. Due to universality of x, there exist two morphisms $u_1 \colon d \to x$ and $u_2 \colon \mathbf{F}(c) \to x$ in category $\mathbf{D}$, so that $u_1; \pi_n^x = \pi_n^d; i_n$ and $u_2; \pi_n^x = \mathbf{F}(\pi_n^c)$. Let $i \colon d' \to \mathbf{F}(c)$ and $i' \colon d' \to d$ be the pullback of u_1 and u_2 in $\mathbf{D}$. The product of the collection is $\langle d', c, i \rangle$ with projections $\langle i'; \pi_n^d, \pi_n^c \rangle$ for $n \in N$, as in the commuting diagram in $\mathbf{D}$ below.



The universality of the product is a consequence of the universality of the three products and the pullback used in the construction.

□

The proof of Theorem 4.2 follows from Lemma 4.16 using the assumption that **Sig** is (finitely) (co)complete and the fact that **Set** is also (finitely) (co)complete. □

Chapter 5

Signatures with Dependencies

5.1 Introduction

Dependencies between symbols are present in most typical signatures used for specifications. For example, in algebraic many-sorted signatures, sets of operation symbols are indexed by finite sequences of sort symbols (cf. Sect. 3.3.1). This yields natural dependency between operation symbols and their arity and result sorts. Another example are signatures of parameterised modules (cf. Sect. 3.4), where each symbol from the result signature is (potentially) dependent on all symbols from the parameters signature.

In this chapter we propose a formalism suitable to explicitly express dependencies between symbols in signatures. This is not only to capture the basic dependency relation derived from the structure of the signatures, but also to allow its extension.

5.2 Signatures with Dependencies

Most of the content of this section comes from [Mar12], where we introduced many-sorted algebraic signatures with dependency structure. Here we slightly generalise the approach to cover any signatures formalised as category **Sig** meeting the assumptions from Sect. 3.5.

5.2.1 Dependency Relation

We choose the relation suitable to represent dependencies between symbols and the properties of morphisms between them. Typical dependencies are transitive. The non-transitive dependencies (e.g. resulting from software

layering, cf. [GS94]) can be represented by information hiding and need not be addressed directly here. Before we impose further properties of the relation, we formalise transitive relations as R-sets (cf. Def. 3.1).

Morphisms between R-sets must be monotonic and somehow reflect the dependency. We choose the weak reflection of dependency structures described by the conditions of p-morphism (cf. Def. 3.2).

The table[1] below summarises the existence of (co)limits[2] in the category **Rset**$\downarrow$ and three of its full subcategories: **Preord**$\downarrow$, **Soset**$\downarrow$, and $\mathbf{Soset_b}\downarrow$ (cf. Def. 3.3). We do not include partial orders as a candidate for dependency relation, because the antisymmetry of partial orders causes that, in the case of the algebraic many-sorted signatures with partial-order dependencies, there are no coequalisers.

category	**relation**	**equal.**	**final obj.**	**non-empty product**	**coeq.**	**coprod.**
Rset$\downarrow$	transitive	yes	no?	no?	yes	yes
Preord$\downarrow$	preorder	yes	yes	no?	yes	yes
Soset$\downarrow$	strict order	yes	no	no?	no	yes
$\mathbf{Soset_b}\downarrow$	bounded strict order	yes	no	yes	yes	yes (finite)

We choose $\mathbf{Soset_b}\downarrow$ (cf. Def. 3.3), the category of bounded strictly ordered sets, as our category of dependency structures. The boundedness requirement is needed to show properties by structural induction on the dependency structure[3] (e.g. see the proof of Lemma 6.51 in the next chapter). It is also used to prove the existence of non-empty products (cf. [Mar12]) and coequalisers (cf. the proof of Theorem 5.1 below).

In this thesis we rely on existence of the finite colimits in $\mathbf{Soset_b}\downarrow$, there-

[1] "No?" in the table means that absence of the property is a plausible conjecture (cf. [Mar12]).

[2] The final object is the product of the empty set. In the table the final object and nonempty products are presented in separate columns, because in some categories apparently their existence does not coincide.

[3] It is needed, because we do not have a general assumption that signatures in **Sig** are finite, cf. the footnote to assumption (2) in Sect. 3.5.

fore, we show the following result.

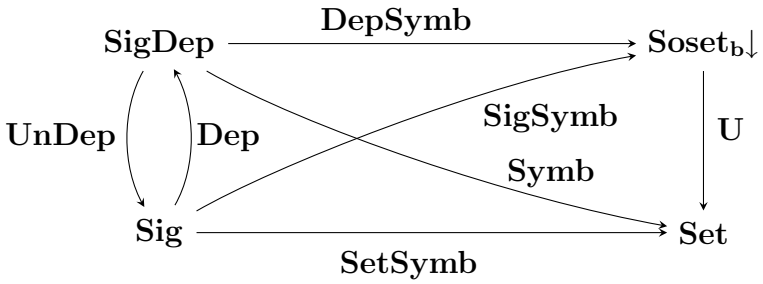
Theorem 5.1 *Category* $\mathbf{Soset_b}\downarrow$ *is finitely cocomplete.*

The construction of colimits in $\mathbf{Soset_b}\downarrow$ is essentially the same as in **Set**. There is no need for transitive closure of the strict order in the construction of the coequaliser. The coequaliser of two morphisms $f, g\colon A_< \to B_<$ in $\mathbf{Soset_b}\downarrow$ is $e\colon B_< \to C_<$, where e is the coequaliser of f and g in **Set** and the relation is simply given as $<_C = e(<_B)$. The transitivity of $<_C$ follows from the p-morphism conditions of f and g. The complete proof is in Appendix 5.A.

5.2.2 Signatures with Dependency Structure

As described in Sect. 3.5, we assume that **Sig** comes equipped with the *basic dependency* functor $\mathbf{SigSymb}\colon \mathbf{Sig} \to \mathbf{Soset_b}\downarrow$ that maps signatures to bounded sosets of their symbols. In examples we use the basic dependency functor $\mathbf{SigSymb_{AlgSig}}$ from Sect. 3.3.1.

Consider the following diagram of functors among different categories of signatures and their fragments. We define all functors and categories from the diagram.



The category **SigDep** is an extension of **Sig** by strict bounded dependency structure, with **SigDep**-objects being pairs $\Sigma_< = \langle \Sigma, <_\Sigma \rangle$ of a signature $\Sigma \in \mathbf{Sig}$ and a dependency relation $<_\Sigma \subseteq \mathbf{SetSymb}(\Sigma) \times \mathbf{SetSymb}(\Sigma)$ such that it is a bounded strict order that extends the basic dependency given by the functor **SigSymb**, i.e. $\langle \mathbf{SetSymb}(\Sigma), <_\Sigma \rangle \in \mathbf{Soset_b}\downarrow$ and for $\langle A, <_A \rangle = \mathbf{SigSymb}(\Sigma)$, it holds that $<_A \subseteq <_\Sigma$. The **SigDep**-morphisms

are signature morphisms $\sigma\colon \Sigma \to \Sigma'$ such that the function $\mathbf{SetSymb}(\sigma)$ is a $\mathbf{Soset_b}\downarrow$-morphism (cf. Def. 3.2 and Def. 3.3).

Notation: In examples we usually omit the index of the order where no confusion is possible, e.g. we write $<$ instead of $<_\Sigma$.

Example 5.2 *Consider the following algebraic signature with dependencies.*

$$\Sigma_< = \left|\begin{array}{ll} \textbf{sort} & Nat; \\ \textbf{ops} & zero\colon Nat, \\ & succ\colon Nat \to Nat, \\ & plus\colon Nat \times Nat \to Nat, \\ \textbf{deps} & succ < plus, \\ & zero < succ \end{array}\right|$$

$$\begin{array}{c} plus : Nat \times Nat \to Nat \\ \downarrow \\ succ : Nat \to Nat \\ \downarrow \\ zero : Nat \\ \downarrow \\ Nat \end{array}$$

*Here and in the examples that follow, the basic dependency (i.e., zero, succ and plus depend on Nat in this case) is omitted in our notation, whereas an additional dependency (in above example that are: plus depends on succ and succ depends on zero) is given after the keyword **deps**. Dependency relation $<_\Sigma$ is given as the transitive closure of the union of basic dependency and explicitly given additional dependency.*

Let **DepSymb**, **UnDep** and **U** be the obvious projections. We define functor $\mathbf{Dep}\colon \mathbf{Sig} \to \mathbf{SigDep}$ as the embedding based on the basic dependency relation, i.e. such that for any $\Sigma \in \mathbf{Sig}$, $\mathbf{Dep}(\Sigma) = \langle \Sigma, <_{\mathbf{SigSymb}(\Sigma)} \rangle$. Functor **Symb** is given as the composition of **UnDep** and **SetSymb**.

The *dependency bound of* $\Sigma_<$ is defined as the dependency bound of its underlying strictly ordered set (cf. Def. 3.4), i.e. $db(\Sigma_<) = db(\mathbf{DepSymb}(\Sigma_<))$.

Category **SigDep** has all finite colimits. The coequaliser of two morphisms $f, g\colon \langle \Sigma, <_\Sigma \rangle \to \langle \Sigma', <_{\Sigma'} \rangle$ in **SigDep** is $h\colon \langle \Sigma', <_{\Sigma'} \rangle \to \langle \Sigma'', <_{\Sigma''} \rangle$, where $h\colon \Sigma' \to \Sigma''$ is the coequaliser of f and g in **Sig**, and the bounded strict order $<_{\Sigma''} = \mathbf{SetSymb}(h)(<_{\Sigma'})$ (cf. Theorem 5.1 and its proof where we show that $<_{\Sigma''}$ is transitive, irreflexive and bounded; this is provided that **SetSymb** preserves the finite colimits, as assumed in Sect. 3.5). The initial object in **SigDep** is $\mathbf{Dep}(\Sigma_\emptyset)$, where $\Sigma_\emptyset$ is the initial signature in

Sig. Binary coproducts in **SigDep** are binary coproducts in **Sig** ordered by the union of the component orders. The resulting order is bounded by the maximum of the bounds of component orders. It is transitive, because the orders of components are transitive, and, since **SetSymb** preserves finite colimits, no symbols are shared in the coproduct object, as in **Set**. Other finite coproducts are defined in the same way.

Functor **Symb** preserves and reflects pushouts, because **UnDep** and **SetSymb** do.

The inclusion system of $\mathbf{Soset_b}\downarrow$ comes directly from the inclusion system of **Set**. Given a $\mathbf{Soset_b}\downarrow$-morphism $f\colon A_< \to B_<$ let $f_e\colon A \to C$ and $f_i\colon C \to B$ be the factorisation of $\mathbf{U}(f)$ in **Set**. We define $C_< = \langle C, <_B|_{f_i}\rangle$. It is easy to prove that $C_< \in \mathbf{Soset_b}\downarrow$ and that f_e and f_i are $\mathbf{Soset_b}\downarrow$-morphisms. In other words, $C_< \subseteq B_<$ iff $C \subseteq B$ and $<_C = <_B|_C$.

Category **SigDep** has the inclusion system derived from the inclusion systems of **Sig** and $\mathbf{Soset_b}\downarrow$. For a **SigDep**-morphism $\sigma\colon \Sigma_< \to \Sigma'_<$, the factorisation $\langle \sigma_e, \sigma_i\rangle$ of $\mathbf{UnDep}(\sigma)$ in **Sig** gives the factorisation of σ in **SigDep**. The proof is by straightforward use of the assumption that **SetSymb** preserves the inclusions (cf. Sect. 3.5).

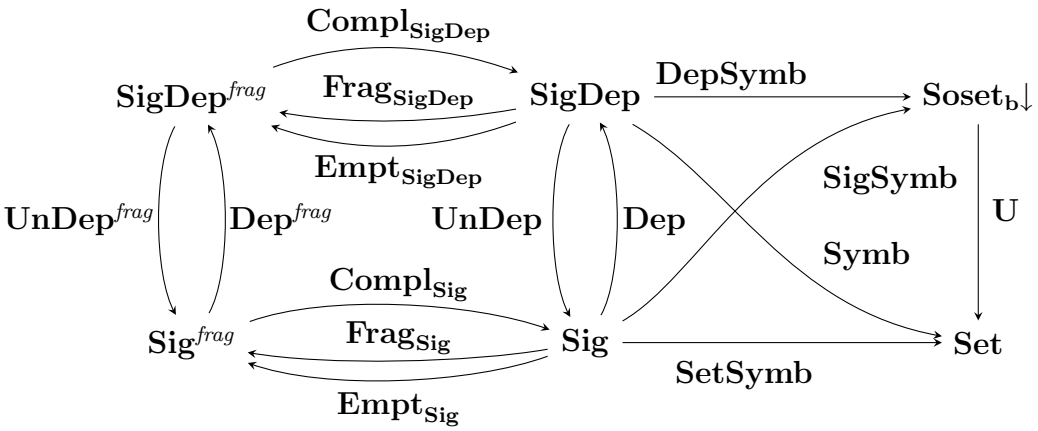
We say that two symbols of a signature with dependency structure $\Sigma_<$ are *independent* in $\Sigma_<$ if they are not related by $<_\Sigma$. A set of $\Sigma_<$-symbols is an *independent set* if its elements are pairwise independent.

5.3 Fragments of Signatures with Dependencies

Let $\mathbf{SigDep}^{frag}$ denote the category of *fragments of signatures with dependencies*. It is defined analogously to Def. 4.1 for **SigDep** (instead of **Sig**) as the concrete category with **Symb** (instead of **SetSymb**) as the concretisation functor. Objects in $\mathbf{SigDep}^{frag}$ are $\underline{\mathcal{S}} = \langle A, \Sigma_<, f\colon A \to \mathbf{Symb}(\Sigma_<)\rangle$. $\mathbf{SigDep}^{frag}$-morphisms between $\mathbf{SigDep}^{frag}$-objects $\underline{\mathcal{S}}$ and $\underline{\mathcal{S}}'$ are pairs $\underline{\sigma} = \langle g\colon A \to A', \sigma\colon \Sigma_< \to \Sigma'_<\rangle\colon \underline{\mathcal{S}} \to \underline{\mathcal{S}}'$, where g is a function and σ is a **SigDep**-morphism, such that $g; f' = f; \mathbf{Symb}(\sigma)$.

Notation. We use the underlined calligraphic font (e.g. $\underline{\mathcal{S}}$) to denote fragments of signatures with dependencies. Sometimes we implicitly assume that the internals of $\underline{\mathcal{S}}$ are given as $\langle A, \Sigma_<, f\colon A \to \mathbf{Symb}(\Sigma_<)\rangle$ and we use symbols A, f, Σ, $<_\Sigma$ directly without prior introduction. When the confusion is not possible we omit the indexes of dependency relations, e.g. we write $<$ instead of $<_\Sigma$.

Consider the following diagram that contains fragments-related categories and functors in addition to the categories and functors from the diagram in Sect. 5.2.2:



Category $\mathbf{Sig}^{frag}$ along functors $\mathbf{Frag}_{\mathbf{Sig}}$, $\mathbf{Compl}_{\mathbf{Sig}}$, and $\mathbf{Empt}_{\mathbf{Sig}}$ are defined in Sect. 4.2. In addition to category $\mathbf{SigDep}^{frag}$ (explicitly defined at the beginning of this section), the definition and the results from Sect. 4.2 applied to category **SigDep** (instead of **Sig**) and functor **Symb** (instead of **SetSymb**) in the obvious way yield functors $\mathbf{Frag}_{\mathbf{SigDep}}$, $\mathbf{Compl}_{\mathbf{SigDep}}$, and $\mathbf{Empt}_{\mathbf{SigDep}}$.

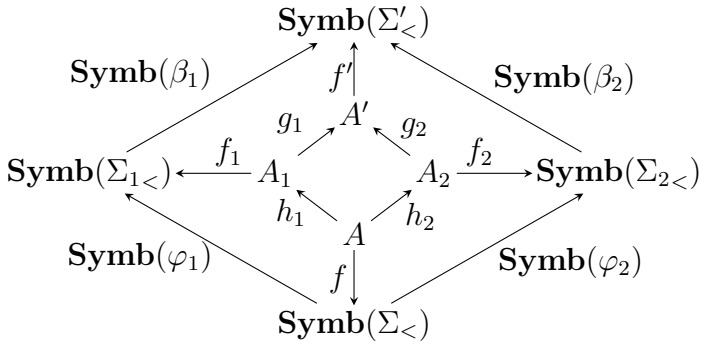
Notation. As it was already mentioned in Chapter 4, we omit the indices **Sig** and **SigDep** and simply write **Compl**, **Frag** and **Empt** where no confusion is possible.

Functor $\mathbf{Dep}^{frag}$ maps a $\mathbf{Sig}^{frag}$-object $\langle A, \Sigma, f\colon A \to \mathbf{SetSymb}(\Sigma)\rangle$ to a $\mathbf{SigDep}^{frag}$-object $\langle A, \mathbf{Dep}(\Sigma), f\colon A \to \mathbf{Symb}(\mathbf{Dep}(\Sigma))\rangle$ and a $\mathbf{Sig}^{frag}$-morphism to itself.

Functor $\mathbf{UnDep}^{frag}$ simply removes dependency relations, like **UnDep**. It maps an $\mathbf{SigDep}^{frag}$-object $\langle A, \Sigma_<, f\colon A \to \mathbf{Symb}(\Sigma_<)\rangle$ to a $\mathbf{Sig}^{frag}$-object

$\langle A, \mathbf{UnDep}(\Sigma_<), f\colon A \to \mathbf{SetSymb}(\mathbf{UnDep}(\Sigma_<))\rangle$. It is an identity on morphisms.

By Theorem 4.2 (with the definitions and results from Sect. 4.2 applied to **SigDep** and **Symb**, as indicated above), category $\mathbf{SigDep}^{frag}$ has all finite colimits. Let us explicitly give the construction of pushouts in $\mathbf{SigDep}^{frag}$. Consider two morphisms $\underline{\varphi_1}\colon \underline{\mathcal{S}} \to \underline{\mathcal{S}_1}$ and $\underline{\varphi_2}\colon \underline{\mathcal{S}} \to \underline{\mathcal{S}_2}$ such that $\underline{\varphi_1} = \langle h_1, \varphi_1\rangle$ and $\underline{\varphi_2} = \langle h_2, \varphi_2\rangle$. Their pushout is a pair of morphism $\underline{\beta_1}\colon \underline{\mathcal{S}_1} \to \underline{\mathcal{S}'}$ and $\underline{\beta_2}\colon \underline{\mathcal{S}_2} \to \underline{\mathcal{S}'}$ given as $\underline{\beta_1} = \langle g_1, \beta_1\rangle$ and $\underline{\beta_2} = \langle g_2, \beta_2\rangle$ where $\beta_1\colon \Sigma_{1<} \to \Sigma'_<$ and $\beta_2\colon \Sigma_{2<} \to \Sigma'_<$ form the pushout of φ_1 and φ_2 in **SigDep**, and $g_1\colon A_1 \to A'$ and $g_2\colon A_2 \to A'$ form the pushout of h_1 and h_2 in **Set**. The $\mathbf{SigDep}^{frag}$-object $\underline{\mathcal{S}'} = \langle A', \Sigma'_<, f'\colon A' \to \mathbf{Symb}(\Sigma'_<)\rangle$ has the inner mapping f' given as the universal morphism in **Set** for $f_1; \mathbf{Symb}(\beta_1)$ and $f_2; \mathbf{Symb}(\beta_2)$ w.r.t. the pushout of h_1 and h_2, as on the following diagram in **Set**.



Category $\mathbf{SigDep}^{frag}$ has the inclusion system built over the inclusion systems of **Set** and **SigDep**, like $\mathbf{Sig}^{frag}$ in Sect. 4.2 (cf. also diagonal-fill lemma in [Dia08]).

The *dependency bound of* $\underline{\mathcal{S}} = \langle A, \Sigma_<, f\colon A \to \mathbf{Symb}(\Sigma_<)\rangle$ is the dependency bound of the underlying signature with dependencies (cf. Sect. 5.2.2), i.e. $db(\underline{\mathcal{S}}) = db(\Sigma_<)$.

Notation. To simplify notation we write $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ instead of $A \subseteq \mathbf{Symb}(\mathbf{Compl}(\underline{\mathcal{S}}))$. Moreover, let us recall that defined symbols are considered as elements of the signature fragment, i.e.,we write $a \in \underline{\mathcal{S}}$ and this means that a is defined in $\underline{\mathcal{S}}$. Assumed symbols are not elements of the fragment, but they are elements of the completion of the fragment, i.e., for

an assumed a we write $a \in \mathbf{Compl}(\underline{\mathcal{S}})$, $a \notin \underline{\mathcal{S}}$. The same notation extends to sets of symbols and we write $A \subseteq \underline{\mathcal{S}}$ when for all $a \in A$, $a \in \underline{\mathcal{S}}$.

For a signature fragment $\underline{\mathcal{S}} = \langle A, \Sigma_<, f\colon A \to \mathbf{Symb}(\Sigma_<)\rangle \in \mathbf{SigDep}^{frag}$ and any symbol $b \in \mathbf{Symb}(\Sigma_<)$, let

$$S_b = \{a \in \mathbf{Symb}(\Sigma_<) \mid a < b\}$$

be the subset of $\mathbf{Symb}(\Sigma_<)$ containing symbols that b depends on. Let Σ_{S_b} be the reconstruction of a Σ-subsignature from S_b w.r.t. **SigSymb** (cf. assumptions in Sect. 3.5), by $\Sigma_<^{S_b}$ we denote $\Sigma_<$-subsignature $\Sigma_<^{S_b} = \langle \Sigma_{S_b}, <_{S_b}\rangle$, where $<_{S_b} = <_\Sigma|_{S_b}$. Of course we have $\mathbf{Symb}(\Sigma_<^{S_b}) = S_b$.

The *dependency structure below* b is defined as

$$\underline{\mathcal{S}}^b\Downarrow = \langle f^{-1}(S_b), \Sigma_<^{S_b}, f'\colon f^{-1}(S_b) \to \mathbf{Symb}(\Sigma_<^{S_b})\rangle \in \mathbf{SigDep}^{frag}\ .$$

where f' is the appropriate restriction of f.

Let T_b be the smallest subset of $\mathbf{Symb}(\Sigma_<)$ containing b such that it is closed-down w.r.t. $<_\Sigma$. It is defined as

$$T_b = S_b \cup \{b\}$$

Let $\Sigma_<^{T_b}$ be the signature with dependency structure reconstructed as a subsignature of $\Sigma_<$, analogously to $\Sigma_<^{S_b}$, but from T_b instead of S_b.

We define the *dependency structure of* b as

$$\underline{\mathcal{S}}^b\downarrow = \langle f^{-1}(T_b), \Sigma_<^{T_b}, f''\colon f^{-1}(T_b) \to \mathbf{Symb}(\Sigma_<^{T_b})\rangle \in \mathbf{SigDep}^{frag}$$

where f'' is the appropriate restriction of f. Notice that b is any symbol, whether defined or assumed.

By $\underline{\mathcal{S}}^B\Downarrow$ we denote the *dependency structure below a set of independent symbols* $B \subseteq \mathbf{Symb}(\Sigma_<)$. It is the union of dependency structures below all symbols in B. The requirement of independency of symbols in B is important, because only then $B \cap \mathbf{Symb}(\mathbf{Compl}(\underline{\mathcal{S}}^B\Downarrow)) = \emptyset$.

The *dependency structure of a set of symbols* $C \subseteq \mathbf{Symb}(\Sigma_<)$ is denoted

by $\underline{\mathcal{S}}^{C}\!\downarrow$ and is the union of dependency structures of all symbols in C. Note that symbols in C need not be independent.

To briefly summarize:

- $\underline{\mathcal{S}}^{a}\!\Downarrow$ is the dependency structure below a symbol a, *excluding* a,
- $\underline{\mathcal{S}}^{B}\!\Downarrow$ is the dependency structure below a set of independent symbols B, *excluding* B,
- $\underline{\mathcal{S}}^{a}\!\downarrow$ is the dependency structure of a symbol a, *including* a,
- $\underline{\mathcal{S}}^{B}\!\downarrow$ is the dependency structure of a set of symbols B, *including* B,
- $\underline{\mathcal{S}}^{\emptyset}\!\downarrow$ is the empty fragment of $\Sigma_{\emptyset}$, the initial signature in **Sig**, i.e.

$$\underline{\mathcal{S}}^{\emptyset}\!\downarrow = \langle \emptyset, \Sigma_{\emptyset<}, \emptyset \to \mathbf{Symb}(\Sigma_{\emptyset<}) \rangle$$

For any non-empty set $B \subseteq \mathbf{Symb}(\Sigma_{<})$ and $a \in B$ the following inclusions hold in $\mathbf{SigDep}^{frag}$:

$$\underline{\mathcal{S}}^{\emptyset}\!\downarrow \subseteq \underline{\mathcal{S}}^{a}\!\Downarrow \subset \underline{\mathcal{S}}^{a}\!\downarrow \subseteq \underline{\mathcal{S}}^{B}\!\downarrow \subseteq \underline{\mathcal{S}}$$

Notation. In the text that follows we stick to notation simplifications concerning the use of functors from the diagram in the beginning of this section. We sometimes omit them, but only when the identity of the omitted functor is clear from the context. For example, given $\underline{\mathcal{S}} \in \mathbf{SigDep}^{frag}$, we write $[\![\underline{\mathcal{S}}]\!]$ and $\mathbf{Spec}(\underline{\mathcal{S}})$ instead of $[\![\mathbf{UnDep}(\mathbf{Compl}(\underline{\mathcal{S}}))]\!]$ and $\mathbf{Spec}(\mathbf{UnDep}(\mathbf{Compl}(\underline{\mathcal{S}})))$, respectively.

We also use (sub)objects instead of inclusion morphisms, e.g. given a model $M \in [\![\underline{\mathcal{S}}]\!]$ and a subsignature $\underline{\mathcal{S}}' \subseteq \underline{\mathcal{S}}$ we write $M|_{\underline{\mathcal{S}}'}$ instead of $M|_{\mathbf{UnDep}(\mathbf{Compl}(\iota))}$, where $\iota\colon \underline{\mathcal{S}}' \to \underline{\mathcal{S}}$ is the corresponding inclusion morphism. Similarly, for a specification $SP \in \mathbf{Spec}(\underline{\mathcal{S}})$ we simply write $SP|_{\underline{\mathcal{S}}'}$ to denote the hiding of SP via the morphism $\mathbf{UnDep}(\mathbf{Compl}(\iota))$.

5.A Appendix: Proofs

Proof of Theorem 5.1. It is enough to show the existence of the coequalisers and the initial object. The lemma given below is used in the proof of this theorem.

Lemma 5.3 *Given two* $\mathbf{Soset_b}\downarrow$*-morphisms* $f, g\colon A_< \to B_<$ *let there be a relation* $\sim \subseteq B \times B$ *defined as* $b \sim b'$ *iff there exists* $a \in A$ *such that* $b = f(a)$ *and* $b' = g(a)$*. By* $\equiv$ *we denote the reflexive, symmetric and transitive closure of* $\sim$*. It holds that for any* $b_1, b_2 \in B$*, if* $b_1 <_B b_2$ *then for any* $b_2' \in B$ *such that* $b_2' \equiv b_2$*, there exists* $b_1' \in B$ *such that* $b_1' \equiv b_1$ *and* $b_1' <_B b_2'$*.*

Proof. Let there be $b_1 <_B b_2$ and $b_2' \equiv b_2$. Notice that $b_2 \equiv b_2'$ iff $b_2\ (\sim \cup \sim^{-1})^n\ b_2'$ where n is the length of a sequence of relations ($\sim$ or $\sim^{-1}$) between b_2 and b_2'. Let us prove the lemma by induction on n.

In the base case, when $n = 0$, we have $b_2 = b_2'$, thus b_1 is such that $b_1 \equiv b_1$ and $b_1 <_B b_2$, as required.

In the induction step let $n = i{+}1$, i.e., $b_2\ (\sim \cup \sim^{-1})^{i+1}\ b_2'$, assume that the lemma works for sequences of the length i. Without loss of generality assume that there is $a_2 \in A$ such that $f(a_2) = b_2$ and $g(a_2)\ (\sim \cup \sim^{-1})^i\ b_2'$. This of course makes $g(a_2) \equiv b_2'$. Since f is an $\mathbf{Rset}\downarrow$-morphism, by requirement (2) of Def. 3.2, there exists $a_1 \in A$ such that $f(a_1) = b_1$ and $a_1 <_A a_2$. Since g is also an $\mathbf{Rset}\downarrow$-morphism, by requirement (1) of the same definition, it is monotone, i.e., $g(a_1) <_B g(a_2)$. By definition of $\sim$ it holds that $b_1 \sim g(a_1)$, thus $g(a_1) \equiv b_1$. From the inductive assumption we can use the lemma for sequences of the length i. The lemma applied to $b_2' \equiv g(a_2)$ such that $g(a_1) <_B g(a_2)$ gives us the existence of $b_1' \equiv g(a_1)$ such that $b_1' <_B b_2'$. From $g(a_1) \equiv b_1$ and $b_1' \equiv g(a_1)$, by symmetry and transitivity of $\equiv$, we get $b_1' \equiv b_1$, as required. □

Let $\equiv$ be the equivalence relation from Lemma 5.3. The coequaliser of two morphisms $f, g\colon A_< \to B_<$ in $\mathbf{Soset_b}\downarrow$ is $e\colon B_< \to C_<$, where e is the coequaliser of f and g in **Set** with C defined as $B/\equiv$ and the relation $<_C\ = e(<_B)$. We prove that $<_C$ is a bounded strict order (irreflexive, transitive

and bounded) and that e is a $\mathbf{Soset_b}\downarrow$-morphism. We skip the detailed proof of the coequaliser properties, because they follow easily from the properties of e as the coequaliser of f and g in **Set** and the fact that e is surjective (by the definition of C).

The irreflexivity of $<_C$ follows from the fact that for any $b_1, b_2 \in B$, if $b_1 < b_2$ then $b_1 \not\equiv b_2$. In order to prove this, let us assume that there are $b_1, b_2 \in B$ such that $b_1 < b_2$. The proof goes by induction on the length of the descending chain lower to b_1 w.r.t. $<$ (the chain is finite, because $<_B$ is bounded). In the base case, let for all $b \in B$, $b \not< b_1$ and suppose $b_1 \equiv b_2$. By Lemma 5.3, since $b_1 \equiv b_2$ and $b_1 < b_2$, there must exist $b_1' \in B$ such that $b_1' \equiv b_1$ and $b_1' < b_1$; contradiction. In the induction step, let us assume that for all $b \in B$, if $b < b_1$ then $b \not\equiv b_1$. Again suppose $b_1 \equiv b_2$ and again by Lemma 5.3, since $b_1 \equiv b_2$ and $b_1 < b_2$, we get the existence of $b_1' \in B$ such that $b_1' < b_1$ and $b_1' \equiv b_1$; contradiction. Therefore, $<_C$ is irreflexive.

The transitivity of $<_C$ is not obvious, because there is no transitive closure in its definition. To prove this we use the transitivity of $<_B$ and the fact that if there are $b_1, b_2, b_3, b_4 \in B$ such that $b_1 <_B b_2$ and $b_3 <_B b_4$ and $e(b_2) = e(b_3)$ then $e(b_1) <_C e(b_4)$. This follows from Lemma 5.3, which gives us the existence of $b_1' \in B$ such that $b_1' \equiv b_1$ and $b_1' <_B b_3$; then, by transitivity of $<_B$, we get $b_1' <_B b_4$, hence $e(b_1) = e(b_1') <_C e(b_4)$.

The relation $<_C$ is bounded, because it is irreflexive and it is defined as $e(<_B)$, where $<_B$ is bounded and e is surjective.

The two conditions from Def. 3.2, needed to prove that e is a $\mathbf{Soset_b}\downarrow$-morphism, are easily discharged by the observation that e is monotone (by the definition of $<_C$) and surjective (by the definition of C).

The empty set ordered by the empty relation is an initial object in $\mathbf{Soset_b}\downarrow$. A binary coproduct of $\langle A, <_A\rangle$ and $\langle B, <_B\rangle$ is $\langle A \uplus B, <_A \uplus <_B\rangle$. Other finite coproducts are defined in the same way. Not all infinite coproducts exist in $\mathbf{Soset_b}\downarrow$, because the resulting structure may be not bounded.

□

Chapter 6

Constructions

6.1 Introduction

This chapter introduces *constructions*, a notion that uniformly covers non-parameterised and parameterised modules and constitute basic building blocks of architectural decomposition of systems.

As already discussed in Sect. 3.4, typically models of parameterised modules map (parameter signature) models to (result signature) models, so the dependency between the parameter and the result is external to the parameter and result signatures. In our approach the dependencies between symbols are encoded into construction signatures. Internally, inside a construction signature, we mark the result symbols (called defined) leaving the rest as parameter symbols (called assumed). Such a representation of construction signatures eliminates the distinction between the signatures of non-parameterised and parameterised modules. Interestingly, it also handles uniformly the first- and higher-order parameterisation.

The internal dependency relation between symbols is more fine-grained than external definition of a parameter signature and a result signature. It allows one to partially instantiate a construction and to use the partial result, even though not all parameter symbols are given to the instantiation operation.

We define a sum of two construction signatures in such a way that it guarantees that the sharing of symbols is explicit and only possible via assumed symbols. This means that if composed constructions share a symbol, it must not be defined in both construction signatures of the components. As a consequence of this natural assumption we get an explicit information about the origin of the definition for every symbol in the composite construction

signature.

The content of this chapter is provided independently of any particular choice of the base institution $\mathbb{I}$ that satisfies all assumptions listed in Sect. 3.5.

6.2 Construction Signatures

Construction signatures are defined as injective fragments of signatures with dependency structure.

Definition 6.1 (Construction Signature) *A* construction signature *is an injective fragment of a signature with dependencies* $\underline{\mathcal{S}} \in \mathbf{SigDep}^{frag}$*, i.e., for* $\underline{\mathcal{S}} = \langle A, \Sigma_<, f\colon A \to \mathbf{Symb}(\Sigma_<)\rangle$*,* f *is an injection (cf. Sect. 5.3). A* construction signature morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ *is a* $\mathbf{SigDep}^{frag}$*-morphism, so it is such that* $\mathbf{DepSymb}(\mathbf{Compl}(\underline{\omega}))$ *is a p-morphism.*

Defined symbols of a construction signature indicate its *result part*, and assumed symbols constitute the *parameter part* of a construction. Symbols from the parameter part are sometimes called *parameters*. The dependency structure of a symbol lists all symbols that may be given during its construction.

A *complete construction signature* is a construction signature such that all its symbols are defined (cf. complete fragments in Sect. 4.2), therefore, the whole signature is the result part and there are no parameters. Complete construction signatures correspond to signatures of simple (non-parameterised) modules.

An *empty construction signature* has all symbols assumed (cf. empty fragments in Sect. 4.2), i.e. it has only a parameter part.

Let us remind the reader of the p-morphism requirements (cf. Def. 3.2) posed on every construction signature morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$:

1. *(monotonic)* For all $b_1, b_1' \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$, if $b_1' < b_1$ then $\underline{\omega}(b_1') < \underline{\omega}(b_1)$;
2. *(weakly reflect <)* for all $b_1 \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$, $b_2' \in \mathbf{Compl}(\underline{\mathcal{S}}_2)$, if $b_2' < \underline{\omega}(b_1)$, then there exists $b_1' \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$ such that $b_1' < b_1$ and $\underline{\omega}(b_1') = b_2'$.

The above-given conditions guarantee preservation of the dependency structure and that no new symbols are either added to or removed from the dependency structure of any symbol. The idea is that each symbol of a construction signature is inseparable from its dependency structure and construction signature morphisms map it only to symbols with bisimilar dependency structure (cf. Def. 3.2). In Sect. 7.2 we will define another kind of morphisms[1] between construction signatures, allowing addition of new symbols to the dependency structure.

Notation. In what follows, examples of algebraic many-sorted construction signatures use the syntax of algebraic many-sorted signatures (cf. Sect. 3.3) extended in the following way (as in Example 5.2): (1) assumed symbols are underlined; (2) defined symbols are not underlined; (3) similarly to the notation used in example in Sect. 5.2.2, the additional dependency structure is given after the keyword **deps**; (4) basic dependency is implicit and may be omitted. Dependency relation $<_\Sigma$ is given as the transitive closure of the union of the basic dependency and the explicitly given additional dependency.

6.2.1 Signatures of Modules as Construction Signatures

The construction signatures are suitable to represent signatures of parameterised and simple (non-parameterised) modules. In this section we give an exemplary conversion from module signatures to construction signatures. We begin by presenting the conversion for simple modules and later on for parameterised modules.

A signature of a non-parameterised module is usually given as a signature $\Sigma \in \mathbf{Sig}$. The corresponding construction signature is $\underline{\mathcal{S}_\Sigma} = \mathbf{Frag}(\mathbf{Dep}(\Sigma))$. All symbols in $\underline{\mathcal{S}_\Sigma}$ are defined, i.e. it is a complete construction signature.

In the standard approach a signature of a parameterised module is a signature morphism $\sigma\colon \Sigma_P \to \Sigma_R$ (cf. Sect. 3.4), where Σ_P is a signature of the module parameter and Σ_R is a signature of the module result.

Let A_σ and D_σ, denoting the set of symbols assumed (parameters) and

[1] Construction signature refinement morphisms defined in Def. 7.1.

defined by σ, respectively, be given as

$$A_\sigma = \mathbf{SetSymb}(\sigma)(\mathbf{SetSymb}(\Sigma_P)),$$
$$D_\sigma = \mathbf{SetSymb}(\Sigma_R) \setminus A_\sigma.$$

The construction signature corresponding to σ is given as

$$\underline{\mathcal{S}}_\sigma = \langle D_\sigma, \Sigma_{R<}, \iota\colon D_\sigma \to \mathbf{Symb}(\Sigma_{R<})\rangle,$$

where $\Sigma_{R<} = \langle \Sigma_R, <_R\rangle$ with $<_R \subseteq \mathbf{SetSymb}(\Sigma_R)\times\mathbf{SetSymb}(\Sigma_R)$ is defined as the smallest strict order such that:

1. $<_R$ includes the basic dependency of Σ_R,
2. for any $a_p \in A_\sigma$ and any $a_d \in D_\sigma$, it holds that $a_p <_R a_d$, i.e. every defined symbol depends on all assumed (parameter) symbols.

Relation $<_R$ is a bounded strict order, because the bound of $\underline{\mathcal{S}}_\sigma$ is at most $2n+1$, where n is the bound of the basic dependency of Σ_R.

The above-described conversion assumes maximal dependency between result and parameter symbols. Moreover, it does not pose any additional dependency between assumed symbols. At this level of generality this is the only sensible approach; however, given more detailed information about symbol inter-dependencies (e.g. from the specifications) the dependency structure may be fine-tuned during the conversion (cf. Sect. 6.4.3 below for a discussion about additional dependencies between assumed symbols during the conversion of parameterised module specification into a construction specification).

Example 6.2 *Consider a signature of parameterised module given as inclusion*

$$\sigma_1\colon (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s) \to (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ b : s).$$

The corresponding construction signature is

$$\underline{\mathcal{S}}_1 = (\mathbf{sort}\ \underline{s};\ \mathbf{ops}\ \underline{a : s},\ b : s;\ \mathbf{dep}\ a < b)$$

In some approaches, like ACT2 (cf. [EM90]), signatures of parameterised modules are of an extended form and contain also import and export signatures. In general, it is only possible to convert such extended module signatures to simple diagrams of constructions (cf. Sect. 8.2 below). We leave this task for future work.

A construction signature $\underline{\mathcal{S}}$ corresponds[2] to a (first-order) parameterised module signature if no assumed symbol in $\underline{\mathcal{S}}$ depends on a defined symbol in $\underline{\mathcal{S}}$. For example, the construction signature

$$(\textbf{sorts } \underline{s}, t;\ \textbf{ops } \underline{a : s},\ b : t;\ \textbf{deps } a < b,\ a < t)$$

corresponds to the parameterised module signature given as the inclusion

$$(\textbf{sort } s;\ \textbf{ops } a : s) \to (\textbf{sort } s, t; \textbf{ops } a : s,\ b : t).$$

In general, the conversion from constructions to parameterised modules is possible only if we allow for higher-order parameterisation. For instance, the construction signature (**sort** s; **op** $\underline{a : s}$) has no corresponding first-order parameterised module signature, because it is parametric on operation $a : s$ that in turn is parametric on sort s. The corresponding higher order parameterised module would have the following signature given as the inclusion $((\textbf{sort } s) \to (\textbf{sort } s;\ \textbf{op } a : s)) \to (\textbf{sort } s;\ \textbf{op } a : s)$. More examples follow.

Examples of the higher-order parameterised module signatures are given here for illustration only (in particular in Example 6.4 below). In our thesis we have no intend to analyse or formalise higher-order parameterised module signatures otherwise than by investigations of our constructions and their signatures.

Example 6.3 *To reverse the conversion from Example 6.2, consider con-*

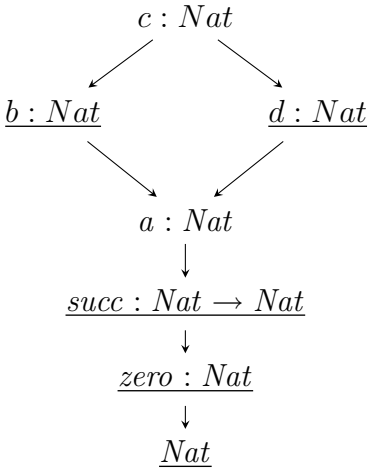
[2]The term "corresponds" is used semi-formally here, because the dependency structure need not be preserved and reflected by this correspondence. Some dependencies may disappear, because all dependencies among defined symbols and among assumed symbols in $\underline{\mathcal{S}}$ that are beyond the basic dependency of $\underline{\mathcal{S}}$ are not present in the corresponding parameterised module signature. Some new dependencies may be formed, because each defined symbol in $\underline{\mathcal{S}}$ becomes dependent on all assumed symbols of **AlgSig** in the corresponding parameterised module signature.

struction signature $\underline{\mathcal{S}}_1$ given there. Sort s and operation a are assumed; they constitute the parameter part of the construction signature. Operation b is defined, thus b is the result part of $\underline{\mathcal{S}}_1$. Symbol b is dependent on s and a, therefore, as expected, construction signature $\underline{\mathcal{S}}_1$ corresponds to the signature of parameterised module σ_1 from Example 6.2.

Example 6.4 *Consider another construction signature and a graph representation of its dependency structure.*

$\underline{\mathcal{S}}_2 =$

sort	$\underline{Nat}$;
ops	$\underline{zero}\colon Nat$,
	$\underline{succ}\colon Nat \to Nat$,
	$a\colon Nat,\ \underline{b}\colon Nat$,
	$c\colon Nat,\ \underline{d}\colon Nat$;
deps	$zero < succ$,
	$succ < a$,
	$a < b,\ a < d$,
	$b < c,\ d < c$



There are two defined and five assumed symbols. Any construction over $\underline{\mathcal{S}}_2$ provides an implementation of a and c. All other symbols are parameters. The dependency structure indicates exactly which symbols can be potentially used to construct other symbols. For instance, symbols b and d are potentially needed to construct symbol c, but a is constructed independently of b and d. Sort Nat together with operations zero and succ are assumed and they do not depend on any defined symbol; therefore, they correspond to first-order parameters in signatures of parameterised modules. Symbol a is defined and depends on the first-order parameter, i.e., it is enough to provide the implementation of Nat, zero and succ for a to be uniquely constructed. Symbols b and d are assumed and depend on defined a; this makes b and d second-order parameters. Potential implementations of b and d may depend only on a (and its dependency structure). Symbol c is defined and depends on

both second-order parameters b and d (as well as on a and the first-order parameters).

Construction signature $\underline{\mathcal{S}}_2$ corresponds to the signature of the higher order parameterised module given below.

$$\begin{aligned}
&(((\textbf{sort } Nat;\ \textbf{ops } zero\colon Nat, succ\colon Nat \to Nat)\\
&\qquad \to (\textbf{sort } Nat;\ \textbf{ops } zero\colon Nat, succ\colon Nat \to Nat,\ a : Nat))\\
&\quad \to (\textbf{sort } Nat;\ \textbf{ops } zero\colon Nat, succ\colon Nat \to Nat,\\
&\qquad\qquad a : Nat,\ b : Nat,\ d : Nat))\\
&\to (\textbf{sort } Nat;\ \textbf{ops } zero\colon Nat, succ\colon Nat \to Nat,\\
&\qquad\qquad a : Nat,\ b : Nat,\ c : Nat,\ d : Nat)
\end{aligned}$$

6.3 Construction Models

The definition of construction models reflects the intuition that all defined symbols in the construction signature are uniquely constructed for every model of dependency structure below them.

Definition 6.5 (Construction Model) A construction model of a construction signature $\underline{\mathcal{S}}$ *is a class of models (from the base institution $\mathbb{I}$ described in Sect. 3.5) $Con \subseteq [\![\mathbf{UnDep}(\mathbf{Compl}(\underline{\mathcal{S}}))]\!]$ such that, for any defined symbol $a \in \underline{\mathcal{S}}$ and any two models $M, M' \in Con$, if $M|_{\underline{\mathcal{S}}^{a\Downarrow}} = M'|_{\underline{\mathcal{S}}^{a\Downarrow}}$ then $M|_{\underline{\mathcal{S}}^{a\downarrow}} = M'|_{\underline{\mathcal{S}}^{a\downarrow}}$.*

The definition shall be read as follows. For each defined symbol a in the construction signature, for any implementation of the dependency structure below a existing in the construction model, there is (in the construction model) *a unique construction* of the model of dependency structure of a (including a).

Notation. By $[\![\underline{\mathcal{S}}]\!]^c$ we denote the class of all construction models of $\underline{\mathcal{S}}$. We use notation simplifications introduced at the end of Sect. 5.3. For example we write $[\![\underline{\mathcal{S}}]\!]$ instead of $[\![\mathbf{UnDep}(\mathbf{Compl}(\underline{\mathcal{S}}))]\!]$ and $M|_{\underline{\mathcal{S}}^{a\downarrow}}$ instead of clumsy

$M|_{\mathbf{UnDep}(\mathbf{Compl}(\iota_{\underline{\mathcal{S}}^a\downarrow}))}$, where $\underline{\iota_{\underline{\mathcal{S}}^a\downarrow}}$ is the $\mathbf{SigDep}^{frag}$-morphism for inclusion $\underline{\mathcal{S}}^a\downarrow \subseteq \underline{\mathcal{S}}$.

It is easy to check that, due to the implicative nature of the above definition, the empty class of models is a construction model for any $\underline{\mathcal{S}}$, i.e. $\emptyset \in [\![\underline{\mathcal{S}}]\!]^c$. Such construction models are called trivial.

Example 6.6 *Consider the construction signature*

$$\underline{\mathcal{S}}_3 = (\mathbf{sort}\ \underline{s};\ \mathbf{ops}\ a : s,\ \underline{b : s},\ c : s;\ \mathbf{deps}\ a < b,\ b < c)$$

and let there be two $\mathbf{Compl}(\underline{\mathcal{S}}_3)$*-models*

$$M_1 = (s = Nat,\ a : s = 5,\ b : s = 10,\ c : s = 0)$$
$$M_2 = (s = Nat,\ a : s = 5,\ b : s = 15,\ c : s = 1))$$

where Nat denotes the set of natural numbers. It is easy to check that the class of models $Con = \{M_1, M_2\}$ *is a construction model of* $\underline{\mathcal{S}}_3$*, i.e.* $Con \in [\![\underline{\mathcal{S}}_3]\!]^c$*. The different values of defined symbol c in* M_1 *and* M_2 *do not pose a problem, because the models of dependency structures below c are also different in them.*

Let us now give two non-examples of $\underline{\mathcal{S}}_3$*-construction models. Let there be two other* $\mathbf{Compl}(\underline{\mathcal{S}}_3)$*-models*

$$M_3 = (s = Nat,\ a : s = 6,\ b : s = 10,\ c : s = 0)$$
$$M_4 = (s = Nat,\ a : s = 5,\ b : s = 10,\ c : s = 1)$$

The first non-example is the class of models $Con' = \{M_1, M_2, M_3\}$ *that is not a construction model of* $\underline{\mathcal{S}}_3$*, because the dependency structures below symbol a are equal in* M_1 *and* M_3 *(sort s is Nat in both models), i.e.* $M_1|_{\underline{\mathcal{S}}^a\Downarrow} = M_3|_{\underline{\mathcal{S}}^a\Downarrow}$*, but the dependency structures of a are different (*$a = 6$ *in* M_1 *and* $a = 5$ *in* M_3*), i.e.* $M_1|_{\underline{\mathcal{S}}^a\downarrow} \neq M_3|_{\underline{\mathcal{S}}^a\downarrow}$

The second non-example is $Con'' = \{M_1, M_2, M_4\}$*.* Con'' *is not a construction model of* $\underline{\mathcal{S}}_3$*, because the value of defined symbol c differs in* M_1 *and* M_4 *even though the two models agree on the dependency structure below c.*

Directly from the definition we have that in a nontrivial construction model *Con*, any defined symbol that is not dependent on assumed symbols is interpreted in the same way in all models in *Con*. Therefore, if the considered construction signature is complete, *Con* is a singleton. This means that there is one-to-one correspondence between nontrivial construction models of a complete construction signature $\underline{\mathcal{S}} \in \mathbf{SigDep}^{frag}$ and models of $\mathbf{Compl}(\underline{\mathcal{S}}) \in \mathbf{Sig}$.

Reduct of construction models is defined pointwise.

Definition 6.7 (Reduct of Construction Model) *Consider a construction signature morphism* $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$*, for some construction signatures* $\underline{\mathcal{S}}_1$ *and* $\underline{\mathcal{S}}_2$*. The* reduct of an $\underline{\mathcal{S}}_2$-construction model Con_2 w.r.t. ω *is the* $\underline{\mathcal{S}}_1$*-construction model* $Con_2|_{\underline{\omega}} = \{M|_{\underline{\omega}} \mid M \in Con_2\}$.

Notation. Following the notation simplifications introduced at the end of Sect. 5.3 we write $M|_{\underline{\omega}}$ instead of the bulky $M|_{\mathbf{UnDep}(\mathbf{Compl}(\underline{\omega}))}$.

The following lemma shows that the result of the reduct operation is indeed a construction model.

Lemma 6.8 *The reduct of a construction model w.r.t. a construction signature morphism is a construction model.*

The proof uses the fact that construction signature morphisms are p-morphism, see Appendix 6.A for details.

The requirements posed on construction models simply guarantee that the construction of defined symbols is solely parametric on their dependencies. Nothing is said with regard to assumed symbols and possibility of partial instantiation. Consider the following example.

Example 6.9 *Let there be the following construction signature*

$$\underline{\mathcal{S}}_4 = (\mathbf{sort}\ s;\ \mathbf{ops}\ \underline{a : s},\ \underline{b : s})$$

and the construction model of $\underline{\mathcal{S}}_4$

$$\begin{aligned} Con_4 = \{&(s = Nat,\ a : s = 1,\ b : s = 1),\\ &(s = Nat,\ a : s = 0,\ b : s = 0),\\ &(s = Nat,\ a : s = 1,\ b : s = 0)\} \end{aligned}$$

Symbols a *and* b *are independent in* $\underline{\mathcal{S}}_4$*. The set of values for* a *and* b *is* $\{0, 1\}$*; however, after the symbol* a *is instantiated to* 0 *(becomes defined), the set of values for* b *is reduced only to* $\{0\}$*. This poses an undesired dependency between* a *and* b*. A construction model that is ready for partial instantiation should contain models of all possible combinations of values of its assumed symbols.*

The concept of *well-grouped construction models* is a response to the need depicted by the above example. Informally speaking, a well-grouped construction model contains models with all independent combinations of assumed (parameter) symbols.

Definition 6.10 (Well-grouped Construction Model) *Given a construction signature* $\underline{\mathcal{S}}$ *and a construction model* $Con \in [\![\underline{\mathcal{S}}]\!]^c$*, we say that* Con *is* a well-grouped construction model *iff for all* $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ *and all* $M \in [\![\underline{\mathcal{S}}]\!]$*, if for all* $a \in A$*,* $M|_{\underline{\mathcal{S}}^{a\downarrow}} \in Con|_{\underline{\mathcal{S}}^{a\downarrow}}$*, then* $M|_{\underline{\mathcal{S}}^{A\downarrow}} \in Con|_{\underline{\mathcal{S}}^{A\downarrow}}$*.*

Example 6.11 *(cont. of Example 6.9) Construction model* Con_4 *is not a well-grouped construction model. The following construction model of* $\underline{\mathcal{S}}_4$ *is well-grouped*

$$\begin{aligned} Con'_4 = \{&(s = Nat,\ a : s = 1,\ b : s = 1),\ (s = Nat,\ a : s = 0,\ b : s = 0),\\ &(s = Nat,\ a : s = 1,\ b : s = 0), (s = Nat,\ a : s = 0,\ b : s = 1)\} \end{aligned}$$

6.3.1 Models of Modules as Construction Models

A model of a non-parameterised module signature Σ is simply a Σ-model M. The corresponding construction model of $\underline{\mathcal{S}}_\Sigma$ (cf. Sect. 6.2.1) is a singleton

class containing M,

$$Con_M = \{M\} \ .$$

Given a signature of parameterised module $\sigma\colon \Sigma_P \to \Sigma_R$, a persistent parameterised module model of σ is a (partial) map of models $\kappa\colon [\![\Sigma_P]\!] \to [\![\Sigma_R]\!]$ such that for all $M \in dom(\kappa)$, $\kappa(M)|_\sigma = M$ (cf. Sect. 3.4).

The corresponding construction model is the range of κ,

$$Con_\kappa = \{\kappa(M) \mid M \in dom(\kappa)\} \ .$$

Con_κ is a construction model of construction signature $\underline{\mathcal{S}}_\sigma$ (cf. Sect. 6.2.1), because in $\underline{\mathcal{S}}_\sigma$ every defined symbol depends on all assumed (parameter) symbols, so the requirement from Def. 6.5 is discharged easily.

Note that Con_κ in general does not need to be well-grouped, but it is well-grouped when the domain of κ forms a well-grouped model of Σ_P with all symbols assumed; in particular, Con_κ is well-grouped when κ is total.

Every construction model $Con \in [\![\underline{\mathcal{S}}_\sigma]\!]^c$ can be represented as a persistent parameterised module model of σ,

$$\kappa_\sigma = \{\langle M|_\sigma, M\rangle \mid M \in Con\}.$$

6.4 Construction Specifications

As it has been already explained, a construction signature contains assumed symbols that form its parameter part and defined symbols that are its result part. Similarly to construction signatures, construction specifications describe the construction as a whole, uniformly in non-parameterised and parameterised cases.

This approach differs from the typical specification of (first-order) parameterised modules consisting usually of separate specifications of the parameter and of the result.

Definition 6.12 *A* construction specification $\underline{SP}$ *over a construction signature* $\underline{\mathcal{S}}$ *is a pair* $\langle \underline{\mathcal{S}}, SP\rangle$*, where* $SP \in \mathbf{Spec}(\mathbf{UnDep}(\mathbf{Compl}(\underline{\mathcal{S}})))$.

Notation: When no confusion is possible, we omit the projection and write $\underline{SP}$ for $\pi_2(\underline{SP})$, that is the specification SP.

The distinction between assumed and defined symbols plays a crucial role in the definition of satisfaction relation. The below-given requirements expose the difference between them coherently with the intuition that the assumed and defined symbols are the parameter part and the result part of the construction signature, respectively.

Definition 6.13 (Satisfaction Relation) *A construction model* $Con \in [\![\underline{\mathcal{S}}]\!]^c$ *is a model of a construction specification* $\underline{SP}$ *over* $\underline{\mathcal{S}}$*, denoted* $Con \models^c \underline{SP}$*, iff*

1. *(construction) for all defined* $a \in \underline{\mathcal{S}}$ *and all* $M \in Con$*,*
   if $M|_{\underline{\mathcal{S}}^{a\Downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\Downarrow}}$ *then* $M|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$*,*
2. *(completeness) for all assumed* $a \in \mathbf{Compl}(\underline{\mathcal{S}})$*,* $a \notin \underline{\mathcal{S}}$*, and all* $M \models \underline{SP}$*,*
   if $M|_{\underline{\mathcal{S}}^{a\Downarrow}} \in Con|_{\underline{\mathcal{S}}^{a\Downarrow}}$ *then* $M|_{\underline{\mathcal{S}}^{a\downarrow}} \in Con|_{\underline{\mathcal{S}}^{a\downarrow}}$*,*
3. *(grouping) Con is a well-grouped construction model (cf. Def. 6.10),*
4. *(Con-dependency-wise) for all* $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ *and all* $M \in Con$*,*
   if for all $a \in A$*,* $M|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$ *then* $M|_{\underline{\mathcal{S}}^{A\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A\downarrow}}$*.*

Notation: By $[\![\underline{SP}]\!]^c$ we denote the class of all construction models of $\underline{SP}$. In the definition above we use notation simplifications introduced at the end of Sect. 5.3 and write $M|_{\underline{\mathcal{S}}^{a\downarrow}}$ and $M|_{\underline{\mathcal{S}}^{a\Downarrow}}$ instead of $M|_{\mathbf{UnDep}(\mathbf{Compl}(\underline{\iota_{\mathcal{S}^{a\downarrow}}}))}$ and $M|_{\mathbf{UnDep}(\mathbf{Compl}(\underline{\iota_{\mathcal{S}^{a\Downarrow}}}))}$, respectively. Here $\underline{\iota_{\mathcal{S}^{a\Downarrow}}}$ and $\underline{\iota_{\mathcal{S}^{a\downarrow}}}$ are defined as the $\mathbf{SigDep}^{frag}$-morphisms for inclusions $\underline{\mathcal{S}^{a}\Downarrow} \subseteq \underline{\mathcal{S}^{a}\downarrow} \subseteq \underline{\mathcal{S}}$. Similarly for $\underline{SP}|_{\underline{\mathcal{S}}^{a\Downarrow}}$ and $\underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$.

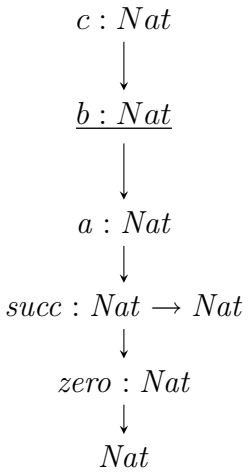
Let us explain the satisfaction requirements one by one. All models in examples below interpret defined symbols *Nat*, *zero* and *succ* as the set of natural numbers, 0 and the successor function, respectively.

1. (construction) For every defined symbol a, if a model from the construction model satisfies the specification reduced to the dependency structure below a then it must also satisfy the specification reduced

to the dependency structure of a (including a). This is to ensure that all defined symbols, i.e., the results of the construction, satisfy the specification, provided the model reducts to their proper dependency structures do.

Example 6.14 *Consider the following construction signature*

$$\underline{\mathcal{S}}_5 = \left(\begin{array}{ll} \textbf{sort} & Nat; \\ \textbf{ops} & zero\colon Nat, \\ & succ\colon Nat \to Nat, \\ & a\colon Nat,\ \underline{b\colon Nat}, \\ & c\colon Nat; \\ \textbf{deps} & zero < succ, \\ & succ < a, \\ & a < b,\ b < c \end{array}\right)$$



and let $\underline{SP}_1$ *be construction specification over* $\underline{\mathcal{S}}_5$ *consisting of the sentence*

$$b = zero \wedge c = succ(b).$$

We have

- $\{(a = 0, b = 0, c = 1)\} \models^c \underline{SP}_1$, *because b is 0 and c is 1,*
- $\{(a = 0, b = 0, c = 2)\} \not\models^c \underline{SP}_1$, *because b is 0 and c is not 1,*
- $\{(a = 0, b = 1, c = 3), (a = 0, b = 0, c = 1)\} \models^c \underline{SP}_1$, *because in the model where b is 0 the value of c is 1, the value of c in the other model does not matter for the satisfaction, because b (the symbol in the dependency structure of c) does not satisfy the specification* $\underline{SP}_1$*; to be precise the reduct model* $(a = 0, b = 1, c = 3)|_{\underline{S}^{c\Downarrow}}$, *i.e., the model* $(a = 0, b = 1)$, *is not the reduct of any model of* $\underline{SP}_1$*; and, since b is an assumed symbol, the condition (1) for symbol c holds trivially in this case.*

2. (parameter completeness) For every assumed symbol a, all values of a in the models of the specification compatible with the dependency structure shall be in the models in *Con*. This is to guarantee that all possible values of assumed symbols, i.e., the parameters of the construction, are allowed by the construction, as long as this is consistent with the construction specification.

   Example 6.15 *Let $\underline{\mathcal{S}}_5$ be the construction signature from Example 6.14 and let specification $\underline{SP}_2$ over $\underline{\mathcal{S}}_5$ be given by*

   $$(b = zero \vee b = succ(zero)) \wedge c = succ(b).$$

   The following holds:

   - $\{(a = 0, b = 0, c = 1)\} \not\models^c \underline{SP}_2$*, since there is no model with b equal to 1,*

     $\{(a = 0, b = 1, c = 2)\} \not\models^c \underline{SP}_2$*, since there is no model with b equal to 0,*

   - $\{(a = 0, b = 1, c = 2), (a = 0, b = 0, c = 1)\} \models^c \underline{SP}_2$*; there are as many models as possible values of b.*

3. (grouping) Following Def. 6.10, for a set of symbols from $\mathbf{Compl}(\underline{\mathcal{S}})$, if a model of the signature $\mathbf{UnDep}(\mathbf{Compl}(\underline{\mathcal{S}}))$ reduced to the dependency structure of every element of the set is in the likewise reduced construction model, then the model reduced to the dependency structure of the set as a whole must also be in the accordingly reduced construction model.

   This condition does not depend on the specification, therefore, it applies to all models, also those that have assumed symbols incompatible with the specification. Nevertheless, it is included as a part of the satisfaction relation, because it complements the completeness condition (2) – it requires that for any set of symbols, all combinations of their different interpretations must be allowed by models in the construction

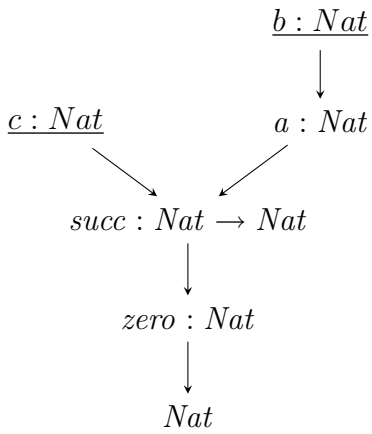
model. Since the interpretation of defined symbols is uniquely given by their dependency structure, in fact the condition concerns only assumed symbols.

This condition guarantees that construction models satisfying a construction specification are nontrivial, i.e. they are nonempty classes of models. This is by taking the empty set of symbols and any model of **Compl**($\underline{\mathcal{S}}$) in Def. 6.10, which must exist, in consequence of assumptions from Sect. 3.5.

Example 6.16 *Consider construction signature* $\underline{\mathcal{S}}_6$ *and the corresponding dependency graph given below.*

$\underline{\mathcal{S}}_6 =$

sort	Nat;
ops	$zero : Nat$,
	$succ : Nat \to Nat$,
	$a : Nat$,
	$\underline{b : Nat}$,
	$\underline{c : Nat}$;
deps	$zero < succ$,
	$succ < a$,
	$a < b$,
	$succ < c$



Here b and c are assumed symbols and symbol c is independent of both a and b. Only b depends on defined symbol a. Let the construction specification $\underline{SP}_3$ *over* $\underline{\mathcal{S}}_6$ *be given by the sentence*

$$(b = succ(a) \vee b = zero) \wedge (c = succ(zero) \vee c = zero).$$

It holds that

- $\{(a = 0, b = 1, c = 1), (a = 0, b = 1, c = 0), (a = 0, b = 0, c = 0), (a = 0, b = 0, c = 1)\} \models^c \underline{SP}_3$*; according to* $\underline{SP}_3$ *the assumed symbols b and c can have two values and all four models with*

all possible combinations of them are present in the construction model;

- $\{(a = 0, b = 0, c = 1), (a = 0, b = 1, c = 0), (a = 0, b = 1, c = 1)\} \not\models^c \underline{SP}_3$, *because the construction model includes a model that interprets* b *as* 0 *and a model with* c *interpreted as* 0*, but it does not include a model having them both interpreted as* 0.

4. (*Con*-dependency-wise) For any set of symbols of $\mathbf{Compl}(\underline{\mathcal{S}})$ and any model in the construction model *Con*, if the model reduced to the dependency structure of every symbol satisfies the reduced specification, the model reduced to the dependency structure of the whole set must satisfy the same reduced specification. This is to make sure that the specifications of symbols do not depend on symbols beyond their dependency structure. In a way, this condition constraints specification $\underline{SP}$ rather then the construction model – see Sect. 6.4.1.

Example 6.17 *Consider construction signature* $\underline{\mathcal{S}}_6$ *from Example 6.16. Let* $\underline{SP}_4$ *be the construction specification over* $\underline{\mathcal{S}}_6$ *(where* c *is an assumed symbol that does not depend on* a*) given by the sentence*

$$c = succ(a).$$

No construction models in $[\![\underline{SP}_4]\!]^c$ *are extensions of the standard model of natural numbers, because on the one hand,* c *is an assumed symbol, therefore, by condition (2) in the construction model there should be as many models as possible values of* c*. On the other hand,* a *is a defined symbol dependent only on* Nat *(with zero and succ), therefore, all models in the construction model must interpret* a *as the same value. Condition (4) puts requirements on* c *and* a *together, thus only singleton construction models with* Nat *interpreted as a singleton set may possibly satisfy the given conditions. If the specification additionally required* Nat *to be a set of natural numbers, condition (4) would render the specification inconsistent.*

Example 6.18 *Consider construction signature* $\underline{\mathcal{S}}_6$ *from Example 6.16. Let* $\underline{SP}'_4$ *be the construction specification over* $\underline{\mathcal{S}}_6$ *consisting of the sentence*

$$a = zero \wedge b = zero \wedge c = succ(a) \ .$$

Then, perhaps surprisingly, we have $\{(a = 0, b = 0, c = 1)\} \models^c \underline{SP}'_4$*, because both* a *and* c *depend on succ and zero and* $(c = succ(a))$ *is equivalent to* $(c = succ(zero))$*, because* $(a = zero)$*; this example shows that a Con-dependency-wise specification can correlate symbols that are not directly related by the dependency structure from the signature when the correlation goes via symbols that they depend on (zero in this case).*

In Sect. 6.4.3 below we discuss how the satisfaction of construction specifications corresponds to the typical satisfaction of specifications of parameterised module.

6.4.1 Consistency of Construction Specification

A *construction specification is consistent* iff there exists a construction model that satisfies all conditions of Def. 6.13.

Lemma 6.19 *Every consistent construction specification is consistent in the base institution* $\mathbb{I}$*.*

The proof uses conditions (3) and (4) of Def. 6.13 for the empty set of symbols. The details are in Appendix 6.A.

There are two main reasons of inconsistency of a construction specification $\underline{SP} = \langle \underline{\mathcal{S}}, SP \rangle$:

- inconsistency of specification SP in the base institution $\mathbb{I}$;
- the condition (4) of Def. 6.13, i.e. the fact that the specification does not match the dependency structure of its signature (cf. Example 6.17 above).

A stronger version of condition (4) of Def. 6.13 may be given without any reference to satisfaction condition or construction models.

Definition 6.20 (Dependency-wise Construction Specification) *A construction specification $\underline{SP}$ over a construction signature $\underline{\mathcal{S}}$ is* dependency-wise *iff for all $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ and all $M \in [\![\underline{\mathcal{S}}]\!]$, if for all $a \in A$, $M|_{\underline{\mathcal{S}}^a\downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^a\downarrow}$ then $M|_{\underline{\mathcal{S}}^A\downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^A\downarrow}$.*

The difference between condition (4) of Def. 6.13 and Def. 6.20 is that the former limits the models to those in construction model *Con*, whereas the latter refers to any models. Still, the idea is simple: a dependency-wise construction specification does not directly relate symbols that are not related by dependency relation in the construction signature.

Lemma 6.21 *For any construction specification $\underline{SP} = \langle \underline{\mathcal{S}}, SP \rangle$ and a construction model $Con \in [\![\underline{\mathcal{S}}]\!]^c$. If $\underline{SP}$ and Con meet conditions (1-3) of Def. 6.13 and additionally $\underline{SP}$ is dependency-wise, then $Con \models^c \underline{SP}$.*

The proof is easy and we omit it here. Obviously if the dependency-wise condition works for all models in $[\![\underline{\mathcal{S}}]\!]$ (Def. 6.20), it also works for all models in *Con* (condition (4) of Def. 6.13).

All dependency-wise construction specifications seen as specifications in the base institution $\mathbb{I}$ are consistent. From the assumptions of Sect. 3.5 we get that every signature in **Sig** has a model. Consider $\underline{SP} = \langle \underline{\mathcal{S}}, SP \rangle$ and, by the requirement of Def. 6.20 for the empty set of symbols, for any $M \in [\![\underline{\mathcal{S}}]\!]$ we have $M|_{\Sigma_\emptyset} \models SP|_{\Sigma_\emptyset}$, where $\Sigma_\emptyset$ is the initial object in **Sig**. This means that there exists $M' \in [\![SP]\!]$ such that $M'|_{\Sigma_\emptyset} = M|_{\Sigma_\emptyset}$, thus SP is consistent.

The following theorem gives sufficient condition for construction specification to be consistent as such. Notice the finiteness requirement.

Theorem 6.22 *Every dependency-wise construction specification over a finite construction signature is consistent, i.e., it has a construction model.*

The proof is in Appendix 6.A. The opposite implication does not hold, because there are examples of consistent construction specifications that are not dependency-wise. Below we present one of them.

Example 6.23 *Consider $\underline{SP} = \langle \underline{\mathcal{S}}, SP \rangle$ with $\underline{\mathcal{S}} = (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s, b : s)$ and $SP = \{a = b\}$. Clearly $\underline{SP}$ is not dependency-wise, because, for the set*

of symbols $A = \{a, b\}$ *and* $M = (s = \{0, 1\},\ a = 1,\ b = 0)$, *even though* $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models SP|_{\underline{\mathcal{S}}^{a}\downarrow}$ *and* $M|_{\underline{\mathcal{S}}^{b}\downarrow} \models SP|_{\underline{\mathcal{S}}^{b}\downarrow}$, *we get* $M|_{\underline{\mathcal{S}}^{A}\downarrow} \not\models SP|_{\underline{\mathcal{S}}^{A}\downarrow}$. *Nevertheless, it is consistent. Take* $M' = (s = \{0, 1\},\ a = 0,\ b = 0)$ *and we have* $\{M'\} \models^{c} \underline{SP}$, *which makes* $\underline{SP}$ *consistent.*

6.4.2 Cleaning Operator

It is not always the case that given $Con \models^{c} \underline{SP}$, all models inside the construction model Con satisfy the specification, i.e., that for all $M \in Con$, $M \models \underline{SP}$. We follow the design by contract approach, where the result is required to meet a specification only if the parameter does. Therefore, all models that do not meet the specification on their assumed symbols (parameter part) do not have to meet the specification on the result part. As a consequence construction models may contain some "junk", i.e., models that do not satisfy the specification.

The following operator cleans the construction models up from the "junk" models.

Definition 6.24 (Cleaning operator) *For a construction specification* $\underline{SP} \in \mathbf{Spec}(\underline{\mathcal{S}})$ *and* $Con \in [\![\underline{\mathcal{S}}]\!]^{c}$, *the* cleaning operator $\mathbf{Clean}_{\underline{SP}} \colon [\![\underline{SP}]\!]^{c} \to [\![\underline{SP}]\!]^{c}$ *is defined as*

$$\mathbf{Clean}_{\underline{SP}}(Con) = \{M \in Con \mid M \models \underline{SP}\}\ .$$

The theorem below ensures that the cleaning operator is well defined.

Theorem 6.25 *The cleaning operator from Def. 6.24 is well defined, i.e. for a construction specification and a construction model* $Con \in [\![\underline{SP}]\!]^{c}$, *it holds that* $\mathbf{Clean}_{\underline{SP}}(Con) \in [\![\underline{SP}]\!]^{c}$.

This theorem may seem trivial, but the proof turns out to be quite elaborate. The main challenge is to prove that there are enough models in $\mathbf{Clean}_{\underline{SP}}(Con)$ to satisfy conditions (2) and (3) of Def. 6.13. Both conditions require a kind of completeness of the construction model with regard to the assumed symbols, i.e. the parameters of the construction. It is not enough to use directly the corresponding conditions w.r.t. $Con \in [\![\underline{SP}]\!]^{c}$, because in the

result we get the existence of models in *Con* such that their reduct to some subset of signature symbols satisfies the appropriately reduced specification and we still need to prove that they satisfy the specification as a whole, i.e. that they belong to $\mathbf{Clean}_{\underline{SP}}(Con)$. The solution is that by induction we can build a model that satisfies the specification and, when reduced to the above mentioned subset of signature symbols, equals to the original model reduced to the same subset. The detailed proof is given in Appendix 6.A below.

Notation. A construction model *Con* such that $Con \models^c \underline{SP}$ and $Con = \mathbf{Clean}_{\underline{SP}}(Con)$ is called a *clean construction model of* $\underline{SP}$. By $\mathbf{Clean}_{\underline{SP}}([\![\underline{SP}]\!]^c)$ we denote the class of all clean construction models of $\underline{SP}$.

6.4.3 Module Specifications as Construction Specifications

A non-parameterised module specification is typically a specification SP over a signature $\Sigma \in \mathbf{Sig}$. The corresponding construction specification is

$$\underline{SP} = \langle \underline{\mathcal{S}}_\Sigma, SP \rangle$$

where $\underline{\mathcal{S}}_\Sigma$ is a construction signature corresponding to Σ (cf. Sect. 6.2.1). A construction model of $\underline{SP}$ is a singleton $\{M\}$ such that $M \in [\![SP]\!]$.

For a parameterised module specification $SP_M = \langle \sigma, SP_P, SP_R \rangle$ over a parameterised module signature $\sigma\colon \Sigma_P \to \Sigma_R$, the corresponding construction specification is given as

$$\underline{SP}_M = \langle \underline{\mathcal{S}}_\sigma, \sigma(SP_P) \cup SP_R \rangle$$

where $\underline{\mathcal{S}}_\sigma$ is a construction signature corresponding to σ defined in Sect. 6.2.1. Recall that a model (called persistent parameterised module model, cf. Sect. 3.4) of a parameterised modules specification SP_M, as above, is a (partial) map $\kappa\colon [\![\Sigma_P]\!] \to [\![\Sigma_R]\!]$ such that for all $M_p \in [\![SP_P]\!]$, $\kappa(M_p)$ is defined, $\kappa(M_p) \in [\![SP_R]\!]$ and $\kappa(M_p)|_\sigma = M_p$.

Even if SP_M is a consistent parameterised module specification, $\underline{SP}_M$ may

fail to be a consistent construction specification, because it does not need to be dependency-wise (cf. (4) of Def. 6.13 and Def. 6.20). If its construction signature is finite, it is enough to add additional dependencies between assumed symbols of $\underline{\mathcal{S}}_\sigma$ to make it dependency-wise and, by Theorem 6.22, also consistent.

The need for additional dependencies between assumed symbols results from the different level of parameterisation in the parameterised modules and constructions. The former have whole signatures as parameters and only in presence of a model of the entire parameter signature, the result model of the result signature is given. The latter admits parameterisation on the symbol level (via assumed symbols) and any parameter symbol may be instantiated on its own, as long as the dependencies are respected.

The following example shows how additional dependencies added to the construction signature make the construction specification consistent.

Example 6.26 *Consider a parameterised module specification* $\langle \sigma, SP_P, SP_R \rangle$ *over a parameterised module signature* $\sigma\colon \Sigma_P \to \Sigma_R$*, where*

$$\Sigma_P = (\textbf{sort } Nat;\ \textbf{ops } a : Nat,\ b : Nat),$$
$$\Sigma_R = (\textbf{sort } Nat;\ \textbf{ops } a : Nat,\ b : Nat,\ c : Nat),$$

σ *is an inclusion and*

$$SP_P = \{(a = 0 \wedge b = 1) \vee (a = 1 \wedge b = 0)\},$$
$$SP_R = SP_P \cup \{c = a + b\}.$$

To simplify the notation assume that constants $0, 1$ *and operation* $+$ *come together with the sort of natural numbers* Nat*. The corresponding construction specification is* $\underline{SP} = \langle \underline{\mathcal{S}}_\sigma, \sigma(SP_P) \cup SP_R \rangle$ *with*

$$\underline{\mathcal{S}}_\sigma = (\textbf{sort } \underline{Nat};\ \textbf{ops } \underline{a : Nat},\ \underline{b : Nat},\ c : Nat)$$
$$\textbf{deps } a < c,\ b < c)$$

It is easy to show that $\langle \sigma, SP_P, SP_R \rangle$ *is a consistent parameterised module*

specification and that $\underline{SP}$ *is an inconsistent construction specification (for Nat interpreted as the set of natural numbers). The reason is that* $\underline{SP}$ *is not dependency-wise. Take* $A = \{a, b\}$ *and* $M \in [\![\underline{\mathcal{S}}_\sigma]\!]$, $M = (a = 1, b = 1)$. *It holds that* $M|_{\underline{\mathcal{S}}^a_{\sigma\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^a_{\sigma\downarrow}}$ *and* $M|_{\underline{\mathcal{S}}^b_{\sigma\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^b_{\sigma\downarrow}}$, *but* $M|_{\underline{\mathcal{S}}^A_{\sigma\downarrow}} \not\models \underline{SP}|_{\underline{\mathcal{S}}^A_{\sigma\downarrow}}$. *It is enough to add a dependency between a and b in* $\underline{\mathcal{S}}_\sigma$ *to make it dependency-wise. Let*

$$\underline{\mathcal{S}}'_\sigma = (\textbf{sort } \underline{Nat};\ \textbf{ops } \underline{a : Nat},\ \underline{b : Nat},\ c : Nat)$$
$$\textbf{deps } a < b,\ b < c)$$

It is easy to check that $\underline{SP}' = \langle \underline{\mathcal{S}}'_\sigma, \sigma(SP_P) \cup SP_R \rangle$ *is dependency-wise, therefore, by Theorem 6.22, it is consistent.*

In what follows we assume that $\underline{SP}_M$ is a consistent construction specification, either directly or, if we assume finiteness of signatures, after adding extra dependencies between assumed symbols in $\underline{\mathcal{S}}_\sigma$. It may be done by taking any linear order that extends the dependency relation of $\underline{\mathcal{S}}_\sigma$, then a consistent parameterised module specification SP_M is a consistent construction specification (over a construction signature with a linear dependency relation). Note that this would not work for signatures with infinite set of symbols, as the linear dependency imposed on them would not be bounded. Finiteness is also required by Theorem 6.22.

Let *Con* be a construction model of $\underline{SP}_M$. It is easy to show that *Con* is a class of Σ_R-models such that for any $M_P \models SP_P$ there exists a unique $M \in Con$ such that $M|_\sigma = M_P$ and $M \models SP_R$, i.e. the map $\{M_P \mapsto M \mid M \in Con, M_P = M|_\sigma, M_P \models SP_P, \}$ is a persistent parameterised module model of SP_M. To sketch the proof, we first notice in the definition of parameterised module specification in Sect. 3.4 that $[\![SP_P]\!] \subseteq [\![SP_R|_\sigma]\!]$. From $M_P \models SP_P$ we have existence of $M_R \models SP_R$ such that $M_R|_\sigma = M_P$, thus $M_R \models (\sigma(SP_P) \cup SP_R)$. All the symbols from Σ_P are assumed in $\underline{\mathcal{S}}_\sigma$ (denoted by set A_σ in Sect. 6.2.1), so, by inductive use of condition (2) and (3) of Def. 6.13, we eventually get $M_R|_{\underline{\mathcal{S}}^{A_\sigma}{}_\downarrow} \in Con|_{\underline{\mathcal{S}}^{A_\sigma}{}_\downarrow}$. Therefore, there is $M \in Con$ such that $M|_{\underline{\mathcal{S}}^{A_\sigma}{}_\downarrow} = M_R|_{\underline{\mathcal{S}}^{A_\sigma}{}_\downarrow}$ and consequently $M|_\sigma = M_P$. All the symbols from Σ_R that are not coming from Σ_P (i.e. that are not assumed) are

defined in $\underline{\mathcal{S}}_\sigma$ (denoted by set D_σ in Sect. 6.2.1). No assumed symbol depends on defined symbol in $\underline{\mathcal{S}}_\sigma$, so, by inductive use of (1) and (4) of Def. 6.13, we get $M \models \sigma(SP_P) \cup SP_R$, thus $M \models SP_R$.

Let κ be a persistent module model that satisfies SP_M. Unfortunately, in general the corresponding construction model Con_κ (cf. Sect. 6.3.1) does not have to satisfy $\underline{SP}_M$ directly. This is because satisfaction of construction specifications is slightly stronger than satisfaction of parameterised module specifications. The difference concerns parameter models that do not satisfy SP_P. The condition (3) of Def. 6.13 requires that given a set of symbols $A \subseteq \Sigma_R$ and a Σ_R-model M such that for every $a \in A$, $M|_{\underline{\mathcal{S}}^a_{\sigma\downarrow}} \in Con_\kappa|_{\underline{\mathcal{S}}^a_{\sigma\downarrow}}$, then $M|_{\underline{\mathcal{S}}^A_{\sigma\downarrow}} \in Con_\kappa|_{\underline{\mathcal{S}}^A_{\sigma\downarrow}}$. If $M|_\sigma \not\models (\sigma(SP_P) \cup SP_R)|_\sigma$, and so $M|_\sigma \not\models SP_P$, this implies that there must exist $M' \in Con_\kappa$ such that $M'|_{\underline{\mathcal{S}}^A{\downarrow}} = M|_{\underline{\mathcal{S}}^A{\downarrow}}$. There is no guarantee, however, that there is such $M' \in Con_\kappa$, because satisfaction of parameterised module specification conditions posed on κ concerns only parameter models that satisfy SP_P. Therefore, we use the cleaning operator and we get:

$$\mathbf{Clean}_{\underline{SP}}(Con_\kappa) \models^c \underline{SP}.$$

The following example illustrates the situation.

Example 6.27 *Consider a parameterised module specification* $\langle \sigma, SP_P, SP_R \rangle$ *over a parameterised module signature* $\sigma\colon \Sigma_P \to \Sigma_R$*, where*

$$\Sigma_P = (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ b : s,\ c : s),$$
$$\Sigma_R = (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ b : s,\ c : s,\ d : s),$$

σ *is an inclusion and*

$$SP_P = \{a = c\ \wedge\ b \neq c\ \},$$
$$SP_R = SP_P \cup \{d = b\}.$$

The corresponding construction signature is

$$\underline{\mathcal{S}} = (\mathbf{sort}\ \underline{s};\ \mathbf{ops}\ \underline{a : s},\ \underline{b : s},\ \underline{c : s},\ d : s;\ \mathbf{deps}\ c < a,\ c < b,\ a < d,\ b < d)$$

Note the additional dependency in $\underline{\mathcal{S}}$, among a, b and c, added to make the construction specification dependency-wise, as described above.

The corresponding construction specification is $\underline{SP} = \langle \underline{\mathcal{S}}, SP_R \rangle$.

Axiom $b \neq c$ from specification SP_P makes the carrier of sort s to have at least two elements. In order to make this example readable, let us assume that sort s is a set of exactly two elements (the general case is analogous to this restricted one).

Consider four Σ_P-models

$$\begin{aligned} M_1^P &= (s = \{0,1\},\ a = 1,\ b = 0,\ c = 1), \\ M_2^P &= (s = \{0,1\},\ a = 0,\ b = 1,\ c = 0), \\ M_3^P &= (s = \{0,1\},\ a = 1,\ b = 1,\ c = 0), \\ M_4^P &= (s = \{0,1\},\ a = 0,\ b = 0,\ c = 0) \end{aligned}$$

and four Σ_R-models

$$\begin{aligned} M_1^R &= (s = \{0,1\},\ a = 1,\ b = 0,\ c = 1, d = 0), \\ M_2^R &= (s = \{0,1\},\ a = 0,\ b = 1,\ c = 0, d = 1), \\ M_3^R &= (s = \{0,1\},\ a = 1,\ b = 1,\ c = 0, d = 1), \\ M_4^R &= (s = \{0,1\},\ a = 0,\ b = 0,\ c = 0, d = 0) \end{aligned}$$

and a partial mapping $\kappa \colon [\![\Sigma_P]\!] \to [\![\Sigma_R]\!]$ given as

$$\kappa = \{\langle M_i^P, M_i^R \rangle \mid 1 \leq i \leq 4\} \ .$$

It is easy to show that κ is a persistent parameterised module model of $\langle \sigma, SP_P, SP_R \rangle$. For $1 \leq i \leq 2$ we have $M_i^P \models SP_P$, $M_i^R \models SP_R$ and $M_i^R|_\sigma = M_i^P$; for $3 \leq i \leq 4$ we have $M_i^P \not\models SP_P$ and as a consequence $M_i^R \not\models SP_R$.

The corresponding construction model is

$$Con = \{M_i^R \mid 1 \leq i \leq 4\}$$

and we have

$$Con \not\models^{c} \underline{SP} \ ,$$

because of condition (3) of Def. 6.13, for $A = \{a, b\}$ *and* Σ_R*-model* $M' = (s = \{0, 1\},\ a = 1,\ b = 0,\ c = 0, d = 0)$*, even though* $M'|_{\underline{S}^{a}\downarrow} \in Con|_{\underline{S}^{a}\downarrow}$ *and* $M'|_{\underline{S}^{b}\downarrow} \in Con|_{\underline{S}^{b}\downarrow}$*, we have* $M'|_{\underline{S}^{A}\downarrow} \notin Con|_{\underline{S}^{A}\downarrow}$*.*

However, it is easy to check that $\mathbf{Clean}_{\underline{SP}}(Con) = \{M_1^R, M_2^R\}$ *is a construction model of* $\underline{SP}$*:*

$$\mathbf{Clean}_{\underline{SP}}(Con) \models^{c} \underline{SP}.$$

6.5 Construction Fittings and Sum

Architectural decomposition of the system into a number of basic building blocks makes sense only under the assumption that the blocks may be composed together later into the system as a whole. In algebraic specifications the system decomposition is typically described by a diagram of specifications connected by (fitting) morphisms, and the colimit operation acts as such a composition operation for specifications, with amalgamation of models used to put together their "implementations".

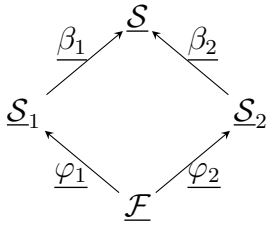
We take a similar approach. In order to connect symbols from two construction signatures, we define fitting spans and we use the pushout as the sum operation. It subsumes union, composition, translation and application operations, as usually defined. This is possible due to fitting spans being defined as external to construction signatures. Depending on the actual connections between construction signature symbols, the sum operation captures cases traditionally treated separately.

6.5.1 Construction Fittings and Sum of Construction Signatures

The definition of construction fittings as spans in $\mathbf{SigDep}^{frag}$ makes them suitable for symmetric and asymmetric connections between construction signatures.

Definition 6.28 (Construction Fitting) *A* construction fitting $ft\colon \underline{\mathcal{S}}_1 \nwarrow\!\!\nearrow \underline{\mathcal{S}}_2$ *between two construction signatures* $\underline{\mathcal{S}}_1$ *and* $\underline{\mathcal{S}}_2$ *is a span in* $\mathbf{SigDep}^{frag}$ $ft = \langle \underline{\varphi_1}\colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_1, \underline{\varphi_2}\colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_2 \rangle$, *such that*

1. $\underline{\mathcal{F}}$ *is an empty signature fragment (cf. Sect. 4)*
2. *the pushout* $\underline{\beta_1}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}$ *and* $\underline{\beta_2}\colon \underline{\mathcal{S}}_2 \to \underline{\mathcal{S}}$ *of* $\underline{\varphi_1}$ *and* $\underline{\varphi_2}$ *in* $\mathbf{SigDep}^{frag}$ *yields a construction signature* $\underline{\mathcal{S}}$.



The pushout signature $\underline{\mathcal{S}}$ is called *the sum of* $\underline{\mathcal{S}}_1$ *and* $\underline{\mathcal{S}}_2$ *w.r.t. fitting* ft.

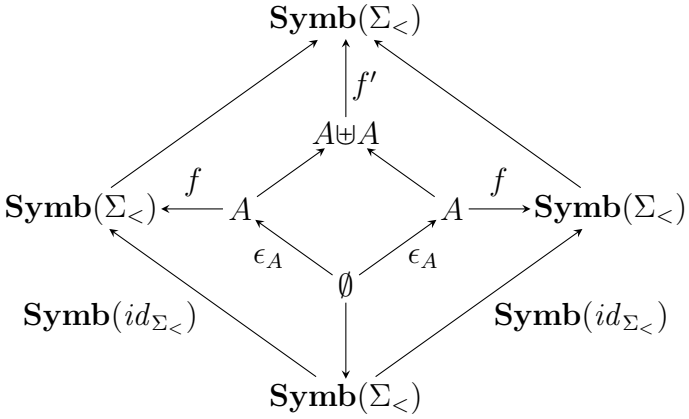
Notation. The symbols used in the above definition will be used as the default notation for construction fittings and their pushouts. This means that every time when we introduce a construction fitting ft without an explicit definition, we assume that $ft = \langle \underline{\varphi_1}\colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_1, \underline{\varphi_2}\colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_2 \rangle$ and that its pushout morphisms are $\underline{\beta_1}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}$ and $\underline{\beta_2}\colon \underline{\mathcal{S}}_2 \to \underline{\mathcal{S}}$. Sometimes we may write $\underline{\mathcal{S}}_1 \oplus_{ft} \underline{\mathcal{S}}_2$ to denote the sum construction signature $\underline{\mathcal{S}}$.

Let us briefly describe the meaning of the requirements of the definition above. Requirement (1) says that the fitting source $\underline{\mathcal{F}}$ contains only assumed symbols. Requirement (2) makes the construction of $\underline{\mathcal{S}}$ unique (up-to isomorphism) with the internal mapping of $\underline{\mathcal{S}}$ being an injection (cf. Def. 6.1). The consequence of both requirements of Def. 6.28 is that $\underline{\beta_1}$ and $\underline{\beta_2}$ are injective on defined symbols, and each defined symbol of $\underline{\mathcal{S}}$ comes either from $\underline{\mathcal{S}}_1$ or $\underline{\mathcal{S}}_2$.

The injectivity of $\underline{\beta_1}$ and $\underline{\beta_2}$ on defined symbols is the key for explicit tracking of the origin of defined symbols, because it guarantees that each defined symbol in the result construction signature comes from exactly one component, i.e. there is a unique *origin of symbol definition*. However, it renders some, perhaps expected operations on modules ill-defined.

Example 6.29 *For a construction signature* $\underline{\mathcal{S}} = \langle A, \Sigma_<, f\colon A \to \mathbf{Symb}(\Sigma_<) \rangle$ *containing some defined symbols,* $A \neq \emptyset$, *the pair of identity morphisms*

$\langle id_{\underline{\mathcal{S}}}, id_{\underline{\mathcal{S}}}\rangle$ fails to be a construction fitting in the sense of Def. 6.28, because the source of the fitting is not empty. The pair $\langle \underline{\varphi}, \underline{\varphi}\rangle$ with $\underline{\varphi}\colon \mathbf{Empt}(\underline{\mathcal{S}}) \to \underline{\mathcal{S}}$ defined as $\underline{\varphi} = \langle \epsilon_A, id_{\Sigma_<}\rangle$, where $\epsilon_A\colon \emptyset \to A$ also fails to be a construction fitting, because the pushout object is not an injective fragment, thus it is not a construction signature, c.f. the diagram in **Set** *below (f' is not injective).*



We do not take this as a disadvantage. To the contrary, in our approach the sharing of symbols is via parameterisation only. This means that if two constructions are to share a symbol that is defined on the one side, then the construction fitting must map it to an assumed symbol on the other side.

Notation. We say that two *symbols are shared* (symmetrically) if they are targets of a single symbol via the fitting span, i.e. for $a_1 \in \mathbf{Compl}(\underline{\mathcal{S}_1})$ and $a_2 \in \mathbf{Compl}(\underline{\mathcal{S}_2})$ we say that a_1 *and* a_2 *are shared* iff there exists $a \in \mathbf{Compl}(\underline{\mathcal{F}})$ such that $\underline{\varphi_1}(a) = a_1$ and $\underline{\varphi_2}(a) = a_2$.

6.5.2 Sum of Construction Models

As the sum of construction signatures connected by a construction fitting is given by a pushout operation, not surprisingly, the sum of construction models uses the amalgamation property (cf. Sect. 3.5) in its definition.

Definition 6.30 *The* sum of construction models Con_1 and Con_2 w.r.t. the fitting ft *is given as*

$$Con_1 \oplus_{ft} Con_2 = \{M \in [\![\underline{\mathcal{S}}]\!] \mid \ M|_{\underline{\beta_1}} \in Con_1,\ M|_{\underline{\beta_2}} \in Con_2\}$$

The sum of construction models contains amalgamations of all models from both sides that prove to be compatible on the fitting source. The operation looks trivial, but its effect is very powerful due to the nature of the fitting and the contents of construction models, as required by Def. 6.5. We make sure that the result of the sum operation also meets all the requirements imposed on construction models.

Theorem 6.31 *The sum of construction models is a construction model.*

The proof is in Appendix 6.A.

6.5.3 Sum of Construction Specifications

The sum of construction specifications is the union of translations of component specifications.

Definition 6.32 (Sum of Construction Specifications) *The sum of construction specifications* $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1 \rangle$ *and* $\underline{SP}_2 = \langle \underline{\mathcal{S}}_2, SP_2 \rangle$ *w.r.t. fitting* $ft \colon \underline{\mathcal{S}}_1 \nwarrow\!\!\nearrow \underline{\mathcal{S}}_2$ *is the construction specification*

$$\underline{SP}_1 \oplus_{ft} \underline{SP}_2 = \langle \underline{\mathcal{S}}_1 \oplus_{ft} \underline{\mathcal{S}}_2, \underline{\beta_1}(SP_1) \cup \underline{\beta_2}(SP_2) \rangle.$$

It is not guaranteed that the sum of two consistent construction specifications connected via any construction fitting gives rise to a consistent construction specification.

Example 6.33 *Consider two construction specifications*

$$\underline{SP}_1 = \langle (\textbf{sort } \underline{s}; \textbf{ops } \underline{a : s}), \{\forall\ x : s\ \cdot x = a\} \rangle$$
$$\underline{SP}_2 = \langle (\textbf{sort } s; \textbf{ops } a : s,\ b : s), \{a \neq b\} \rangle$$

and a construction fitting $ft = \langle \varphi_1, \varphi_2 \rangle$*, where* $\varphi_1 \colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_1$ *and* $\varphi_2 \colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_2$ *are inclusions and* $\underline{\mathcal{F}} = (\textbf{sort } \underline{s}; \textbf{ops } \underline{a : s})$.

Both $\underline{SP}_1$ *and* $\underline{SP}_2$ *are consistent; however, their sum* $\underline{SP}_1 \oplus_{ft} \underline{SP}_2$ *is not consistent, because by* SP_1 *sort* s *contains exactly one element and by* $\underline{SP}_2$ *it has at least two different elements.*

In order to address this problem we introduce the concept of compatibility of construction specification w.r.t. the fitting span. The intention is to allow defined symbols (actual parameters) to have stronger specifications than the corresponding assumed symbols (formal parameters). The corresponding assumed symbols on both sides must have equivalent specifications.

Definition 6.34 (Compatible Construction Specifications) *Given a construction fitting* $ft = \langle \underline{\varphi_1}, \underline{\varphi_2} \rangle$ *with* $\underline{\varphi_1} \colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_1$ *and* $\underline{\varphi_2} \colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_2$ *and two consistent construction specifications* $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1 \rangle$ *and* $\underline{SP}_2 = \langle \underline{\mathcal{S}}_2, SP_2 \rangle$*, we say that* $\underline{SP}_1$ is compatible with $\underline{SP}_2$ w.r.t. the fitting *ft iff*

1. *for all sets of independent symbols* $A \subseteq \mathbf{Compl}(\underline{\mathcal{F}})$ *such that for all* $a \in A$, $\underline{\varphi_1}(a) \notin \underline{\mathcal{S}}_1$ *(this means that* $\underline{\varphi_1}(A)$ *is a set of assumed symbols in* $\underline{\mathcal{S}}_1$*) and all* $M_2 \in [\![\underline{\mathcal{S}}_2]\!]$ *such that* $M_2 \models \underline{SP}_2$*, if* $(M_2|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{A_{\Downarrow}}} \models (SP_1|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{A_{\Downarrow}}}$ *then* $(M_2|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{A_{\downarrow}}} \models (SP_1|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{A_{\downarrow}}}$

2. *for all sets of independent symbols* $A \subseteq \mathbf{Compl}(\underline{\mathcal{F}})$ *such that for all* $a \in A$, $\underline{\varphi_2}(A) \notin \underline{\mathcal{S}}_2$ *(this means that* $\underline{\varphi_2}(A)$ *is a set of assumed symbols in* $\underline{\mathcal{S}}_2$*) and all* $M_1 \in [\![\underline{\mathcal{S}}_1]\!]$ *such that* $M_1 \models \underline{SP}_1$*, if* $(M_1|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{A_{\Downarrow}}} \models (SP_2|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{A_{\Downarrow}}}$ *then* $(M_1|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{A_{\downarrow}}} \models (SP_2|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{A_{\downarrow}}}$

The two compatibility conditions are symmetric, so it is enough to comment only the first one. It says that for any set of independent assumed symbols A shared between the two constructions (i.e. A is a subset of $\mathbf{Compl}(\underline{\mathcal{F}})$) such that the corresponding symbols in $\underline{\mathcal{S}}_1$ are also assumed, for any model of $\underline{SP}_2$, if the model reduced to the dependency structure below A satisfies $\underline{SP}_1$ reduced to the same signature, then this is also the case on the dependency structure of A, including A. This implies that the specification $\underline{SP}_1$ is not stronger than $\underline{SP}_2$ on the shared symbols that are assumed in $\underline{\mathcal{S}}_1$. If the corresponding symbols in $\underline{\mathcal{S}}_2$ are also assumed, then, by symmetry, both specifications have to be equivalent. However, if the corresponding symbols in $\underline{\mathcal{S}}_2$ are defined, then $\underline{SP}_2$ may be stronger with respect to them.

Such behavior is inspired by the typical situations arising from the application of parameterised modules. When a parameterised module is instantiated with an actual parameter, the specification of the actual parameter

(defined symbols) has to ensure the specification of the formal parameter (the corresponding assumed symbols) in the parameterised module specification, but of course may be stronger than this specification.

The compatibility of two consistent construction specifications guarantees the consistency of their sum in the base institution $\mathbb{I}$, assuming that both construction specifications are dependency-wise (cf. Def. 6.20) and over finite construction signatures.

Lemma 6.35 *Given two dependency-wise construction specifications, $\underline{SP}_1$ and $\underline{SP}_2$ over finite construction signatures such that they are compatible w.r.t. a construction fitting ft, their sum $(\underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}_2))$ is a consistent specification in the base institution $\mathbb{I}$.*

The proof is in Appendix 6.A. The requirement of finiteness is due to Theorem 6.22 used in the proof.

The following theorem shows that the definition of the sum of construction specifications and the sum of clean construction models match. Moreover, the sum of clean models yields a clean model.

Theorem 6.36 *Given two compatible (cf. Def. 6.34) construction specifications $\underline{SP}_1$ and $\underline{SP}_2$ connected by a construction fitting ft, for any two clean (cf. Sect. 6.4.2) construction models $Con_1 \models^c \underline{SP}_1$ and $Con_2 \models^c \underline{SP}_2$, the following holds*

$$Con \models^c \underline{SP} \text{ and } Con \text{ is clean,}$$

where $Con = Con_1 \oplus_{ft} Con_2$ and $\underline{SP} = \underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}_2)$.

The above theorem ensures that our method is sound and the decomposition of the system specification into the smaller parts makes sense. Given models of the component specifications we can join them and obtain the composite that satisfies the specification of the whole. The proof is in Appendix 6.A.

The examples below show different aspects of (in)compatibility of construction specifications.

Example 6.37 *Consider construction signatures*

$$\underline{\mathcal{S}}_1 = (\textbf{sort } \underline{s};\ \textbf{ops } \underline{a\colon s},\ \underline{b\colon s},\ \underline{f\colon s \to s};\ \textbf{deps } a < b,\ b < f),$$
$$\underline{\mathcal{S}}_2 = (\textbf{sort } \underline{s};\ \textbf{ops } \underline{a\colon s},\ b\colon s,\ f\colon s \to s;\ \textbf{deps } a < b,\ b < f),$$

a fitting $ft = \langle \underline{\varphi_1}, \underline{\varphi_2} \rangle$ *with* $\underline{\varphi_1}\colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_1$, $\underline{\varphi_2}\colon \underline{\mathcal{F}} \to \underline{\mathcal{S}}_2$ *such that* $\underline{\mathcal{F}} = \underline{\mathcal{S}}_1$ *and* $\underline{\varphi_1}$, $\underline{\varphi_2}$ *are inclusions, and construction specifications* $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1 \rangle$, $\underline{SP}_2 = \langle \underline{\mathcal{S}}_2, SP_2 \rangle$ *with*

$$SP_1 = \{f(b) = b,\ f(a) = a\},$$
$$SP_2 = \{b = a, f(a) = b\}.$$

Construction specifications $\underline{SP}_1$ *and* $\underline{SP}_2$ *are compatible w.r.t. construction fitting* ft *and, by Theorem 6.36, for any clean* $Con_1 \models^c \underline{SP}_1$ *and* $Con_2 \models^c \underline{SP}_2$ *we have* $Con_1 \oplus_{ft} Con_2 \models^c (\underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}_2))$. *This is because* b *and* f *are defined in* $\underline{\mathcal{S}}_2$ *and the specification* $\underline{SP}_2$ *is stronger on* b *and* f *than* $\underline{SP}_1$ *where, in* $\underline{\mathcal{S}}_1$ *symbols* b *and* f *are assumed. The specifications of* a *in* $\underline{SP}_1$ *and* $\underline{SP}_2$ *are equivalent.*

Example 6.38 *(Cont. of Example 6.37) In order to show a non-example let us change to the institution of the first-order logic* $\mathbb{FOEQ}$ *(to allow inequality sentences) and take* $\underline{SP}'_2 = \langle \underline{\mathcal{S}}_2, SP'_2 \rangle$ *with*

$$SP'_2 = \{b \neq a, f(a) = b\}.$$

Construction specifications $\underline{SP}_1$ *and* $\underline{SP}'_2$ *are not compatible w.r.t. construction fitting* ft*, because* $\underline{SP}'_2$ *is inconsistent with* $\underline{SP}_1$ *in the base institution* $\mathbb{I}$ *(which in the case of this example is* $\mathbb{FOEQ}$*). By Lemma 6.35, since* $\underline{SP}_1$ *and* $\underline{SP}'_2$ *are dependency-wise construction specifications (all symbol dependencies expressed by specifications are statically present in construction signatures* $\underline{\mathcal{S}}_1$ *and* $\underline{\mathcal{S}}_2$*) if they were compatible then their sum* $(\underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}'_2))$ *would be a consistent specification in the base institution, but it is not. More directly, note than the requirements on* b *and* f *imposed by* SP'_2 *do not entail those imposed by* SP_1*. In this case for any* $Con_1 \models^c \underline{SP}_1$ *and* $Con_2 \models^c \underline{SP}'_2$ *we*

have $Con_1 \oplus_{ft} Con_2 = \emptyset$.

Example 6.39 *(Cont. of Example 6.38) To present a more subtle non-example, consider* $\underline{SP}''_2 = \langle \underline{\mathcal{S}}_2, SP''_2 \rangle$ *with*

$$SP''_2 = \{f(a) = a\}.$$

Construction specifications $\underline{SP}_1$ *and* $\underline{SP}''_2$ *are not compatible w.r.t. construction fitting ft, because the specification of defined symbol* f *in* $\underline{SP}''_2$ *is weaker than in* $\underline{SP}_1$*. Their sum, however, is consistent in the base institution, so the situation is different than before. It is easy to check that condition (1) of Def. 6.34 is violated for* $A = \{f\}$*. It does not hold that any model of* $\underline{SP}''_2$ *that satisfies* $\underline{SP}_1|_{\{s,a,b\}}$ *also satisfies* $\underline{SP}_1|_{\{s,a,b,f\}}$*, because* SP_1 *requires* $f(b) = b$ *which is not ensured by* SP''_2*. As a counterexample take model* $M_2 = (s = \{0,1\}, a = 0, b = 1, f = \{\langle 0,0\rangle, \langle 1,0\rangle\})$*. We have* $M_2 \models SP''_2$ *and* $M_2|_{\{s,a,b\}} \models SP_1|_{\{s,a,b\}}$*, but* $M_2|_{\{s,a,b,f\}} \not\models SP_1|_{\{s,a,b,f\}}$*. In fact, there exist clean construction models* $Con_2 \models^c \underline{SP}''_2$ *such that for any* $Con_1 \models^c \underline{SP}_1$ *it holds that* $Con_1 \oplus_{ft} Con_2 \models^c (\underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}_2))$*, but there are also clean* $Con'_2 \models^c \underline{SP}_1$ *such that* $Con_1 \oplus_{ft} Con'_2 \not\models^c (\underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}_2))$*,*

Example 6.40 *(Cont. of Example 6.39) The following example shows that the compatibility condition is stronger than it is actually necessary for the conclusion of Theorem 6.36. Consider* $\underline{SP}'''_2 = \langle \underline{\mathcal{S}}'''_2, SP'''_2 \rangle$ *with*

$$\underline{\mathcal{S}}'''_2 = (\textbf{sort } \underline{s};\ \textbf{ops } \underline{a : s},\ b : s,\ \underline{f : s \to s};\ \textbf{deps } a < b,\ b < f),$$
$$SP'''_2 = \{f(a) = a\}.$$

Construction specifications $\underline{SP}_1$ *and* $\underline{SP}'''_2$ *are not compatible w.r.t. construction fitting ft, because the specification of assumed symbol* f *in* $\underline{SP}'''_2$ *is weaker than in* $\underline{SP}_1$*. Note that symbol* f *is assumed in both* $\underline{\mathcal{S}}_1$ *and* $\underline{\mathcal{S}}'''_2$*, so this example is quite different from the previous one. In this case it is easy to check that for any clean* $Con_2 \models^c \underline{SP}'''_2$ *and clean* $Con_1 \models^c \underline{SP}_1$ *it holds that* $Con_1 \oplus_{ft} Con_2 \models^c (\underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}'''_2))$*.*

In fact, the requirement that specifications of assumed symbols must be equivalent is not directly needed, because the theorem does not require the sum construction model to persistently extend the summand construction models in any way.

6.5.4 Other Operations as Sum Operation

The sum of construction specifications subsumes many typical operations on parameterised module specifications. This is possible because construction fittings are external to construction signatures, therefore, sums of the same construction specifications w.r.t. different construction fittings may give completely different results. We list some of the typical operations on parameterised module specifications (mostly taken from [EM90]) and represent them as sums w.r.t. some construction fittings. For every operation we give a simple example of the corresponding sum of construction signatures.

The parameterisation exercised in the framework of construction specifications is on a different level than in the parameterised module specifications. The former have parameterisation on individual symbols (together with their dependency structures), the latter only on whole signatures.

As the result, additional work may be required to represent operations on parameterised module specifications as sums of construction specifications. The main reason is that in construction fittings the dependency structures of shared symbols have to be same on both sides of the fitting, but there is no equivalent requirement on the fittings (morphisms or spans) between parameterised module specifications. Technically speaking, the problem is that some fittings fail to be $\mathbf{SigDep}^{frag}$-morphisms.

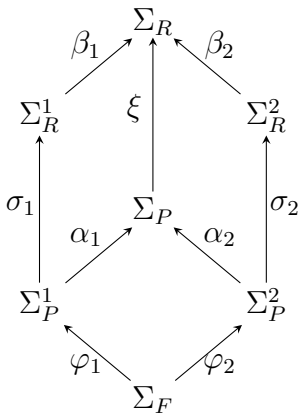
It should be noted, however, that construction specifications, thanks to the lower level of parameterisation, are always ready for partial and mutual applications without any additional assumptions that are needed in the case of parameterised module specifications.

Union. Given two parameterised module specifications, the union operation transforms them into one parameterised module specification, containing the union of both modules. The sharing of symbols is explicit via

an injective fitting span. Injectivity assures that no symbols from any signature are merged during the operation.

Union of parameterised module specifications corresponds directly to the sum of construction specifications connected by a construction fitting such that only assumed symbols are shared. Other sharing is excluded firstly, because the sharing of assumed and defined symbols corresponds to the application operation (explained below) and does not happen in a union operation, and secondly, because the sharing of defined symbols is inexpressible via a construction fitting definition (cf. discussion in Sect. 6.5.1). Another requirement concerns the dependency structure of shared symbols that needs to be exactly the same on both sides.

Consider parameterised module specifications $SP^1_M = \langle \sigma_1, SP^1_P, SP^1_R \rangle$ and $SP^2_M = \langle \sigma_2, SP^2_P, SP^2_R \rangle$ over some parameterised module signatures $\sigma_1 \colon \Sigma^1_P \to \Sigma^1_R$ and $\sigma_2 \colon \Sigma^2_P \to \Sigma^2_R$, respectively. Let there be an inclusive fitting span $\langle \varphi_1 \colon \Sigma_F \to \Sigma^1_P, \varphi_2 \colon \Sigma_F \to \Sigma^2_P \rangle$; we get $\langle \alpha_1, \alpha_2 \rangle$ as the pushout of $\langle \varphi_1, \varphi_2 \rangle$ and $\langle \beta_1, \beta_2 \rangle$ as the pushout of $\langle \varphi_1; \sigma_1, \varphi_2; \sigma_2 \rangle$. The situation is depicted on the following commuting diagram in **Sig**, where $\xi \colon \Sigma_P \to \Sigma_R$ is the universal arrow.



The union of SP^1_M and SP^2_M is the parameterised module specification $SP_M = \langle \xi, SP_P, SP_R \rangle$ with SP_P and SP_R obtained via pushout operations, i.e. $SP_P = \alpha_1(SP^1_P) \cup \alpha_2(SP^2_P)$ and $SP_R = \beta_1(SP^1_R) \cup \beta_2(SP^2_R)$.

Coming to constructions, let $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1 \rangle$ and $\underline{SP}_2 = \langle \underline{\mathcal{S}}_2, SP_2 \rangle$

be the construction specifications corresponding to SP^1_M and SP^2_M, respectively, as described in Sect. 6.4.3. In some cases, the simple dependency structure of symbols in construction signatures (from Sect. 6.2.1) must be adapted to make the specifications dependency-wise (cf. discussion in Sect. 6.4.3). Here we assume that construction signatures $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$ are defined in such a way that their shared symbols have the same dependency structure. For example given any linear dependency of shared symbols, any its extensions to linear orders in $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$ is sufficient. Let $\underline{\mathcal{F}}$ be defined as an empty construction signature $\mathbf{Empt}(\mathbf{Dep}(\Sigma_F))$ with all the symbols from Σ_F with the additional above-mentioned dependencies. Let $ft = \langle \underline{\varphi_1}, \underline{\varphi_2} \rangle$ be the construction fitting with $\underline{\varphi_1} = \langle \varepsilon_1, \varphi^1_P \rangle$ and $\underline{\varphi_2} = \langle \varepsilon_2, \varphi^2_P \rangle$, where $\varepsilon_1 \colon \emptyset \to D_{\sigma^1}$ and $\varepsilon_2 \colon \emptyset \to D_{\sigma^2}$ are the unique functions from the empty set to the sets of defined symbols in $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$, respectively. Clearly, by definition, ft is a construction fitting from Def. 6.28. The sum $\underline{SP}_1 \oplus_{ft} \underline{SP}_2$ corresponds to the union of SP^1_M and SP^2_M, given above.

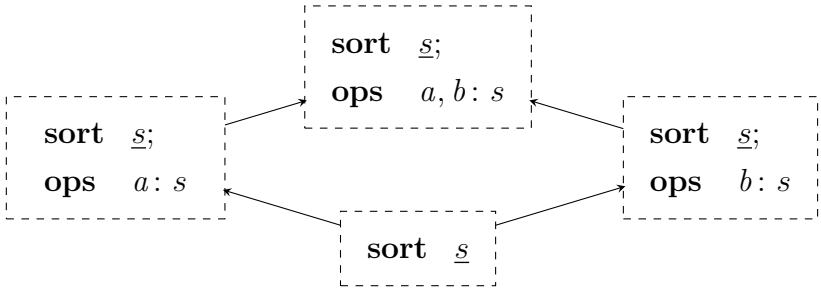
Example 6.41 *Consider the union of two parameterised module specifications:*

$$\langle (\mathbf{sort}\ s), \emptyset \rangle \to \langle (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s), \emptyset \rangle,$$
$$\langle (\mathbf{sort}\ s), \emptyset \rangle \to \langle (\mathbf{sort}\ s;\ \mathbf{ops}\ b : s), \emptyset \rangle$$

via the fitting span $\langle (\mathbf{sort}\ s) \to (\mathbf{sort}\ s), (\mathbf{sort}\ s) \to (\mathbf{sort}\ s) \rangle$. *The result is the parameterised module specification*

$$\langle (\mathbf{sort}\ s), \emptyset \rangle \to \langle (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ b : s), \emptyset \rangle.$$

The following sum of two construction signatures corresponds to the union operation



where all morphisms are inclusions and the basic dependency ($s < a$, $s < b$) is omitted. Symbol s is a shared assumed symbol. Symbol a is defined by the left construction signature, whereas defined symbol b is defined by the right construction signature.

Example 6.42 *To show a problematic case, consider the union of two parameterised module specifications:*

$$\begin{aligned}
&\langle(\textbf{sort } s;\ \textbf{ops } a : s,\ b : s), \{a = b\}\rangle \rightarrow \\
&\qquad\qquad \langle(\textbf{sort } s;\ \textbf{ops } a : s,\ b : s,\ c : s), \{a = b, b = c\}\rangle, \\
&\langle(\textbf{sort } s;\ \textbf{ops } a : s,\ d : s), \{a = d\}\rangle \rightarrow \\
&\qquad\qquad \langle(\textbf{sort } s;\ \textbf{ops } a : s,\ d : s,\ e : s), \{a = d, a = e\}\rangle
\end{aligned}$$

via the fitting span

$$\begin{aligned}
&\langle(\textbf{sort } s;\ \textbf{ops } a : s) \rightarrow (\textbf{sort } s;\ \textbf{ops } a : s,\ b : s), \\
&\ (\textbf{sort } s;\ \textbf{ops } a : s) \rightarrow (\textbf{sort } s;\ \textbf{ops } a : s,\ d : s)\rangle.
\end{aligned}$$

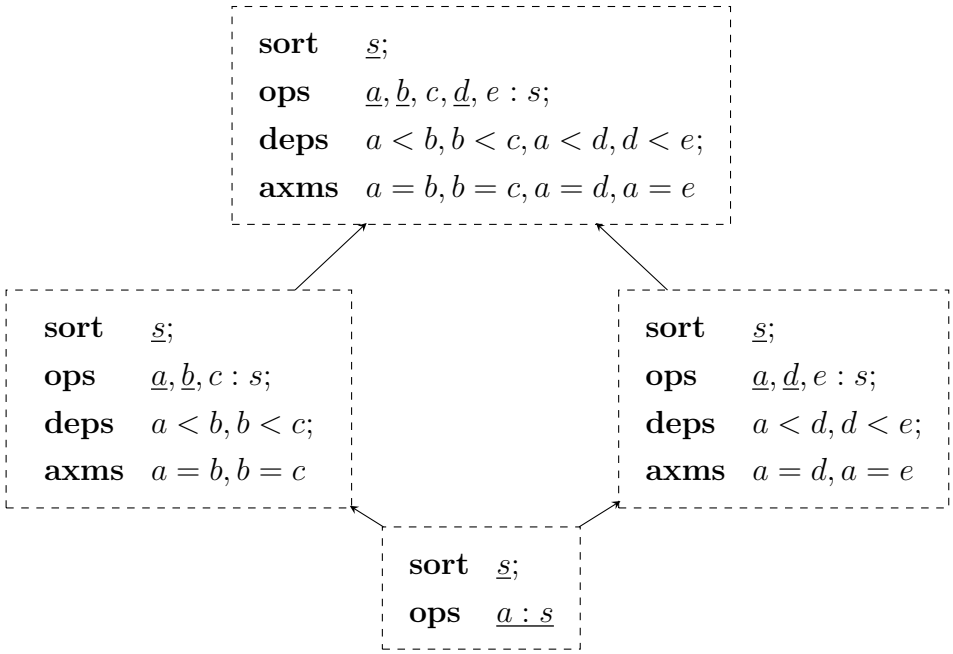
The result is the parameterised module specification

$$\begin{aligned}
&\langle(\textbf{sort } s;\ \textbf{ops } a : s,\ b : s,\ d : s), \{a = b, a = d\}\rangle \rightarrow \\
&\ \langle(\textbf{sort } s;\ \textbf{ops } a : s,\ b : s,\ c : s,\ d : s), \{a = b, b = c, a = d, a = e\}\}\rangle.
\end{aligned}$$

While transforming above parameterised module specifications to construction specifications, in order to make them dependency-wise, one needs to add some dependency between a and b in the first construction

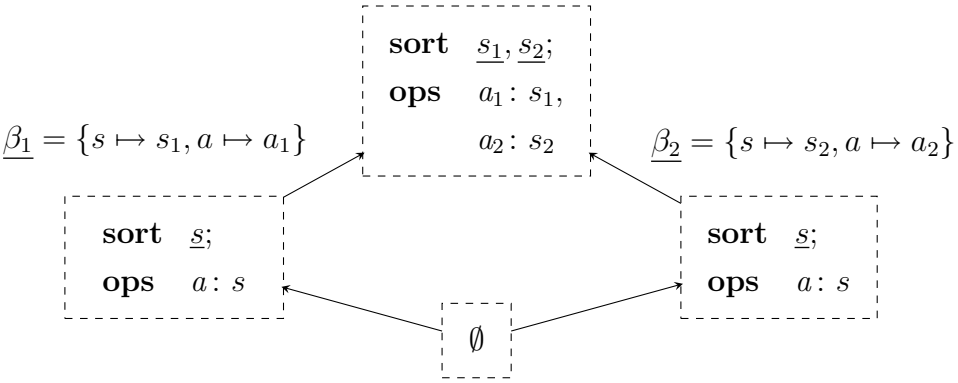
specification and another one between a and d in the second one. In the first case, both choices, $a < b$ and $b < a$, are equivalent when the first construction specification is concerned in isolation. However, to ensure compatibility with the second construction specification, only $a < b$ is the right choice, because $b < a$ makes the shared symbol a dependent on non-shared symbol b, therefore, the above-given fitting fails to be a p-morphism. The same applies to the second construction specification, where $a < d$ is the only choice.

The following sum of two construction specifications corresponds to the union operation



Disjoint union. For a parameterised module specification, a disjoint union operation is a kind of union operation (as described above) that duplicates all the otherwise shared symbols, because the fitting span with the empty source (which is always a construction fitting) is used. A special case is the disjoint union of a parameterised module with itself.

Example 6.43 *The same construction signature connected by a fitting with no symbols in its source corresponds to the disjoint union operation*



where the basic dependency is omitted. No symbols are shared.

Composition. Given two parameterised module specifications, the composition operation uses a fitting morphism to connect the result of the first parameterised module to the parameter of the second one.

Consider parameterised module specifications $SP^1_M = \langle \sigma_1, SP^1_P, SP^1_R \rangle$ and $SP^2_M = \langle \sigma_2, SP^2_P, SP^2_R \rangle$ over some parameterised module signatures $\sigma_1 \colon \Sigma^1_P \to \Sigma^1_R$ and $\sigma_2 \colon \Sigma^2_P \to \Sigma^2_R$, respectively. Let $\varphi \colon SP^2_P \to SP^1_R$ be a specification morphism; let $\sigma'_2 \colon \Sigma^1_R \to \Sigma_R$ and $\varphi' \colon \Sigma^2_R \to \Sigma_R$ be the pushout of φ and σ_2, as on the commuting diagram in **Sig** below.

$$\begin{array}{ccc} \Sigma^1_P & \xrightarrow{\sigma_1} & \Sigma^1_R \\ & & \downarrow{\scriptstyle \varphi} \\ \Sigma^2_P & \xrightarrow{\sigma_2} & \Sigma^2_R \end{array} \qquad \begin{array}{ccc} \Sigma^1_R & \xrightarrow{\sigma'_2} & \Sigma_R \\ \downarrow{\scriptstyle \varphi} & & \downarrow{\scriptstyle \varphi'} \\ \Sigma^2_P & \xrightarrow{\sigma_2} & \Sigma^2_R \end{array}$$

The result of the composition of SP^1_M and SP^2_M via φ is the parameterised module specification $SP_M = \langle \sigma_1; \sigma'_2, SP^1_P, SP_R \rangle$ over $\sigma_1; \sigma'_2$ with $SP_R = \sigma'_2(SP^1_R) \cup \varphi'(SP^2_R)$.

In order to present the composition as a sum of construction specifications, let $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1 \rangle$ and $\underline{SP}_2 = \langle \underline{\mathcal{S}}_2, SP_2 \rangle$ be the construction specifications corresponding to SP^1_M and SP^2_M, respectively, defined in such a way that they are dependency-wise construction specifications

(cf. discussion in Sect. 6.4.3). Moreover the care must be taken to prepare correctly construction signatures $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$, so that every assumed symbol from $\underline{\mathcal{S}}_2$ has the same dependency structure as the related (by φ) symbol from $\underline{\mathcal{S}}_1$. If this is impossible without adding new symbols to $\underline{\mathcal{S}}_2$, either $\underline{SP}_2$ needs to be altered so that the (part of) the parameter specification from $\underline{SP}_1$ is added to $\underline{SP}_2$ or some refactoring of the whole setting is required (cf. Sect. 8.3 below for proposed approach in such case). Let $\underline{\mathcal{F}} = \mathbf{Empt}(\mathbf{Dep}(\Sigma_P^2))$ and $ft = \langle \underline{\varphi}, \underline{\sigma_2} \rangle$ with $\underline{\varphi} = \langle \varepsilon_1, \varphi \rangle$ and $\underline{\sigma_2} = \langle \varepsilon_2, \sigma_2 \rangle$, where $\varepsilon_1 \colon \emptyset \to D_{\sigma_1}$ and $\varepsilon_2 \colon \emptyset \to D_{\sigma_2}$ are the unique functions from the empty set to the sets of defined symbols in $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$, respectively. Assuming that the dependency structures of related symbols coincide, as discussed above, $\underline{\varphi}$ and $\underline{\sigma_2}$ are $\mathbf{SigDep}^{frag}$-morphisms, and also ft is a construction fitting. The sum of $\underline{SP}_1$ and $\underline{SP}_2$ w.r.t. ft corresponds to the composition of SP_M^1 and SP_M^2 via φ.

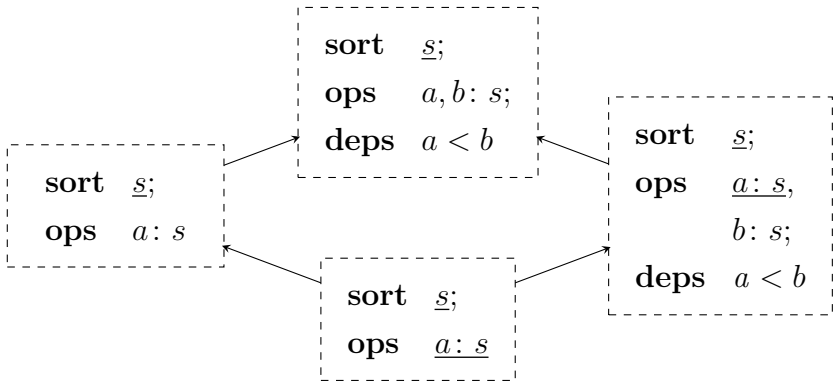
Example 6.44 *Consider the composition of the following parameterised module specifications:*

$$\langle (\mathbf{sort}\ s), \emptyset \rangle \to \langle (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s), \emptyset \rangle,$$
$$\langle (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s), \emptyset \rangle \to \langle (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ b : s), \emptyset \rangle$$

via the fitting morphism $(\mathbf{sort}\ s;\ \mathbf{ops}\ a : s) \to (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s)$. *The result is the parameterised module specification*

$$\langle (\mathbf{sort}\ s), \emptyset \rangle \to \langle (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ b : s), \emptyset \rangle.$$

The below-given sum of two construction signatures corresponds to the above composition



where all morphisms are inclusions and the basic dependency is omitted. Both a and s are shared; however, there is a difference between them. The symbol s is a shared parameter, whereas a is defined in the left construction. Symbol b is defined in the right construction; it depends on a that is a parameter there. In the resulting construction only the symbol s is a parameter. This example, read literally, can also be interpreted as a partial application (see below)*.*

Example 6.45 *The following composition of parameterised module specifications requires some transformation prior to its representation as a sum of construction specifications. Consider the composition of two parameterised module specifications*

$$\langle(\textbf{sort } s;\ \textbf{ops } a : s), \emptyset\rangle \to \langle(\textbf{sort } s;\ \textbf{ops } a : s,\ b : s), \emptyset\rangle,$$
$$\langle(\textbf{sort } s;\ \textbf{ops } b : s), \emptyset\rangle \to \langle(\textbf{sort } s;\ \textbf{ops } b : s,\ c : s), \emptyset\rangle$$

via the inclusive fitting morphism $(\textbf{sort } s;\ \textbf{ops } b : s) \to (\textbf{sort } s;\ \textbf{ops } a : s,\ b : s)$*.*

The result is the parameterised module specification

$$\langle(\textbf{sort } s;\ \textbf{ops } a : s), \emptyset\rangle \to \langle(\textbf{sort } s;\ \textbf{ops } a : s,\ b : s,\ c : s), \emptyset\rangle.$$

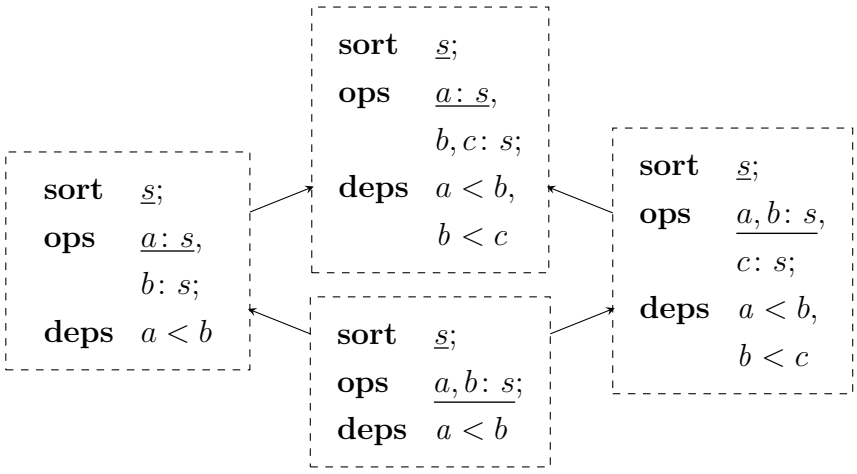
The construction specifications corresponding to the above parameterised

module specifications are

$$\underline{SP}_1 = \langle(\textbf{sort } \underline{s};\ \textbf{ops } \underline{a : s},\ b : s;\ \textbf{deps } a < b), \emptyset\rangle,$$
$$\underline{SP}_2 = \langle(\textbf{sort } \underline{s};\ \textbf{ops } \underline{b : s},\ c : s;\ \textbf{deps } b < c), \emptyset\rangle.$$

The problem is that the fitting morphism is not a $\mathbf{SigDep}^{frag}$*-morphism, because in the target construction signature we have* $a < b$*, and in the source construction signature there is no* $a : s$*, so for no constant* x*,* $x < b$*, and the morphism is not a p-morphism.*

One way to handle such a situation is to amend the second construction specification simply by adding $a : s$ *as an assumed symbol to its construction signature. The following sum of two construction signatures corresponds to the above composition after the amendment of the second construction specification*



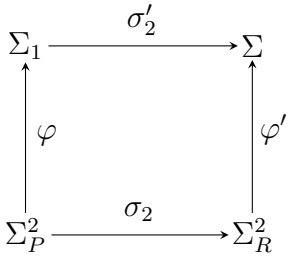
where all morphisms are inclusions.

Another approach to handle this kind of problem is presented in Example 8.13 below.

Application. For a non-parameterised module specification, a parameterised module specification and a fitting morphism, the application operation applies the second one to the first one via the given fitting. As result,

we get a non-parameterised module specification. The application can be seen as a special case of a composition (as described above).

Consider a non-parameterised module specification SP_1 over Σ_1 and a parameterised module specification $SP^2_M = \langle \sigma_2, SP^2_P, SP^2_R \rangle$, where $\sigma_2 \colon \Sigma^2_P \to \Sigma^2_R$. Let there be a specification morphism $\varphi \colon SP^2_P \to SP_1$ and let $\sigma'_2 \colon \Sigma_1 \to \Sigma$ and $\varphi' \colon \Sigma^2_R \to \Sigma$ be the pushout of φ and σ_2.



The result of the application of SP^2_M to SP_1 via φ is a non-parameterised module specification $SP = \sigma'_2(SP_1) \cup \varphi'(SP^2_R)$ over Σ.

To present the application as a sum of construction specifications, let $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1 \rangle$ and $\underline{SP}_2 = \langle \underline{\mathcal{S}}_2, SP_2 \rangle$ be the construction specifications corresponding to SP_1 and SP^2_M, respectively (cf. Sect. 6.4.3). By default they are defined in such a way that they are construction specifications (cf. Sect. 6.4.3) and every assumed symbol in $\underline{\mathcal{S}}_2$ has the same dependency structure (including basic dependency) as the related (by φ) symbol from $\underline{\mathcal{S}}_1$. Let $\underline{\mathcal{F}} = \mathbf{Empt}(\mathbf{Dep}(\Sigma^2_P))$ and $ft = \langle \underline{\varphi}, \underline{\sigma_2} \rangle$ with $\underline{\varphi} = \langle \varepsilon_1, \varphi \rangle$ and $\underline{\sigma_2} = \langle \varepsilon_2, \sigma_2 \rangle$, where $\varepsilon_1 \colon \emptyset \to \mathbf{SetSymb}(\Sigma_1)$ and $\varepsilon_2 \colon \emptyset \to D_{\sigma_2}$ are the unique functions from the empty set to the sets of defined symbols in $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$, respectively. By assumption about the same dependency structures, *ft* is a construction fitting and the sum of $\underline{SP}_1$ and $\underline{SP}_2$ w.r.t. *ft* corresponds to the application of SP^2_M to SP^1 via φ.

Example 6.46 *When the parameterised module specification*

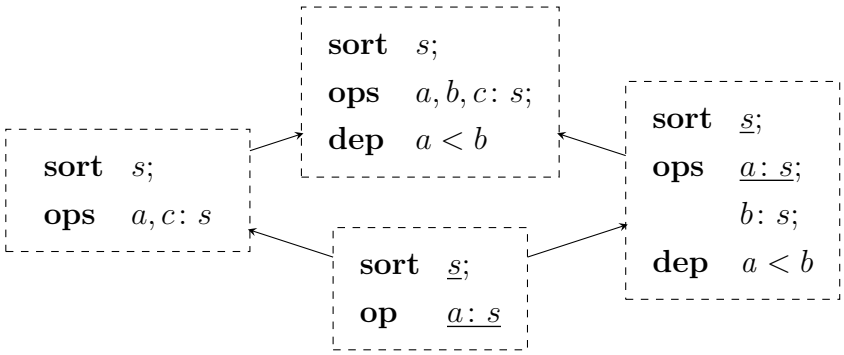
$$\langle (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s), \emptyset \rangle \to \langle (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ b : s), \emptyset \rangle$$

is applied to the module specification $\langle (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ c : s), \emptyset \rangle$ *via*

the inclusive fitting morphism (**sort** s; **ops** $a : s$) $\rightarrow$ (**sort** s; **ops** $a : s,\ c : s$)*, the result is the non-parameterised module specification*

$$\langle(\textbf{sort}\ s;\ \textbf{ops}\ a : s,\ b : s,\ c : s), \emptyset\rangle.$$

The sum of construction signatures, as depicted below, corresponds to the above-given application operation



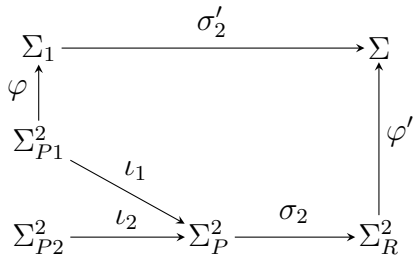
where all morphisms are inclusions and the basic dependency is omitted. Both a and s are shared. They are defined by the left construction and they are a parameter of the right construction. The left construction additionally defines c that appears in the result of the application.

Partial application. Partial application is similar to application, but not all parameters are instantiated. The result is a parameterised module specification.

To use the partial application we need to require some additional structure on the parameter of the parameterised module. Here we assume that the specification of a parameter is a coproduct of two specifications, but in general any colimit may be used for that purpose.

Consider a non-parameterised module specification SP_1 over a signature Σ_1 and a parameterised module specification $SP^2_M = \langle \sigma_2, SP^2_P, SP^2_R\rangle$ over $\sigma_2 \colon \Sigma^2_P \rightarrow \Sigma^2_R$. Let there be two specifications SP^2_{P1} and SP^2_{P2} over Σ^2_{P1} and Σ^2_{P2}, respectively, such that Σ^2_P is a coproduct of Σ^2_{P1} and Σ^2_{P2} via $\iota_1 \colon \Sigma^2_{P1} \rightarrow \Sigma^2_P$ and $\iota_2 \colon \Sigma^2_{P2} \rightarrow \Sigma^2_P$ and $[\![SP^2_P]\!] = [\![\iota_1(SP^2_{P1}) \cup \iota_2(SP^2_{P2})]\!]$.

Let there also be a specification morphism $\varphi\colon SP^2_{P1} \to SP_1$. Let $\sigma'_2\colon \Sigma_1 \to \Sigma$ and $\varphi'\colon \Sigma^2_R \to \Sigma$ be the pushout of φ and $\iota_1;\sigma_2$.



The result of the partial application of SP^2_M to SP^1 via φ is a parameterised module specification $SP_M = \langle SP^2_{P2}, SP\rangle$ over $\iota_2;\sigma_2;\varphi'$ with $SP = \sigma'_2(SP_1) \cup \varphi'(SP^2_R)$.

In the framework of construction specifications, the explicit decomposition of SP^2_P into SP^2_{P1} and SP^2_{P2} is not necessary. Let $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1\rangle$ and $\underline{SP}_2 = \langle \underline{\mathcal{S}}_2, SP_2\rangle$ be the construction specifications corresponding to SP_1 and SP^2_M, respectively (cf. Sect. 6.4.3). By default they are construction specifications indeed (cf. Sect. 6.4.3) and every assumed symbol from $\underline{\mathcal{S}}_2$ (from Σ^2_{P1} to be precise) has the same dependency structure (including basic dependency) in $\underline{\mathcal{S}}_2$ as in $\underline{\mathcal{S}}_1$ (via φ). Let $\underline{\mathcal{F}} = \mathbf{Empt}(\mathbf{Dep}(\Sigma^2_{P1}))$ and $ft = \langle \underline{\varphi}, \underline{\sigma_2}\rangle$ with $\underline{\varphi} = \langle \varepsilon_1, \varphi\rangle$ and $\underline{\sigma_2} = \langle \varepsilon_2, (\iota_1;\sigma_2)\rangle$, where $\varepsilon_1\colon \emptyset \to \mathbf{SetSymb}(\Sigma^1)$ and $\varepsilon_2\colon \emptyset \to D_{\sigma_2}$ are the unique functions from the empty set to the sets of defined symbols in $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$, respectively. The above assumptions make ft a construction fitting. The sum of $\underline{SP}_1$ and $\underline{SP}_2$ w.r.t. ft corresponds to the partial application of SP^2_M to SP^1 via φ.

Example 6.47 *Consider the parameterised module specification*

$$\langle(\mathbf{sort}\ s,\ t;\ \mathbf{ops}\ a : s,\ b : t), \emptyset\rangle \to \langle(\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ b : t,\ c : s), \emptyset\rangle$$

with the parameter being the coproduct of $\langle(\mathbf{sort}\ s;\ \mathbf{ops}\ a : s), \emptyset\rangle$ *and*

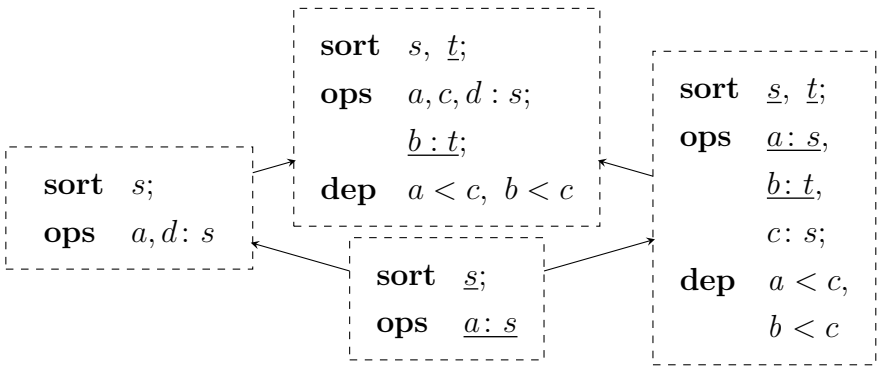
$\langle(\textbf{sort}\ t;\ \textbf{ops}\ b : t), \emptyset\rangle$. *Let there also be the module specification*

$$\langle(\textbf{sort}\ s;\ \textbf{ops}\ a : s,\ d : s), \emptyset\rangle$$

and the inclusive fitting morphism $(\textbf{sort}\ s;\ \textbf{ops}\ a : s) \to (\textbf{sort}\ s;\ \textbf{ops}\ a : s,\ d : s)$. *The result of the partial application is*

$$\langle(\textbf{sort}\ t;\ \textbf{ops}\ b : t), \emptyset\rangle \to \langle(\textbf{sort}\ s, t;\ \textbf{ops}\ a : s,\ b : t,\ c : s,\ d : s), \emptyset\rangle.$$

The following sum of construction signatures corresponds to the above partial application



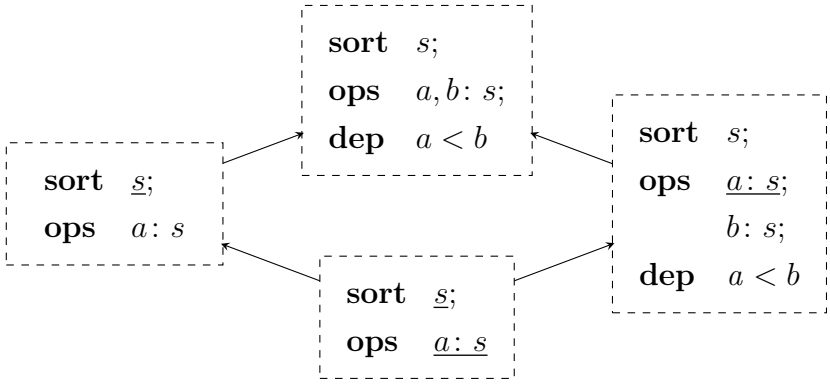
where all morphisms are inclusions and the basic dependency is omitted. The symbol s is shared. It is defined in the left construction and it is a part of the parameter of the right construction. The symbol a is a parameter on the right side and it remains a parameter in the result.

Mutual application. Given two parameterised modules, mutual application instantiates parameters on both sides simultaneously. This operation is a generalisation of partial application.

We do not formalize mutual application of parameterised modules, because in general this operation is not well defined and all special cases are slightly too complex to consider them as basic operations.

However, for construction specifications, the mutual (partial) application may be easily represented as a sum operation w.r.t. the construction fitting that shares assumed and defined symbols from both sides.

Example 6.48 *The diagram below is an example of a sum of construction signatures corresponding to mutual application*



where all morphisms are inclusions and the basic dependency is omitted. The symbol s is a parameter of the left construction and it is defined by the right construction. The opposite situation concerns a, which is defined by the left construction and it is a parameter in the right one. Both s and a are shared. The result is a construction without assumed symbols (i.e. parameters), corresponding to a non-parameterised module.

6.A Appendix: Proofs

The following definition and lemma are going to be useful throughout the proofs.

Definition 6.49 (Dependency Structure Morphisms) *Given two signature fragments $\underline{\mathcal{S}}_1, \underline{\mathcal{S}}_2 \in \mathbf{SigDep}^{frag}$, a set $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}}_1)$ and a morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$,*

- *the* closed-down subsignature fragment morphism induced by *A, denoted by $\underline{\omega}_A\colon \underline{\mathcal{S}}_1^A{\downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(A)}{\downarrow}$, is given as $\underline{\omega}_A = \underline{\omega}|_{\underline{\mathcal{S}}_1^A{\downarrow}}$*
- *if A is a set of independent symbols in $\mathbf{Compl}(\underline{\mathcal{S}}_1)$ and $\underline{\omega}(A)$ is a set of independent symbols*[3] *in $\mathbf{Compl}(\underline{\mathcal{S}}_2)$, the* dependency structure subsignature fragment morphism induced by *A, $\underline{\omega}_A^-\colon \underline{\mathcal{S}}_1^A{\Downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(A)}{\Downarrow}$, is given as $\underline{\omega}_A^- = \underline{\omega}|_{\underline{\mathcal{S}}_1^A{\Downarrow}}$*

$$\begin{array}{ccc}
\underline{\mathcal{S}}_1 & \xrightarrow{\underline{\omega}} & \underline{\mathcal{S}}_2 \\
\uparrow & & \uparrow \\
\underline{\mathcal{S}}_1^A{\downarrow} & \xrightarrow{\underline{\omega}_A} & \underline{\mathcal{S}}_2^{\underline{\omega}(A)}{\downarrow} \\
\uparrow & & \uparrow \\
\underline{\mathcal{S}}_1^A{\Downarrow} & \xrightarrow{\underline{\omega}_A^-} & \underline{\mathcal{S}}_2^{\underline{\omega}(A)}{\Downarrow}
\end{array}$$

The above definition uses the fact that $\mathbf{SigDep}^{frag}$ has an inclusion system. It is easy to check that $\underline{\omega}_A, \underline{\omega}_A^- \in \mathbf{SigDep}^{frag}$ and the diagram in Def. 6.49 is commuting.

Notation. When $A = \{b\}$, we sometimes write $\underline{\omega}_b\colon \underline{\mathcal{S}}_1^b{\downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(b)}{\downarrow}$ and $\underline{\omega}_b^-\colon \underline{\mathcal{S}}_1^b{\Downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(b)}{\Downarrow}$, i.e. we use an element, instead of a singleton set that consists of this element.

Lemma 6.50 *Closed-down subsignature morphisms and dependency structure morphisms are surjective on symbols, i.e. both $\mathbf{Symb}(\mathbf{Compl}(\underline{\omega}_A))$ and $\mathbf{Symb}(\mathbf{Compl}(\underline{\omega}_A^-))$ are surjections (using notation from Def. 6.49).*

[3] An important assumption for $\underline{\omega}$ such that it is not injective on symbols.

Proof. P-morphisms by definition are surjective on the dependency structure of every element of the source signature. □

Proof of Lemma 6.8. Let there be a construction signature morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ and a construction model $Con_2 \in [\![\underline{\mathcal{S}}_2]\!]^c$. Let $a \in \underline{\mathcal{S}}_1$ and let there be two models $M_1, M_1' \in Con_2|_{\underline{\omega}}$ such that $M_1|_{\underline{\mathcal{S}}_1^a\Downarrow} = M_1'|_{\underline{\mathcal{S}}_1^a\Downarrow}$. We have to show that $M_1|_{\underline{\mathcal{S}}_1^a\downarrow} = M_1'|_{\underline{\mathcal{S}}_1^a\downarrow}$. There exist two models $M_2, M_2' \in Con_2$ such that $M_2|_{\underline{\omega}} = M_1$ and $M_2'|_{\underline{\omega}} = M_1'$. From $(M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^a\Downarrow} = (M_2'|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^a\Downarrow}$, by commutativity of the diagram in Def. 6.49, we obtain $(M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow})|_{\underline{\omega_a^-}} = (M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow}))|_{\underline{\omega_a^-}}$. By Lemma 6.50, the morphism $\underline{\omega_a^-}\colon \underline{\mathcal{S}}_1^a\Downarrow \to \underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow$ is surjective, therefore, by assumption from the beginning of Sect. 3.5, the reduct functor $\_|_{\underline{\omega_a^-}}\colon [\![\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow]\!] \to [\![\underline{\mathcal{S}}_1^a\Downarrow]\!]$ is injective on models. Consequently, we get $M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow}$ and, by Def. 6.5, $M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow}$. This yields $M_1|_{\underline{\mathcal{S}}_1^a\downarrow} = M_1'|_{\underline{\mathcal{S}}_1^a\downarrow}$, because $M_1|_{\underline{\mathcal{S}}_1^a\downarrow} = (M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow})|_{\underline{\omega_a}} = (M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow})|_{\underline{\omega_a}} = M_1'|_{\underline{\mathcal{S}}_1^a\downarrow}$ □

Proof of Lemma 6.19. In consequence of the assumptions from Sect. 3.5, every signature in **Sig** has a model. It is easy to check that, by condition (3) of Def. 6.13 (for $A = \emptyset$), every construction model satisfying construction specification is non-trivial, i.e., there exists a model (from the base institution $\mathbb{I}$) in it. Then, by condition (4) of the same definition, the said model reduced to initial signature is required to satisfy the specification (reduced to the initial signature as well) in the base institution $\mathbb{I}$ and this makes the specification consistent. □

Proof of Theorem 6.22. Let there be a dependency-wise construction specification $\underline{SP} = \langle \underline{\mathcal{S}}, SP \rangle$ over a finite construction signature $\underline{\mathcal{S}}$. Assumptions from Sect. 3.5 ensure that every signature in **Sig** has a model, so let there be a model $M \in [\![\underline{\mathcal{S}}]\!]$. Using the assumption that $\underline{SP}$ is dependency-wise, for $A = \emptyset$, we get $M|_{\Sigma_\emptyset} \models SP|_{\Sigma_\emptyset}$, where $\Sigma_\emptyset$ is the initial object in **Sig**; this means that there exists $M' \in [\![SP]\!]$ such that $M'|_{\Sigma_\emptyset} = M|_{\Sigma_\emptyset}$, i.e. SP is consistent in the base institution $\mathbb{I}$.

By induction we define a finite sequence of sets of symbols from $\mathbf{Compl}(\underline{\mathcal{S}})$ $\langle A_i\rangle_{0\leq i\leq n}$, where $n = db(\underline{\mathcal{S}})$ is the dependency bound of $\underline{\mathcal{S}}$ (cf. Sect.5.3). Let $A_0 = \emptyset$ and $A_{i+1} = \{a \in \underline{\mathcal{S}} \mid \text{for all } a' \in \mathbf{Compl}(\underline{\mathcal{S}}) \text{ such that } a' < a, \text{ it holds } a' \in A_i \text{ or } a' \notin \underline{\mathcal{S}}\}$. For $i \in \{0,\ldots,n-1\}$, A_i is a set of defined symbols such that $A_i \subseteq A_{i+1}$, moreover $A_{i+1} \setminus A_i$ is a set of independent symbols.

Let $Con_0 = [\![\underline{SP}]\!]$. If $n > 0$, for $i \in \{0,\ldots,n-1\}$ let Con_{i+1} be a maximal subset of Con_i such that for any $M, M' \in Con_{i+1}$ and for each $a \in A_{i+1}$, $a \in \underline{\mathcal{S}}$, if $M|_{\underline{\mathcal{S}}^{a\Downarrow}} = M'|_{\underline{\mathcal{S}}^{a\Downarrow}}$ then $M|_{\underline{\mathcal{S}}^{a\downarrow}} = M'|_{\underline{\mathcal{S}}^{a\downarrow}}$. The existence of Con_{i+1} is guaranteed by the Kuratowski-Zorn lemma (hence also by the axiom of choice). It is easily visible that Con_n is a construction model.

Let us now prove that Con_n satisfies $\underline{SP}$ according to Def. 6.13.

Condition (1) is discharged by the construction of Con_n, because for any $M \in Con_n \subseteq Con_0$, $M \models SP$.

We prove that Con_i satisfies condition (2) by induction on $i \in \{0\ldots n\}$. For $i = 0$ this is obvious. Suppose it holds for $i \in \{0\ldots n-1\}$, and consider an assumed symbol $a \in \mathbf{Compl}(\underline{\mathcal{S}})$, $a \notin \underline{\mathcal{S}}$ and a model $M \models SP$, $M|_{\underline{\mathcal{S}}^{a\Downarrow}} \in Con_{i+1}|_{\underline{\mathcal{S}}^{a\Downarrow}}$.

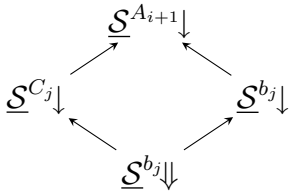
There are two cases.

1. Assume $a \in \mathbf{Compl}(\underline{\mathcal{S}}^{A_{i+1}\downarrow})$. Since $Con_{i+1} \subseteq Con_i$, we have $M|_{\underline{\mathcal{S}}^{a\Downarrow}} \in Con_i|_{\underline{\mathcal{S}}^{a\Downarrow}}$; therefore, by the inductive assumption, $M|_{\underline{\mathcal{S}}^{a\downarrow}} \in Con_i|_{\underline{\mathcal{S}}^{a\downarrow}}$. This means that there exists $M' \in Con_i$ such that, $M'|_{\underline{\mathcal{S}}^{a\downarrow}} = M|_{\underline{\mathcal{S}}^{a\downarrow}}$. If $M' \in Con_{i+1}$ then $M|_{\underline{\mathcal{S}}^{a\downarrow}} \in Con_{i+1}|_{\underline{\mathcal{S}}^{a\downarrow}}$, as required.

   Otherwise, if $M' \notin Con_{i+1}$, let $B = \{b \in A_{i+1} \mid \text{there exists } M_b \in Con_{i+1} \text{ such that } M_b|_{\underline{\mathcal{S}}^{b\Downarrow}} = M'|_{\underline{\mathcal{S}}^{b\Downarrow}} \text{ and } M_b|_{\underline{\mathcal{S}}^{b\downarrow}} \neq M'|_{\underline{\mathcal{S}}^{b\downarrow}}\}$. This means that for all $a' \in (A_{i+1} \setminus B)$, for any $M'' \in Con_{i+1}$, if $M''|_{\underline{\mathcal{S}}^{a'\Downarrow}} = M'|_{\underline{\mathcal{S}}^{a'\Downarrow}}$ then $M''|_{\underline{\mathcal{S}}^{a'\downarrow}} = M'|_{\underline{\mathcal{S}}^{a'\downarrow}}$. We notice that $B \subseteq (A_{i+1} \setminus A_i)$, because for each $b \in (B \cap A_i)$, for any $M'' \in Con_{i+1}$ such that $M''|_{\underline{\mathcal{S}}^{b\Downarrow}} = M'|_{\underline{\mathcal{S}}^{b\Downarrow}}$, since $M', M'' \in Con_i$, by definition of Con_i, $M''|_{\underline{\mathcal{S}}^{b\downarrow}} = M'|_{\underline{\mathcal{S}}^{b\downarrow}}$. Therefore, B is a set of independent symbols. If $a \leq b$ for some $b \in B$, from $a \neq b$ (symbol a is assumed and B is a set of defined symbols) we have $a < b$ and there is $M_b \in Con_{i+1}$ such that $M_b|_{\underline{\mathcal{S}}^{b\Downarrow}} = M'|_{\underline{\mathcal{S}}^{b\Downarrow}}$. We

get $M_b|_{\underline{\mathcal{S}}^{a}\downarrow} = M'|_{\underline{\mathcal{S}}^{a}\downarrow} = M|_{\underline{\mathcal{S}}^{a}\downarrow}$, hence, $M|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con_{i+1}|_{\underline{\mathcal{S}}^{a}\downarrow}$, as required.

Assume $a \notin \mathbf{Compl}(\underline{\mathcal{S}}^{B}\downarrow)$. By assumption that $\underline{\mathcal{S}}$ is finite, B also is finite, so let $m = |B|$, let us arbitrarily name its elements as $B = \{b_1 \dots b_m\}$ and let the corresponding models from Con_{i+1} be named as $M^b_1, \dots, M^b_m$. For $1 \leq j \leq m$, let $C_j = (A_{i+1} \backslash \{b_j\}) \cup \mathbf{Symb}(\mathbf{Compl}(\underline{\mathcal{S}}^{b_j}\Downarrow))$, the following square of inclusions is a pushout in $\mathbf{SigDep}^{frag}$.



Let $M_1 = M'$, by induction on $j \in \{2, \dots, m\}$, given $M_{j-1} \models SP$ we construct $M_j \models SP$ in the following way. By definition of B and construction of M_{j-1}, we have $M^b_j|_{\underline{\mathcal{S}}^{b_j}\Downarrow} = M'|_{\underline{\mathcal{S}}^{b_j}\Downarrow} = M_{j-1}|_{\underline{\mathcal{S}}^{b_j}\Downarrow}$. Let $M''_j \in [\![\underline{\mathcal{S}}^{A_{i+1}}\downarrow]\!]$ be the amalgamation of $M_{j-1}|_{\underline{\mathcal{S}}^{C_j}\downarrow}$ and $M^b_j|_{\underline{\mathcal{S}}^{b_j}\downarrow}$ w.r.t. the above pushout diagram. By assumption (7) from Sect. 3.5 there exists a model $M'''_j \in [\![\underline{\mathcal{S}}]\!]$ such that $M''_j = M'''_j|_{\underline{\mathcal{S}}^{A_{i+1}}\downarrow}$. Since $\underline{SP}$ is dependency-wise, from $M'''_j|_{\underline{\mathcal{S}}^{C_j}\downarrow} \models SP|_{\underline{\mathcal{S}}^{C_j}\downarrow}$ and $M'''_j|_{\underline{\mathcal{S}}^{b_j}\downarrow} \models SP|_{\underline{\mathcal{S}}^{b_j}\downarrow}$ we get $M'''_j|_{\underline{\mathcal{S}}^{A_{i+1}}\downarrow} \models SP|_{\underline{\mathcal{S}}^{A_{i+1}}\downarrow}$. This means that there exists $M_j \models SP$ such that $M_j|_{\underline{\mathcal{S}}^{A_{i+1}}\downarrow} = M'''_j|_{\underline{\mathcal{S}}^{A_{i+1}}\downarrow}$.

By induction on $j \in \{0, \dots, i+1\}$ we prove that $M_m \in Con_j$. For $j = 0$, this is obvious, because $M_m \models SP$. Let us assume $M_m \in Con_j$ for some $j \in \{0, \dots, i\}$.

By contradiction, assume that $M_m \notin Con_{j+1}$. This means that there exists $M^Y \in Con_{j+1}$ and $a' \in A_{j+1}$ such that $M^Y|_{\underline{\mathcal{S}}^{a'}\Downarrow} = M_m|_{\underline{\mathcal{S}}^{a'}\Downarrow}$ and $M^Y|_{\underline{\mathcal{S}}^{a'}\downarrow} \neq M_m|_{\underline{\mathcal{S}}^{a'}\downarrow}$.

For $j < i$, from $A_{j+1} \subseteq A_i$ we have $a' \in A_i$, hence $a' \in C_k$, for all $1 \leq k \leq m$. From $Con_i \subseteq Con_{j+1}$ it holds $M_m|_{\underline{\mathcal{S}}^{a'}\downarrow} = M'''_m|_{\underline{\mathcal{S}}^{a'}\downarrow} = M''_m|_{\underline{\mathcal{S}}^{a'}\downarrow} = M_{m-1}|_{\underline{\mathcal{S}}^{a'}\downarrow} = \dots = M_1|_{\underline{\mathcal{S}}^{a'}\downarrow} = M'|_{\underline{\mathcal{S}}^{a'}\downarrow} \in Con_i|_{\underline{\mathcal{S}}^{a'}\downarrow} \subseteq Con_{j+1}|_{\underline{\mathcal{S}}^{a'}\downarrow}$. This means that there exists $M^Z \in Con_{j+1}$ such that $M^Z|_{\underline{\mathcal{S}}^{a'}\downarrow} = M_m|_{\underline{\mathcal{S}}^{a'}\downarrow}$. We have $M^Y|_{\underline{\mathcal{S}}^{a'}\Downarrow} = M_m|_{\underline{\mathcal{S}}^{a'}\Downarrow} = M^Z|_{\underline{\mathcal{S}}^{a'}\Downarrow}$, thus, since $M^Y, M^Z \in Con_{j+1}$,

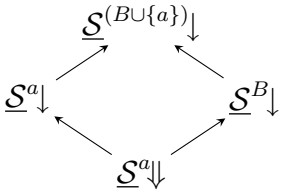
$M^Y|_{\underline{\mathcal{S}}^{a'}\downarrow} = M^Z|_{\underline{\mathcal{S}}^{a'}\downarrow} = M_m|_{\underline{\mathcal{S}}^{a'}\downarrow}$. Contradiction.

For $j = i$, if $a' \in B$ then for some $k \in \{1, \ldots, m\}$, $a' = b_k$, thus $M_m|_{\underline{\mathcal{S}}^{a'}\downarrow} = M'''_m|_{\underline{\mathcal{S}}^{a'}\downarrow} = M''_m|_{\underline{\mathcal{S}}^{a'}\downarrow} = \cdots = M_k|_{\underline{\mathcal{S}}^{a'}\downarrow} = M^b_k|_{\underline{\mathcal{S}}^{a'}\downarrow} \in Con_{i+1}|_{\underline{\mathcal{S}}^{a'}\downarrow} = Con_{j+1}|_{\underline{\mathcal{S}}^{a'}\downarrow}$. As above, this leads to contradiction.

If $a' \in (A_{i+1} \setminus B)$, by definition of B, since $M^Y \in Con_{i+1}$ and $M^Y|_{\underline{\mathcal{S}}^{a'}\Downarrow} = M_m|_{\underline{\mathcal{S}}^{a'}\Downarrow} = M'''_m|_{\underline{\mathcal{S}}^{a'}\Downarrow} = M''_m|_{\underline{\mathcal{S}}^{a'}\Downarrow} = M_{m-1}|_{\underline{\mathcal{S}}^{a'}\Downarrow} = \cdots = M_1|_{\underline{\mathcal{S}}^{a'}\Downarrow} = M'|_{\underline{\mathcal{S}}^{a'}\Downarrow}$, we have $M^Y|_{\underline{\mathcal{S}}^{a'}\downarrow} = M'|_{\underline{\mathcal{S}}^{a'}\downarrow} = M_1|_{\underline{\mathcal{S}}^{a'}\downarrow} = \cdots = M_m|_{\underline{\mathcal{S}}^{a'}\downarrow}$. Contradiction.

This means that $M_m \in Con_{i+1}$, thus, since $M|_{\underline{\mathcal{S}}^{a}\downarrow} = M'|_{\underline{\mathcal{S}}^{a}\downarrow} = M_1|_{\underline{\mathcal{S}}^{a}\downarrow} = \cdots = M_m|_{\underline{\mathcal{S}}^{a}\downarrow}$, we get $M|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con_{i+1}|_{\underline{\mathcal{S}}^{a}\downarrow}$, as required.

2. Assume $a \notin \mathbf{Compl}(\underline{\mathcal{S}}^{A_{i+1}}\downarrow)$. For $B = \mathbf{Symb}(\mathbf{Compl}(\underline{\mathcal{S}}^{a}\Downarrow)) \cup A_{i+1}$, the following square of inclusions is a pushout in $\mathbf{SigDep}^{frag}$.



From $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \in Con_{i+1}|_{\underline{\mathcal{S}}^{a}\Downarrow}$, there exists $M' \in Con_{i+1}$ such that $M|_{\underline{\mathcal{S}}^{a}\Downarrow} = M'|_{\underline{\mathcal{S}}^{a}\Downarrow}$. Let $M'' \in [\![\underline{\mathcal{S}}^{(B\cup\{a\})}\downarrow]\!]$ be the amalgamation of $M|_{\underline{\mathcal{S}}^{a}\downarrow}$ and $M'|_{\underline{\mathcal{S}}^{B}\downarrow}$ w.r.t. the above pushout diagram. By assumption (7) from Sect. 3.5 there exists a model $M''' \in [\![\underline{\mathcal{S}}]\!]$ such that $M'' = M'''|_{\underline{\mathcal{S}}^{(B\cup\{a\})}\downarrow}$. Since $\underline{SP}$ is dependency-wise, from $M'''|_{\underline{\mathcal{S}}^{a}\downarrow} \models SP|_{\underline{\mathcal{S}}^{a}\downarrow}$ and $M'''|_{\underline{\mathcal{S}}^{B}\downarrow} \models SP|_{\underline{\mathcal{S}}^{B}\downarrow}$ we get $M'''|_{\underline{\mathcal{S}}^{(B\cup\{a\})}\downarrow} \models SP|_{\underline{\mathcal{S}}^{(B\cup\{a\})}\downarrow}$. This means that there exists $M^X \models SP$ such that $M^X|_{\underline{\mathcal{S}}^{(B\cup\{a\})}\downarrow} = M'''|_{\underline{\mathcal{S}}^{(B\cup\{a\})}\downarrow}$. By induction on $j \in \{0, \ldots, i+1\}$ we prove that $M^X \in Con_j$. For $j = 0$, this is obvious. Let us assume $M^X \in Con_j$ for some $j \in \{0, \ldots, i\}$. It holds that $M^X \in Con_{j+1}$, because otherwise there would exist $M^Y \in Con_{j+1}$ and $a' \in A_{j+1}$ such that $M^Y|_{\underline{\mathcal{S}}^{a'}\Downarrow} = M^X|_{\underline{\mathcal{S}}^{a'}\Downarrow}$ and $M^Y|_{\underline{\mathcal{S}}^{a'}\downarrow} \neq M^X|_{\underline{\mathcal{S}}^{a'}\downarrow}$, which is impossible by definition of Con_{j+1}, because $A_{j+1} \subseteq A_{i+1}$ thus $a' \in A_{i+1}$ and $Con_{i+1} \subseteq Con_{j+1}$ thus $M^X|_{\underline{\mathcal{S}}^{a'}\downarrow} = M'''|_{\underline{\mathcal{S}}^{a'}\downarrow} = M''|_{\underline{\mathcal{S}}^{a'}\downarrow} = M'|_{\underline{\mathcal{S}}^{a'}\downarrow} \in Con_{i+1}|_{\underline{\mathcal{S}}^{a'}\downarrow} \subseteq Con_{j+1}|_{\underline{\mathcal{S}}^{a'}\downarrow}$. For $j = i$ this gives $M^X \in Con_{i+1}$ and, since $M|_{\underline{\mathcal{S}}^{a}\downarrow} = M''|_{\underline{\mathcal{S}}^{a}\downarrow} = M'''|_{\underline{\mathcal{S}}^{a}\downarrow} = M^X|_{\underline{\mathcal{S}}^{a}\downarrow}$, we get $M|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con_{i+1}|_{\underline{\mathcal{S}}^{a}\downarrow}$, as

required.

Regarding condition (3), let there be any $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ and a model $M \in [\![\underline{\mathcal{S}}]\!]$ such that for all $a \in A$, $M|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con_n|_{\underline{\mathcal{S}}^{a}\downarrow}$, then by construction of Con_n we have $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^{a}\downarrow}$. From the fact that $\underline{SP}$ is a dependency-wise construction specification we get $M|_{\underline{\mathcal{S}}^{A}\downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^{A}\downarrow}$, therefore, by construction of Con_n we have $M|_{\underline{\mathcal{S}}^{A}\downarrow} \in Con_n|_{\underline{\mathcal{S}}^{A}\downarrow}$, as required.

Condition (4) follows directly from the fact that $\underline{SP}$ is a dependency-wise construction specification.

□

Proof of Theorem 6.25. Let us prove the following lemma prior to showing the main fact.

Lemma 6.51 *Given a construction specification $\underline{SP} \in \mathbf{Spec}(\underline{\mathcal{S}})$ and a construction model $Con \models^{c} \underline{SP}$, for any $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ and $M \in [\![\underline{\mathcal{S}}]\!]$, if*

$$M|_{\underline{\mathcal{S}}^{A}\downarrow} \in Con|_{\underline{\mathcal{S}}^{A}\downarrow} \text{ and } M|_{\underline{\mathcal{S}}^{A}\downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^{A}\downarrow}$$

then

$$M|_{\underline{\mathcal{S}}^{A}\downarrow} \in \mathbf{Clean}_{\underline{SP}}(Con)|_{\underline{\mathcal{S}}^{A}\downarrow}$$

Proof. Let there be a consistent specification $\underline{SP} \in \mathbf{Spec}(\underline{\mathcal{S}})$, a construction model $Con \models^{c} \underline{SP}$ and a set $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$. Let us name $Con' = \mathbf{Clean}_{\underline{SP}}(Con)$.

By induction we define a chain of sets $A_0 \subseteq A_1 \subseteq \cdots \subseteq A_n$ as $A_0 = \mathbf{Compl}(\underline{\mathcal{S}}^{A}\downarrow)$ and $A_{i+1} = \{a \in \mathbf{Compl}(\underline{\mathcal{S}}) \mid \mathbf{Compl}(\underline{\mathcal{S}}^{a}\Downarrow) \subseteq A_i\}$. Let n be the smallest natural number such that $A_n = \mathbf{Compl}(\underline{\mathcal{S}})$. The dependency structure is a bounded strict order (cf. Sect. 5.3), so by definition n is finite and $n \leq db(\underline{\mathcal{S}})$.

Let us prove the internal lemma saying that for any $0 \leq i \leq n$ and any model $M \in [\![\underline{\mathcal{S}}]\!]$, if $M|_{\underline{\mathcal{S}}^{A_i}\downarrow} \in Con|_{\underline{\mathcal{S}}^{A_i}\downarrow}$ and $M|_{\underline{\mathcal{S}}^{A_i}\downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^{A_i}\downarrow}$ then $M|_{\underline{\mathcal{S}}^{A_i}\downarrow} \in Con'|_{\underline{\mathcal{S}}^{A_i}\downarrow}$. The proof is by induction on $i \in \langle n, \ldots, 0\rangle$.

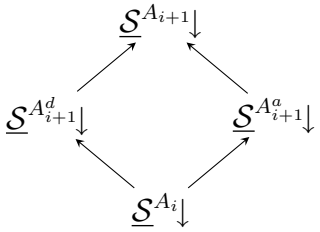
In the base case, for $i = n$, let us have $M \in [\![\underline{\mathcal{S}}]\!]$ such that $M|_{\underline{\mathcal{S}}^{A_i\downarrow}} \in Con|_{\underline{\mathcal{S}}^{A_i\downarrow}}$ and $M|_{\underline{\mathcal{S}}^{A_i\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A_i\downarrow}}$. We have $A_i = A_n = \mathbf{Compl}(\underline{\mathcal{S}})$, so $M \in Con$ and $M \models \underline{SP}$ thus, by definition, $M \in Con'$, i.e. $M|_{\underline{\mathcal{S}}^{A_i\downarrow}} \in Con'|_{\underline{\mathcal{S}}^{A_i\downarrow}}$.

As for the induction step, let us assume that the internal lemma holds for $i+1$, where $0 \leq i < n$, i.e. for any model $M \in [\![\underline{\mathcal{S}}]\!]$, if $M|_{\underline{\mathcal{S}}^{A_{i+1}\downarrow}} \in Con|_{\underline{\mathcal{S}}^{A_{i+1}\downarrow}}$ and $M|_{\underline{\mathcal{S}}^{A_{i+1}\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A_{i+1}\downarrow}}$, then $M|_{\underline{\mathcal{S}}^{A_i+1\downarrow}} \in Con'|_{\underline{\mathcal{S}}^{A_{i+1}\downarrow}}$. In what follows, we prove the same for i.

Consider a model $M \in [\![\underline{\mathcal{S}}]\!]$ such that $M|_{\underline{\mathcal{S}}^{A_i\downarrow}} \in Con|_{\underline{\mathcal{S}}^{A_i\downarrow}}$ and $M|_{\underline{\mathcal{S}}^{A_i\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A_i\downarrow}}$. By definition of the reduct, there exist $M' \models \underline{SP}$ and $M'' \in Con$ such that $M|_{\underline{\mathcal{S}}^{A_i\downarrow}} = M'|_{\underline{\mathcal{S}}^{A_i\downarrow}} = M''|_{\underline{\mathcal{S}}^{A_i\downarrow}}$. Let us define two sets

- $A^a_{i+1} = A_i \cup \{a \in A_{i+1} \mid a \notin \underline{\mathcal{S}}\}$
- $A^d_{i+1} = A_i \cup \{a \in A_{i+1} \mid a \in \underline{\mathcal{S}}\}$

The following square of inclusions is a pushout in $\mathbf{SigDep}^{frag}$.



Let $M_s \in [\![\underline{\mathcal{S}}^{A_{i+1}\downarrow}]\!]$ be the amalgamation of $M''|_{\underline{\mathcal{S}}^{A^d_{i+1}\downarrow}}$ and $M'|_{\underline{\mathcal{S}}^{A^a_{i+1}\downarrow}}$ w.r.t. the above pushout diagram. Let $M_l \in [\![\underline{\mathcal{S}}]\!]$ be any model such that $M_l|_{\underline{\mathcal{S}}^{A_{i+1}\downarrow}} = M_s$ (by assumption (7) from Sect. 3.5)

We show that for any $a \in A_{i+1}$ it holds that $M_l|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\downarrow}$. Let $a \in A_{i+1}$,

- if $a \in A_i$, then by assumption, $M_l|_{\underline{\mathcal{S}}^{a}\downarrow} = M|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\downarrow}$,
- if $a \notin A_i$ and $a \in A^d_{i+1}$, i.e. a is defined, $M_l|_{\underline{\mathcal{S}}^{a}\downarrow} = M''|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\downarrow}$,
- if $a \notin A_i$ and $a \in A^a_{i+1}$, i.e. a is assumed, we have $M_l|_{\underline{\mathcal{S}}^{a}\Downarrow} = M'|_{\underline{\mathcal{S}}^{a}\Downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\Downarrow}$, thus, since $M' \models \underline{SP}$, by condition (2) of Def. 6.13 for $Con \models^c \underline{SP}$, $M'|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\downarrow}$, and since $M'|_{\underline{\mathcal{S}}^{a}\downarrow} = M_l|_{\underline{\mathcal{S}}^{a}\downarrow}$, we get $M_l|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\downarrow}$.

By condition (3) of Def. 6.13, we obtain $M_l|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}} \in Con|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}}$. That is, for some $M_x \in Con$, $M_x|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}} = M_l|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}}$.

Now we show that for any $a \in A_{i+1}$ it holds that $M_x|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$. Let $a \in A_{i+1}$,

- if $a \in A_i$, then by assumption, $M_x|_{\underline{\mathcal{S}}^{a\downarrow}} = M_l|_{\underline{\mathcal{S}}^{a\downarrow}} = M|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$;
- if $a \notin A_i$ and $a \in A^a_{i+1}$, i.e. a is assumed, $M_x|_{\underline{\mathcal{S}}^{a\downarrow}} = M_l|_{\underline{\mathcal{S}}^{a\downarrow}} = M'|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$;
- if $a \notin A_i$ and $a \in A^d_{i+1}$, i.e. a is defined, we have $M_x|_{\underline{\mathcal{S}}^{a\downarrow}} = M_l|_{\underline{\mathcal{S}}^{a\Downarrow}} = M''|_{\underline{\mathcal{S}}^{a\Downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\Downarrow}}$; thus, since $M'' \in Con$, by condition (1) of Def. 6.13 for $Con \models^c \underline{SP}$, $M'|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$, and since $M''|_{\underline{\mathcal{S}}^{a\downarrow}} = M_l|_{\underline{\mathcal{S}}^{a\downarrow}} = M_x|_{\underline{\mathcal{S}}^{a\downarrow}}$, we get $M_x|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$.

Consequently, by condition (4) of Def. 6.13, we get $M_l|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}} = M_x|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}}$.

By inductive assumption from $M_l|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}} \in Con|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}}$ and $M_l|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}}$ we conclude that $M_l|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}} \in Con'|_{\underline{\mathcal{S}}^{A_{i+1\downarrow}}}$, therefore, $M|_{\underline{\mathcal{S}}^{A_{i\downarrow}}} = M_s|_{\underline{\mathcal{S}}^{A_{i\downarrow}}} = M_l|_{\underline{\mathcal{S}}^{A_{i\downarrow}}} \in Con'|_{\underline{\mathcal{S}}^{A_{i\downarrow}}}$. This completes the proof of the internal lemma.

Finally, to prove Lemma 6.51, let there be a model $M \in [\![\underline{\mathcal{S}}]\!]$ such that $M|_{\underline{\mathcal{S}}^{A_\downarrow}} \in Con|_{\underline{\mathcal{S}}^{A_\downarrow}}$ and $M|_{\underline{\mathcal{S}}^{A_\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A_\downarrow}}$. By definition $A = A_0$ and by the proven internal lemma, $M|_{\underline{\mathcal{S}}^{A_\downarrow}} \in Con'|_{\underline{\mathcal{S}}^{A_\downarrow}}$.

□

To prove Theorem 6.25, let there be a construction specification $\underline{SP}$ and a construction model Con such that $Con \models^c \underline{SP}$. This means that $\underline{SP}$ is a consistent construction specification, thus, by Lemma 6.19, the specification $\pi_2(\underline{SP})$ is consistent in the base institution $\mathbb{I}$. Let $Con' = \mathbf{Clean}_{\underline{SP}}(Con)$. Clearly, since Con is a construction model, Con' is a construction model as well. We prove that $Con' \models^c \underline{SP}$ by showing all conditions from Def. 6.13.

To show condition (1), let there be $a \in \underline{\mathcal{S}}$ and a model $M \in Con'$ such that $M|_{\underline{\mathcal{S}}^{a\Downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\Downarrow}}$. By definition, $M \models \underline{SP}$, thus $M|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$.

For condition (2), let us have $a \in \mathbf{Compl}(\underline{\mathcal{S}})$ such that $a \notin \underline{\mathcal{S}}$ and let there be a model $M \models \underline{SP}$ such that $M|_{\underline{\mathcal{S}}^{a\Downarrow}} \in Con'|_{\underline{\mathcal{S}}^{a\Downarrow}}$. From $Con \models^c \underline{SP}$,

since $Con' \subseteq Con$, by condition (2) of Def. 6.13 for $Con \models^c \underline{SP}$, we have $M|_{\underline{\mathcal{S}}^{a\downarrow}} \in Con|_{\underline{\mathcal{S}}^{a\downarrow}}$. By Lemma 6.51, for $A = \{a\}$, from $M \models \underline{SP}$ and $M|_{\underline{\mathcal{S}}^{a\downarrow}} \in Con|_{\underline{\mathcal{S}}^{a\downarrow}}$, we obtain $M|_{\underline{\mathcal{S}}^{a\downarrow}} \in Con'|_{\underline{\mathcal{S}}^{a\downarrow}}$, as required.

Regarding condition (3), let there be a set $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ and a model $M \in [\![\underline{\mathcal{S}}]\!]$ such that for all $a \in A$, $M|_{\underline{\mathcal{S}}^{a\downarrow}} \in Con'|_{\underline{\mathcal{S}}^{a\downarrow}}$. Again, since $Con' \subseteq Con$, by condition (3) of Def. 6.13 for $Con \models^c \underline{SP}$, we have $M|_{\underline{\mathcal{S}}^{A\downarrow}} \in Con|_{\underline{\mathcal{S}}^{A\downarrow}}$. This means that there exists $M' \in Con$ such that $M'|_{\underline{\mathcal{S}}^{A\downarrow}} = M|_{\underline{\mathcal{S}}^{A\downarrow}}$ and for all $a \in A$, $M'|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$. By condition (4) of Def. 6.13 for $Con \models^c \underline{SP}$, we have $M'|_{\underline{\mathcal{S}}^{A\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A\downarrow}}$, therefore, $M|_{\underline{\mathcal{S}}^{A\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A\downarrow}}$. By Lemma 6.51, from $M|_{\underline{\mathcal{S}}^{A\downarrow}} \in Con|_{\underline{\mathcal{S}}^{A\downarrow}}$ and $M|_{\underline{\mathcal{S}}^{A\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A\downarrow}}$ we get $M|_{\underline{\mathcal{S}}^{A\downarrow}} \in Con'|_{\underline{\mathcal{S}}^{A\downarrow}}$, as required.

Finally, to prove condition (4) let there be a set $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ and a model $M \in Con'$ such that for all $a \in A$, $M|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{a\downarrow}}$. Since $M \in Con$, by condition (4) of Def. 6.13 for $Con \models^c \underline{SP}$, we get $M|_{\underline{\mathcal{S}}^{A\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}^{A\downarrow}}$, as required.

□

Lemma 6.52 *Cleaning operator is idempotent, i.e.*

$$\mathbf{Clean}_{\underline{SP}}(\mathbf{Clean}_{\underline{SP}}(Con)) = \mathbf{Clean}_{\underline{SP}}(Con)$$

Proof. Directly from definition. □

Proof of Theorem 6.31. Let the pushout of ft be as on the diagram of Def. 6.28, and let us name $Con = Con_1 \oplus_{ft} Con_2$. To check that Con fulfills the requirement in Def. 6.5, let $a \in \underline{\mathcal{S}}$ be a defined symbol, and let $M, M' \in Con$ be two models such that $M|_{\underline{\mathcal{S}}^{a\Downarrow}} = M'|_{\underline{\mathcal{S}}^{a\Downarrow}}$.

Without loss of generality we can assume that there exists $a_1 \in \underline{\mathcal{S}}_1$ such that $\underline{\beta_1}(a_1) = a$. Let us name $M_1 = M|_{\underline{\beta_1}}$ and $M'_1 = M'|_{\underline{\beta_1}}$. By definition $M_1, M'_1 \in Con_1$. From $M|_{\underline{\mathcal{S}}^{a\Downarrow}} = M'|_{\underline{\mathcal{S}}^{a\Downarrow}}$ we obtain $(M|_{\underline{\mathcal{S}}^{a\Downarrow}})|_{(\beta_1)^-_{a1}} = (M|_{\underline{\mathcal{S}}^{a\Downarrow}})|_{(\beta_1)^-_{a1}}$ (cf. Def. 6.49). Hence, we get $(M|_{\underline{\beta_1}})|_{\underline{\mathcal{S}}_1^{a_1\Downarrow}} = (M'|_{\underline{\beta_1}})|_{\underline{\mathcal{S}}_1^{a_1\Downarrow}}$, thus $M_1|_{\underline{\mathcal{S}}_1^{a_1\Downarrow}} = M'_1|_{\underline{\mathcal{S}}_1^{a_1\Downarrow}}$. Since Con_1 is a construction model, by Def. 6.5, $M_1|_{\underline{\mathcal{S}}_1^{a_1\downarrow}} = M'_1|_{\underline{\mathcal{S}}_1^{a_1\downarrow}}$.

By Lemma 6.50, the morphism $\underline{(\beta_1)^-_{a_1}} : \underline{\mathcal{S}}_1^{a_1}\Downarrow \to \underline{\mathcal{S}}^{a}\Downarrow$ is surjective, therefore, by assumption from the beginning of Sect. 3.5, the reduct functor $-|_{\underline{(\beta_1)^-_{a_1}}} : [\![\underline{\mathcal{S}}^{a}\downarrow]\!] \to [\![\underline{\mathcal{S}}_1^{a_1}\downarrow]\!]$ is injective on models, thus we get $M|_{\underline{\mathcal{S}}^{a}\downarrow} = M'|_{\underline{\mathcal{S}}^{a}\downarrow}$, as required.

□

Proof of Lemma 6.35. Let there be two dependency-wise construction specifications $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1 \rangle$ and $\underline{SP}_2 = \langle \underline{\mathcal{S}}_2, SP_2 \rangle$ compatible w.r.t. a fitting $ft = \langle \underline{\varphi_1}, \underline{\varphi_2} \rangle$ such that $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$ are finite. We use the notation from Def. 6.28 (see the diagram there).

By Theorem 6.22, dependency-wise construction specifications over finite construction signatures are consistent. By Lemma 6.19, they are also consistent in the base institution $\mathbb{I}$.

The compatibility definition (cf. Def. 6.34) provides us with certain properties w.r.t. a given set of independent symbols $A \subseteq \mathbf{Compl}(\underline{\mathcal{F}})$. Using those properties, we show by induction on the height of dependency structure of symbols from A that there exists a model $M \in [\![\underline{\mathcal{F}}]\!]$ such that $M|_{\underline{\mathcal{F}}^{A}\downarrow} \models (SP_1|_{\varphi_1})|_{\underline{\mathcal{F}}^{A}\downarrow}$ and $M|_{\underline{\mathcal{F}}^{A}\downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A}\downarrow}$.

Let us define a sequence of sets of independent symbols from $\mathbf{Compl}(\underline{\mathcal{F}})$, $\langle A_i^{=} \rangle_{0 \leq i \leq db(\underline{\mathcal{F}})}$ given as

$$A_i^{=} = \{a \in \mathbf{Compl}(\underline{\mathcal{F}}) \mid db(\underline{\mathcal{F}}^{a}\downarrow) = i\}$$

Then, let there be another sequence of sets of independent symbols from $\mathbf{Compl}(\underline{\mathcal{F}})$, $\langle A_i \rangle_{0 \leq i \leq db(\underline{\mathcal{F}})}$ given by induction as $A_0 = \emptyset$ and for $i \in \{1, \ldots, db(\underline{\mathcal{F}})\}$

$$A_i = \{a \in \mathbf{Compl}(\underline{\mathcal{F}}) \mid a \in A_i^{=} \text{ or } a \in A_{i-1} \text{ and there is no } a' \in A_i^{=} \text{ such that } a < a'\}.$$

The sequence $\langle A_i \rangle_{0 \leq i \leq db(\underline{\mathcal{F}})}$ is such that in A_i there is at least one symbol with the dependency structure of height i; A_i is the set of maximal elements in $\bigcup_{0 \leq k \leq i} A_k^{=}$. We have $\underline{\mathcal{F}}^{A_{db(\mathcal{F})}}\downarrow = \underline{\mathcal{F}}$ and also for any $i \in \{1, \ldots, db(\underline{\mathcal{F}})\}$, $\underline{\mathcal{F}}^{A_i}\Downarrow \subseteq \underline{\mathcal{F}}^{A_{i-1}}\downarrow$.

Let us also define two sequences of sets of independent symbols from $\mathbf{Compl}(\underline{\mathcal{F}})$ corresponding to $\langle A_i \rangle$ that are mapped via φ_1 and φ_2 to assumed symbols in $\mathbf{Compl}(\underline{\mathcal{S}}_1)$ and $\mathbf{Compl}(\underline{\mathcal{S}}_2)$, respectively, $\langle A_i^k \rangle_{0 \leq i \leq db(\underline{\mathcal{F}})}$, given as

$$A_i^k = \{a \in A_i \mid \varphi_k(a) \notin \underline{\mathcal{S}}_k\}$$

for $k \in \{1, 2\}$. Sets A_i^1 and A_i^2 need not be disjoint. Of course for any $i \in \{0, \ldots, db(\underline{\mathcal{F}})\}$, $A_i^k \subseteq A_i$ for $k \in \{1, 2\}$ and $A_i = A_i^1 \cup A_i^2$.

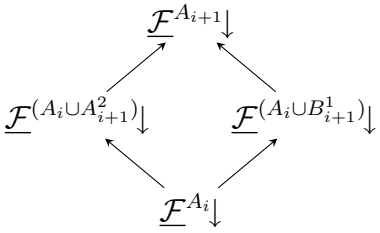
In the base case, we have $A_0 = \emptyset$, we take a model $M_1 \models SP_1$ (it exists, because SP_1 is consistent) and of course it holds that $(M_1|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{\emptyset}\downarrow} \models (SP_2|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{\emptyset}\downarrow}$, because $\underline{\mathcal{F}}^{\emptyset}\downarrow$ is the initial signature and SP_2 is consistent. This makes $M_0 = M_1|_{\underline{\varphi_1}}$ a model of $\mathbf{Compl}(\underline{\mathcal{F}})$ such that $M_0|_{\underline{\mathcal{F}}^{A_0}\downarrow} \models (SP_1|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_0}\downarrow}$ and $M_0|_{\underline{\mathcal{F}}^{A_0}\downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_0}\downarrow}$.

In the induction step, for a natural number i such that $0 \leq i < db(\underline{\mathcal{F}})$ we assume that there exist a model $M^i \in [\![\underline{\mathcal{F}}]\!]$ such that $M^i|_{\underline{\mathcal{F}}^{A_i}\downarrow} \models (SP_1|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_i}\downarrow}$ and $M_i|_{\underline{\mathcal{F}}^{A_i}\downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_i}\downarrow}$. We prove the existence of a model $M^{i+1} \in [\![\underline{\mathcal{F}}]\!]$ such that $M^{i+1}|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow} \models (SP_1|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow}$ and $M_{i+1}|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow}$.

Let $M_1' \models SP_1$ be such that $(M_1'|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_i}\downarrow} = M^i|_{\underline{\mathcal{F}}^{A_i}\downarrow}$. By the inductive assumption we have $(M_1'|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_i}\downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_i}\downarrow}$ thus, by $\underline{\mathcal{F}}^{A_{i+1}}\Downarrow \subseteq \underline{\mathcal{F}}^{A_i}\downarrow$, we get $(M_1'|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_{i+1}}\Downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_{i+1}}\Downarrow}$. Since $A_{i+1}^2 \subseteq A_{i+1}$, we have $(M_1'|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_{i+1}^2}\Downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_{i+1}^2}\Downarrow}$. By compatibility, condition (2) from Def. 6.34, we get $(M_1'|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_{i+1}^2}\downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_{i+1}^2}\downarrow}$.

Let $M_2' \models SP_2$ be such that $(M_2'|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_i}\downarrow} = M^i|_{\underline{\mathcal{F}}^{A_i}\downarrow}$. Using the same reasoning as above, but for exchanged indexes (1 for 2 and vice versa), we get $(M_2'|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_{i+1}^1}\downarrow} \models (SP_1|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_{i+1}^1}\downarrow}$.

Let $B_{i+1}^1 = \{a \in A_{i+1}^1 \mid a \notin A_{i+1}^2\}$. The following diagram is a pushout in $\mathbf{SigDep}^{frag}$



Let $M' \in [\![\underline{\mathcal{F}}^{A_{i+1}\downarrow}]\!]$ be the amalgamation of $M'_1|_{\underline{\mathcal{F}}^{(A_i \cup A^2_{i+1})}\downarrow}$ and $M'_2|_{(A_i \cup B^1_{i+1})}$. Then, by the assumptions from Sect. 3.5, there exists a model $M^{i+1} \in [\![\underline{\mathcal{F}}]\!]$ such that $M^{i+1}|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow} = M'$. We use the fact that $\underline{SP}_1$ and $\underline{SP}_2$ are dependency-wise construction specifications to show that $M^{i+1}|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow} \models (SP_1|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow}$ and $M^{i+1}|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow}$. Since both conditions are symmetrical, we show only the first one. By Def. 6.20, it is enough to show that for any $a \in A_{i+1}$, $M^{i+1}|_{\underline{\mathcal{F}}^{a}\downarrow} \models (SP_1|_{\varphi_1})|_{\underline{\mathcal{F}}^{a}\downarrow}$. So consider $a \in A_{i+1}$. If $a \in A_i$ then by inductive assumption we have $M^{i+1}|_{\underline{\mathcal{F}}^{a}\downarrow} \models SP_1|_{\underline{\mathcal{F}}^{a}\downarrow}$; if $a \in A^2_{i+1}$ then, since $M'_1 \models SP_1$, we get $M^{i+1}|_{\underline{\mathcal{F}}^{a}\downarrow} \models SP_1|_{\underline{\mathcal{F}}^{a}\downarrow}$; if $a \in B^1_{i+1}$ then, since $(M'_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A^1_{i+1}}\downarrow} \models (SP_1|_{\varphi_1})|_{\underline{\mathcal{F}}^{A^1_{i+1}}\downarrow}$, we have $M^{i+1}|_{\underline{\mathcal{F}}^{a}\downarrow} \models SP_1|_{\underline{\mathcal{F}}^{a}\downarrow}$. Therefore, $M^{i+1}|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow} \models (SP_1|_{\varphi_1})|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow}$ and, by symmetry, $M^{i+1}|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow} \models (SP_2|_{\varphi_2})|_{\underline{\mathcal{F}}^{A_{i+1}}\downarrow}$.

From the above-given proof by induction, since $\underline{\mathcal{F}}^{A_{db(\mathcal{F})}}\downarrow = \underline{\mathcal{F}}$, we get $M^{db(\mathcal{F})} \models SP_1|_{\varphi_1}$ and $M^{db(\mathcal{F})} \models SP_2|_{\varphi_2}$. This means that there exist $M_1 \models SP_1$ and $M_2 \models SP_2$ such that $M_1|_{\varphi_1} = M^{db(\mathcal{F})}$ and $M_2|_{\varphi_2} = M^{db(\mathcal{F})}$. By amalgamation of M_1 and M_2 w.r.t. the diagram from Def. 6.28 we get a model $M \in [\![\underline{\mathcal{S}}]\!]$, and moreover, $M \models (\underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}_2))$. This proves that $\underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}_2)$ is a consistent specification in the base institution $\mathbb{I}$. □

Proof of Theorem 6.36. We show two lemmas before we prove the theorem. The lemma below shows that the sum of clean models yields a clean model.

Lemma 6.53 *Given two construction specifications $\underline{SP}_1$ and $\underline{SP}_2$ connected by a construction fitting ft and two clean construction models $Con_1 \models^c \underline{SP}_1$ and $Con_2 \models^c \underline{SP}_2$, it holds that*

$$\mathbf{Clean}_{\underline{SP}}(Con) = Con$$

where $Con = Con_1 \oplus_{ft} Con_2$ and $\underline{SP} = \underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}_2)$.

Proof. Let there be $M \in Con$. By Def. 6.30, $M|_{\underline{\beta_i}} \in Con_i$, thus for $i \in \{1, 2\}$, $M|_{\underline{\beta_i}} \models \underline{SP}_i$, so $M \models \underline{SP}$. □

Lemma 6.54 *Consider two compatible (cf. Def. 6.34) construction specifications* $\underline{SP}_1$ *and* $\underline{SP}_2$ *connected by a construction fitting ft and two clean construction models* $Con_1 \models^c \underline{SP}_1$ *and* $Con_2 \models^c \underline{SP}_2$. *Given a set* $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ *let us name* $A_1 = \underline{\beta_1}^{-1}(\mathbf{Compl}(\underline{\mathcal{S}}^{A}\downarrow))$ *and* $A_2 = \underline{\beta_2}^{-1}(\mathbf{Compl}(\underline{\mathcal{S}}^{A}\downarrow))$. *For* $M \in [\![\underline{\mathcal{S}}]\!]$, *if*

$$(M|_{\underline{\beta_1}})|_{\underline{\mathcal{S}}_1^{A_1}\downarrow} \in Con_1|_{\underline{\mathcal{S}}_1^{A_1}\downarrow} \text{ and } (M|_{\underline{\beta_2}})|_{\underline{\mathcal{S}}_2^{A_2}\downarrow} \in Con_2|_{\underline{\mathcal{S}}_2^{A_2}\downarrow}$$

then $M|_{\underline{\mathcal{S}}^{A}\downarrow} \in (Con_1 \oplus Con_2)|_{\underline{\mathcal{S}}^{A}\downarrow}$.

Proof. Let there be *ft*, $\underline{SP}_1$, $\underline{SP}_2$, Con_1, Con_2, A, A_1 and A_2, as described in the statement of the lemma. Consider a model $M \in [\![\underline{\mathcal{S}}]\!]$, such that $(M|_{\underline{\beta_1}})|_{\underline{\mathcal{S}}_1^{A_1}\downarrow} \in Con_1|_{\underline{\mathcal{S}}_1^{A_1}\downarrow}$ and $(M|_{\underline{\beta_2}})|_{\underline{\mathcal{S}}_2^{A_2}\downarrow} \in Con_2|_{\underline{\mathcal{S}}_2^{A_2}\downarrow}$. We have then $M_i \in Con_i$ such that $(M|_{\underline{\beta_i}})|_{\underline{\mathcal{S}}_i^{A_i}\downarrow} = M_i|_{\underline{\mathcal{S}}_i^{A_i}\downarrow}$, for $i \in \{1,2\}$. Since $\mathbf{Clean}_{\underline{SP}_i}(Con_i) = Con_i$, $M_i \models \underline{SP}_i$ for $i \in \{1,2\}$.

To show that $M|_{\underline{\mathcal{S}}^{A}\downarrow} \in (Con_1 \oplus Con_2)|_{\underline{\mathcal{S}}^{A}\downarrow}$ we first search for $M_1' \in Con_1$ and $M_2' \in Con_2$ such that $M_1'|_{\underline{\varphi_1}} = M_2'|_{\underline{\varphi_2}}$ and for $i \in \{1,2\}$, $M_i'|_{\underline{\mathcal{S}}_i^{A_i}\downarrow} = M_i|_{\underline{\mathcal{S}}_i^{A_i}\downarrow}$.

Let $A_F = \underline{\varphi_1}^{-1}(\mathbf{Compl}(\underline{\mathcal{S}}_1^{A_1}\downarrow)) = \underline{\varphi_2}^{-1}(\mathbf{Compl}(\underline{\mathcal{S}}_2^{A_2}\downarrow))$. By induction we define a chain of sets $A_F^0 \subseteq A_F^1 \subseteq \cdots \subseteq A_F^n$ as $A_F^0 = \mathbf{Compl}(\underline{\mathcal{F}}^{A_F}\downarrow)$ and $A_F^{i+1} = \{a \in \mathbf{Compl}(\underline{\mathcal{F}}) \mid \mathbf{Compl}(\underline{\mathcal{F}}^{a}\Downarrow) \subseteq A_F^i\}$. Dependency structures are bounded strict orders (cf. Sect. 5.3), therefore, there exists the least natural number n such that $A_F^n = \mathbf{Compl}(\underline{\mathcal{F}})$. For every $i \in \{0,\ldots,n\}$ let us name $A_1^i = \underline{\varphi_1}(A_F^i)$ and $A_2^i = \underline{\varphi_2}(A_F^i)$.

By induction on $i \in \langle 0,\ldots,n\rangle$ we define two series of models $M_1^i \in Con_1$ and $M_2^i \in Con_2$ such that

$$M_1^i|_{\underline{\mathcal{S}}_1^{A_1}\downarrow} = M_1|_{\underline{\mathcal{S}}_1^{A_1}\downarrow},\ M_2^i|_{\underline{\mathcal{S}}_2^{A_2}\downarrow} = M_2|_{\underline{\mathcal{S}}_2^{A_2}\downarrow}$$

and

$$(M_1^i|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{A_F^i}\downarrow} = (M_2^i|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{A_F^i}\downarrow}$$

In the base case, for $i = 0$, let $M_1^0 = M_1$ and$M_2^0 = M_2$. It is easy to check that the above conditions are met.

In the induction step case, let us assume that there are M_1^i and M_2^i satisfying the above conditions. In what follows we construct M_1^{i+1} and M_2^{i+1} and prove that they have the required properties.

Let us define two sets $X = \{a \in A_F^{i+1} \mid a \notin A_F^i \text{ and } \underline{\varphi_1}(a) \notin \underline{\mathcal{S}}_1\}$ and $Y = \{a \in A_F^{i+1} \mid a \notin A_F^i \text{ and } a \notin X\}$. The set X contains $\underline{\mathcal{F}}$-symbols such that they belong to A_F^{i+1}, they don't belong to A_F^i and their images w.r.t. $\underline{\varphi_1}$ in $\underline{\mathcal{S}}_1$ are assumed symbols. For any $a, a' \in X$, $db(\underline{\mathcal{F}}^a\downarrow) = db(\underline{\mathcal{F}}^{a'}\downarrow)$, therefore, X is a set of independent symbols in $\underline{\mathcal{F}}$. The set Y contains all the $\underline{\mathcal{F}}$-symbols such that they belong to A_F^{i+1} and they belong neither to A_F^i nor to X. It is easy to show that their images w.r.t. $\underline{\varphi_2}$ in $\underline{\mathcal{S}}_2$ are assumed symbols and that Y is a set of independent symbols in $\underline{\mathcal{F}}$. Let us name $X_1 = \underline{\varphi_1}(X)$ and $Y_2 = \underline{\varphi_2}(Y)$. Since construction signature morphisms do not change the dependency structure of symbols (thus they have the same height), it also holds that X_1 and Y_2 are sets of independent assumed symbols in $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$, respectively.

From compatibility (cf. Def. 6.34), since $M_2^i \models \underline{SP}_2$, $(M_2^i|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{X}\Downarrow} \models (\underline{SP}_1|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{X}\Downarrow}$ and X is a set of independent symbols of $\underline{\mathcal{F}}$ that are assumed in $\underline{\mathcal{S}}_1$, we get $(M_2^i|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{X}\downarrow} \models (\underline{SP}_1|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{X}\downarrow}$, i.e. there exists $M_1^X \models \underline{SP}_1$ such that

$$(M_1^X|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{X}\downarrow} = (M_2^i|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{X}\downarrow}$$

Since both X and X_1 are sets of independent symbols in $\underline{\mathcal{F}}$ and $\underline{\mathcal{S}}_1$, respectively, the morphism $\underline{(\varphi_1)}_X^-\colon \underline{\mathcal{F}}^X\Downarrow \to \underline{\mathcal{S}}_1^{X_1}\Downarrow$ is a surjection (cf. Lemma 6.50). By assumption from the beginning of Sect. 3.5, the reduct functor $\_|_{\underline{(\varphi_1)}_X^-}$ is injective on models. Consequently, from

$$(M_1^i|_{\varphi_1})|_{\underline{\mathcal{F}}^{X}\Downarrow} = (M_2^i|_{\varphi_2})|_{\underline{\mathcal{F}}^{X}\Downarrow} = (M_1^X|_{\varphi_1})|_{\underline{\mathcal{F}}^{X}\Downarrow}$$

by commutativity of the diagram from Def. 6.49 and injectivity of $\_|_{\underline{(\varphi_1)}_X^-}$, we get

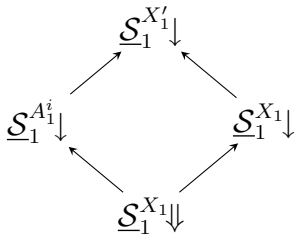
$$M_1^i|_{\underline{\mathcal{S}}_1^{X_1}\Downarrow} = M_1^X|_{\underline{\mathcal{S}}_1^{X_1}\Downarrow}$$

This means that $M_1^X|_{\underline{\mathcal{S}}_1^{X_1}\Downarrow} \in Con_1|_{\underline{\mathcal{S}}_1^{X_1}\Downarrow}$, thus for every $b \in X_1$ we have $M_1^X|_{\underline{\mathcal{S}}_1^{b}\Downarrow} \in Con_1|_{\underline{\mathcal{S}}_1^{b}\Downarrow}$, therefore, by (2) of Def. 6.13, $M_1^X|_{\underline{\mathcal{S}}_1^{b}\downarrow} \in Con_1|_{\underline{\mathcal{S}}_1^{b}\downarrow}$. Con-

sequently, by (3) of Def. 6.13,

$$M_1^X|_{\underline{\mathcal{S}}_1^{X_1\downarrow}} \in Con_1|_{\underline{\mathcal{S}}_1^{X_1\downarrow}}$$

Let $X_1' = X_1 \cup A_1^i$. The following square of inclusions is a pushout in $\mathbf{SigDep}^{frag}$.



Let $M_1^{X'} \in [\![\underline{\mathcal{S}}_1^{X'}\downarrow]\!]$ be the result of amalgamation of $M_1^i|_{\underline{\mathcal{S}}_1^{A_1^i\downarrow}}$ and $M_1^X|_{\underline{\mathcal{S}}_1^{X_1\downarrow}}$ w.r.t. the pushout given above. Let $M_1^s \in [\![\underline{\mathcal{S}}_1]\!]$ be such that $M_1^s|_{\underline{\mathcal{S}}_1^{X_1'\downarrow}} = M_1^{X'}$ (cf. the assumption about the base institution $\mathbb{I}$ at the beginning of Sect. 3.5). From (3) of Def. 6.13, since for all $a \in X_1'$, $M_1^s|_{\underline{\mathcal{S}}_1^{a\downarrow}} \in Con_1|_{\underline{\mathcal{S}}_1^{a\downarrow}}$ we get $M_1^s|_{\underline{\mathcal{S}}_1^{X_1'\downarrow}} \in Con_1|_{\underline{\mathcal{S}}_1^{X_1'\downarrow}}$. As a consequence, there exists $M_1^{i+1} \in Con_1$ such that $M_1^s|_{\underline{\mathcal{S}}_1^{X_1'\downarrow}} = M_1^{i+1}|_{\underline{\mathcal{S}}_1^{X_1'\downarrow}}$. Obviously $M_1^{i+1}|_{\underline{\mathcal{S}}_1^{A_1\downarrow}} = M_1|_{\underline{\mathcal{S}}_1^{A_1\downarrow}}$. Let $X' = X \cup A_F^i$. By construction we have

$$(M_1^{i+1}|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{X'\downarrow}} = (M_2|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{X'\downarrow}}$$

Now, in order to construct M_2^{i+1} we repeat the similar reasoning as above, but for Y instead of X:

From compatibility (cf. Def. 6.34), since $M_1^{i+1} \models \underline{SP}_1$, $(M_1^{i+1}|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{Y\Downarrow}} \models (\underline{SP}_2|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{Y\Downarrow}}$ and Y is a set of independent symbols of $\underline{\mathcal{F}}$ that are assumed in $\underline{\mathcal{S}}_2$, we get $(M_1^{i+1}|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{Y\downarrow}} \models (\underline{SP}_2|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{Y\downarrow}}$, i.e. there exists $M_2^Y \models \underline{SP}_2$ such that

$$(M_2^Y|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{Y\downarrow}} = (M_1^{i+1}|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{Y\downarrow}}$$

Since both Y and Y_2 are sets of independent symbols in $\underline{\mathcal{F}}$ and $\underline{\mathcal{S}}_2$, respectively, the morphism $\underline{(\varphi_2)_Y^-}\colon \underline{\mathcal{F}}^{Y\Downarrow} \to \underline{\mathcal{S}}_2^{Y_2\Downarrow}$ is a surjection (cf. Lemma 6.50). By assumption from the beginning of Sect. 3.5, the reduct functor $\_|_{\underline{(\varphi_2)_Y^-}}$ is

injective on models. Consequently, from

$$(M_2^i|_{\varphi_2})|_{\underline{\mathcal{F}}^{Y}\Downarrow} = (M_1^{i+1}|_{\varphi_1})|_{\underline{\mathcal{F}}^{Y}\Downarrow} = (M_2^Y|_{\varphi_2})|_{\underline{\mathcal{F}}^{Y}\Downarrow}$$

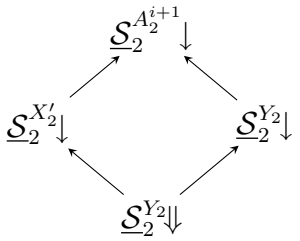
by commutativity of the diagram from Def. 6.49 and injectivity of $\_|_{\underline{(\varphi_2)^-_Y}}$, we get

$$M_2^i|_{\underline{\mathcal{S}}_2^{Y_2}\Downarrow} = M_2^Y|_{\underline{\mathcal{S}}_2^{Y_2}\Downarrow}$$

This means that $M_2^Y|_{\underline{\mathcal{S}}_1^{Y_2}\Downarrow} \in Con_2|_{\underline{\mathcal{S}}_2^{Y_2}\Downarrow}$, thus for every $b \in Y_2$ we have $M_2^Y|_{\underline{\mathcal{S}}_2^{b}\Downarrow} \in Con_2|_{\underline{\mathcal{S}}_2^{b}\Downarrow}$, therefore, by (2) of Def. 6.13, $M_2^Y|_{\underline{\mathcal{S}}_2^{b}\downarrow} \in Con_2|_{\underline{\mathcal{S}}_2^{b}\downarrow}$. Consequently, by (3) of Def. 6.13,

$$M_2^Y|_{\underline{\mathcal{S}}_2^{Y_2}\downarrow} \in Con_2|_{\underline{\mathcal{S}}_2^{Y_2}\downarrow}$$

Let $X_2' = \varphi_2(X) \cup A_2^i$. The following square of inclusions is a pushout in $\mathbf{SigDep}^{frag}$.



Let $M_2^{Y'} \in [\![\underline{\mathcal{S}}_2^{A_2^{i+1}}\downarrow]\!]$ be the result of amalgamation of $M_2^i|_{\underline{\mathcal{S}}_2^{X_2'}\downarrow}$ and $M_2^Y|_{\underline{\mathcal{S}}_2^{Y_2}\downarrow}$ w.r.t. the pushout given above. Let $M_2^s \in [\![\underline{\mathcal{S}}_2]\!]$ be such that $M_2^s|_{\underline{\mathcal{S}}_2^{A_2^{i+1}}\downarrow} = M_2^{Y'}$ (cf. assumption (7) in Sect. 3.5). From (3) of Def. 6.13, since for all $a \in A_2^{i+1}$, $M_2^s|_{\underline{\mathcal{S}}_2^{a}\downarrow} \in Con_2|_{\underline{\mathcal{S}}_2^{a}\downarrow}$ we get $M_2^s|_{\underline{\mathcal{S}}_2^{A_2^{i+1}}\downarrow} \in Con_2|_{\underline{\mathcal{S}}_2^{A_2^{i+1}}\downarrow}$. As a consequence, there exists $M_2^{i+1} \in Con_2$ such that $M_2^s|_{\underline{\mathcal{S}}_2^{A_2^{i+1}}\downarrow} = M_2^{i+1}|_{\underline{\mathcal{S}}_2^{A_2^{i+1}}\downarrow}$. Obviously $M_2^{i+1}|_{\underline{\mathcal{S}}_2^{A_2}\downarrow} = M_2|_{\underline{\mathcal{S}}_2^{A_2}\downarrow}$. Moreover, by construction we have

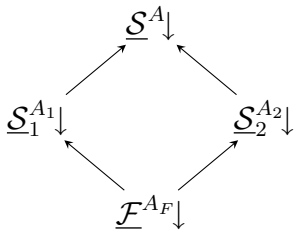
$$(M_1^{i+1}|_{\underline{\varphi_1}})|_{\underline{\mathcal{F}}^{A_F^{i+1}}\downarrow} = (M_2^{i+1}|_{\underline{\varphi_2}})|_{\underline{\mathcal{F}}^{A_F^{i+1}}\downarrow}$$

This finishes the induction step construction of M_1^{i+1} and M_2^{i+1}.

Put $M_1' = M_1^n$ and $M_2' = M_2^n$. Since $A_F^n = \mathbf{Compl}(\underline{\mathcal{F}})$, we indeed have $M_1^n|_{\underline{\varphi_1}} = M_2^n|_{\underline{\varphi_2}}$ and, by the construction, $M_i' \in Con_i$, $M_i'|_{\underline{\mathcal{S}}_i^{A_i}\downarrow} = M_i|_{\underline{\mathcal{S}}_i^{A_i}\downarrow}$, for

$i \in \{1, 2\}$, as it was required.

Now, to prove the main fact in Lemma 6.54, let M' be the result of amalgamation of M'_1 and M'_2. By definition of the sum of construction models, $M' \in Con_1 \oplus_{ft} Con_2$. The following square of inclusions is a pushout in $\mathbf{SigDep}^{frag}$



where $A_F = \underline{\varphi_1}^{-1}(A_1) = \underline{\varphi_2}^{-1}(A_2)$. We have $M'|_{\underline{\mathcal{S}}^{A}\downarrow} = M|_{\underline{\mathcal{S}}^{A}\downarrow}$, because for $i \in \{1, 2\}$, $(M'|_{\underline{\beta_i}})|_{\underline{\mathcal{S}}_i^{A_i}\downarrow} = M_1|_{\underline{\mathcal{S}}_i^{A_i}\downarrow}$ and by uniqueness of the amalgamation of models with respect to the above pushout. Therefore, $M|_{\underline{\mathcal{S}}^{A}\downarrow} \in (Con_1 \oplus Con_2)|_{\underline{\mathcal{S}}^{A}\downarrow}$, as required. □

Now, to prove Theorem 6.36 let us have two construction specifications, $\underline{SP}_1$ and $\underline{SP}_2$, connected by a fitting $\langle \varphi_1, \varphi_2 \rangle$ and let there be two clean construction models $Con_1 \models^c \underline{SP}_1$ and $Con_2 \models^c \underline{SP}_2$. We name $Con = Con_1 \oplus_{ft} Con_2 \in [\![\underline{\mathcal{S}}]\!]^c$ and $\underline{SP} = \underline{\beta_1}(\underline{SP}_1) \cup \underline{\beta_2}(\underline{SP}_2)$. By Theorem 6.31 Con is a construction model. By Lemma 6.53 Con is clean, i.e. $\mathbf{Clean}_{\underline{SP}}(Con) = Con$. To show that $Con \models^c \underline{SP}$ we prove the four conditions of Def. 6.13.

As for condition (1), let $a \in \underline{\mathcal{S}}$ and let there be $M \in Con$ such that $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^{a}\Downarrow}$. Since Con is clean, from $M \in Con$ and Lemma 6.53 we get $M \models \underline{SP}$, therefore, $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^{a}\downarrow}$.

Regarding condition (2), let $a \in \mathbf{Compl}(\underline{\mathcal{S}})$ be such that $a \notin \underline{\mathcal{S}}$ and let there be $M \models \underline{SP}$ such that $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\Downarrow}$. For $i \in \{1, 2\}$, let $A_i = \beta_i^{-1}(\mathbf{Symb}(\underline{\mathcal{S}}^{a}\downarrow))$. Let $M_i = M|_{\underline{\beta_i}}$. Of course $M_i \models \underline{SP}_i$. For every $a_i \in A_i$, $M_i|_{\underline{\mathcal{S}}_i^{a_i}\Downarrow} \in Con_i|_{\underline{\mathcal{S}}_i^{a_i}\Downarrow}$. By the definition of the construction signature morphism, $a_i \in \mathbf{Compl}(\underline{\mathcal{S}}_i)$ and $a_i \notin \underline{\mathcal{S}}_i$, because otherwise $a \in \underline{\mathcal{S}}$. By condition (2) of Def. 6.13 for Con_i, we get $M_i|_{\underline{\mathcal{S}}_i^{a_i}\downarrow} \in Con_i|_{\underline{\mathcal{S}}_i^{a_i}\downarrow}$. By the condition (3) of the same definition we have $M_i|_{\underline{\mathcal{S}}_i^{A_i}\downarrow} \in Con_i|_{\underline{\mathcal{S}}_i^{A_i}\downarrow}$. By Lemma 6.54 we get $M|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\downarrow}$, as required.

To prove condition (3), let there be $A \subseteq \underline{\mathcal{S}}$ and a model $M \in [\![\underline{\mathcal{S}}]\!]$. Let us assume that for every $a \in A$, $M|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\downarrow}$. For $i \in \{1, 2\}$, let $A_i = \beta_i^{-1}(\mathbf{Symb}(\underline{\mathcal{S}}^{A}\downarrow))$. Let $M_i = M|_{\underline{\beta_i}}$, for every $a_i \in A_i$, $M_i|_{\underline{\mathcal{S}}_i^{a_i}\downarrow} \in Con|_{\underline{\mathcal{S}}_i^{a_i}\downarrow}$. By condition (3) of Def. 6.13 for Con_i, we get $M_i|_{\underline{\mathcal{S}}_i^{A_i}\downarrow} \in Con_i|_{\underline{\mathcal{S}}_i^{A_i}\downarrow}$. By Lemma 6.54 we obtain $M|_{\underline{\mathcal{S}}^{A}\downarrow} \in Con|_{\underline{\mathcal{S}}^{A}\downarrow}$, as required.

Condition (4) follows: let there be $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ and a model $M \in Con$ such that for all $a \in A$ $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^{a}\downarrow}$. By Lemma 6.53 (since Con_1 and Con_2 are assumed to be clean), $Con = \mathbf{Clean}_{\underline{SP}}(Con)$, i.e. $M \models \underline{SP}$, thus $M|_{\underline{\mathcal{S}}^{A}\downarrow} \models \underline{SP}|_{\underline{\mathcal{S}}^{A}\downarrow}$. □

Chapter 7

Refinements

7.1 Introduction

Specifications refinements are a means to strengthen the specification in order to add more implementation details. The following types of refinement are described in the literature (cf. [GB80, ST88, ST12]).

Simple refinement. Specification SP_1 refines SP_0, $SP_0 \rightsquigarrow SP_1$, when every SP_1-model is an SP_0-model. Specification refinements compose (vertically), i.e. $SP_0 \rightsquigarrow SP_1$ and $SP_1 \rightsquigarrow SP_2$ implies $SP_0 \rightsquigarrow SP_2$. Implementation is a sequence of refinement steps

$$SP_0 \rightsquigarrow SP_1 \rightsquigarrow \ldots \rightsquigarrow SP_n$$

from the most abstract SP_0 to the most detailed SP_n. All models of SP_n are also models of SP_0.

Stepwise refinement via constructors. Constructor implementation, denoted by $SP_0 \overset{\rightsquigarrow}{\kappa_1} SP_1$, is a refinement via constructor $\kappa_1 \colon [\![Sig(SP_1)]\!] \to [\![Sig(SP_0)]\!]$. A constructor implementation is correct if for every model $M \in [\![SP_1]\!]$, $\kappa_1(M) \in [\![SP_0]\!]$. In this approach, an implementation of specification SP_0 is a sequence of refinement steps via constructors

$$SP_0 \overset{\rightsquigarrow}{\kappa_1} SP1 \overset{\rightsquigarrow}{\kappa_2} \ldots \overset{\rightsquigarrow}{\kappa_n} SP_n = EMPTY$$

where $EMPTY$ stands for the empty specification. If all constructor implementations in the sequence are correct, a model of SP_0 is obtained by application of constructors, starting from the unique trivial model

$M_E \in [\![EMPTY]\!]$,

$$\kappa_1(\kappa_2(\ldots \kappa_n(M_E)\ldots)) \in [\![SP_0]\!].$$

Branching stepwise refinement via multi-parameter constructors. Multi-parameter constructor implementation $SP_0 \overset{\kappa_1}{\rightsquigarrow} (SP_1^1, \ldots, SP_n^1)$ is a refinement via constructor $\kappa_1 \colon [\![Sig(SP_1^1)]\!] \times \cdots \times [\![Sig(SP_n^1)]\!] \to [\![Sig(SP_0)]\!]$. Similarly to the single-parameter case, a multi-parameter constructor implementation is correct if for all models $M_1 \in [\![SP_1^1]\!], \ldots, M_n^1 \in [\![SP_1^n]\!]$, it holds that $\kappa_1(M_1, \ldots, M_n) \in [\![SP_0]\!]$. In this approach, refinement steps introduce branching and the implementation process of SP_0 is presented as a tree

$$SP_0 \overset{\kappa}{\rightsquigarrow} \begin{cases} SP_1 \overset{\kappa_1}{\rightsquigarrow} EMPTY \\ \ldots \\ SP_n \overset{\kappa_n}{\rightsquigarrow} \begin{cases} SP_{n1} \overset{\kappa_{n1}}{\rightsquigarrow} EMPTY \\ \ldots \\ SP_{nm} \overset{\kappa_{nm}}{\rightsquigarrow} EMPTY \end{cases} \end{cases}$$

with specification $EMPTY$ in all leaves. Again, if all constructor implementation steps in the tree are correct, the model of SP_0 is obtained by application of constructions to trivial models of $EMPTY$.

Constructors in the latter two approaches are the same concept as single- and multi-parameter parameterised modules (cf. Sect. 3.4). Constructions introduced in the previous chapter also serve the similar purpose.

In this chapter we introduce the notion of *construction specification refinement* corresponding to the simple refinement of specifications, but defined for construction specifications. The construction fittings together with the sum operation described in Sect. 6.5 are suitable for specification of branching.

Our approach to specification implementation combines the simple refinement and the branching stepwise refinement. We call it *diagrams of constructions* (cf. Chapter 8). Implementation process of SP_0 is presented as a tree

$$\underline{SP}_0 \overset{\underline{\omega}}{\rightsquigarrow}{}^{\mathrm{c}}\, \underline{SP}'_0 \;\overset{}{\underset{ft_0}{\rightsquigarrow}}\; \left\{ \begin{array}{l} \underline{SP}_1 \overset{\underline{\omega_1}}{\rightsquigarrow}{}^{\mathrm{c}}\, \underline{SP}'_1 \\ \underline{SP}_2 \overset{\underline{\omega_2}}{\rightsquigarrow}{}^{\mathrm{c}}\, \underline{SP}'_2 \;\underset{ft_2}{\rightsquigarrow}\; \left\{ \begin{array}{l} \underline{SP}_{21} \overset{\underline{\omega}}{\rightsquigarrow}{}^{\mathrm{c}}\, \underline{SP}'_{21} \;\underset{ft_{21}}{\rightsquigarrow}\; \left\{ \begin{array}{l} SP_{211} \\ \underline{SP}_{212} \end{array} \right. \\ \underline{SP}_{22} \overset{\underline{\omega_{22}}}{\rightsquigarrow}{}^{\mathrm{c}}\, SP'_{22} \end{array} \right. \end{array} \right.$$

where $\underline{SP}_0 \overset{\underline{\omega}}{\rightsquigarrow}{}^{\mathrm{c}}\, \underline{SP}'_0$ says that $\underline{SP}'_0$ is a construction specification refinement of $\underline{SP}_0$ along construction signature refinement morphism (cf. Def. 7.15 below) and $\underline{SP}'_0 \underset{ft_0}{\rightsquigarrow} (\underline{SP}_1, \underline{SP}_2)$ says that $\underline{SP}'_0 = \underline{SP}_1 \oplus_{ft_0} \underline{SP}_2$ (cf. Def. 6.32), i.e. that $\underline{SP}'_0$ is a sum of construction specifications $\underline{SP}_1$ and $\underline{SP}_2$ w.r.t. construction fitting ft. If the construction specifications in every branch are compatible, the construction model of (the most abstract) $\underline{SP}_0$ is obtained by a series of sums of construction models and reducts along construction refinement morphisms (cf. Def. 7.4 below), starting from (the most detailed) construction models of the construction specification in the leaves of the tree.

7.2 Construction Signature Refinement Morphisms

The construction signature morphisms considered so far (cf. Def. 6.1) preserve and reflect the dependency structure of each symbol. This means that adding new or removing old dependencies is illegal. In this section, we introduce another type of morphism between construction signatures called *construction signature refinement morphism*. It allows for adding new symbols dependent on the existing signature symbols, but at the cost of requiring injectivity on assumed symbols. The details follow.

Definition 7.1 (Construction Signature Refinement Morphism) *A* construction signature refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ *is a* $\mathbf{Sig}^{frag}$*-morphism subject to the following conditions:*

1. (injective on assumed) *for all* $b_1, b'_1 \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$*, if* $b_1, b'_1 \notin \underline{\mathcal{S}}_1$ *and* $\underline{\omega}(b_1) = \underline{\omega}(b'_1)$ *then* $b_1 = b'_1$*,*

2. (assumed stay so) *for all* $b_1 \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$, *if* $b_1 \notin \underline{\mathcal{S}}_1$ *then* $\underline{\omega}(b_1) \notin \underline{\mathcal{S}}_2$,

3. (new are defined) *for all* $b_2 \in \mathbf{Compl}(\underline{\mathcal{S}}_2)$, *if* $b_2 \notin img(\omega)$ *then* $b_2 \in \underline{\mathcal{S}}_2$,

4. (p-morphism within the image of $\underline{\omega}$)

   (a) (monotonic) *for all* $b_1, b_1' \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$, *if* $b_1' < b_1$ *then* $\underline{\omega}(b_1') < \underline{\omega}(b_1)$,

   (b) (weakly reflected $<$ within the range) *for all* $b_1, b_1' \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$, *if* $\underline{\omega}(b_1') < \underline{\omega}(b_1)$, *then there exists* $b_1'' \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$ *such that* $b_1'' < b_1$ *and* $\underline{\omega}(b_1'') = \underline{\omega}(b_1')$,

The above requirements say that assumed elements must be injectively mapped and they must not become defined. The assumed elements represent the parameters of the construction, therefore one must ensure that no implicit application is performed and that independent symbols remain independent in the refined signature. All new symbols are defined in the target signature, because they are considered auxiliary symbols, thus cannot be assumed there. Refinement morphisms are monotone and within their range they reflect weakly the dependency relation. This is to ensure that the refinement neither removes nor adds any dependencies in the target signature between symbols that existed in the source signature. The highlight of refinement morphisms is that there is no restriction concerning dependencies to and from new symbols.

It is easy to check that construction signature refinement morphisms compose and that the identity morphisms from $\mathbf{Sig}^{frag}$ are also construction signature refinement morphisms.

Definition 7.2 *By* $\mathbf{SigDep}^{ref}$ *we denote the category of construction signatures with objects from* $\mathbf{SigDep}^{frag}$ *and construction signature refinement morphisms introduced by Def. 7.1.*

In general, construction signature morphisms are not construction signature refinement morphisms and vice versa. The former are suitable for application, i.e. an assumed symbol may be mapped to a defined symbol.

The latter are suitable to express implementation details, because they permit addition of new auxiliary symbols on which the existing symbols may depend.

Example 7.3 *Consider the following three construction signatures*

$$\underline{\mathcal{S}}_1 = (\textbf{sort } \underline{s};\ \textbf{ops } \underline{a : s},\ b : s;\ \textbf{dep } a < b),$$
$$\underline{\mathcal{S}}_2 = (\textbf{sort } \underline{s};\ \textbf{ops } \underline{a : s},\ b : s,\ c : s;\ \textbf{dep } a < b,\ c < b),$$
$$\underline{\mathcal{S}}_3 = (\textbf{sort } \underline{s};\ \textbf{ops } b : s,\ c : s;\ \textbf{dep } c < b)$$

and the two $\mathbf{Sig}^{frag}$*-morphisms*

$$\underline{\omega} \colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2 = \{s \mapsto s, a \mapsto a, b \mapsto b\},$$
$$\underline{\varphi} \colon \underline{\mathcal{S}}_2 \to \underline{\mathcal{S}}_3 = \{s \mapsto s, a \mapsto c, b \mapsto b, c \mapsto c\}.$$

Morphism $\underline{\omega}$ *adds a new defined symbol* c *into the dependency structure of* b*. Morphism* $\underline{\varphi}$ *merges assumed* a *with defined* c*, what corresponds to the self-application operation. It is easy to check that* $\underline{\omega} \in \mathbf{SigDep}^{ref}$ *and* $\underline{\varphi} \in \mathbf{SigDep}^{frag}$*, i.e.* $\underline{\omega}$ *is a construction signature refinement morphism, whereas* $\underline{\varphi}$ *is a construction signature morphism. We also note that* $\varphi \notin \mathbf{SigDep}^{ref}$ *and* $\omega \notin \mathbf{SigDep}^{frag}$*.*

The above notation assumes the existence of the coercion of $\mathbf{SigDep}^{ref}$ into $\mathbf{Sig}^{frag}$. The coercion functor $\mathbf{UnDep}^{ref} : \mathbf{SigDep}^{ref} \to \mathbf{Sig}^{frag}$ simply removes dependency relation from objects and acts as an identity on morphisms, analogously to $\mathbf{UnDep}^{frag}$ (defined in Sect. 5.3).

Let us now define a reduct of construction model along a construction signature refinement morphism.

Definition 7.4 (Reduct along Construction Refinement Morphism) *Given a* $\mathbf{SigDep}^{ref}$*-morphism* $\underline{\omega} \colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ *and a construction model* $Con_2 \in [\![\underline{\mathcal{S}}_2]\!]^c$*, the* reduct of the construction model Con_2 along $\underline{\omega}$ *is defined as*

$$Con_2|_{\underline{\omega}} = \{M|_{\mathbf{Compl}(\underline{\omega})} \mid M \in Con_2\}$$

The definition of the reduct along construction refinement morphism copies the definition of the reduct along construction signature morphism given in Def. 6.7. Nevertheless, since morphisms are different, the following lemma is needed.

Lemma 7.5 *The reduct of a construction model of $\underline{\mathcal{S}}_2$ along a construction refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ is a construction model of $\underline{\mathcal{S}}_1$.*

The proof of the lemma is in Appendix 7.A.

The following lemma shows that the reduct along a construction refinement morphism preserves the property of being well-grouped construction model (cf. Def. 6.10). The proof relies on the finiteness of the source construction signature.

Lemma 7.6 *The reduct of a well-grouped construction model of a construction signature $\underline{\mathcal{S}}_2$ along a construction refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$, for a finite construction signature $\underline{\mathcal{S}}_1$, is a well-grouped construction model of $\underline{\mathcal{S}}_1$.*

The proof of the lemma is in Appendix 7.A.

7.3 Construction Specification Refinements

The obvious standard notion of the simple refinement of two construction specifications $\underline{SP}_1$ and $\underline{SP}_2$ over the same construction signature $\underline{\mathcal{S}}$ is $[\![\underline{SP}_2]\!]^c \subseteq [\![\underline{SP}_1]\!]^c$ (cf. [ST12]). We find it useful, however, to introduce this notion differently, relying directly on the concepts from the base institution $\mathbb{I}$. The resulting notion, although technically slightly different, will serve the same purpose in all practical situations.

Definition 7.7 (Construction Specification Refinement) *Given two construction specifications $\underline{SP}_1$ and $\underline{SP}_2$ over the same construction signature $\underline{\mathcal{S}}$, $\underline{SP}_1$ is a* construction specification refinement *of $\underline{SP}_1$, $\underline{SP}_1 \rightsquigarrow^c \underline{SP}_2$, iff*

1. *(refinement) $[\![\underline{SP}_2]\!] \subseteq [\![\underline{SP}_1]\!]$;*

2. *(stronger only on defined) for* $a \in \mathbf{Compl}(\underline{\mathcal{S}})$ *and a model* $M \in [\![\underline{\mathcal{S}}]\!]$, *if* $M|_{\underline{\mathcal{S}}^{a\Downarrow}} \models \underline{SP_2}|_{\underline{\mathcal{S}}^{a\Downarrow}}$ *and* $M|_{\underline{\mathcal{S}}^{a\downarrow}} \models \underline{SP_1}|_{\underline{\mathcal{S}}^{a\downarrow}}$, *but* $M|_{\underline{\mathcal{S}}^{a\downarrow}} \not\models \underline{SP_2}|_{\underline{\mathcal{S}}^{a\downarrow}}$ *then* $a \in \underline{\mathcal{S}}$, *i.e. a is a defined symbol.*

The above conditions allow the specification $\underline{SP_2}$ to be stronger only on defined symbols. This corresponds to the typical refinement of parameterised specification, where only the specification of the result may be strengthened.

When we compare Def. 7.7 and simple $[\![\underline{SP_2}]\!]^c \subseteq [\![\underline{SP_1}]\!]^c$, it is clear that the former is stronger, because it requires the equivalent specification of assumed symbols in both specifications, whereas the latter admits the weakening of the said specifications.

Example 7.8 *Consider the construction signature*

$$\underline{\mathcal{S}} = (\mathbf{sort}\ s;\ \mathbf{ops}\ \underline{a : s},\ \underline{b : s},\ c : s;\ \mathbf{deps}\ a < b\ , b < c)$$

and three construction specifications over $\underline{\mathcal{S}}$

$$\begin{aligned}\underline{SP_1} &= \langle \underline{\mathcal{S}}, \{b = a\}\rangle,\\ \underline{SP_2} &= \langle \underline{\mathcal{S}}, \{b = c\}\rangle,\\ \underline{SP_3} &= \langle \underline{\mathcal{S}}, \{b = a, b = c\}\rangle.\end{aligned}$$

It holds that

$$\underline{SP_1} \not\rightsquigarrow^c \underline{SP_2} \text{ and } \underline{SP_1} \rightsquigarrow^c \underline{SP_3}$$

but

$$[\![\underline{SP_2}]\!]^c \subseteq [\![\underline{SP_1}]\!]^c \text{ and } [\![\underline{SP_3}]\!]^c \subseteq [\![\underline{SP_1}]\!]^c.$$

The following lemma shows that Def. 7.7 does indeed yield the refinement of construction specifications.

Lemma 7.9 *Given a construction specification refinement* $\underline{SP_1} \rightsquigarrow^c \underline{SP_2}$ *and a construction model* $Con \models^c \underline{SP_2}$, *it holds that* $Con \models^c \underline{SP_1}$.

The proof is in Appendix 7.A.

It is worth to note that a clean model of a refinement is also a clean model of the original specification.

Lemma 7.10 *For* $\underline{SP}_1 \rightsquigarrow^c \underline{SP}_2$ *and a construction model* $Con \models^c \underline{SP}_2$, $\mathbf{Clean}_{\underline{SP}_2}(Con) = \mathbf{Clean}_{\underline{SP}_1}(\mathbf{Clean}_{\underline{SP}_2}(Con))$.

The proof is a direct application of condition (1) from Def. 7.7.

A special kind of construction refinement morphism, compatible with the specification is needed to prove the satisfaction condition.

Definition 7.11 (Conservative Construction Refinement Morphism) *Given a construction specification* $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1 \rangle \in \mathbf{Spec}(\underline{\mathcal{S}}_1)$ *and a construction refinement morphism* $\underline{\omega} \colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$, *we say that* $\underline{\omega}$ is a conservative construction refinement morphism w.r.t. $\underline{SP}_1$ *iff* $\underline{\omega} \colon SP_1 \to \underline{\omega}(SP_1)$ *is a conservative specification morphism.*

It is easy to show that all injective construction specification morphisms are conservative.

Lemma 7.12 *For a construction specification* $\underline{SP}_1 \in \mathbf{Spec}(\underline{\mathcal{S}}_1)$ *and a construction refinement morphism* $\underline{\omega} \colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$, *if* $\underline{\omega}$ *is injective on symbols then* $\underline{\omega}$ *is a conservative construction refinement morphism w.r.t.* $\underline{SP}_1$.

Proof. The condition from Def. 7.11 is discharged directly by assumption (7) from Sect. 3.5. □

The following example shows a non-injective construction refinement morphism that is conservative for some construction specification and fails to be conservative for another construction specification.

Example 7.13 *Consider two construction signatures*

$$\underline{\mathcal{S}}_1 = (\mathbf{sort}\ s;\ \mathbf{ops}\ a_1 : s,\ a_2 : s, b : s;\ \mathbf{deps}\ b < a_1,\ b < a_2),$$
$$\underline{\mathcal{S}}_2 = (\mathbf{sort}\ s;\ \mathbf{ops}\ a : s,\ b : s,\ c : s;\ \mathbf{deps}\ c < b,\ b < a)$$

and the construction refinement morphism $\underline{\omega} \colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ *given by* $\underline{\omega}(s) = s$, $\underline{\omega}(a_1) = a$ *and* $\underline{\omega}(a_2) = a$. *Let there be the two construction specifications*

$$\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, \{a_1 = b, a_2 = b\} \rangle,$$
$$\underline{SP}'_1 = \langle \underline{\mathcal{S}}_1, \emptyset \rangle.$$

Morphism $\underline{\omega}$ is conservative w.r.t. $\underline{SP}_1$, but it is not conservative w.r.t. $\underline{SP}'_1$.

The satisfaction condition (as discussed in Sect. 3.3) legitimates the use of translation on construction specifications. It also constitutes a step towards the definition of an institution of constructions. The following theorem proves the "if" part of the satisfaction condition. The second half, the "only-if" part, is not needed for our purposes and we leave it for the future work. The proof uses Lemma 7.6 and thus requires the finiteness of the source construction signature (cf. the comment on assumption (2) in Sect. 3.5).

Theorem 7.14 (Satisfaction Condition - the "if" part) *Consider a signature refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ and a construction specification $\underline{SP} \in \mathbf{Spec}(\underline{\mathcal{S}}_1)$ such that $\underline{\mathcal{S}}_1$ is finite and $\underline{\omega}$ is a conservative construction refinement morphism w.r.t. $\underline{SP}$ (cf. Def. 7.11); given a construction model $Con_2 \in [\![\underline{\mathcal{S}}_2]\!]^c$ such that $Con_2 = \mathbf{Clean}_{\underline{\omega}(\underline{SP})}(Con_2)$,*

$$\textit{if } Con_2 \models^c \underline{\omega}(\underline{SP}) \textit{ then } Con_2|_{\underline{\omega}} \models^c \underline{SP}$$

The proof is in Appendix 7.A.

We are now ready to introduce formally a refinement of a construction specification along a signature refinement morphism.

Definition 7.15 (Construction Specification Refinement) *For a construction signature refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ and two construction specifications $\underline{SP}_1$ and $\underline{SP}_2$ over $\underline{\mathcal{S}}_1$ and $\underline{\mathcal{S}}_2$, respectively, we say that* $\underline{SP}_2$ is a construction specification refinement of $\underline{SP}_1$ along $\underline{\omega}$, *denoted by $\underline{SP}_1 \overset{\underline{\omega}}{\rightsquigarrow}{}^c \underline{SP}_2$, iff*

1. *$\underline{\omega}$ is a conservative construction refinement morphism w.r.t. $\underline{SP}_1$,*
2. *$\underline{\omega}(\underline{SP}_1) \rightsquigarrow^c \underline{SP}_2$ (cf. Def. 7.7).*

As a consequence of Lemma 7.9, Lemma 7.10, Theorem 6.25 and Theorem 7.14, Def. 7.15 describes the refinement of construction specifications via the reduct w.r.t. the construction signature refinement morphism.

Corollary 7.16 *Consider a construction specification refinement along signature refinement morphism* $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$*, for a finite construction signature* $\underline{\mathcal{S}}_1$*,*

$$\underline{SP}_1 \sim\!\!\overset{\underline{\omega}}{\rightsquigarrow}{}^{c}\, \underline{SP}_2.$$

For any clean construction model $Con_2 \models^c \underline{SP}_2$*, it holds that*

$$Con_2|_{\underline{\omega}} \models^c \underline{SP}_1$$

and also

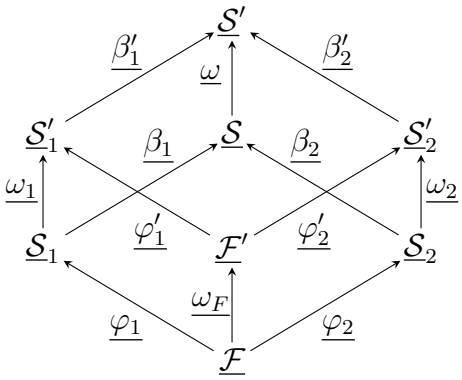
$$\mathbf{Clean}_{\underline{SP}_1}(Con_2|_{\underline{\omega}}) = Con_2|_{\underline{\omega}}\ .$$

The above corollary justifies the notion of refinement of construction specifications in diagrams of constructions, to be presented in Chapter 8.

7.4 Refinement Compositionality

Having defined the construction specification refinement we check how it can be used with the sum operation from Sect. 6.5. At the beginning we prove compositionality of vertical morphisms (refinement) w.r.t. horizontal morphisms (sum operation) (cf. [ST06]). Yet later we discover limits of its applicability, as it turns out that it does not cover some interesting cases of the stepwise system construction method.

Theorem 7.17 (Refinement and Fittings Compositionality) *Given two construction fitting spans* $\langle \varphi_1, \varphi_2 \rangle\colon \underline{\mathcal{S}}_1 \nwarrow\!\!\nearrow \underline{\mathcal{S}}_2$ *and* $\langle \varphi'_1, \varphi'_2 \rangle\colon \underline{\mathcal{S}}'_1 \nwarrow\!\!\nearrow \underline{\mathcal{S}}'_2$ *and their pushouts* β_1*,* β_2 *and* β'_1*,* β'_2*; given also three construction refinement morphisms* $\underline{\omega_1}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}'_1$*,* $\underline{\omega_2}\colon \underline{\mathcal{S}}_2 \to \underline{\mathcal{S}}'_2$ *and* $\underline{\omega_F}\colon \underline{\mathcal{F}} \to \underline{\mathcal{F}}'$ *such that* $\underline{\omega_F}; \underline{\varphi'_1} = \underline{\varphi_1}; \underline{\omega_1}$ *and* $\underline{\omega_F}; \underline{\varphi'_2} = \underline{\varphi_2}; \underline{\omega_2}$ *in* $\mathbf{Sig}^{frag}$*; there exists the unique construction refinement morphism* $\underline{\omega}\colon \underline{\mathcal{S}} \to \underline{\mathcal{S}}'$ *such that the following diagram commutes in* $\mathbf{Sig}^{frag}$



where $\underline{\mathcal{S}}$ and $\underline{\mathcal{S}'}$ are the construction signatures of the sum (cf. Def. 6.28) of $\langle \underline{\varphi_1}, \underline{\varphi_2} \rangle$ and $\langle \underline{\varphi'_1}, \underline{\varphi'_2} \rangle$ respectively.

The proof is in Appendix 7.A. The result is not a trivial consequence of pushout properties, because the above diagram is in $\mathbf{Sig}^{frag}$ and we require $\underline{\omega}$ te be a $\mathbf{SigDep}^{ref}$-morphism. Let us remind that both $\mathbf{SigDep}^{frag}$ and $\mathbf{SigDep}^{ref}$ are embeddable into $\mathbf{Sig}^{frag}$ via functors $\mathbf{UnDep}^{frag}$ and $\mathbf{UnDep}^{ref}$ respectively.

Applicability of Theorem 7.17 is limited, because the conditions put on construction fittings and refinement morphisms are somehow incompatible and do not cover all cases of interest. On the one hand, a construction signature being the source of a construction fitting span is required to be an empty fragment, i.e., with all elements being assumed. On the other hand, new elements that may appear in the target of a construction refinement morphism have to be defined. As a consequence, refinement morphism $\underline{\omega_F}$ from Theorem 7.17 must not add any new elements to its target and it must be injective on assumed symbols. Therefore, since construction signature morphisms are p-morphisms, refinements $\underline{\omega_1}$ and $\underline{\omega_2}$ must not add any new dependencies to shared elements, i.e., elements that come from from $\underline{\mathcal{F}}$. As a consequence, no refinement (on either side) of shared symbols is possible.

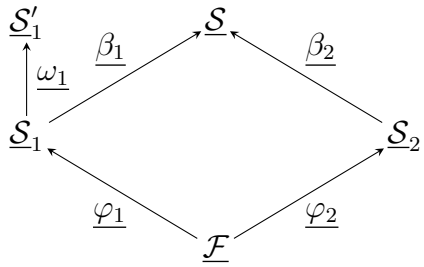
The following example shows some limitations of the compositionality given by Theorem 7.17.

Example 7.18 *We show a simple typical situation where refinement con-*

cerns shared symbols. Consider the following construction signatures

$$\begin{aligned}\underline{\mathcal{S}} &= (\textbf{sort } \underline{s};\ \textbf{ops } a : s,\ b : s),\\ \underline{\mathcal{S}}_1 &= (\textbf{sort } \underline{s};\ \textbf{op } a : s),\\ \underline{\mathcal{S}}_2 &= (\textbf{sort } \underline{s};\ \textbf{ops } \underline{a : s},\ b : s),\\ \underline{\mathcal{F}} &= (\textbf{sort } \underline{s};\ \textbf{op } \underline{a : s})\end{aligned}$$

connected by inclusions



and $\underline{\mathcal{S}}'_1 = (\textbf{sorts } \underline{s}, u;\ \textbf{ops } a : s,\ c : u;\ \textbf{dep } c < a)$ *being a refinement of* $\underline{\mathcal{S}}_1$ *via inclusion.*

There exists no refinement of $\underline{\mathcal{S}}_2$ *that may be a part of the cube from Theorem 7.17. Let us analyse why.*

First we notice that in fact $\underline{\varphi_1}$ *and* $\underline{\varphi_2}$ *are* $\mathbf{SigDep}^{frag}$*-morphisms. Let* $ft = \langle \underline{\varphi_1}, \underline{\varphi_2} \rangle$*, which is a construction fitting, and* $\underline{\beta_1}$ *and* $\underline{\beta_2}$ *are the pushout of* ft*, moreover* $\underline{\omega_1}$ *is a* $\mathbf{SigDep}^{ref}$*-morphism. Since construction signature refinement morphism* $\underline{\omega_1}$ *adds a new defined symbol* c *to the dependency structure of shared symbol* a *in* $\underline{\mathcal{S}}'_1$*, every refinement of* $\underline{\mathcal{S}}_2$ *(via any* $\underline{\omega_2}$*) would also have to make* a *dependent on the new defined symbol* c*. Consequently, if there existed construction signature* $\underline{\mathcal{F}'}$ *and a pair of morphisms* $\underline{\varphi'_1} \colon \underline{\mathcal{F}'} \to \underline{\mathcal{S}}_1$ *and* $\underline{\varphi'_2} \colon \underline{\mathcal{F}'} \to \underline{\mathcal{S}}_2$*, it would also contain this dependency. Construction fitting sources are empty fragments (all symbols assumed) and construction signature refinement morphisms are bijective on assumed symbols, so the inclusion between* $\underline{\mathcal{F}}$ *and* $\underline{\mathcal{F}'}$ *must have been an isomorphism. The pair* $\langle \varphi'_1, \varphi'_2 \rangle$ *would fail to be a construction fitting, because its pushout object would not be a construction signature (internal mapping would fail to be injective).*

The proper approach in the top-down style of the system construction is to to add a dependency to a shared symbol before it is decomposed via the sum operation.

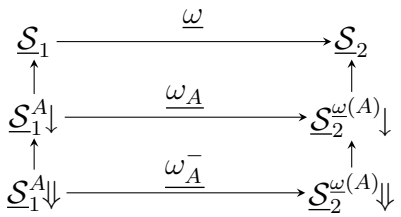
The next chapter introduces diagrams of constructions without assumption that local refinements compose with sums in any way.

7.A Appendix: Proofs

The following definition and lemma will be used in the subsequent proofs.

Definition 7.19 *Given two signature fragments* $\underline{\mathcal{S}}_1, \underline{\mathcal{S}}_2 \in \mathbf{SigDep}^{ref}$*, a set* $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}}_1)$ *and a* $\mathbf{SigDep}^{ref}$*-morphism* $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$*,*

- *the* $\mathbf{SigDep}^{ref}$*-morphism* $\underline{\omega}_A\colon \underline{\mathcal{S}}_1^{A\downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(A)\downarrow}$ *is given as* $\underline{\omega}_A = \underline{\omega}|_{\underline{\mathcal{S}}_1^{A\downarrow}}$*,*
- *if* A *is a set of independent symbols in* $\mathbf{Compl}(\underline{\mathcal{S}}_1)$ *and* $\underline{\omega}(A)$ *is a set of independent symbols*[1] *in* $\mathbf{Compl}(\underline{\mathcal{S}}_2)$*, the* $\mathbf{SigDep}^{ref}$*-morphism* $\underline{\omega}_A^-\colon \underline{\mathcal{S}}_1^{A\Downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(A)\Downarrow}$*, is given as* $\underline{\omega}_A^- = \underline{\omega}|_{\underline{\mathcal{S}}_1^{A\Downarrow}}$*,*
- *the following diagram, where all unnamed arrows are inclusions, is a commuting diagram in* $\mathbf{SigDep}^{ref}$



Cf. Def. 6.49 for analogous definition for $\mathbf{SigDep}^{frag}$-morphisms.

Notation. We sometimes use an element, instead of a singleton set that consists of this element. When $A = \{b\}$, instead of $\underline{\omega}_A\colon \underline{\mathcal{S}}_1^{A\downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(A)\downarrow}$ and $\underline{\omega}_A^-\colon \underline{\mathcal{S}}_1^{A\Downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(A)\Downarrow}$, we write $\underline{\omega}_b\colon \underline{\mathcal{S}}_1^{b\downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(b)\downarrow}$ and $\underline{\omega}_b^-\colon \underline{\mathcal{S}}_1^{b\Downarrow} \to \underline{\mathcal{S}}_2^{\underline{\omega}(b)\Downarrow}$, respectively.

Lemma 7.20 *Given a construction refinement morphism* $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$*, a construction model* $Con_2 \in [\![\underline{\mathcal{S}}_2]\!]^c$*, two models* $M_2, M_2' \in Con_2$*, and* $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}}_1)$*,*

1. *if* $(M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{A\downarrow}} = (M_2'|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{A\downarrow}}$*, then* $M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(A)\downarrow}} = M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(A)\downarrow}}$*,*

[1] Again, as in Def. 6.49, this is an important assumption for all $\underline{\omega}$ that are not injective on symbols.

2. *if A is a set of independent symbols in* $\mathbf{Compl}(\underline{\mathcal{S}}_1)$ *and* $\underline{\omega}(A)$ *is a set of independent symbols in* $\mathbf{Compl}(\underline{\mathcal{S}}_2)$*, if* $(M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{A}\Downarrow} = (M_2'|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{A}\Downarrow}$*, then* $M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow}$.

Proof. We prove the lemma for case (2). The proof for case (1) is analogous.

The morphism $\underline{\omega_A^-}\colon \underline{\mathcal{S}}_1^{A}\Downarrow \to \underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow$ does not need to be surjective. Let $\underline{\omega_{e_A}^-}$ be an abstract surjection $\underline{\omega_{e_A}^-}\colon \underline{\mathcal{S}}_1^{A}\Downarrow \to \underline{\omega_A^-}(\underline{\mathcal{S}}_1^{A}\Downarrow)$ given by factorisation of $\underline{\omega_A^-}$ to $\underline{\omega_{e_A}^-}$ and $\underline{\omega_{i_A}^-}$. Assumption (3) from Sect. 3.5 implies that abstract surjections in **Sig** are surjective on their symbols, thus $\underline{\omega_{e_A}^-}$ is surjective on its symbols. Therefore, by assumption (6) from Sect. 3.5, the reduct functor $\_|_{\underline{\omega_{e_A}^-}}\colon [\![\underline{\omega_A^-}(\underline{\mathcal{S}}_1^{A}\Downarrow)]\!] \to [\![\underline{\mathcal{S}}_1^{A}\Downarrow]\!]$ is injective on models. Consequently, we get $M_2|_{\omega_e^-(\underline{\mathcal{S}}_1^{A}\Downarrow)} = M_2'|_{\omega_e^-(\underline{\mathcal{S}}_1^{A}\Downarrow)}$.

By induction on the dependency bound (cf. Def. 3.4) of elements in $\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow$ we prove that for every $b \in \mathbf{Compl}(\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow)$, $M_2|_{\underline{\mathcal{S}}_2^{b}\downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{b}\downarrow}$.

(Base case) For $i = 0$, there are no $b \in \mathbf{Compl}(\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow)$, such that $db(\underline{\mathcal{S}}_2^{b}\downarrow) = 0$, so the implication trivially holds.

(Induction step) Let $0 < i < db(\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow)$ and for all $c \in \mathbf{Compl}(\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow)$ such that $db(\underline{\mathcal{S}}_2^{c}\downarrow) < i$, we have $M_2|_{\underline{\mathcal{S}}_2^{c}\downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{c}\downarrow}$. Let $b \in \mathbf{Compl}(\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow)$ be such that $db(\underline{\mathcal{S}}_2^{b}\downarrow) = i$. By inductive assumption we have $M_2|_{\underline{\mathcal{S}}_2^{b}\Downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{b}\Downarrow}$ and

1. either $b \in \underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow$ (b is a defined element) and since $M_2, M_2' \in \mathit{Con}$, by Def. 6.5, from $M_2|_{\underline{\mathcal{S}}_2^{b}\Downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{b}\Downarrow}$ we get $M_2|_{\underline{\mathcal{S}}_2^{b}\downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{b}\downarrow}$,

2. or $b \notin \underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow$ (b is an assumed element), thus, by conditions (3) of Def. 7.1, $b \in \mathbf{Compl}(\omega_e^-(\underline{\mathcal{S}}_1^{A}\Downarrow))$, therefore, by inductive assumption $M_2|_{\underline{\mathcal{S}}_2^{b}\Downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{b}\Downarrow}$ and $M_2|_{\omega_e^-(\underline{\mathcal{S}}_1^{A}\Downarrow)} = M_2'|_{\omega_e^-(\underline{\mathcal{S}}_1^{A}\Downarrow)}$, we get $M_2|_{\underline{\mathcal{S}}_2^{b}\downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{b}\downarrow}$.

So indeed, for every $b \in \mathbf{Compl}(\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow)$, $M_2|_{\underline{\mathcal{S}}_2^{b}\downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{b}\downarrow}$, from which we conclude that $M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(A)}\Downarrow}$.

□

Proof of Lemma 7.5. Let there be a construction refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ and a construction model $\mathit{Con}_2 \in [\![\underline{\mathcal{S}}_2]\!]^c$. Let us check that $\mathit{Con}|_{\underline{\omega}}$ meets the condition of Def. 6.5.

Let $a \in \underline{\mathcal{S}}_1$ and let there be two models $M_1, M_1' \in Con_2|_{\underline{\omega}}$ such that $M_1|_{\underline{\mathcal{S}}^{a}\Downarrow} = M_1'|_{\underline{\mathcal{S}}^{a}\Downarrow}$. There exist two models $M_2, M_2' \in Con_2$ such that $M_2|_{\underline{\omega}} = M_1$ and $M_2'|_{\underline{\omega}} = M_1'$, thus $(M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}^{a}\Downarrow} = (M_2'|_{\underline{\omega}})|_{\underline{\mathcal{S}}^{a}\Downarrow}$. From Lemma 7.20 we get $M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow}$. Therefore, since $\underline{\omega}(a) \in \underline{\mathcal{S}}_2$, by Def. 6.5, we get $M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow} = M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow}$. Finally, $M_1|_{\underline{\mathcal{S}}_1^{a}\downarrow} = M_1'|_{\underline{\mathcal{S}}_1^{a}\downarrow}$, as required. □

Proof of Lemma 7.6. Consider a construction refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ such that construction signature $\underline{\mathcal{S}}_1$ is finite, a well-grouped construction model $Con_2 \in [\![\underline{\mathcal{S}}_2]\!]^c$, a set $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}}_1)$ and a model $M_1 \in [\![\mathbf{Compl}(\underline{\mathcal{S}}_1)]\!]$ such that for all $a \in A$, $M_1|_{\underline{\mathcal{S}}_1^{a}\downarrow} \in (Con_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a}\downarrow}$. This means that for all $a \in \mathbf{Compl}(\underline{\mathcal{S}}_1^{A}\downarrow)$, $M_1|_{\underline{\mathcal{S}}_1^{a}\downarrow} \in (Con_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a}\downarrow}$. Therefore, without loss of generality we assume that $A = \mathbf{Compl}(\underline{\mathcal{S}}_1^{A}\downarrow)$. Thus for any $a \in A$ there is $M_2^a \in Con_2$ such that $(M_2^a|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a}\downarrow} = M_1|_{\underline{\mathcal{S}}_1^{a}\downarrow}$.

We first prove that for all $a, a' \in A$ such that $a \neq a'$ and $\underline{\omega}(a) = \underline{\omega}(a')$, $M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow} = M_2^{a'}|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a')}\downarrow}$. In what follows such pairs of symbols are called non-injective pairs. The proof is by induction on the dependency bound of non-injective pairs of symbols in the dependency structure of a and a' (we notice that due to condition (4) of Def. 7.1, the dependency bound of both symbols in a non-injective pair is the same). In the base case let $a_1, a_2 \in A$ be a minimal non-injective pair such that $a_1 \leq a$, $a_2 \leq a'$. By minimal we mean that there are no non-injective pairs below it in the dependency structure, i.e., for all $b_1, b_2 \in A$ such that $b_1 < a_1$ and $b_2 < a_2$, if $\underline{\omega}(b_1) = \underline{\omega}(b_2)$ then $b_1 = b_2$. This means that $\underline{\mathcal{S}}_1^{a_1}\Downarrow = \underline{\mathcal{S}}_1^{a_2}\Downarrow$, because for all $b \in A$ such that $b < a_1$, we have $\underline{\omega}(b) < \underline{\omega}(a_2)$, therefore, by (4b) from Def. 7.1, there exists $b' < a_2$ such that $\underline{\omega}(b) = \underline{\omega}(b')$, thus, by above conditions, $b = b'$ and $b < a_2$. We have $(M_2^a|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a_1}\Downarrow} = (M_2^a|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a_2}\Downarrow} = (M_2^{a'}|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a_2}\Downarrow}$. By Lemma 7.20, we get $M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_1)}\Downarrow} = M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_2)}\Downarrow} = M_2^{a'}|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_2)}\Downarrow}$. Non-injectivity of $\underline{\omega}$ on a_1 and a_2, by (1) from Def. 7.1, implies that a_1 and a_2 are defined in $\underline{\mathcal{S}}_1$. Therefore, by Def. 6.5 from $M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_1)}\Downarrow} = M_2^{a'}|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_2)}\Downarrow}$ we get $M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_1)}\downarrow} = M_2^{a'}|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_2)}\downarrow}$. In the induction step, let us have an non-injective pair $a_1, a_2 \in A$ such that $a_1 \leq a$, $a_2 \leq a'$, we assume that for all non-injective pairs $a_1', a_2' \in A$ such that $a_1' < a_1$ and $a_2' < a_2$, it holds that $M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_1')}\downarrow} = M_2^{a'}|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_2')}\downarrow}$. For all $b \in A$ such that $b < a_1$ and $b < a_2$, we have $(M_2^a|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{b}\downarrow} = M_1|_{\underline{\mathcal{S}}_1^{b}\downarrow} = (M_2^{a'}|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{b}\downarrow}$,

thus, by Lemma 7.20, we get $M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(b)}\downarrow} = M_2^{a'}|_{\underline{\mathcal{S}}_2^{\underline{\omega}(b)}\downarrow}$. Altogether this means that for all $c \in \mathbf{Compl}(\underline{\mathcal{S}}_2^{\underline{\omega}(a_1)}\Downarrow)$ such that there is $b \in A$, $\underline{\omega}(b) = c$ we have $M_2^a|_{\underline{\mathcal{S}}_2^c\downarrow} = M^{a'_2}|_{\underline{\mathcal{S}}_2^c\downarrow}$. All other symbols in $\mathbf{Compl}(\underline{\mathcal{S}}_2^{\underline{\omega}(a_1)}\Downarrow)$ are defined (cf. (3) from Def. 7.1,), therefore, by repetitive use of Def. 6.5, we get $M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_1)}\downarrow} = M_2^{a'}|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a_2)}\downarrow}$ and finally $M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow} = M_2^{a'}|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a')}\downarrow}$, as required.

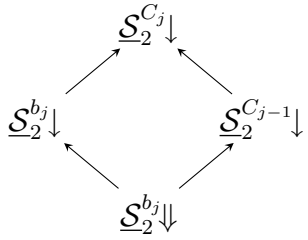
Let us now prove that $M_1|_{\underline{\mathcal{S}}_1^A\downarrow} \in (Con_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^A\downarrow}$. Let $n = db(\underline{\mathcal{S}}_1^A\downarrow)$, define, for $i \in \{0, \ldots, n\}$, $A_i^{=} = \{a \in A \mid db(\underline{\mathcal{S}}_1^a\downarrow) = i\}$ and $A_i = \bigcup_{0 \leq j \leq i} A_j^{=}$. By definition $A_0 = \emptyset$ and $A_n = A$. We prove by induction on i that there exists $M_2^i \in Con_2$ such that $(M_2^i|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{A_i}\downarrow} = M_1|_{\underline{\mathcal{S}}_1^{A_i}\downarrow}$.

In the base case, for $i = 0$, $A_0 = \emptyset$, let us take any $M_2 \in [\![\underline{\mathcal{S}}_2]\!]$ such that $M_1|_{\Sigma_\emptyset} = M_2|_{\Sigma_\emptyset}$ (cf. assumption (8) from Sect. 3.5), where signature $\Sigma_\emptyset = \mathbf{UnDep}(\mathbf{Compl}(\underline{\mathcal{S}}_1^\emptyset\downarrow)) = \mathbf{UnDep}(\mathbf{Compl}(\underline{\mathcal{S}}_2^\emptyset\downarrow))$ is the initial object of **Sig** (cf. Sect. 5.3); then from $Con_2 \models^c \underline{\omega}(\underline{SP})$ we get $M_1|_{\Sigma_\emptyset} = M_2|_{\Sigma_\emptyset} \in Con_2|_{\Sigma_\emptyset}$. By Def. 7.4 there exists $M_2^0 \in Con_2$ such that $(M_2^0|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{A_0}\downarrow} = M_2|_{\Sigma_\emptyset} = M_1|_{\Sigma_\emptyset} = M_1|_{\underline{\mathcal{S}}_1^{A_0}\downarrow}$.

In the induction step, we assume that for some $i \in \{0, \ldots, n-1\}$, there exists $M_2^i \in Con_2$ such that $(M_2^i|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{A_i}\downarrow} = M_1|_{\underline{\mathcal{S}}_1^{A_i}\downarrow}$. We show the same for $i+1$. By assumption that $\underline{\mathcal{S}}_1$ is finite we can assume that $A_{i+1}^{=}$ is finite, let $B_{i+1}^{=} = \underline{\omega}(A_{i+1}^{=})$ and let us name its elements, i.e. let $B_{i+1}^{=} = \{b_1, \ldots, b_k\}$. By induction on elements from $B_{i+1}^{=}$ we construct $M_2^{i+1} \in Con_2$ such that $(M_2^{i+1}|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{A_{i+1}}\downarrow} = M_1|_{\underline{\mathcal{S}}_1^{A_{i+1}}\downarrow}$.

For any $j \in \{1, \ldots, k\}$, let $a_j \in A_{i+1}^{=}$ be such that $\underline{\omega}(a_j) = b_j$, from assumption $M_1|_{\underline{\mathcal{S}}_1^{a_j}\downarrow} \in (Con_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a_j}\downarrow}$ there exists $M_2^{a_j} \in Con_2$ such that $(M_2^{a_j}|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a_j}\downarrow} = M_1|_{\underline{\mathcal{S}}_1^{a_j}\downarrow}$. By the inductive assumption, it holds that $(M_2^{a_j}|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a_j}\Downarrow} = M_1|_{\underline{\mathcal{S}}_1^{a_j}\Downarrow} = (M_2^i|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a_j}\Downarrow}$, because $\underline{\mathcal{S}}_1^{a_j}\Downarrow \subseteq A_i$. By Lemma 7.20 we get $M_2^{a_j}|_{\underline{\mathcal{S}}_2^{b_j}\Downarrow} = M_2^i|_{\underline{\mathcal{S}}_2^{b_j}\Downarrow}$. Notice that the choice of a_j is unimportant, because, as it was already proven above, for all $a, a' \in A$ such that $a \neq a'$ and $\underline{\omega}(a) = \underline{\omega}(a')$, $M_2^a|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow} = M_2^{a'}|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a')}\downarrow}$.

Consider a sequence of sets $\langle C_j \rangle_{j \in \{0,\ldots,k\}}$ defined inductively as $C_0 = \mathbf{Symb}(\mathbf{Compl}(\underline{\mathcal{S}}_2^{A_{i+1}}\Downarrow))$ and $C_{j+1} = C_j \cup \{b_j\}$. We have $\underline{\mathcal{S}}_2^{C_k}\downarrow = \underline{\mathcal{S}}_2^{\underline{\omega}(A_{i+1})}\downarrow$. For $j \in \{1, \ldots, k\}$, the following diagram is a pushout in $\mathbf{SigDep}^{frag}$



Let $N_2^0 \in [\![\underline{\mathcal{S}}_2]\!]$ be defined as $N_2^0 = M_2^i$. By induction we construct $N_2^j \in [\![\underline{\mathcal{S}}_2]\!]$, for $j \in \{1, \ldots, k\}$. Let $N'_j \in [\![\underline{\mathcal{S}}_2^{C_j\downarrow}]\!]$ be the amalgamation of $M_2^{a_j}|_{\underline{\mathcal{S}}_2^{b_j\downarrow}}$ and $N_2^{j-1}|_{\underline{\mathcal{S}}_2^{C_{j-1}\Downarrow}}$ w.r.t. the above pushout. Amalgamation is possible, because we showed above that $M_2^{a_j}|_{\underline{\mathcal{S}}_2^{b_j\Downarrow}} = M_2^i|_{\underline{\mathcal{S}}_2^{b_j\Downarrow}}$. By assumption (7) from Sect. 3.5, define N_2^j as any model from $[\![\underline{\mathcal{S}}_2]\!]$ such that $N_2^j|_{\underline{\mathcal{S}}_2^{C_j\downarrow}} = N'_j$.

Let $M'_2 = N_2^k$. By construction, for any $d \in \mathbf{Compl}(\underline{\mathcal{S}}_2)^{A_{i+1}\downarrow}$, we get $M'_2|_{\underline{\mathcal{S}}_2^{d\downarrow}} \in Con_2|_{\underline{\mathcal{S}}_2^{d\downarrow}}$. By assumption that Con_2 is well-grouped we have $M'_2|_{\underline{\mathcal{S}}_2^{\omega(A_{i+1})\downarrow}} \in Con_2|_{\underline{\mathcal{S}}_2^{\omega(A_{i+1})\downarrow}}$. Therefore, there exists $M_2^{i+1} \in Con_2$ such that $M_2^{i+1}|_{\underline{\mathcal{S}}_2^{\omega(A_{i+1})\downarrow}} = M'_2|_{\underline{\mathcal{S}}_2^{\omega(A_{i+1})\downarrow}}$. By construction $(M_2^{i+1}|_\omega)|_{\underline{\mathcal{S}}_1^{A_{i+1}\downarrow}} = M_1|_{\underline{\mathcal{S}}_1^{A_{i+1}\downarrow}}$, as required.

Taking the just proven fact for $i = n$, we obtain $M_2^n \in Con_2$ such that $(M_2^n|_\omega)|_{\underline{\mathcal{S}}_1^{A_n\downarrow}} = M_1|_{\underline{\mathcal{S}}_1^{A_n\downarrow}}$. Therefore, since $A_n = A$, we have $M_1|_{\underline{\mathcal{S}}_1^{A\downarrow}} \in (Con_2|_\omega)|_{\underline{\mathcal{S}}_1^{A\downarrow}}$, as required. □

Proof of Lemma 7.9. Let us show a useful lemma first.

Lemma 7.21 *Consider a construction signature $\underline{\mathcal{S}}$ and two construction specifications $\underline{SP}_1$ and $\underline{SP}_2$ over $\underline{\mathcal{S}}$ such that $\underline{SP}_1 \rightsquigarrow^c \underline{SP}_2$. Let there be a construction model $Con \models^c \underline{SP}_2$ and a set of symbols $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$. For all models $M \in Con$ such that $M|_{\underline{\mathcal{S}}^{A\downarrow}} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{A\downarrow}}$, it holds that $M|_{\underline{\mathcal{S}}^{A\downarrow}} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{A\downarrow}}$.*

Proof. By induction on $db(\underline{\mathcal{S}}^{A\downarrow})$. Let there be a construction signature $\underline{\mathcal{S}}$ and two construction specifications $\underline{SP}_1$ and $\underline{SP}_2$ over $\underline{\mathcal{S}}$ such that $\underline{SP}_1 \rightsquigarrow^c \underline{SP}_2$ and a construction model $Con \models^c \underline{SP}_2$.

In the base case, let A be such that $db(\underline{\mathcal{S}}^{A\downarrow}) = 0$. Notice that $\underline{\mathcal{S}}^{A\downarrow}$ is then the initial construction signature. Since $\underline{SP}_2$ is consistent (consequence of existence of $Con \models^c \underline{SP}_2$ and Lemma 6.19), there exists $M_2 \models \underline{SP}_2$, thus $M|_{\underline{\mathcal{S}}^{A\downarrow}} = M_2|_{\underline{\mathcal{S}}^{A\downarrow}} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{A\downarrow}}$.

In the induction step case, assume that the lemma holds for all A' such that $db(\underline{\mathcal{S}}^{A'}\downarrow) \leq i$. We prove the lemma for A such that $db(\underline{\mathcal{S}}^{A}\downarrow) \leq i+1$. For all A such that $db(\underline{\mathcal{S}}^{A}\downarrow) < i+1$ the lemma is already covered by the inductive assumption. Assume then that $db(\underline{\mathcal{S}}^{A}\downarrow) = i+1$. For all $a \in A$, from $M|_{\underline{\mathcal{S}}^{A}\downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{A}\downarrow}$ we get $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\downarrow}$ and $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\Downarrow}$, therefore, since $db(\underline{\mathcal{S}}^{a}\Downarrow) \leq i$, by the inductive assumption, we get $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\Downarrow}$. If a is a defined symbol, by condition (1) of Def. 6.13 for $Con \models^c \underline{SP}_2$, since $M \in Con$ and $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\Downarrow}$, we get $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\downarrow}$. Otherwise, if a is an assumed symbol, by condition (2) of Def. 7.7 for $\underline{SP}_1 \rightsquigarrow^c \underline{SP}_2$, since $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\Downarrow}$ and $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\downarrow}$, we also get $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\downarrow}$. This means that for all $a \in A$, $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\downarrow}$. By condition (4) of Def. 6.13 for $Con \models^c \underline{SP}_2$, since $M \in Con$ and for all $a \in A$, $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\downarrow}$, we get $M|_{\underline{\mathcal{S}}^{A}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{A}\downarrow}$, as required. □

Now, let us prove Lemma 7.9. Consider two construction specifications $\underline{SP}_1$ and $\underline{SP}_2$ over the same construction signature $\underline{\mathcal{S}}$ such that $\underline{SP}_1 \rightsquigarrow^c \underline{SP}_2$ (as in Def. 7.7). Let there be a construction model $Con \models^c \underline{SP}_2$, we prove that $Con \models^c \underline{SP}_1$ by showing all conditions from Def. 6.13.

To prove condition (1), consider a defined symbol $a \in \underline{\mathcal{S}}$ and a model $M \in Con$ such that $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\Downarrow}$. Let $A = \mathbf{Symb}(\mathbf{Compl}(\underline{\mathcal{S}}^{a}\Downarrow))$. By Lemma 7.21, from $M|_{\underline{\mathcal{S}}^{A}\downarrow} = M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\Downarrow} = \underline{SP}_1|_{\underline{\mathcal{S}}^{A}\downarrow}$ we get $M|_{\underline{\mathcal{S}}^{A}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{A}\downarrow}$, thus $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\Downarrow}$. By condition (1) of Def. 6.13 for $Con \models^c \underline{SP}_2$, since $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\Downarrow}$, we get $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\downarrow}$, therefore, $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\downarrow}$, as required.

To prove condition (2), consider an assumed symbol $a \in \mathbf{Compl}(\underline{\mathcal{S}})$, $a \notin \underline{\mathcal{S}}$ and a model $M \models \underline{SP}_1$ such that $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\Downarrow}$. This means that there exists $M' \in Con$ such that $M'|_{\underline{\mathcal{S}}^{a}\Downarrow} = M|_{\underline{\mathcal{S}}^{a}\Downarrow}$, thus $M'|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\Downarrow}$. Again let $A = \mathbf{Symb}(\mathbf{Compl}(\underline{\mathcal{S}}^{a}\Downarrow))$. By Lemma 7.21, from $M'|_{\underline{\mathcal{S}}^{A}\downarrow} = M'|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\Downarrow} = \underline{SP}_1|_{\underline{\mathcal{S}}^{A}\downarrow}$ we get $M'|_{\underline{\mathcal{S}}^{A}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{A}\downarrow}$, thus $M|_{\underline{\mathcal{S}}^{a}\Downarrow} = M'|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\Downarrow}$. By condition (2) of Def. 7.7 from $M|_{\underline{\mathcal{S}}^{a}\Downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\Downarrow}$ and $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\downarrow}$, and by the fact that a is an assumed symbol, we get $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\downarrow}$. This means that there exists $M_2 \models \underline{SP}_2$ such that $M_2|_{\underline{\mathcal{S}}^{a}\downarrow} = M|_{\underline{\mathcal{S}}^{a}\downarrow}$ and consequently $M_2|_{\underline{\mathcal{S}}^{a}\Downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\Downarrow}$. Therefore, since $Con \models^c \underline{SP}_2$, by condition

(2) of Def. 6.13, for a and M_2 such that $M_2|_{\underline{\mathcal{S}}^{a}\Downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\Downarrow}$, we obtain $M_2|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\downarrow}$. By $M_2|_{\underline{\mathcal{S}}^{a}\downarrow} = M|_{\underline{\mathcal{S}}^{a}\downarrow}$ we get $M|_{\underline{\mathcal{S}}^{a}\downarrow} \in Con|_{\underline{\mathcal{S}}^{a}\downarrow}$, as required.

Condition (3) for $Con \models^{c} \underline{SP}_1$ and $Con \models^{c} \underline{SP}_2$ is the same, thus since it holds for the latter, it also holds for the former.

As for condition (4), consider a set $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ and a model $M \in Con$ such that for all $a \in A$, $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\downarrow}$. By Lemma 7.21, for $A = \{a\}$, from $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{a}\downarrow}$ we have $M|_{\underline{\mathcal{S}}^{a}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{a}\downarrow}$. Thus, by condition (4) of Def. 6.13 for $Con \models^{c} \underline{SP}_2$, we get $M|_{\underline{\mathcal{S}}^{A}\downarrow} \models \underline{SP}_2|_{\underline{\mathcal{S}}^{A}\downarrow}$, thus, by condition (1) of Def. 7.7 , $M|_{\underline{\mathcal{S}}^{A}\downarrow} \models \underline{SP}_1|_{\underline{\mathcal{S}}^{A}\downarrow}$, as required. □

Proof of Theorem 7.14. Let us show a useful lemma before we proceed with the proof of the theorem.

Lemma 7.22 *Given a construction specification $\underline{SP}_1 = \langle \underline{\mathcal{S}}_1, SP_1 \rangle \in \mathbf{Spec}(\underline{\mathcal{S}}_1)$ and a conservative construction refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ w.r.t. SP_1 (cf. Def. 7.11), for any $M_1 \models SP_1$,*

1. *there exists $M_2 \models \underline{\omega}(\underline{SP}_1)$ such that $M_2|_{\underline{\omega}} = M_1$;*
2. *for any assumed symbol $a \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$, $a \notin \underline{\mathcal{S}}_1$, and a model $M_2 \models \underline{\omega}(SP_1)$ such that $(M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a}\Downarrow} = M_1|_{\underline{\mathcal{S}}_1^{a}\Downarrow}$, there exists a model $M_2' \models \underline{\omega}(SP_1)$ such that*
   (a) *$(M_2'|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a}\downarrow} = M_1|_{\underline{\mathcal{S}}_1^{a}\downarrow}$;*
   (b) *$M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow} = M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow}$.*

Proof. The proof is organized as follows. We start by proving the lemma with the assumption that $\underline{\omega}$ is injective on symbols. Later we generalise the result to all morphisms.

Consider a signature refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ such that $\underline{\omega}$ is injective on symbols, a construction specification $\underline{SP}_1 \in \mathbf{Spec}(\underline{\mathcal{S}}_1)$ and a model $M_1 \models SP_1$. Let us show the claims from Lemma 7.22.

Claim (1) is discharged directly by assumption (7) from Sect. 3.5.

To prove claim (2) consider an assumed symbol $a \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$, $a \notin \underline{\mathcal{S}}_1$ and a model $M_2 \models \underline{\omega}(SP_1)$ such that $(M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a}\Downarrow} = M_1|_{\underline{\mathcal{S}}_1^{a}\Downarrow}$. Consider

also a commuting diagram in $\mathbf{SigDep}^{ref}$ (later referred to as diagram (1)) being a part of a variant of the diagram from Def. 7.19 (unnamed arrows are inclusions).

$$\begin{array}{ccc} \underline{\mathcal{S}}_1^{a}\downarrow & \xrightarrow{\underline{\omega_a}} & \underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow \\ \uparrow & & \uparrow \\ \underline{\mathcal{S}}_1^{a}\Downarrow & \xrightarrow{\underline{\omega_a^-}} & \underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow \end{array}$$

The above square is a pushout in $\mathbf{SigDep}^{ref}$, because we assumed that $\underline{\omega}$ is injective on symbols and consequently $\underline{\omega_a}$ and $\underline{\omega_a^-}$ are also injective on symbols.

Define a sequence of sets of symbols by induction as $A_1 = \{b \in \mathbf{Compl}(\underline{\mathcal{S}}_1) \mid \text{for all } b' \in \mathbf{Compl}(\underline{\mathcal{S}}_1),\ b \not< b'\}$ and further $A_{i+1} = \{b \in \mathbf{Compl}(\underline{\mathcal{S}}_1) \mid b \notin \bigcup(A_j)_{1\leq j\leq i}$ and for all $b' \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$, if $b < b'$ then there exists $k \leq i$ such that $b' \in A_k\}$. It is easy to prove that for $i \in \{1, \ldots, db(\underline{\mathcal{S}}_1)\}$, A_i is a set of independent symbols. Moreover, for $i < db(\underline{\mathcal{S}}_1)$, $\underline{\mathcal{S}}_1^{A_i}\Downarrow = \underline{\mathcal{S}}_1^{A_{i+1}}\downarrow$. It also holds that $\bigcup A_i = \mathbf{Symb}(\underline{\mathcal{S}}_1)$ and $\underline{\mathcal{S}}_1^{A_1}\downarrow = \underline{\mathcal{S}}_1$. Obviously, there exists $i_a \in \{1, \ldots, db(\underline{\mathcal{S}}_1)\}$ such that $a \in A_{i_a}$. In order to make the notation more concise, we introduce another sequence of sets given as $A_i^a = \{a\} \cup A_i$, for $i \in \{1, \ldots, db(\underline{\mathcal{S}}_1)\}$ and $A_i = \{a\}$, for $i = db(\underline{\mathcal{S}}_1) + 1$. It is easy to prove that for all $1 \leq i \leq i_a$, $\underline{\mathcal{S}}_1^{A_i}\downarrow = \underline{\mathcal{S}}_1^{A_i^a}\downarrow$. As a consequence, we have $\underline{\mathcal{S}}_1^{A_1^a}\downarrow = \underline{\mathcal{S}}_1^{A_1}\downarrow = \underline{\mathcal{S}}_1$.

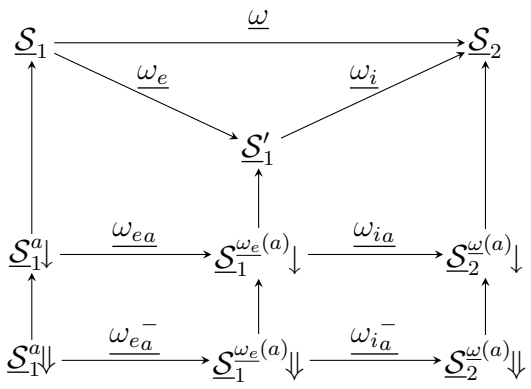
For any $i \in \{1, \ldots, db(\underline{\mathcal{S}}_1)\}$ consider the following commuting diagram in $\mathbf{SigDep}^{ref}$ ($\underline{\alpha_i}$, $\underline{\beta_i}$ and $\underline{\gamma_i}$ are inclusions), later referred to as diagram (2).

$$\begin{array}{ccccc} \underline{\mathcal{S}}_1^{A_i^a}\downarrow & & \xrightarrow{\underline{\omega_{A_i^a}}} & & \underline{\mathcal{S}}_2^{\underline{\omega}(A_i^a)}\downarrow \\ \uparrow \underline{\alpha_i} & & & & \uparrow \underline{\gamma_i} \\ \underline{\mathcal{S}}_1^{A_{i+1}^a}\downarrow & \xrightarrow{\underline{\omega_{A_{i+1}^a}}} & \underline{\mathcal{S}}_2^{\underline{\omega}(A_{i+1}^a)}\downarrow & \xrightarrow{\underline{\beta_i}} & \underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow \cup \underline{\mathcal{S}}_2^{\underline{\omega}(A_i)}\Downarrow \end{array}$$

The square of $\underline{\alpha_i}$, $\underline{\omega_{A_i^a}}$, $(\underline{\omega_{A_{i+1}^a}}; \underline{\beta_i})$, $\underline{\gamma_i}$ is a pushout in **Sig** (via coertion functor $\mathbf{UnDep}^{ref}; \mathbf{Compl}_{\mathbf{Sig}}$). It is, however, not always a pushout in $\mathbf{SigDep}^{ref}$ (the target of $\underline{\gamma_i}$ may include dependencies that are present neither in its source nor in the source of $\underline{\omega_{A_i^a}}$).

Let us construct a model $M_2' \models \underline{\omega}(SP_1)$ such that it meets the requirements from (2) of Lemma 7.22. We begin by using diagram (1), a pushout square, to amalgamate $M_1|_{\underline{\mathcal{S}}_1^a\downarrow}$ and $M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow}$. We name a resulting object $M_2^a \in [\![\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow]\!]$. For $i = db(\underline{\mathcal{S}}_1)$, $A_{i+1}^a = \{a\}$ and we have $\underline{\mathcal{S}}_1^{A_{i+1}^a}\downarrow = \underline{\mathcal{S}}_1^a\downarrow$ and $\underline{\mathcal{S}}_2^{\underline{\omega}(A_{i+1}^a)}\downarrow = \underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow$; we name $N_2^{i+1} = M_2^a$ and use diagram (2) to continue the construction. Morphism $\underline{\beta_1}$ is an inclusion, so it is injective on symbols, thus, by assumption (7) from Sect. 3.5 for $\underline{\beta_i}$ and N_2^{i+1}, there exists $M_2^{i+1} \in [\![\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\downarrow \cup \underline{\mathcal{S}}_2^{\underline{\omega}(A_i)}\Downarrow]\!]$ such that $M_2^{i+1}|_{\underline{\beta_i}} = N_2^{i+1}$. Diagram (2) is a pushout in **Sig**, so we amalgamate $M_1|_{\underline{\mathcal{S}}_1^{A_i^a}\downarrow}$ with M_2^{i+1} and obtain a model $N_2^i \in [\![\underline{\mathcal{S}}_2^{\underline{\omega}(A_i^a)}\downarrow]\!]$. By repeating the above described procedure $db(\underline{\mathcal{S}}_1) - 1$ times we get to $i = 1$ and we have $N_2^1 \in [\![\underline{\mathcal{S}}_2^{\underline{\omega}(A_1^a)}\downarrow]\!]$ such that $N_2^1|_{\underline{\omega_{A_i^a}}} = M_1|_{\underline{\mathcal{S}}_1^{A_1^a}\downarrow} = M_1|_{\underline{\mathcal{S}}_1} = M_1$. Let $\underline{\delta}\colon \underline{\mathcal{S}}_2^{\underline{\omega}(A_1^a)}\downarrow \to \underline{\mathcal{S}}_2$ be the inclusion. It holds that $\underline{\omega} = (\underline{\omega_{A_1^a}}); \underline{\delta}$. By assumption (7) from Sect. 3.5 for $\underline{\delta}$ and N_2^1, there exists $M_2' \in [\![\underline{\mathcal{S}}_2]\!]$ such that $M_2'|_{\underline{\delta}} = N_2^1$. This means that $M_2'|_{\underline{\omega}} = M_2'|_{\underline{(\omega_{A_1^a});\delta}} = N_2^1|_{\underline{\omega_{A_1^a}}} = M_1$, therefore, since $M_1 \models \underline{SP}_1$, we have $M_2' \models \underline{\omega}(SP_1)$. By construction of M_2' we have $(M_2'|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^a\downarrow} = M_1|_{\underline{\mathcal{S}}_1^a\downarrow}$ and $M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow} = M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow}$, as required.

Let us now generalise the above proof to any $\underline{\omega}$, without the injectivity assumption. Let us factorise $\underline{\omega}$ into the abstract surjection $\underline{\omega_e}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_1'$ and the inclusion $\underline{\omega_i}\colon \underline{\mathcal{S}}_1' \to \underline{\mathcal{S}}_2$. Assumption (3) from Sect. 3.5 implies that abstract surjections in **Sig** are surjective on their symbols and inclusions are injective on their symbols, thus $\underline{\omega_e}$ is surjective and $\underline{\omega_i}$ is injective. Consider the following diagram in $\mathbf{SigDep}^{ref}$.



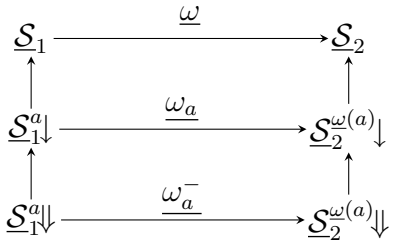
Let there be a model $M_1 \in [\![SP_1]\!]$. To show claim (1) of Lemma 7.22, from the assumption that $\underline{\omega}$ is a conservative refinement construction morphisms w.r.t. $\underline{SP}_1$, by Def. 7.11, we get $M_1 \in [\![(\underline{\omega}(SP_1))|_{\underline{\omega}}]\!]$, so there exists $M_2 \in [\![\underline{\omega}(SP_1)]\!]$ such that $M_2|_{\underline{\omega}} = M_1$, as required.

Regarding claim (2), consider an assumed symbol $a \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$, $a \notin \underline{\mathcal{S}}_1$ and a model $M_2 \models \underline{\omega}(SP_1)$ such that $(M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a\Downarrow}} = M_1|_{\underline{\mathcal{S}}_1^{a\Downarrow}}$. From claim (1) proven above, we get existence of $N_2 \in [\![\underline{\omega}(SP_1)]\!]$ (we name it N_2 to avoid confusion) such that $N_2|_{\underline{\omega}} = M_1$. Let us name $M_1' = N_2|_{\underline{\omega_i}}$. We have $(M_1'|_{\underline{\omega_e}})|_{\underline{\mathcal{S}}_1^{a\Downarrow}} = (N_2|_{\omega})|_{\underline{\mathcal{S}}_1^{a\Downarrow}} = M_1|_{\underline{\mathcal{S}}_1^{a\Downarrow}} = (M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a\Downarrow}}$. By assumption (3) from Sect. 3.5, by surjectivity of $(\underline{\omega_e})_a^-$ and by commutativity of the above diagram, from $(M_1'|_{\underline{\omega_e}})|_{\underline{\mathcal{S}}_1^{a\Downarrow}} = (M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a\Downarrow}}$ we get $M_1'|_{\underline{\mathcal{S}}_1^{\underline{\omega_e}(a)}\Downarrow} = (M_2|_{\underline{\omega_i}})|_{\underline{\mathcal{S}}_1^{\underline{\omega_e}(a)}\Downarrow}$. Morphism $\underline{\omega_i}$ is injective on symbols, thus, by Lemma 7.12 it is a conservative construction refinement morphism w.r.t. $\underline{\omega_e}(\underline{SP})$; therefore, we can use above proven Lemma 7.22 for injective $\underline{\omega_i}\colon \underline{\mathcal{S}}_1' \to \underline{\mathcal{S}}_2$. For the model $M_1' \models \underline{\omega_e}(SP_1)$, the assumed symbol $\underline{\omega_e}(a) \in \mathbf{Compl}(\underline{\mathcal{S}}_1')$, and the model $M_2 \models \underline{\omega_i}(\underline{\omega_e}(SP_1))$ such that $M_1'|_{\underline{\mathcal{S}}_1^{\underline{\omega_e}(a)}\Downarrow} = (M_2|_{\underline{\omega_i}})|_{\underline{\mathcal{S}}_1^{\underline{\omega_e}(a)}\Downarrow}$ we get existence of $M_2' \models \underline{\omega_i}(\underline{\omega_e}(SP_1))$ such that $(M_2'|_{\underline{\omega_i}})|_{\underline{\mathcal{S}}_1^{\underline{\omega_e}(a)}\downarrow} = M_1'|_{\underline{\mathcal{S}}_1^{\underline{\omega_e}(a)}\downarrow}$ and $M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow} = M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}\Downarrow}$. This means that $((M_2'|_{\underline{\omega_i}})|_{\underline{\mathcal{S}}_1^{\underline{\omega_e}(a)}\downarrow})|_{\underline{\omega_e}_a} = (M_1'|_{\underline{\mathcal{S}}_1^{\underline{\omega_e}(a)}\downarrow})|_{\underline{\omega_e}_a}$, thus, by commutativity of the above diagram, we get $(M_2'|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^{a}\downarrow} = (M_1'|_{\underline{\omega_e}})|_{\underline{\mathcal{S}}_1^{a}\downarrow} = M_1|_{\underline{\mathcal{S}}_1^{a}\downarrow}$, as required.

□

Now, let us prove Theorem 7.14. Consider a signature refinement morphism $\underline{\omega}\colon \underline{\mathcal{S}}_1 \to \underline{\mathcal{S}}_2$ such that $\underline{\mathcal{S}}_1$ is finite and a construction specification $\underline{SP} \in \mathbf{Spec}(\underline{\mathcal{S}}_1)$ such that $\underline{\omega}$ is a conservative construction refinement morphism w.r.t. $\underline{SP}$. Consider a construction model $Con_2 \in [\![\underline{\mathcal{S}}_2]\!]^c$ such that $Con_2 = \mathbf{Clean}_{\underline{\omega}(\underline{SP})}(Con_2)$.

Assume that $Con_2 \models^c \underline{\omega}(\underline{SP})$. To prove that $Con_2|_{\underline{\omega}} \models^c \underline{SP}$ we need to show the four conditions from Def. 6.13. Let $a \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$. Consider a variant of the commuting diagram in $\mathbf{SigDep}^{ref}$ from Def. 7.19.



Regarding condition (1) of Def. 6.13, let $a \in \underline{\mathcal{S}}_1$ and $M_1 \in Con_2|_{\underline{\omega}}$ are such that $M_1|_{\underline{\mathcal{S}}_1^a{\Downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}_1^a{\Downarrow}}$. By Def. 7.4 there exists $M_2 \in Con_2$ such that $M_2|_{\underline{\omega}} = M_1$ and, by assumption that $Con_2 = \mathbf{Clean}_{\underline{\omega}(\underline{SP})}(Con_2)$, $M_2 \models \underline{\omega}(\underline{SP})$, thus $M_1 \models \underline{SP}$, so $M_1|_{\underline{\mathcal{S}}_1^a{\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}_1^a{\downarrow}}$, as required.

To prove condition (2), consider $a \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$ such that $a \notin \underline{\mathcal{S}}_1$ and model $M_1 \models \underline{SP}$ such that $M_1|_{\underline{\mathcal{S}}_1^a{\Downarrow}} \in (Con_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}^a{\Downarrow}}$. By Def. 7.4 there exists $M_2 \in Con_2$ such that $(M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^a{\Downarrow}} = M_1|_{\underline{\mathcal{S}}_1^a{\Downarrow}}$. By assumption about cleanness of Con_2 we know that $M_2 \models \underline{\omega}(\underline{SP})$, therefore, by assumption that $\underline{\omega}$ is a conservative construction refinement morphism w.r.t. $\underline{SP}$, and by Lemma 7.22, there exists $M_2' \models \underline{\omega}(\underline{SP})$ such that $(M_2'|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^a{\downarrow}} = M_1|_{\underline{\mathcal{S}}_1^a{\downarrow}}$ and $M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}{\Downarrow}} = M_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}{\Downarrow}}$; hence $M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}{\Downarrow}} \in Con_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}{\Downarrow}}$. By condition (2) of Def. 6.13 for $Con_2 \models^c \underline{\omega}(\underline{SP})$, for $\underline{\omega}(a) \in \mathbf{Compl}(\underline{\mathcal{S}}_2)$, which by Def. 7.1 is an assumed symbol, from $M_2'|_{\underline{\mathcal{S}}_1^{\underline{\omega}(a)}{\Downarrow}} \in Con_2|_{\underline{\mathcal{S}}_1^{\underline{\omega}(a)}{\Downarrow}}$ we get $M_2'|_{\underline{\mathcal{S}}_1^{\underline{\omega}(a)}{\downarrow}} \in Con_2|_{\underline{\mathcal{S}}_1^{\underline{\omega}(a)}{\downarrow}}$. From the commutativity of the above diagram we obtain as required:

$$M_1|_{\underline{\mathcal{S}}_1^a{\downarrow}} = (M_2'|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}{\downarrow}})|_{\underline{\omega}_a} \in (Con_2|_{\underline{\mathcal{S}}_2^{\underline{\omega}(a)}{\downarrow}})|_{\underline{\omega}_a} = (Con_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^a{\downarrow}}$$

Condition (3) is discharged directly by Lemma 7.6.

As for condition (4), let there be a set $A \subseteq \mathbf{Compl}(\underline{\mathcal{S}})$ and a model $M_1 \in Con_2|_{\underline{\omega}}$ such that for all $a \in A$, $M_1|_{\underline{\mathcal{S}}^a{\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}_1^a{\downarrow}}$. By Def. 7.4 there exists $M_2 \in Con_2$ such that $M_2|_{\underline{\omega}} = M_1$ and, by the assumption that $Con_2 = \mathbf{Clean}_{\underline{\omega}(\underline{SP})}(Con_2)$, we have $M_2 \models \underline{\omega}(\underline{SP})$, therefore

$$(M_2|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^A{\downarrow}} \models (\underline{\omega}(\underline{SP})|_{\underline{\omega}})|_{\underline{\mathcal{S}}_1^A{\downarrow}}$$

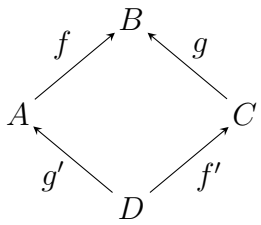
As a consequence, we get $M_1|_{\underline{\mathcal{S}}_1^A{\downarrow}} \models \underline{SP}|_{\underline{\mathcal{S}}_1^A{\downarrow}}$, as required. □

Proof of Theorem 7.17. Pushouts in **SigDep** are also pushouts in **Sig**. Similarly, by definition of fragments (cf. Def. 4.1), pushouts in $\mathbf{SigDep}^{frag}$ are also pushouts in $\mathbf{Sig}^{frag}$. Therefore, since $\underline{\varphi_1}$, $\underline{\varphi_2}$, $\underline{\beta_1}$ and $\underline{\beta_2}$ form a pushout in $\mathbf{SigDep}^{frag}$, they also do so in $\mathbf{Sig}^{frag}$. We have $\underline{\varphi_1};\underline{\omega_1};\underline{\beta'_1} = \underline{\varphi_2};\underline{\omega_2};\underline{\beta'_2}$ in $\mathbf{Sig}^{frag}$, thus there exists the unique universal $\mathbf{Sig}^{frag}$-morphism $\underline{\omega}\colon \underline{\mathcal{S}} \to \underline{\mathcal{S}'}$ such that the whole diagram from the theorem commutes in $\mathbf{Sig}^{frag}$. Let us now prove that $\underline{\omega}$ is a $\mathbf{SigDep}^{ref}$-morphism.

We notice that $\underline{\mathcal{F}'}$, as a source of the construction fitting span, is required to be an empty fragment, i.e. containing only assumed elements, therefore, the construction refinement morphism ω_F is a bijection. This is a consequence of requirements (1), (2) and (3) of Def. 7.1.

The following lemma will be used in the rest of the proof of Theorem 7.17.

Lemma 7.23 *Given four functions, $f\colon A \to B$, $g\colon C \to B$, $f'\colon D \to C$, $g'\colon D \to A$, such that $f';g = g';f$ and f, g are the pushout of f', g' in* **Set**,



for any $a_1 \in A$ define an undirected graph $\langle V, E\rangle$ as $V = D' \uplus f'(D') \uplus g'(D')$, $E = f'|_{D'} \uplus g'|_{D'}$, where $D' \subseteq D$ is given as $D' = (f';g)^{-1}(\{f(a_1)\})$,

1. *for any $a_2 \in A$ such that $a_1 \neq a_2$,*
   $f(a_1) = f(a_2)$ iff
   $a_1 \in img(g')$ and there exists a path from a_1 to a_2 in $\langle V, E\rangle$;
2. *for any $c_2 \in C$,*
   $f(a_1) = g(c_2)$ iff
   $a_1 \in img(g')$ and there exists a path from a_1 to c_2 in $\langle V, E\rangle$.

Proof. Obvious by the construction of pushouts in **Set**. □

Let us check that $\underline{\omega}$ satisfies all conditions from Def. 7.1.

Conditions (1), (2) and (3) are met only if $\underline{\omega}$ is bijective on assumed elements. It is enough to prove that $\underline{\omega}$ is both injective and surjective on assumed elements.

Regarding its injectivity on assumed symbols, let us have assumed $a, b \in \mathbf{Compl}(\underline{\mathcal{S}})$ such that $a \neq b$. By contradiction, let us assume that $\underline{\omega}(a) = \underline{\omega}(b)$. Without loss of generality there exists an assumed $a_1 \in \mathbf{Compl}(\underline{\mathcal{S}_1})$ such that $\underline{\beta_1}(a_1) = a$ and of course $\underline{\beta_1}(a_1) \neq b$. Now, either there exists an assumed $b_1 \in \mathbf{Compl}(\underline{\mathcal{S}_1})$ such that $\underline{\beta_1}(b_1) = b$ (first case) or there exists an assumed $b_2 \in \mathbf{Compl}(\underline{\mathcal{S}_2})$ such that $\underline{\beta_2}(b_2) = b$ (second case). Let us name $a'_1 = \underline{\omega_1}(a_1)$ and $b'_1 = \underline{\omega_1}(b_1)$ and $b'_2 = \underline{\omega_2}(b_2)$. All a'_1, b'_1 and b'_2 are assumed elements, because $\underline{\omega_1}$ and $\underline{\omega_2}$ are bijective on assumed elements. In the first case, $a'_1 \neq b'_1$ and $\underline{\beta'_1}(a'_1) = \underline{\beta'_1}(b'_1)$. In the second case, $\underline{\beta'_1}(a'_1) = \underline{\beta'_2}(b'_1)$. By Lemma 7.23 both cases yield the existence of $a'_f, b'_f \in \mathbf{Compl}(\underline{\mathcal{F}})$ such that $\underline{\varphi'_1}(a'_f) = a'_1$ and $\underline{\varphi'_1}(b'_f) = b'_1$, in the first case, and $\underline{\varphi'_2}(b'_f) = b'_2$, in the second case. Moreover, by Lemma 7.23, there exists a path in the graph (for $\langle \underline{\varphi'_1}, \underline{\varphi'_2} \rangle$) between a'_1 and b'_1 or b'_2, in respective cases. Since $\underline{\omega_F}$ is a bijection, there exist $a_f, b_f \in \mathbf{Compl}(\underline{\mathcal{F}})$ such that $a_f = \underline{\omega_F^{-1}}(a'_f)$, $b_f = \underline{\omega_F^{-1}}(b'_f)$. Of course $b_1 = \underline{\varphi_1}(b_f)$ and $b_2 = \underline{\varphi_2}(b_f)$, in respective cases. Moreover, since both $\underline{\omega_1}$ and $\underline{\omega_2}$ are bijective on assumed elements, there must exist a path in the graph (for $\langle \underline{\varphi_1}, \underline{\varphi_2} \rangle$) between a_1 and b_1 or b_2, in respective cases. Therefore, by Lemma 7.23, in both cases we get $a = b$, contradiction.

Regarding the surjectivity of $\underline{\omega}$ on assumed symbols, let there be an assumed $b' \in \mathbf{Compl}(\underline{\mathcal{S}'})$. Without loss of generality we have an assumed $b'_1 \in \mathbf{Compl}(\underline{\mathcal{S}'_1})$ and, by bijectivity of $\underline{\omega_1}$ on assumed symbols, there is an assumed $b_1 \in \mathbf{Compl}(\underline{\mathcal{S}_1})$. Since the diagram commutes, we get $\underline{\omega}(\underline{\beta_1}(b_1)) = b'$.

The condition (4) is to prove that $\underline{\omega}$ is monotone and that it weakly reflects dependency within its range.

To prove monotonicity let us have $a < b \in \mathbf{Compl}(\underline{\mathcal{S}})$. Without loss of generality we can assume that there exist $a_1 < b_1 \in \mathbf{Compl}(\underline{\mathcal{S}_1})$ such that $\underline{\beta_1}(a_1) = a$ and $\underline{\beta_1}(b_1) = b$, because $\underline{\beta_1}$ is a p-morphism. Both $\underline{\omega_1}$ and $\underline{\beta'_1}$ are monotone and the diagram is commutative in $\mathbf{Sig}^{frag}$, thus

$$\underline{\omega}(a) = \underline{\beta'_1}(\underline{\omega_1}(a_1)) < \underline{\beta'_1}(\underline{\omega_1}(b_1)) = \underline{\omega}(b)$$

As for weakly reflected dependency within the range of $\underline{\omega}$, let $a, b \in \mathbf{Compl}(\underline{\mathcal{S}})$ be such that $\underline{\omega}(a) < \underline{\omega}(b)$. Without loss of generality we can assume that there exists $b_1 \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$ such that $\underline{\beta_1}(b_1) = b$. Let us name $b'_1 = \omega_1(b_1)$. There exists $a'_1 \in \mathbf{Compl}(\underline{\mathcal{S}}'_1)$ such that $a'_1 < b'_1$ and $\underline{\beta'_1}(a'_1) = \underline{\omega}(a)$, because $\underline{\beta'_1}$ is a p-morphism and $\underline{\omega}(a) < \underline{\omega}(b) = \underline{\beta'_1}(b'_1)$. Let us now prove that $a'_1 \in img(\underline{\omega_1})$. Since pushouts are jointly-epi, we know that there exists $a_1 \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$ such that $\underline{\beta_1}(a_1) = a$ or there exists $a_2 \in \mathbf{Compl}(\underline{\mathcal{S}}_2)$ such that $\underline{\beta_2}(a_1) = a$. In the first case, when it also happens that $\underline{\omega_1}(a_1) = a'_1$, trivially we have $a'_1 \in img(\underline{\omega_1})$. Otherwise, Lemma 7.23 (with assumption that **SetSymb** preserves pushouts, cf. (2) from Sect. 3.5) applied to the first case when $\underline{\omega_1}(a_1) \neq a'_1$ and to the second case yields the existence of $a'_f \in \mathbf{Compl}(\underline{\mathcal{F}'})$ such that $\underline{\varphi'_1}(a'_f) = a'_1$. Using the bijectivity of $\underline{\omega_F}$, let us name $a_f = \underline{\omega}_F^{-1}(a'_f)$. By commutativity of the diagram, we have $\underline{\omega_1}(\underline{\varphi_1}(a_f)) = a'_1$. It proves that $a'_1 \in img(\underline{\omega_1})$ in all cases. By condition (4) of Def. 7.1 for $\underline{\omega_1}$ we get existence of $a_3 \in \mathbf{Compl}(\underline{\mathcal{S}}_1)$ such that $a_3 < b_1$ and $\underline{\omega_1}(a_3) = a'_1$. This means that $\underline{\omega}(\underline{\beta_1}(a_3)) = \underline{\omega}(a)$ and, by monotonicity of $\underline{\beta_1}$, we have

$$\underline{\beta_1}(a_3) < \underline{\beta_1}(b_1) = b$$

as required. □

Chapter 8

Architectures as Diagrams of Constructions

8.1 Introduction

Constructions, construction fittings, sums and refinements, as so far presented in the previous chapters, are all ingredients needed to represent a software architecture as a *diagram of constructions*. Such diagrams, present in some form in most approaches to software modularisation (cf. Chapter 2), give architectural system decomposition. In our case they capture a top-down development process. Interestingly, our approach, in contrast to most other frameworks, provides uniform representation of non-parameterised and parameterised modules of any order in the diagram.

8.2 Diagrams of Constructions

Before we formally define diagrams of constructions, we introduce a category suitable to uniformly represent different kind of morphisms between construction signatures.

Definition 8.1 (Category of Construction Signatures) *Let the category* $\mathbf{SigDep}^{con}$ *have objects of* $\mathbf{SigDep}^{ref}$ *(and* $\mathbf{SigDep}^{frag}$*, since both have the same classes of objects) as its objects and morphisms of* $\mathbf{Sig}^{frag}$ *as its morphisms.*

The $\mathbf{SigDep}^{con}$-objects are constructions signatures, the morphisms are just $\mathbf{Sig}^{frag}$-morphisms, disregarding any conditions related to the depen-

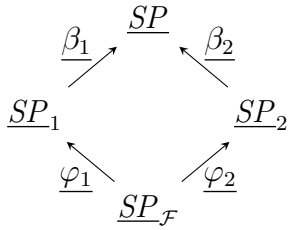
dency structure. Therefore, both $\mathbf{SigDep}^{frag}$ and $\mathbf{SigDep}^{con}$ are subcategories of $\mathbf{SigDep}^{con}$ such that $|\mathbf{SigDep}^{con}| = |\mathbf{SigDep}^{frag}| = |\mathbf{SigDep}^{ref}|$.

Definition 8.2 (Category of Construction Specifications) *The category* $\mathbf{SpecDep}^{con}$ *has as objects construction specifications, i.e. pairs* $\underline{SP} = \langle \underline{\mathcal{S}}, SP \rangle$ *(cf. Def. 6.12).* $\mathbf{SpecDep}^{con}$*-morphisms* $\underline{\sigma} \colon \langle \underline{\mathcal{S}}_1, SP_1 \rangle \to \langle \underline{\mathcal{S}}_2, SP_2 \rangle$ *are* $\mathbf{SigDep}^{con}$*-morphisms such that they are refinements* $SP_1 \overset{\mathbf{Compl}(\underline{\sigma})}{\rightsquigarrow} SP_2$ *in the base institution* $\mathbb{I}$*, i.e.* $[\![SP_2|_{\mathbf{Compl}(\underline{\sigma})}]\!] \subseteq [\![SP_1]\!]$.

Notation. Given a $\mathbf{SpecDep}^{con}$-object $\underline{SP} = \langle \underline{\mathcal{S}}, SP \rangle$, we use projection functions to get its components, $\pi_1(\underline{SP}) = \underline{\mathcal{S}}$ and $\pi_2(\underline{SP}) = SP$.

Definition 8.3 (Diagrams of Constructions) A diagram of construction specifications *(or shortly* a diagram of constructions*)* $\mathfrak{D} \colon \mathbf{J} \to \mathbf{SpecDep}^{con}$*, is a diagram in* $\mathbf{SpecDep}^{con}$ *such that:*

1. *every node in the diagram is a construction specification over a finite construction signature;*
2. *every arrow in the diagram belongs to exactly one of the following sets of arrows in the diagram:*
   (a) refinement arrows *that are* $\mathbf{SigDep}^{ref}$*-morphisms,*
   (b) composition arrows *that are* $\mathbf{SigDep}^{frag}$*-morphisms;*
3. *a* sum square *in the diagram is a pushout of a construction fitting* $\langle \underline{\varphi_1}, \underline{\varphi_2} \rangle$ *in* $\mathbf{SigDep}^{frag}$ *(cf. Def. 6.28)*



consisting of the bottom node $\underline{SP}_{\mathcal{F}} = \langle \underline{\mathcal{F}}, \emptyset \rangle$*, the* side nodes $\underline{SP}_1$, $\underline{SP}_2$ *and the* top node $\underline{SP}$ *such that:*

   (a) *the bottom node* $\underline{SP}_{\mathcal{F}}$ *is an empty construction specification over an empty signature fragment (cf. Def. 6.28);*

(b) *the side nodes, i.e. construction specifications* $\underline{SP}_1$ *and* $\underline{SP}_2$ *over* $\underline{\mathcal{S}}_1$ *and* $\underline{\mathcal{S}}_2$*, respectively, are compatible construction specifications w.r.t. the fitting* $\langle \underline{\varphi}_1, \underline{\varphi}_2 \rangle$ *(cf. Def. 6.34);*

(c) *the top node* $\underline{SP}$ *is given as a sum* $\underline{SP} = \underline{\beta}_1(\underline{SP}_1) \cup \underline{\beta}_2(\underline{SP}_2)$*;*

4. *every composition arrow belongs to some sum square;*

5. *no two sum squares have common nodes;*

6. *every refinement arrow* $\underline{\omega}\colon \underline{SP}_1 \to \underline{SP}_2$ *is a construction specification refinement* $\underline{SP}_1 \overset{\underline{\omega}}{\rightsquigarrow}{}^{c} \underline{SP}_2$ *(cf. Def. 7.15);*

7. *the source of every refinement arrow is:*

   (a) *a single node (not belonging to any sum square) or*

   (b) *a side node of a sum square;*

8. *the target of every refinement arrow is:*

   (a) *a single node (not belonging to any sum square) or*

   (b) *a top node of a sum square;*

9. *every node is the source of at most one refinement arrow;*

10. *the diagram seen as a directed graph with refinement arrows (cf. condition (2) above) as edges in the same direction and the composition arrows (cf. condition (2) above) as edges in the opposite direction meets the following conditions:*

    (a) *the graph is a dag (directed acyclic graph);*

    (b) *the graph is connected;*

    (c) *there is exactly one node, called the* result node*, such that it is not the target of any edge in the graph.*

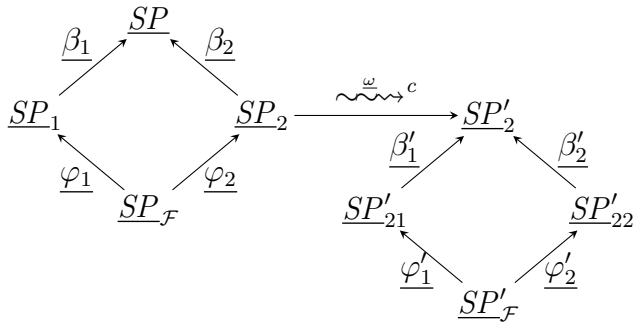
Example 8.4 *Consider the following construction signatures*

$$\begin{aligned}
\underline{\mathcal{S}} &= (\textbf{sorts } s;\ \textbf{ops } a : s,\ c : s;\ a < c),\\
\underline{\mathcal{S}}_1 &= (\textbf{sorts } \underline{s};\ \textbf{ops } \underline{a : s},\ c : s;\ a < c),\\
\underline{\mathcal{S}}_2 &= (\textbf{sorts } s;\ \textbf{ops } a : s),\\
\underline{\mathcal{F}} &= (\textbf{sorts } \underline{s};\ \textbf{op } \underline{a : s}),\\
\underline{\mathcal{S}}'_2 &= (\textbf{sorts } s,t;\ \textbf{ops } a : s,\ b : t,\ f : t \to s;\ \textbf{deps } b < f,\ f < s),\\
\underline{\mathcal{S}}'_{21} &= (\textbf{sorts } \underline{s},\underline{t};\ \textbf{ops } \underline{a : s},\ \underline{b : t},\ \underline{f : t \to s};\ \textbf{deps } b < f,\ f < s),\\
\underline{\mathcal{S}}'_{22} &= (\textbf{sorts } s,t;\ \textbf{ops } a : s,\ b : t,\ f : t \to s;\ \textbf{deps } b < f,\ f < s),\\
\underline{\mathcal{F}}' &= (\textbf{sorts } \underline{s},\underline{t};\ \textbf{op } \underline{a : s},\ \underline{b : t},\ \underline{f : t \to s};\ \textbf{deps } b < f,\ f < s).
\end{aligned}$$

and the following construction specifications

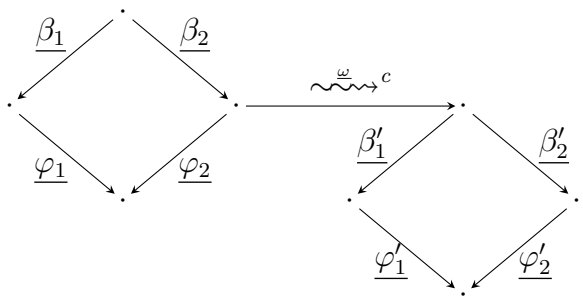
$$\begin{aligned}
\underline{SP} &= \langle \underline{\mathcal{S}}, \{(\forall x \cdot x = a), (a = c)\}\rangle, & \underline{SP}_1 &= \langle \underline{\mathcal{S}}_1, \{c = a\}\rangle,\\
\underline{SP}_2 &= \langle \underline{\mathcal{S}}_2, \{\forall x \cdot x = a\}\rangle, & \underline{SP}'_2 &= \langle \underline{\mathcal{S}}'_2, \{(\forall x \cdot x = a), (f(b) = a)\}\rangle,\\
\underline{SP}'_{21} &= \langle \underline{\mathcal{S}}'_{21}, \{f(b) = a\}\rangle, & \underline{SP}'_{22} &= \langle \underline{\mathcal{S}}'_{22}, \{(\forall x \cdot x = a), (f(b) = a)\}\rangle,\\
\underline{SP}_{\mathcal{F}} &= \langle \underline{\mathcal{F}}, \emptyset\rangle, & \underline{SP}'_{\mathcal{F}} &= \langle \underline{\mathcal{F}}', \emptyset\rangle.
\end{aligned}$$

The following diagram $\mathfrak{D}$ is a diagram of constructions.



where all morphisms are inclusions. In fact, $\underline{\omega}$ is a refinement arrow and all other morphisms are composition arrows; $\langle \underline{\varphi_1}, \underline{\varphi_2}\rangle$ and $\langle \underline{\varphi'_1}, \underline{\varphi'_2}\rangle$ are construction fittings; SP_1 is compatible with SP_2 w.r.t. $\langle \underline{\varphi_1}, \underline{\varphi_2}\rangle$ and SP'_{21} is compatible with SP'_{22} w.r.t. $\langle \underline{\varphi'_1}, \underline{\varphi'_2}\rangle$, thus the two squares on the diagram are the sum squares. The source of the only refinement arrow $\underline{\omega}$ is the side node

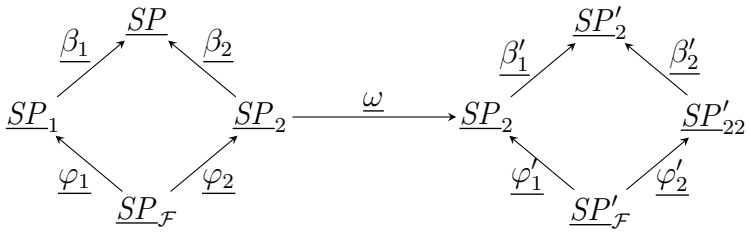
of the left sum square; its target is the top node of the right sum square. The diagram seen as a directed graph (as described in (10) of Def. 8.3) indeed meets all conditions listed in (10) of Def. 8.3.



Example 8.4 depicts the typical situation where a construction specification $\underline{SP}$ is decomposed into a number of components (two in this case) and they (only $\underline{SP_2}$ in this case) are further refined via construction specification refinements (only $\underline{\omega}$ in this case) and the refined signature ($\underline{\mathcal{S}'_2}$ in this case) contains new implementation-specific symbols (sort t and operations $b : t$ and $f : t \to s$ in this case). Additionally the refined specification ($\underline{SP'_2}$) may be stronger than source one ($\underline{SP_2}$) on defined symbols.

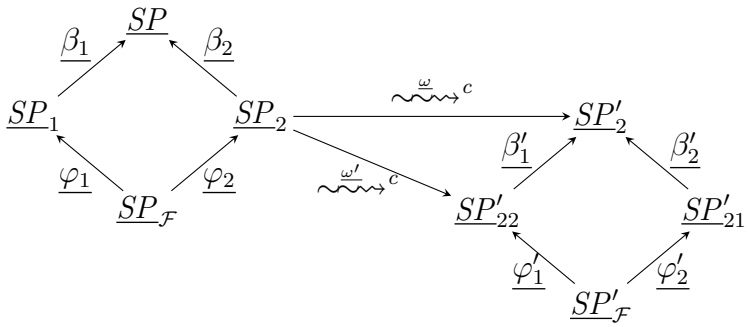
Example 8.5 *(Using the notation from Example 8.4) Let us present some non-examples of diagrams of constructions.*

1. *The following diagram is not a construction diagram, because there are two result nodes, $\underline{SP}$ and $\underline{SP'_2}$. Morphism $\underline{\omega}$ is not marked as a refinement morphism ($\overset{\underline{\omega}}{\rightsquigarrow}{}^{c}$). Even if $\underline{\omega}$ was a refinement morphism, its target is neither a top node nor a single node.*

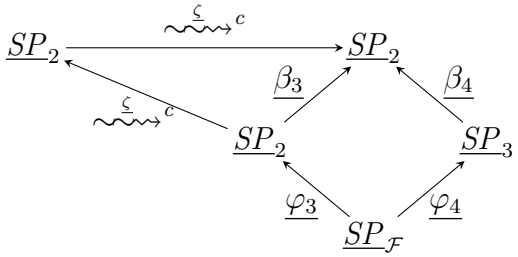


2. *There are two problems with the diagram below. The first is that the target of refinement arrow $\underline{\omega'}$ is a side node of a sum square (not a top*

node). The second is that a node $\underline{SP}_2$ *is the source of two refinement arrows.*



3. *Let there be a construction specification* $\underline{SP}_3 = \langle \underline{\mathcal{F}}, \{true\} \rangle$*. The diagram given below (with all morphisms being inclusions) is not a diagram of constructions, because when it is seen as a directed graph (as described in condition (10) of Def. 8.3), it is not a dag.*



Seeds of the diagram of constructions are those nodes that are allowed to be sources of refinement arrows, but they are not.

Definition 8.6 *Given a diagram of constructions* $\mathfrak{D} \colon \mathbf{J} \to \mathbf{SigDep}^{con}$*, the* set of seed nodes in $\mathfrak{D}$ *is defined as*

$$\begin{aligned}\mathbf{Seeds}(\mathfrak{D}) = \{n \in |\mathbf{J}| \mid\ & n \textit{ is not the source of any refinement arrow and} \\ & n \textit{ either does not belong to any sum square or} \\ & n \textit{ is a side node of a sum square}\}.\end{aligned}$$

Example 8.7 *In the diagram* $\mathfrak{D}$ *from Example 8.4 the nodes marked by* $\underline{SP}_1$*,* $\underline{SP'}_{21}$ *and* $\underline{SP'}_{22}$ *are seed nodes of* $\mathfrak{D}$*.*

Diagrams of constructions talk about construction specifications. Below we precise the method to handle the construction models in a diagram. We begin by defining the category suitable to represent models of construction specifications and construction signature and refinement morphisms. Further we define diagrams of construction models.

Definition 8.8 (Category of Models of Construction Specifications) *The category* $\mathbf{SpecDepM}^{con}$ *has pairs* $\langle \underline{SP}, Con \rangle$ *as objects, where* $\underline{SP} \in \mathbf{SpecDep}^{con}$ *and* $Con \in [\![\underline{SP}]\!]^c$ *is clean, and* $\mathbf{SpecDep}^{con}$*-morphisms as morphisms.*

The requirement that a construction model is clean reflects the technical assumption made in Theorem 7.14, needed to prove the (one side of) the satisfaction condition.

Definition 8.9 (Diagram of Construction Models) *Given a diagram of constructions* $\mathfrak{D} \colon \mathbf{J} \to \mathbf{SpecDep}^{con}$*, a corresponding* diagram of construction specification models *(or shortly* a diagram of construction models*) is a diagram* $\mathfrak{D}_{\mathfrak{M}} \colon \mathbf{J} \to \mathbf{SpecDepM}^{con}$ *such that*

1. *for each* $\mathbf{J}$*-object* n*,* $\pi_1(\mathfrak{D}_{\mathfrak{M}}(n)) = \mathfrak{D}(n)$*;*
2. *for each* $\mathbf{J}$*-morphism* σ*,* $\mathfrak{D}_{\mathfrak{M}}(\sigma) = \mathfrak{D}(\sigma)$*;*
3. *for each* $\mathbf{J}$*-morphism* $\sigma \colon m \to n$ *that is a refinement arrow* $\underline{\omega} = \mathfrak{D}(\sigma)$ *in* $\mathfrak{D}_{\mathfrak{M}}$*, the construction model in* $\underline{\omega}$*'s source,* $Con_m = \pi_2(\mathfrak{D}_{\mathfrak{M}}(m))$*, is the reduct of the construction model in* $\underline{\omega}$*'s target,* $Con_n = \pi_2(\mathfrak{D}_{\mathfrak{M}}(n))$*, i.e.* $Con_m = Con_n|_{\underline{\omega}}$*;*
4. *for each sum square in* $\mathfrak{D}_{\mathfrak{M}}$*, the construction model in the top node is the sum of the construction models in the side nodes w.r.t. the construction fitting from the sum square; and the construction model in the bottom node is the full class of models over the given signature.*

Let us notice that in requirement (3) from Def. 8.9 above, there is no need for extra application of cleaning, because, by Corollary 7.16, we have $\mathbf{Clean}_{\mathfrak{D}(m)}(Con_n|_{\underline{\omega}}) = Con_n|_{\underline{\omega}}$.

Notation. Obviously, given a diagram of constructions $\mathfrak{D}$, it has the same shape as any corresponding diagram of construction models $\mathfrak{D}_{\mathfrak{M}}$. Therefore, if we write *seed nodes of* $\mathfrak{D}_{\mathfrak{M}}$ we mean seed nodes of $\mathfrak{D}$ and we write $\mathbf{Seeds}(\mathfrak{D}_{\mathfrak{M}})$ for $\mathbf{Seeds}(\mathfrak{D})$.

The construction model residing in the result node of a diagram of construction models is considered a result of the system construction.

Definition 8.10 *The* result construction model *of a diagram of construction models* $\mathfrak{D}_{\mathfrak{M}} \colon \mathbf{J} \to \mathbf{SpecDepM}^{con}$*, denoted by* $\mathbf{Result}(\mathfrak{D}_{\mathfrak{M}})$*, is given as* $\pi_2(\mathfrak{D}_{\mathfrak{M}}(n))$*, where* $n \in |\mathbf{J}|$ *is the result node of* $\mathfrak{D}_{\mathfrak{M}}$*.*

It is enough to have a construction model for each seed of the diagram of constructions to obtain the corresponding diagram of construction models and hence the result of the system construction.

Theorem 8.11 *Consider a diagram of constructions* $\mathfrak{D} \colon \mathbf{J} \to \mathbf{SpecDep}^{con}$ *and for each seed node* $n \in \mathbf{Seeds}(\mathfrak{D})$*, a construction model* Con_n *such that* $Con_n \models^c \mathfrak{D}(n)$ *and* $Con = \mathbf{Clean}_{\mathfrak{D}(n)}(Con)$*. This setting yields the unique diagram of construction models* $\mathfrak{D}_{\mathfrak{M}}$ *corresponding to* $\mathfrak{D}$ *such that for each seed node* $n \in \mathbf{Seeds}(\mathfrak{D})$*,* $\pi_2(\mathfrak{D}_{\mathfrak{M}})(n) = \mathfrak{D}(n)$*.*

The proof is in Appendix 8.A. The diagram of construction models is obtained by repeated use of the sum of construction models operation and the reduct along the construction refinement morphism (cf. Corollary 7.16).

Example 8.12 *Let us take the diagram of constructions* $\mathfrak{D}$ *from Example 8.4. By Example 8.7 we know that its seed nodes are* $\underline{SP}_1$*,* $\underline{SP}'_{21}$ *and* $\underline{SP}'_{22}$*. Consider the following construction models of respective construction signatures of seed nodes:*

$$
\begin{aligned}
Con_1 &= \{M \in [\![\underline{\mathcal{S}}_1]\!] \mid a_M = c_M\},\\
Con'_{21} &= \{M \in [\![\underline{\mathcal{S}}'_{21}]\!] \mid f_M(b_M) = a_M\},\\
Con'_{22} &= \{M \in [\![\underline{\mathcal{S}}'_{22}]\!] \mid t_M = \{\heartsuit, \diamondsuit, \clubsuit, \spadesuit\},\ s_M = \{\star\},\ a_M = \star, b_M = \heartsuit,\\
&\qquad\qquad \textit{for all } x \in t_M,\ f_M(x) = \star\}.
\end{aligned}
$$

It is easy to check that $Con_1 \models^c \underline{SP}_1$, $Con'_{21} \models^c \underline{SP}'_{21}$ *and* $Con'_{22} \models^c \underline{SP}'_{22}$.

One-by-one we build the whole diagram of construction models $\mathfrak{D}_{\mathfrak{M}}$ *out from the construction models of the seed nodes.*

A sum of Con'_{21} *and* Con'_{22} *w.r.t. the construction fitting of the right sum square in* $\mathfrak{D}$ *gives the construction model* $Con'_2 = Con'_{22}$ *such that* $Con'_2 \models^c \underline{SP}'_2$.

The reduct of Con'_2 *along* $\underline{\omega}$ *yields a construction model of* $\underline{\mathcal{S}}_2$, *defined as*

$$Con_2 = \{M \in [\![\underline{\mathcal{S}}_2]\!] \mid s_M = \{\star\},\ a_M = \star\}$$

and we have $Con_2 \models^c \underline{SP}_2$.

Finally, we get the result construction model $Con = Con_1 \oplus_{ft} Con_2$, *where ft is the construction fitting of the left sum square of* $\mathfrak{D}$. *Its explicit definition is the following*

$$Con = \{M \in [\![\underline{\mathcal{S}}]\!] \mid s_M = \{\star\},\ a_M = \star,\ c_M = \star\}$$

and of course $Con \models^c \underline{SP}$.

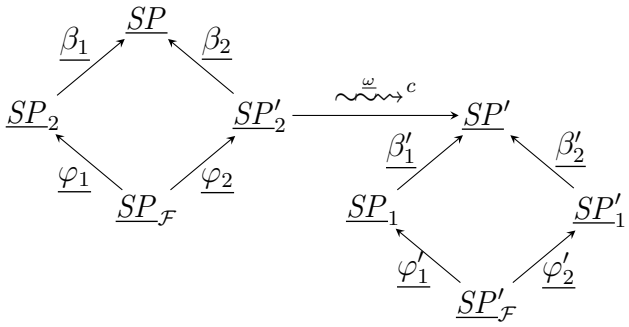
8.3 Operations as Diagrams of Constructions

In this section we look again at the typical connectives between parameterised module specifications and, using an example, we show how they can be represented as diagrams of constructions.

In general the problem is that the fitting spans or morphisms cannot be directly transformed to the corresponding construction fittings, therefore, there are examples of unions, compositions, applications and other operations on parameterised module specifications that fail to be represented directly as one sum of construction specifications. The technical reason is that not all **Sig**-morphism are $\mathbf{SigDep}^{frag}$-morphisms

The proposed solution is to use refinements as a method for abstraction, and refactor the architecture of the given settings of parameterised modules to make them representable as diagrams of constructions.

Example 8.13 *Consider the composition of parameterised module specifications from Example 6.45. The problem presented there concerns the dependency structure of symbol b, which contains symbol a in the construction signature of* $\underline{SP}_1$ *and does not contain a in the construction signature of* $\underline{SP}_2$*. The refactored setting is presented as a diagram of constructions given below*



where, additionally to already defined $\underline{SP}_1$ *and* $\underline{SP}_2$*, we have*

$$\begin{aligned}
\underline{SP} &= \langle(\textbf{sort } s;\ \textbf{ops } b : s,\ c : s;\ \textbf{deps } b < c), \emptyset\rangle,\\
\underline{SP}'_2 &= \langle(\textbf{sort } s;\ \textbf{ops } b : s), \emptyset\rangle,\\
\underline{SP}_{\mathcal{F}} &= \langle(\textbf{sort } \underline{s};\ \textbf{ops } \underline{b : s}), \emptyset\rangle,\\
\underline{SP'} &= \langle(\textbf{sort } s;\ \textbf{ops } a : s,\ b : s;\ \textbf{deps } a < b), \emptyset\rangle,\\
\underline{SP}'_1 &= \langle(\textbf{sort } s;\ \textbf{ops } a : s), \emptyset\rangle,\\
\underline{SP}'_{\mathcal{F}} &= \langle(\textbf{sort } \underline{s};\ \textbf{ops } \underline{a : s}), \emptyset\rangle
\end{aligned}$$

and all morphisms are inclusions. There are three seed nodes: $\underline{SP}_1$*,* $\underline{SP}_2$ *and* $\underline{SP}'_1$*. The first one constructs b (and corresponds to the first parameterised module specification from Example 6.45), the second one constructs c (corresponds to the second parameterised module specification). The last one is a kind of a "parameter" of the above diagram with sort s and operation a; it may be further refined to match the implementation, when provided.*

As illustrated by the above example, the top-down approach to system representation requires that all new dependencies are added prior to the sum-based decomposition that corresponds to the composition or similarly (partial) application operation on parameterised module specifications.

8.A Appendix: Proofs

Proof of Theorem 8.11. Consider a diagram of constructions $\mathfrak{D}\colon \mathbf{J} \to \mathbf{SpecDep}^{con}$ and a collection of construction models indexed by seed nodes of $\mathfrak{D}$, $\langle Con_n \rangle_{n \in \mathbf{Seeds}(\mathfrak{D})}$. Diagram $\mathfrak{D}$ seen as a directed graph (as described in condition (10) of Def. 8.3) is a dag. Its nodes without successors (leaves) are either bottom nodes of sum squares (denote by $\underline{SP_{\mathcal{F}}}$) or seed nodes of $\mathfrak{D}$. We construct the diagram of construction models $\mathfrak{D}_{\mathfrak{M}}\colon \mathbf{J} \to \mathbf{SpecDepM}^{con}$. On morphisms, for each $\mathbf{J}$-morphism σ, we directly set $\mathfrak{D}_{\mathfrak{M}}(\sigma) = \mathfrak{D}(\sigma)$. On nodes we proceed by induction on the structure of the dag.

- Its leaves are either construction models of seed nodes, so we put $\mathfrak{D}_{\mathfrak{M}}(n) = Con_n$ for any seed n, or full classes of models of $\underline{\mathcal{F}}$ (because $\underline{SP_{\mathcal{F}}}$ are empty specifications and every $\underline{\mathcal{F}}$ contains only assumed symbols), so we put $\mathfrak{D}_{\mathfrak{M}}(n) = [\![\pi_1(\mathfrak{D}(m))]\!]$ for any bottom node m in $\mathfrak{D}$;
- Each internal internal node of the tree is either the source of a unique refinement arrow or it is the top node of a unique sum square;
  1. if it is $m \in |\mathbf{J}|$ such that it is the source of a refinement arrow $\underline{\omega}$ in $\mathfrak{D}$, i.e. $\underline{\omega} = \mathfrak{D}(\sigma)$ for $\sigma\colon m \to n \in \mathbf{J}$, we put $\mathfrak{D}_{\mathfrak{M}}(m) = \langle \mathfrak{D}(m), \pi_2(\mathfrak{D}_{\mathfrak{M}}(n))|_{\underline{\omega}} \rangle$,
  2. if it is $p \in |\mathbf{J}|$ such that it is the top node of a sum square in $\mathfrak{D}$, i.e. there are two nodes $o_1, o_2 \in J$ such that $\mathfrak{D}(o_1)$ and $\mathfrak{D}(o_2)$ are the side nodes of a sum square, which is the pushout of some construction fitting $ft = \langle \underline{\varphi_1}, \underline{\varphi_2} \rangle$ in $\mathbf{SigDep}^{frag}$, we put $\mathfrak{D}_{\mathfrak{M}}(p) = \langle \mathfrak{D}(p), \pi_2(\mathfrak{D}_{\mathfrak{M}}(o_1)) \oplus_{ft} \pi_2(\mathfrak{D}_{\mathfrak{M}}(o_2)) \rangle$

By construction diagram $\mathfrak{D}_{\mathfrak{M}}$ is well defined on all morphisms and nodes. Moreover, $\mathfrak{D}_{\mathfrak{M}}$ is a diagram of construction models (meeting all requirements of Def. 8.8 and Def. 8.9). The nontrivial part is the requirement from Def. 8.8 that all construction models are clean. Regarding the point (1) above, by Corollary 7.16, we have

$$\pi_2(\mathfrak{D}_{\mathfrak{M}}(n))|_{\underline{\omega}} = \mathbf{Clean}_{\mathfrak{D}(m)}(\pi_2(\mathfrak{D}_{\mathfrak{M}}(n))|_{\underline{\omega}}).$$

Regarding the point (2) above, in definition Def. 8.3 it is required that the construction specification in the top node of a sum square is the sum of construction specifications from the side nodes. Therefore,

$$\mathbf{Clean}_{\mathfrak{D}(p)}(\pi_2(\mathfrak{D}_{\mathfrak{M}}(o_1)) \oplus_{ft} \pi_2(\mathfrak{D}_{\mathfrak{M}}(o_2))) = \pi_2(\mathfrak{D}_{\mathfrak{M}}(o_1)) \oplus_{ft} \pi_2(\mathfrak{D}_{\mathfrak{M}}(o_2)).$$

Finally, we notice that, by construction, $\mathfrak{D}_{\mathfrak{M}}$ is unique such that for each seed node $n \in \mathbf{Seeds}(\mathfrak{D})$, $\pi_2(\mathfrak{D}_{\mathfrak{M}})(n) = \mathfrak{D}(n)$, as required. □

Chapter 9

Example

9.1 Introduction

In this chapter we provide some steps of development of an example architecture specification. We start by stating a vague description of a system in the natural language. Then we propose the encoding of the first understanding of the system components into construction signatures and construction specifications, and we represent the system decomposition as a diagram of constructions. Further we refine some of the components and provide their subsequent decompositions.

In this chapter, we assume that the base institution is $\mathbb{FOEQF}$, the institution of first order logic with equality with finite many sorted algebraic signatures (cf. Def. 3.9).

9.2 Travel Agency System

The example is about the development of a software system for travel agencies. The system is supposed to manage offices where clients, interested in reservations of flight or train tickets, and also considering taking hotel rooms, may make reservations.

First, from the description we extract the following sorts, useful to formulate the specification of the system:

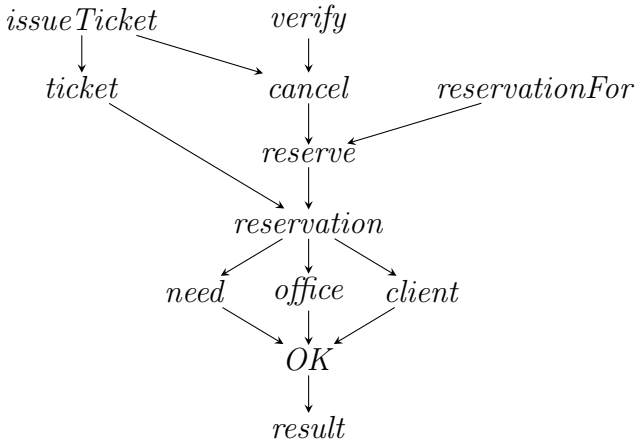
$$\mathit{office}, \mathit{client}, \mathit{need}, \mathit{reservation}, \mathit{ticket}, \mathit{offer}, \mathit{result}$$

We also add the operations:

$$\begin{aligned} \mathit{OK} &: \mathit{result} \\ \mathit{reserve} &: \mathit{office} \times \mathit{client} \times \mathit{need} \to \mathit{reservation} \times \mathit{result} \\ \mathit{cancel} &: \mathit{reservation} \to \mathit{reservation} \times \mathit{result} \\ \mathit{verify} &: \mathit{reservation} \times \mathit{client} \to \mathit{result} \\ \mathit{reservationFor} &: \mathit{reservation} \to \mathit{client} \times \mathit{result} \\ \mathit{issueTicket} &: \mathit{reservation} \to \mathit{ticket} \times \mathit{result} \end{aligned}$$

OK models the successful results. Operation *reserve* covers the situation where a client presents her/his needs to an office and gets a reservation along with the result of the operation. We assume that *need* is a complete description of possible needs, *reservation* encompasses all types and variants of reservations, their statuses, payment information etc. Similarly *result* represents any type of a result, possibly different for different operations. Operation *cancel* allows a client to cancel a reservation. *verify* checks whether a given reservation is good for a given client. Operations *reservationOf* and *reservationFor* give some more information about the reservation. Finally, *issueTicket* converts a reservation into a ticket.

The above-given description of sorts and operations suggests the following dependency relation between them:



Let $\underline{\mathcal{S}}$ be a construction signature that contains the above-given sorts, operations and dependency relation. Let it be a complete signature fragment,

i.e. such that all symbols are defined.

The below-given first-order logic axioms reflect some of the expected properties of the system. All axioms are over the signature $\mathbf{UnDep}(\mathbf{Compl}(\underline{\mathcal{S}}))$.

$$
\begin{aligned}
Ax_{for} &= \forall\, r : reservation; o : office; c : client; n : need \cdot \\
&\quad (reserve(o, c, n) = \langle r, OK\rangle) \Rightarrow (reservationFor(r) = \langle c, OK\rangle) \\
Ax_{ver1} &= \forall\, r : reservation; o : office; c : client; n : need \cdot \\
&\quad (reserve(o, c, n) = \langle r, OK\rangle) \Rightarrow (verify(r, c) = OK) \\
Ax_{ver2} &= \forall\, r : reservation; o : office; c, c' : client; n : need \cdot \\
&\quad (reserve(o, c, n) = \langle r, OK\rangle \wedge \neg(c = c')) \Rightarrow (\neg(verify(r, c') = OK)) \\
Ax_{can} &= \forall\, r, r' : reservation; c : client \cdot \\
&\quad (cancel(r) = \langle r', OK\rangle) \Rightarrow \neg(verify(r', c) = OK) \\
Ax_{tic} &= \forall\, r : reservation; o : office; c : client; n : need; t : ticket \cdot \\
&\quad \neg(reserve(o, c, n) = \langle r, OK\rangle) \Rightarrow \neg(issueTicket(r) = \langle t, OK\rangle)
\end{aligned}
$$

Ax_{for} says that a successful reservation for a client is attributed to the same client. Ax_{ver1} and Ax_{ver2} say that a successful reservation verifies positively only for the same client. Ax_{can} says that a cancelled reservation does not verify positively for any client. Ax_{tic} forbids the successful ticket issue from the failed reservation.

Let $Ax = \{Ax_{ver1}, Ax_{ver2}, Ax_{can}, Ax_{tic}\}$. We define $\underline{SP} = \langle \underline{\mathcal{S}}, \langle \Sigma, Ax\rangle\rangle$.

We expect the monolithic high-level description to be further decomposed and refined. The following construction signatures constitute the candidates for such decomposition (assumed symbols are underlined).

First is the *results part*. All other parts depend on it, because it represents the system common part. Recall that we omit all basic dependencies (dependency of an operation on sorts from its profile).

$$\underline{\mathcal{S}}_{rlt} = (\mathbf{sorts}\ result;\ \mathbf{op}\ OK : result)$$

There is also the *offices part* including clients. It is dependent on result part.

$$\begin{aligned}\underline{\mathcal{S}_{off}} = (&\textbf{sorts } \mathit{office}, \mathit{client}, \underline{\mathit{result}};\\ &\textbf{op } \underline{\mathit{OK} : \mathit{result}};\\ &\textbf{deps } \mathit{OK} < \mathit{office}, \mathit{OK} < \mathit{client})\end{aligned}$$

Next comes the *reservations part* dependent on the office part.

$$\begin{aligned}\underline{\mathcal{S}_{res}} = (&\textbf{sorts } \mathit{reservation}, \mathit{need}, \underline{\mathit{office}}, \underline{\mathit{client}}, \underline{\mathit{result}};\\ &\textbf{ops } \mathit{reserve} : \mathit{office} \times \mathit{client} \times \mathit{need} \rightarrow \mathit{reservation} \times \mathit{result},\\ &\quad \mathit{cancel} : \mathit{reservation} \rightarrow \mathit{reservation} \times \mathit{result},\\ &\quad \mathit{verify} : \mathit{reservation} \times \mathit{client} \rightarrow \mathit{result},\\ &\quad \mathit{reservationFor} : \mathit{reservation} \rightarrow \mathit{client} \times \mathit{result},\\ &\quad \underline{\mathit{OK} : \mathit{result}};\\ &\textbf{deps } \mathit{reserve} < \mathit{cancel}, \mathit{cancel} < \mathit{verify}, \mathit{need} < \mathit{reservation},\\ &\quad \mathit{office} < \mathit{reservation}, \mathit{client} < \mathit{reservation},\\ &\quad \mathit{OK} < \mathit{need}, \mathit{OK} < \mathit{office}, \mathit{OK} < \mathit{client})\end{aligned}$$

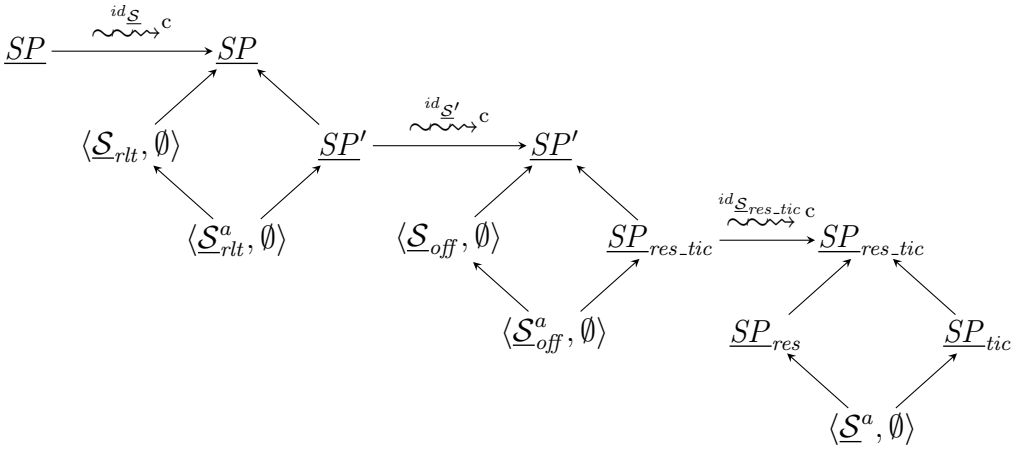
The corresponding specification is $\underline{SP_{res}} = \langle \underline{\mathcal{S}_{res}}, (Ax_{ver1} \cup Ax_{ver2} \cup Ax_{can}) \rangle$.

Finally there is a *tickets part*.

$$\begin{aligned}\underline{\mathcal{S}_{tic}} = (&\textbf{sorts } \mathit{ticket}, \underline{\mathit{reservation}}, \underline{\mathit{need}}, \underline{\mathit{office}}, \underline{\mathit{client}}, \underline{\mathit{result}};\\ &\textbf{ops } \mathit{issueTicket} : \mathit{reservation} \rightarrow \mathit{ticket} \times \mathit{result}\\ &\quad \underline{\mathit{reserve} : \mathit{office} \times \mathit{client} \times \mathit{need} \rightarrow \mathit{reservation} \times \mathit{result}},\\ &\quad \underline{\mathit{cancel} : \mathit{reservation} \rightarrow \mathit{reservation} \times \mathit{result}},\\ &\quad \underline{\mathit{OK} : \mathit{result}};\\ &\textbf{deps } \mathit{reserve} < \mathit{cancel}, \mathit{cancel} < \mathit{issueTicket}, \mathit{need} < \mathit{reservation}\\ &\quad \mathit{office} < \mathit{reservation}, \mathit{client} < \mathit{reservation},\\ &\quad \mathit{OK} < \mathit{need}, \mathit{OK} < \mathit{office}, \mathit{OK} < \mathit{client})\end{aligned}$$

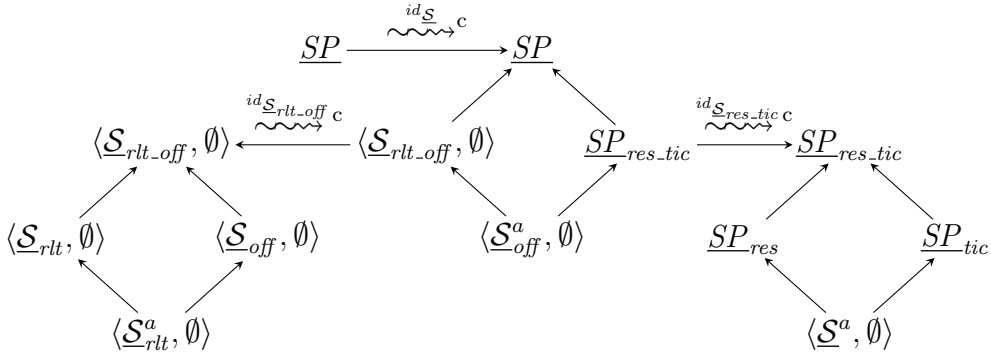
The specification for the tickets part is $\underline{SP_{tic}} = \langle \underline{\mathcal{S}_{tic}}, Ax_{tic} \rangle$.

The decomposition of the system may be given as a diagram of constructions $\mathfrak{D}_1$



with all arrows being inclusions (the refinement morphisms being identities). The construction signature $\underline{\mathcal{S}}^a_{rlt}$ has the symbols and dependency structure as $\underline{\mathcal{S}}_{rlt}$, but with all symbols being assumed. Similarly, the construction signatures $\underline{\mathcal{S}}^a$ and $\underline{\mathcal{S}}^a_{off}$ are the assumed versions of $\underline{\mathcal{S}}$ and $\underline{\mathcal{S}}_{off}$, respectively. All construction signatures indexed by $\_^a$ are empty signature fragments (containing only assumed symbols). Moreover, we define $\underline{SP'} = \langle \underline{\mathcal{S}}', Ax \rangle$ with $\underline{\mathcal{S}}' = (\underline{\mathcal{S}}^a_{rlt} \cup \underline{\mathcal{S}}_{off} \cup \underline{\mathcal{S}}_{res} \cup \underline{\mathcal{S}}_{tic})$ and $\underline{SP}_{res\_tic} = \underline{SP}_{res} \cup \underline{SP}_{tic}$.

Another decomposition may be given as $\mathfrak{D}_2$ depicted below,



again with all arrows being inclusions. Additionally to the construction signatures and specifications introduced in description of $\mathfrak{D}_1$, the construction signature $\underline{\mathcal{S}}_{rlt\_off} = \underline{\mathcal{S}}_{rlt} \cup \underline{\mathcal{S}}_{off}$.

By looking at the seed nodes only, the two above diagrams of construc-

tions are equivalent. However, clearly the architectures described by $\mathfrak{D}_1$ and $\mathfrak{D}_2$ are different. As it will be briefly discussed in Chapter 10, in order to disregard some architectural (unimportant) details of system decomposition, it may be more appropriate to use some sort of architecture development logic to write specifications of the decomposition in more abstract way.

9.3 Further Refinement Steps

In this section we propose refinement steps for some parts of the system. We give the refinement definitions explicitly, but again, we see the need to have a possibility to specify the refinement steps and decomposition in some sort of the architecture development logic.

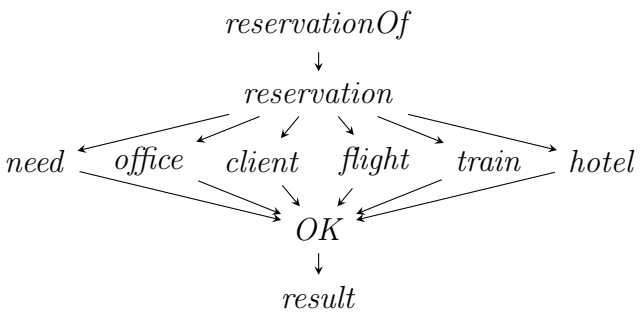
In the refinement we add to $\underline{\mathcal{S}}_{res}$ the following sorts

$$\mathit{flight}, \mathit{train}, \mathit{hotel}$$

representing the respective reservation domains and an operation allowing to get a reservation type from a reservation

$$\mathit{reservationOf} : \mathit{reservation} \rightarrow \mathit{flight} \times \mathit{train} \times \mathit{hotel} \times \mathit{result}$$

The update of dependencies needed to incorporate the new sorts and operation is given by the diagram below.

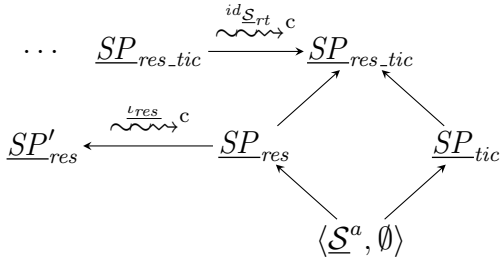


The construction signature of the refined reservations part is the following

$$
\begin{aligned}
\underline{\mathcal{S}}'_{res} = (&\textbf{sorts } \mathit{reservation}, \mathit{need}, \mathit{flight}, \mathit{train}, \mathit{hotel}, \underline{\mathit{office}}, \underline{\mathit{client}}, \underline{\mathit{result}};\\
&\textbf{ops } \mathit{reserve} : \mathit{office} \times \mathit{client} \times \mathit{need} \to \mathit{reservation} \times \mathit{result},\\
&\quad \mathit{cancel} : \mathit{reservation} \to \mathit{reservation} \times \mathit{result},\\
&\quad \mathit{verify} : \mathit{reservation} \times \mathit{client} \to \mathit{result},\\
&\quad \mathit{reservationFor} : \mathit{reservation} \to \mathit{client} \times \mathit{result},\\
&\quad \mathit{reservationOf} : \mathit{reservation} \to \mathit{flight} \times \mathit{train} \times \mathit{hotel} \times \mathit{result},\\
&\quad \underline{\mathit{OK} : \mathit{result}};\\
&\textbf{deps } \mathit{reserve} < \mathit{cancel}, \mathit{cancel} < \mathit{verify},\\
&\quad \mathit{need} < \mathit{reservation}, \mathit{office} < \mathit{reservation},\\
&\quad \mathit{client} < \mathit{reservation}, \mathit{flight} < \mathit{reservation},\\
&\quad \mathit{train} < \mathit{reservation}, \mathit{hotel} < \mathit{reservation},\\
&\quad \mathit{OK} < \mathit{need}, \mathit{OK} < \mathit{office}, \mathit{OK} < \mathit{client},\\
&\quad \mathit{OK} < \mathit{flight}, \mathit{OK} < \mathit{train}, \mathit{OK} < \mathit{hotel})
\end{aligned}
$$

Let $\underline{SP}'_{res} = \langle \underline{\mathcal{S}}'_{res}, (Ax_{ver1} \cup Ax_{ver2} \cup Ax_{can}) \rangle$ and let $\underline{\iota_{res}} : \underline{\mathcal{S}}_{res} \to \underline{\mathcal{S}}'_{res}$ be an inclusion. Clearly ι_{res} a is construction signature refinement morphism and $\underline{SP}_{res} \overset{\iota_{res}}{\rightsquigarrow}{}^{\mathrm{c}} \underline{SP}'_{res}$.

Below-given diagram of constructions $\mathfrak{D}'_1$ is such an extension of $\mathfrak{D}_1$ that contains refinement of $\underline{SP}_{res}$ by $\underline{SP}'_{res}$. The part of the diagram denoted by "..." is the same as in $\mathfrak{D}_1$.



After the refinement of the reservations part added the new sorts representing the reservation domains, the tickets part also needs the addition of those sorts, because only then the right type of the ticket may be matched

to the type of the reservation. The following operation allows one to get the type of ticket and the corresponding reservation:

$$\mathit{ticketInfo} : \mathit{ticket} \rightarrow \mathit{flight} \times \mathit{train} \times \mathit{result}$$

Consider the the construction signature of the refined tickets part.

$$\begin{aligned}
\underline{\mathcal{S}'_{tic}} = (&\textbf{sorts } \mathit{ticket}, \mathit{flight}, \mathit{train}, \underline{\mathit{reservation}}, \underline{\mathit{need}}, \underline{\mathit{office}}, \underline{\mathit{client}}, \underline{\mathit{result}};\\
&\textbf{ops } \mathit{issueTicket} : \mathit{reservation} \rightarrow \mathit{ticket} \times \mathit{result}\\
&\quad \mathit{ticketInfo} : \mathit{ticket} \rightarrow \mathit{flight} \times \mathit{train} \times \mathit{result}\\
&\quad \underline{\mathit{reserve} : \mathit{office} \times \mathit{client} \times \mathit{need} \rightarrow \mathit{reservation} \times \mathit{result}},\\
&\quad \underline{\mathit{cancel} : \mathit{reservation} \rightarrow \mathit{reservation} \times \mathit{result}},\\
&\quad \underline{\mathit{reservationOf} : \mathit{reservation} \rightarrow \mathit{flight} \times \mathit{train} \times \mathit{hotel} \times \mathit{result}},\\
&\quad \underline{\mathit{OK} : \mathit{result}};\\
&\textbf{deps } \mathit{reserve} < \mathit{cancel}, \mathit{cancel} < \mathit{issueTicket},\\
&\quad \mathit{reservationOf} < \mathit{ticketInfo},\\
&\quad \mathit{need} < \mathit{reservation}, \mathit{office} < \mathit{reservation},\\
&\quad \mathit{client} < \mathit{reservation}, \mathit{flight} < \mathit{reservation},\\
&\quad \mathit{train} < \mathit{reservation}, \mathit{hotel} < \mathit{reservation},\\
&\quad \mathit{OK} < \mathit{need}, \mathit{OK} < \mathit{office}, \mathit{OK} < \mathit{client},\\
&\quad \mathit{OK} < \mathit{flight}, \mathit{OK} < \mathit{train}, \mathit{OK} < \mathit{hotel})
\end{aligned}$$

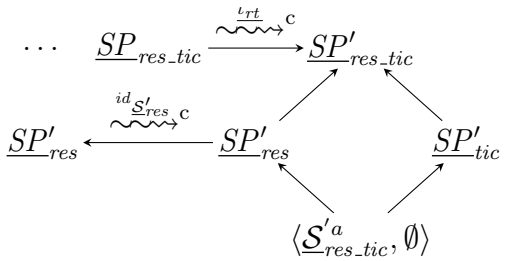
The following axiom expresses the expected property that the ticket issued based on the reservation of a flight or a train, concerns the same transportation.

$$\begin{aligned}
\mathit{Ax}_{\mathit{tran}} = \forall\ & r : \mathit{reservation}; f : \mathit{flight}; tr : \mathit{train}; h : \mathit{hotel}; t : \mathit{ticket} \cdot\\
& (\mathit{issueTicket}(r) = \langle t, \mathit{OK}\rangle) \Rightarrow\\
& \quad ((\mathit{reservationOf}(r) = \langle f, tr, h, \mathit{OK}\rangle) \iff\\
& \quad\quad (\mathit{ticketInfo}(t) = \langle f, tr, \mathit{OK}\rangle))
\end{aligned}$$

Let the refined specification for the tickets part be $\underline{SP'}_{tic} = \langle \underline{\mathcal{S}}_{tic}, Ax_{tran} \cup Ax_{tic} \rangle$.

The independent refinements of two different parts of the system must not add dependencies to the shared symbols (cf. the discussion below Theorem 7.17). Therefore, the only way to consistently add new symbols to both reservations part and tickets part is to do it before the split of $\underline{SP}_{res\_tic}$ into $\underline{SP}_{res}$ and $\underline{SP}_{tic}$.

Consider $\mathfrak{D}''_1$, another extension of $\mathfrak{D}_1$; again, the part of the diagram denoted by "..." is the same as in $\mathfrak{D}_1$



where $\underline{SP'}_{res\_tic} = \underline{SP'}_{res} \cup \underline{SP'}_{tic}$ is a construction specification over a construction signature $\underline{\mathcal{S}'}_{res\_tic}$ and $\underline{\mathcal{S}'^{a}}_{res\_tic}$ is its assumed version; moreover, $\underline{\iota_{rt}} : \underline{\mathcal{S}}_{res\_tic} \rightarrow \underline{\mathcal{S}'}_{res\_tic}$ is an inclusion.

Chapter 10
Summary

In this thesis we presented an approach to development of software architectures via diagrams of constructions. In order to unify the notion of simple and parameterised modules we have equipped the construction signatures with additional dependencies between symbols and the information whether a given symbol is defined or assumed from the outside.

We have started by giving the motivation, discussing the related work, presenting technical preliminaries and setting the assumptions about the base institution $\mathbb{I}$. Then we have provided a formal introduction to signature fragments and signatures with dependency structure (called also signatures with dependencies). Such signatures are equipped with additional relation on symbols that describes the dependency structure of each symbol. Construction signatures are defined as fragments of signatures with dependencies, with bounded strict orders as dependency relations. Morphisms between construction signatures are given as p-morphisms on their symbols, i.e., monotonic maps that weakly reflect the dependency structures of symbols. Construction models are classes of models (from the base institution $\mathbb{I}$) that share interpretation of defined symbols under condition that they share the interpretation of dependency structure of those symbols. The interpretation of assumed symbol may vary, even in presence of the shared interpretation of their dependency structure. Construction model reducts are reducts of all models in the construction model. Construction specifications are then simply specifications in the base institution $\mathbb{I}$. Satisfaction relation requires that the constructed (defined) symbols satisfy the specification, the parameter (assumed) symbols are represented in all variants matching the specification, any partial instantiation respecting the dependency structure is possible, and the specification respects the dependency structure (does not relate the

otherwise independent symbols). Further, we have defined the construction fittings between construction signatures that guarantee the explicit tracking of defined symbols. The sum operation is given by the pushout for construction signatures, the amalgamation operation for construction models, and the specification union for construction specifications. We have given a notion of static compatibility of construction specifications w.r.t. the construction fitting and we have proven that the sum of compatible construction models satisfies the union of corresponding construction specifications.

Along with the definitions of construction signatures, models and specifications, we have presented the correspondence between parameterised modules and constructions. Our sum operation essentially subsumes standard operations (union, composition, application) on parameterised modules, under their representation as constructions. Interestingly, constructions give uniform treatment of first- and higher-order parameterised modules.

Construction signatures, models and specifications do not form an institution (of constructions). This is because the construction satisfaction relation (cf. Def. 6.13) poses completeness requirements w.r.t. assumed symbols, and construction signature morphism may map assumed symbols to defined symbols. The reduct of a construction model along such morphism usually contains fewer models than required by the satisfaction condition. In our approach this is not a problem, because construction signatures are used only for construction fittings and sum embeddings (horizontal composition in terms of parameterised programming). We do not use such morphisms for hiding (i.e., for abstraction). For that purpose we define another type of morphisms.

Construction signature refinement morphisms between construction signatures are injective on assumed symbols, require newly added symbols to be defined and allow for addition of extra dependency between old and new symbols (by new symbols we mean those that are not in the image of the morphism). Such additions represent auxiliary symbols, possibly used to represent more implementation details in the target of the morphism. Further, we have introduced construction specification refinements (vertical composition in terms of parameterised programming) that allow for stronger specifications

of defined symbols. By proving one side of the satisfaction condition for constructions with construction refinement morphisms, we come very close to the definition of the institution of constructions. However, since such an institution is not needed for the purpose of this thesis, we have left the complete proof of the satisfaction condition (opposite side) for the future work. We have shown that the typical symmetric compositionality theorem holds, but its analysis has led to the discovery that its use in typical situations is very limited. As a partial solution to those limitations, we give a single-side compositionality result.

Having constructions as basic architectural units, the sum as a composition operation and construction refinements as a method for abstraction and step-wise refinement we have defined the diagrams of constructions. Diagrams of constructions correspond to software architecture designs. They represent the decomposition structure of the system, and also provide the guideline for the software development process.

We have also presented an example of specification using diagram of constructions.

10.1 Future Work

Many aspects of system specification via diagrams of constructions were not covered in this thesis. We always tried to give the most complete picture of the defined notions, but, as the main goal of this thesis was to provide proper formulation of software architectures as diagram for constructions, some issues were left for the next steps.

The first topic that finally did not find its place in the thesis is architecture specification logic. Such logic has been envisaged to be interpreted upon diagrams of constructions. Roughly, the idea is to extend a diagram of constructions by a global signature and embeddings from every construction signature in the diagram. The logic would be a variant of temporal logic (e.g. based on CTL*) suitable to express architectural properties of the diagram of construction, such as “a given set of symbols is defined only together, i.e. in the same node of the diagram” or “in the refinement structure of some

node there is a node that exhibits an interface (made of assumed symbols) that meets a given specification", etc.

A missing concept is, as it was already mentioned, the institution of constructions and construction signature refinements. It would be favorable to have the constructions framework that forms an institution and benefit from the extensive research base available for institutions.

Another interesting topic for future work is the standard compositionality of construction specifications with respect to refinement. The approach discussed in the thesis has shown to be limited in typical applications.

Finally, a natural follow-up is to implement the concept of constructions as a specification language. Such formalism, code-named by us "dependency-oriented specifications", would allow one to check applicability and usability of our concept, and whether internal complexity of its semantics is really a huge challenge.

Bibliography

[AG97] Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology (TOSEM)* , 6(3):213–249, 1997.

[AHS90] Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and Concrete Categories*. Wiley-Interscience, 1990.

[BG77] Rod M. Burstall and Joseph. A. Goguen. Putting theories together to make specifications. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'77, pages 1045–1058. Morgan Kaufmann Publishers Inc., 1977.

[BG80] Rod M. Burstall and Joseph A. Goguen. The semantics of Clear, a specification language. In Dines Bjørner, editor, *Abstract Software Specifications*, volume 86 of *Lecture Notes in Computer Science*, pages 292–332. Springer, 1980.

[BG92] Rod M. Burstall and Joseph A. Goguen. Institutions: Abstract model theory for specification and programming. *Journal of the ACM (JACM)*, 39(1):95–146, 1992.

[BGM89] Michel Bidoit, Marie-Claude Gaudel, and Anne Mauboussin. How to make algebraic specifications more understandable: An experiment with the Pluss specification language. *Science of Computer Programming*, 12(1):1 – 38, 1989.

[BL69] Rod M. Burstall and Peter J. Landin. Programs and their proofs: an algebraic approach. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence*, volume 4, pages 17–43. Edinburgh University Press, 1969.

[BL70] Garrett Birkhoff and John D. Lipson. Heterogeneous algebras. *Journal of Combinatorial Theory* , 8(1):115 – 133, 1970.

[CRS+11] William Chaves de Souza Carvalho, Pedro Frosi Rosa, Michel dos Santos Soares, Marco Antonio Teixeira da Cunha Jr., and Luiz Carlos Buiatte. A comparative analysis of the agile and traditional software development processes productivity. In *Proceedings of the 2011 30th International Conference of the Chilean Computer Science Society*, SCCC '11, pages 74–82. IEEE Computer Society, 2011.

[Dia08] Răzvan Diaconescu. *Institution-independent Model Theory*. Birkhäuser Basel, 1st edition, 2008.

[EM85] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer, 1985.

[EM90] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*. Springer, 1990.

[FLW03] José Luiz Fiadeiro, Antónia Lopes, and Michel Wermelinger. A mathematical semantics for architectural connectors. In Roland Backhouse and Jeremy Gibbons, editors, *Generic Programming: Advanced Lectures*, number 2793 in Lecture Notes in Computer Science, pages 178–221. Springer, 2003.

[GB80] Joseph A. Goguen and Rod M. Burstall. CAT, a system for the structured elaboration of correct programs from structured specifications. SRI Technical Report CSL 118. Computer Science Laboratory, SRI International, 1980.

[GB84] Joseph A. Goguen and Rod M. Burstall. Introducing institutions. In Edmund Clarke and Dexter Kozen, editors, *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 221–256. Springer, 1984.

[Gog84] Joseph A. Goguen. Parameterized programming. *IEEE Transactions on Software Engineering*, 10(5):528–544, 1984.

[Gog96] Joseph A. Goguen. Parameterized programming and software architecture. In *Proceedings of REUSE 1996*, pages 2–11. IEEE Computer Society Press, 1996.

[GS94] David Garlan and Mary Shaw. An introduction to software architecture. CMU Software Engineering Institute Technical Report CMU/SEI-94-TR-21. Carnegie Mellon University, 1994.

[GS01] H. Peter Gumm and Tobias Schröder. Products of coalgebras. *Algebra Universalis*, 46:163—185, 2001.

[GTW78] Joseph A. Goguen, James W. Thatcher, and Eric G. Wagner. An Initial algebra approach to the specification, correctness and implementation of abstract data types. In Raymond T. Yeh, editor, *Current Trends in Programming Methodology: Data Structuring*, volume 4, pages 80–149. Prentice Hall, 1978.

[GWM+92] Joseph A. Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ3. Technical Report SRI CSL 92-03. Computer Science Laboratory, SRI International, 1992.

[Mac84] David MacQueen. Modules for Standard ML. In *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, LFP '84, pages 198–207. ACM, 1984.

[Mar12] Grzegorz Marczyński. Algebraic signatures enriched by dependency structure. In *Proceedings of WADT 2010*, volume 7137 of *Lecture Notes in Computer Science*, pages 226–250. Springer, 2012.

[ML98] Saunders Mac Lane. *Categories for the Working Mathematician (Graduate Texts in Mathematics)*. Springer, 2nd edition, 1998.

[Mos04] Peter D. Mosses, editor. CASL *Reference Manual, The Complete Documentation of the Common Algebraic Specification*

Language, volume 2960 of *Lecture Notes in Computer Science (IFIP Series)*. Springer, 2004.

[Par72] David Lorge Parnas. A technique for software module specification with examples. *Communications of the Association for Computing (j-CACM)*, 15(5):330–336, May 1972.

[San09] Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems*, 31(4):15:1–15:41, May 2009.

[Seg70] Krister Segerberg. Modal logics with linear alternative relations. *Theoria*, 36(3):301–322, December 1970.

[SST92] Donald Sannella, Stefan Sokolowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Parameterisation revisited. *Acta Informatica*, 29(8):689–736, 1992.

[ST88] Donald Sannella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica*, 25(3):233–281, April 1988.

[ST91] Donald Sannella and Andrzej Tarlecki. A kernel specification formalism with higher-order parameterisation. In Hartmut Ehrig, Klaus P. Jantke, Fernando Orejas, and Horst Reichel, editors, *Recent Trends in Data Type Specification*, volume 534 of *Lecture Notes in Computer Science*, pages 274–296. Springer, 1991.

[ST06] Donald Sannella and Andrzej Tarlecki. Horizontal composability revisited. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning, and Computation*, volume 4060 of *Lecture Notes in Computer Science*, pages 296–316. Springer, 2006.

[ST12] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. EATCS Monographs on Theoretical Computer Science. Springer, 2012.

[SW82] Donald Sannella and Martin Wirsing. Implementation of parameterised specifications. In Mogens Nielsen and ErikMeineche Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 473–488. Springer, 1982.

[SW83] Donald Sannella and Martin Wirsing. A kernel language for algebraic specification and implementation extended abstract. In Marek Karpinski, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 1983.

[WLBF09] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys*, 41(4):19:1–19:36, October 2009.