The Institution of Multialgebras
– a general framework for algebraic software development

Dr. Scient Thesis

Yngve Lamo
email: yla@hib.no

Bergen
23 October 2002

Department of Informatics
University of Bergen

Contents

Acknowledgements

First of all I would like to thank Michał Walicki, my supervisor. Without his many suggestions and constant support the work would have been impossible.

I am also thankful to the people in the programming theory group at the informatics department, university of Bergen, for their feedback on my research. A special thanks to Valentinas Kriauciukas and Uwe Wolter, for their support.

I would also thank the technical staff at the department, especially Jan Berger Henriksen for always helping me when my computer get nondeterministic.

In the first year at the department I had the pleasure of sharing office with Halvard Movik Martinsen, during this year I also met Talal Rahman, thanks to both for their friendship.

I will also thank Bergen University College for their support, and my colleges at the data-engineer education, especially Sven-Olai Høyland for his always positive attitude.

I will also thank you for reading this.

Of course, I am grateful to my family for their patience and *love*. My children Oda Marie and Eivind Mathias, I'm in good mood just thinking about you. My lovely Lise for being the best person in the world, my always supporting parents, Lill Sølvi and Torstein and my sister and brother.

Bergen
2002

Yngve Lamo

Introduction

The need for formal treatment of the system development process is well motivated from large software projects. Algebraic specification techniques offer a promising tool to obtain the goal of correct and efficient data programs. The underlying idea is that programs are modelled as algebras i.e. a set of functions with a corresponding set of axioms specifying the behavior of the functions. There exists several different algebraic specification formalisms in the literature, each of them particulary well suited to specify some aspect of the software developing process. In this thesis we propose that multialgebras could be used as a unifying framework that captures most of the advantages offered by other specification frameworks.

Multialgebras provide a powerful algebraic framework for specification – primarily, but not exclusively, of nondeterministic behavior [22, 55, 27]. Actually multialgebras were introduced to computer science for specifying nondeterministic programs, in a multialgebra a nondeterministic operation returns the set of all possible outcomes of the operation. Hence operations are interpreted as functions from the carrier to the powerset of the carrier, i.e. multialgebras are a direct generalization of classical algebras. In the thesis we don't focus on nondeterministic aspect of multialgebras but we focus on other aspects offered by this generalization of algebras.

It seems to be accepted in the literature that the notion of institution is the right level of abstraction for combining and comparing specification formalisms. We recall the relevant notions of multialgebras and institutions and show that multialgebras is an exact institution.

Since a nondeterministic constant $c :\to s$ denotes a subset of the carrier in a multialgebra, we can use multialgebraic constants to model unary predicates, it means that we easily can represent algebras with unary predicates in multialgebras, (it's also possible to encode algebras with general predicates in multialgebras, in a similar way as predicates are encoded in partial algebras). We exemplify this by constructing the actual embedding of institutions from membership algebras to multialgebras. The particular case when operations in a multialgebra have the empty result set gives straightforwardly a subinstitution of partial algebras. Hence multialgebras can combine the advances of total and partial algebras in one framework. As an example of this we develop a methodology for partiality handling where we start by transforming partial algebra specifications to multialgebra specifications and stepwise refine them with specific error recovery.

To reason about specifications one need a logical entailment system, we give a sound and complete reasoning system for multialgebras. The logic has two atoms: set inclusion, $\prec$ and element equality $\doteq$. The set inclusion $t \prec t'$ holds iff the interpretation of t is included in the interpretation of t', i.e. every possible value for t is a possible value for t'. In other words: the term t is not more nondeterministic than t'. The element equality $t \doteq t'$ holds iff there is a unique element e returned by the terms t and t': i.e. t and t' are deterministic.

Besides axiomatic descriptions of operations, the algebraic frameworks are well suited to combine specifications i.e. one introduces the concept of parameterized specifications. On the other hand one can combine models (algebras), i.e. one can model composition of programs as parameterized datatype specifications. We use the general notion of homomorphism in multialgebras to define parameterized programs essentially as a special kind of subalgebras. The general syntax gives us a presentation of parameterized datatype specifications.

The structure of the thesis is: We start each chapter by an introduction section, where we announce the results to be presented in the chapter. The introduction section also contains a discussion of our results compared with related work. After the introduction section follows the main results of the chapter. Each chapter also has a section with concluding remarks that summarizes the results of the chapter and addresses further work.

The thesis starts by giving the mathematical background to write specifications in chapter 1. We define the institution of multialgebras, and show that it is exact. We also compare multialgebras with other algebraic specification formalisms by use of different maps of institutions. In chapter 2 we develop a sound and complete Raisowa-Sikorski type of logic to reason about multialgebra specifications, this logic offers a canonical strategy for proving sentences, hence it should be well suited for implementation. We transform the Raisowa-Sikorski logic to a Gentzen style logic that is more convenient for doing proofs by hand. On the more practical level we develop a methodology that uses nondeterminism for partiality handling in chapter 3 and show how we can reuse partial algebra specifications and give them particular error recovery with multialgebras. We also give a technique for specifying parameterized datatypes with multialgebras in chapter 4, the technique generalizes the notion of persistency, i.e. we allow the parameter algebra to be a tight subalgebra of the parameterized algebra. We close the chapter by introducing refinement of parameterized datatypes, and we motivate our definitions by several examples. We end the thesis by giving some overall conclusions summarizing the results of the thesis and addresses the most interesting further work.

[The thesis mainly consists of material first presented in technical reports:

Chapter 1 most of this chapter is from the technical report [28]

Chapter 2 is based on the technical report [32]

Chapter 3 is based on the technical report [27], parts also published in [30]

Chapter 4 based on the technical report [29], also published in [31] and [33]]

Chapter 1

Multialgebras

In the first chapter we present the mathematical background for the thesis. We start by recalling the relevant notions about multialgebras and institutions in section 1.1. Multialgebras are a powerful algebraic framework for specification - primarily, but not exclusively, of nondeterministic behavior. A nondeterministic operation returns the set of all possible outcomes of the operation. Hence operations are interpreted as functions from the carrier to the powerset of the carrier. Our definitions of multialgebraic specifications follow the thesis of Walicki, [50], with small differences. In [50] operations are required to return only nonempty result set, while we also allows operations to return the empty set. It means that our definition of multialgebras is a direct generalization of partial algebras, as shown in section 1.4.1. Again following [50], we have two types of atomic formula for specification, set inclusion $t \prec t'$ and element equality $t \doteq t'$. Formulae used for writing specifications are sequents over atomic equalities and inclusions.

There is also a slight difference in our definition of variable assignment for empty sorts. For technical reasons, coming from the design of the logical reasoning system in chapter 2, we let evaluation of variables of an empty sort be the emptyset.

In the remaining part of the chapter we give some new general results about multialgebras that is used in the subsequent chapters.

The concept of institutions [18] has become the standard framework for presenting model-theoretic approaches to logic and, in particular, to algebraic specification. We prove that multialgebras form an institution, $\mathcal{MA}$ in section 1.2. In section 1.2.1 we show that the model functor for multialgebras $\mathsf{Mod}_{\mathcal{MA}}$ sends finite co-limits in **Th** (category of specifications) to limits in **Cat**, i.e. $\mathcal{MA}$ is an exact institution [40] (called institution with composable signatures in [48]). We also mention the well known fact that every exact institution satisfies the amalgamation lemma. The results are not used in this chapter but they form a basis for chapter 4 on structuring multialgebras and their specifications, in particular, on amalgamation and parameterized multialgebras. In study of parameterized specifications the co-limits (actually pushouts) are used for defining parameter instantiation. The exactness of the model functor (actually the amalgamation lemma) ensures that corresponding instantiation can be performed at the semantic level.

The category of multialgebras uses weak homomorphisms to relate multialgebras. In section 1.3 we recall the semantical properties for ensuring initial models for specifications from [22], we get a slightly weaker demand since we allow operations to return emptyset. We must stress that it is still an open question to find general conditions that ensure existence of initial models for multialgebra specifications. The recent work of [52] presents some promising categories with alternative definitions of homomorphisms between multialgebras, that may be worth further studies.

In section 1.4 we show the and we show the formal correspondence between the institution of multialgebras and other specification formalisms. In 1.4.1 multialgebras are related to partial algebras and in 1.4.2 to membership algebras, actually are the two later institutions are embedded into the former. We also indicate how we can transform models of partial algebra and membership algebra specifications to multialgebras and extended the corresponding multialgebra by e.g. nondeterminism.

The structure of the chapter is: We begin, in section 1.1 by introducing some notation and presenting the background definitions of multilagebras and collecting the relevant definitions and results about institutions. In section 1.2 we show that multialgebras form an institution $\mathcal{MA}$ and that it's exact in section 1.2.1. Section 1.3 studies the conditions for the existence of initial models in $\mathcal{MA}$. We compare multialgebras with other specification frameworks in section 1.4 by embedding partial algebras and membership algebras to multialgebras. We close the chapter by summarizing the contributions in section 4.5. Section 1.1 contains only old material, the rest of the sections present some new results about multialgebras.

1.1 Preliminaries

1.1.1 Notation

We use the notation $|\mathbf{C}|$ to denote the objects of a category $\mathbf{C}$. The same notation is used to denote the carrier $|A|$ of an algebra A. (This shouldn't cause any confusion.) Institutions are written with the script font $\mathcal{I}$, categories with bold **Cat**, and functors with Sans Serif Func.

Specifications form categories, hence we write **Spec** for ordinary specifications. Given an arbitrary specification **SP** we will sometimes write $\Sigma(\mathbf{SP})$ to denote its signature.

Sequences $s_1, \ldots, s_k$ will be often denoted by $\overline{s}$. Application of functions are then understood to not distribute over the elements, i.e., $f(\overline{s})$ denotes the term $f(s_1, \ldots, s_k)$. Occasionally, a sequence $s_1 \ldots s_k$ may be denoted by s^* – applications of functions are then understood to distribute over the elements, i.e., $f(s^*)$ denotes the sequence $(f(s_1), \ldots, f(s_k))$. We will denote the disjoint union of sets A, B by $A \uplus B$.

1.1.2 Algebraic Signatures

Signatures for multialgebras are the same as classical *algebraic signatures.*

Definition 1.1.1 *The category of signatures* **Sign** *has:*

- *signatures as objects: a signature* Σ *is a pair of sets* $(\mathbf{S}, \Omega)$ *of symbols for names of sorts and operations. Each operation symbol* $\omega \in \Omega$ *is a (k+2)-tuple:* $\omega : s_1 \times \cdots \times s_k \to s$*, where* $s_1, \ldots, s_k, s \in S$ *and* $k \geq 0$*.* ω *is the name of the operation,* $s_1 \times \ldots \times s_k$ *it's source and* s *its target. If* $k = 0$ *then an operation* $c :\to s$ *is called a constant of sort* s*.*
- *signature morphisms as arrows: a signature morphism* $\mu : \Sigma \to \Sigma'$ *is a pair* $\mu = (\mu_S, \mu_\Omega)$ *of (total) functions:* $\mu_S : S \to S'$*,* $\mu_\Omega : \Omega \to \Omega'$*, such that:* $\mu_\Omega(\omega : s_1 \times \cdots \times s_n \to s) = \omega' : \mu_S(s_1) \times \cdots \times \mu_S(s_n) \to \mu_S(s)$
- *Identities are the identity signature morphisms and morphisms are composed component-wise.*

Example 1.1.2 *Signature for the natural numbers:*

$$
\begin{array}{lll}
\textbf{sign Nat} = & & \\
& \mathbf{S}: & Nat \\
& \Omega: & zero :\to Nat \\
& & succ : Nat \to Nat
\end{array}
$$

The natural numbers has one sort called Nat*, a constant called zero and an unary operation called succ.*

Terms are defined in the standard way, and in the standard way we extend the signature morphism $\mu : \Sigma \to \Sigma'$ to terms, we use the notation $\mathcal{T}_{\Sigma,X}$ for the Σ terms with X as variables.

Definition 1.1.3 *Extension of a signature morphism* μ *to terms* $\tilde{\mu} : \mathcal{T}_{\Sigma,X} \to \mathcal{T}_{\Sigma',X'}$ *is defined by:*

- $\tilde{\mu}(x_s) = x_{\mu(s)}$*, for each variable* $x_s \in X_s$
- $\tilde{\mu}(c) = \mu(c)$
- $\tilde{\mu}(\omega(t_1, \ldots, t_n)) = \mu(\omega)(\tilde{\mu}(t_1), \ldots, \tilde{\mu}(t_n))$

In general variables can be renamed too, but (without loss of generality) we simplify the presentation. We will write $\mu(t)$ instead of $\tilde{\mu}(t)$, for terms $t \in \mathcal{T}_{\Sigma,X}$.

Algebraic signatures have all finite co-limits

It is well known that the category of algebraic signatures is co-complete, see e.g. [17]. We are merely restating here this standard fact. Limiting our attention to finite co-limits, it is sufficient to consider the existence of initial object, co-products (sums) and co-equalizers (see e.g. [3]). Since multialgebras use algebraic signatures all these results will also hold for multialgebraic signatures, i.e **Sign** from 1.1.1.

Fact 1.1.4 *The empty signature $\Sigma_\emptyset$ is initial in* **Sign**

Fact 1.1.5 *The sum of two signatures; $\Sigma + \Sigma'$ is the disjoint union (of sorts and operations), with the natural injections.*

Fact 1.1.6 *Given two signature morphisms $\mu, \nu : \Sigma \to \Sigma'$, let $\sim$ be the least equivalence on Σ' induced by the relation with components:*

- *Sorts: $\sim_{S'} = \{\langle \mu(s), \nu(s) \rangle : s \in \Sigma\}$,*
- *Operations $\sim_{\Omega'} = \{\langle \mu(\omega), \nu(\omega) \rangle : \omega \in \Omega\}$*

Then $\Sigma'/_\sim$ is a co-equalizer object, with canonical signature morphism $\iota : \Sigma' \to \Sigma'/_\sim$, and we have that $\mu;\iota = \nu;\iota$, by construction.

Note that if $\sigma : \Sigma' \to Z$ is a signature morphism such that $\mu;\sigma = \nu;\sigma$, then the kernel of σ has to include $\sim$, so the signature morphism $u_\sigma : \Sigma'/_\sim \to Z$, defined by $u_\sigma([s']_\sim) = \sigma(s')$ and $u_\sigma([\omega']_\sim) = \sigma(\omega')$ is the unique factorization arrow.

Fact 1.1.7 *[17] The category* **Sign** *of algebraic signatures has all (finite) co-limits.*

1.1.3 Multialgebras

We will now summarize the relevant notions about multialgebras (for an overview, see [55, 52]).

A multialgebra for a signature Σ is an algebra where operations may be set-valued.

Definition 1.1.8 *(Multialgebra) A multialgebra A for Σ is given by:*

- *a set s^A, the carrier set, for each sort symbol $s \in \mathbf{S}$*
- *a subset $c^A \in \mathcal{P}(s^A)$, for each constant, $c :\to s$*
- *an operation $\omega^A : {s_1}^A \times \cdots \times {s_k}^A \to \mathcal{P}(s^A)$ for each symbol $\omega : s_1 \times \cdots \times s_k \to s \in \Omega$, where $\mathcal{P}(s^A)$ denotes the power set of s^A. Composition of operations is defined by pointwise extension, i.e.,*
  $f^A(g^A(x)) = \bigcup_{y \in g^A(x)} f^A(y)$.

The carrier set of a multialgebra A is denoted by $|A|$ and the carrier set of the sort s is denoted by s^A. One sometimes demands that constants and operations are total [55, 53], i.e. never return empty set and take values only in $\mathcal{P}^+(s^A)$, the nonempty subsets of s^A, we will not make this assumption. An operation is called partial if it returns the empty set for some arguments. An operation returning more than one value for some arguments is called nondeterministic. So an operation that is neither partial nor nondeterministic is a function. In other words a function is a total deterministic operation.

We also generalize earlier works by allowing carrier sets to be empty. This generalization is crucial when we relate multialgebras with other formalisms in section 1.4, we will also use this in the completeness proof for the logic in chapter 2.

Note that for a constant $c \in \Omega$, c^A denotes a (sub)set of the carrier s^A. This allows one to use constants as predicates (as will be done, in chapter 3).

Example 1.1.9 *N and A below give two examples of Nat-multialgebras for the signature from example 1.1.2:*

$Nat^N = \mathbb{N}$	$Nat^A = \mathbb{N}$
$zero^N = 0$	$zero^A = pos$ *the set of positive integers*
$succ^N(x) = x + 1$, *for* $x \in \mathbb{N}$	$succ^A(x) = \mathbb{N}$, *for* $x \in \mathbb{N}$

The algebra N is the expected algebra of natural numbers, the algebra A illustrates that a nondeterministic constant may be a set and that the nondeterministic operation succ could give any natural number.

As homomorphisms of multialgebras, we will use weak homomorphisms (see [52] for alternative notions).

Definition 1.1.10 *Given two multialgebras A and B, a (weak) homomorphism $h : A \to B$ is a set of functions $h_s : s^A \to s^B$ for each sort $s \in \mathbf{S}$, such that:*

1. *$h_s(c^A) \subseteq c^B$, for each constant $c :\to s$*
2. *$h_s(\omega^A(a_1, \ldots, a_n)) \subseteq \omega^B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$, for each operation $\omega : s_1 \times \cdots \times s_n \to s \in \Omega$ and for all $a_i \in s_i^A$.*

Saying "homomorphism" we will mean weak homomorphism and by algebra will we mean multialgebra, unless something else is stated.

Definition 1.1.11 *The category of Σ-multialgebras,* $\mathbf{MAlg}_\Sigma$, *has Σ-multialgebras as objects and homomorphisms as arrows. The identity arrows are the identity homomorphisms and composition of arrows is the obvious composition of homomorphisms.*

We will now give the formulae used for multialgebra specifications:

Definition 1.1.12 *Given a signature Σ and a set of variables X we define the well formed formulae as the least set $\mathcal{F}_{\Sigma,X}$, such that:*

1. *Atoms, if $t, t' \in \mathcal{T}_{\Sigma,X}$, then:*
   - *$t \doteq t' \in \mathcal{F}_{\Sigma,X}$ (equality), t and t' denote the same one-element set.*
   - *$t \prec t' \in \mathcal{F}_{\Sigma,X}$ (inclusion), the set interpreting t is included in the set interpreting t'.*
2. *Composite formulae, if $\gamma, \phi \in \mathcal{F}_{\Sigma,X}$, then:*
   - *$\neg\gamma \in \mathcal{F}_{\Sigma,X}$, negation.*
   - *$\gamma \vee \phi \in \mathcal{F}_{\Sigma,X}$, disjunction.*

- $\gamma \wedge \phi \in \mathcal{F}_{\Sigma,X}$, *conjunction.*

The implication sign $\rightarrow$ is introduced in the usual way, i.e. $\gamma \rightarrow \phi \equiv \neg\gamma \vee \phi$. Note that formulae in specifications will often be restricted to the form: $a_1 \wedge \ldots \wedge a_n \rightarrow b_1 \vee \ldots \vee b_m$, where either $n > 0$ or $m > 0$ and each a_i and b_j is atomic. We will often drop the symbols $\wedge$ and $\vee$ in specifications, i.e. we write $a_1, \ldots, a_n \rightarrow b_1, \ldots, b_m$ instead of $a_1 \wedge \ldots \wedge a_n \rightarrow b_1 \vee \ldots \vee b_m$.

Given a set of variables X, we define an assignment α by:

Definition 1.1.13 *An assignment α to a multialgebra A is a function $\alpha : X \rightarrow |A| \uplus \{\emptyset\}$ where $\alpha(x_s) = \emptyset \iff s^A = \emptyset$.*

So an assignment assigns an element of the carrier to each variable of a nonempty sort. One could alternatively define assignment as a partial function, $\alpha : X \rightarrow |A|$, with domain the variables over nonempty sorts, as done in [6]. Our formulation gives immediately that any non-ground term with variables over empty sort will be empty, since operations in multialgebra applied to empty set yield empty set. Moreover our assignment definition ensures that $x \doteq x$ holds if and only if the carrier is nonempty, using the alternative assignment definition $x \doteq x$ is a tautology since there is no assignment to a empty sort. This the reason that they need quantifiers to express the predicate $x \doteq x$ in [6].

Definition 1.1.14 *An assignment α induces a unique interpretation $\overline{\alpha}(t)$ in a multialgebra A of any term $t \in \mathcal{T}_{\Sigma,X}$ as follows:*

- $\overline{\alpha}(x) = \{\alpha(x)\}$
- $\overline{\alpha}(c) = c^A$
- $\overline{\alpha}(\omega(t_1, \ldots, t_n)) = \bigcup_{a_i \in \overline{\alpha}(t_i)} \omega^A(a_1, \ldots, a_n)$

Keep in mind that variables are assigned not sets but individual elements of the carrier. We will write $\alpha(t)$ instead of $\overline{\alpha}(t)$.

A pair $\langle A, \alpha \rangle$, where A is a Σ-algebra and α an assignment for A, will be called a Σ-structure.

Satisfaction of formulae in a multialgebra is defined as follows:

Definition 1.1.15 *Let A be a Σ algebra. The satisfaction relation $\models$ is defined by:*

1. $\langle A, \alpha \rangle \models t \prec t' \iff \alpha(t) \subseteq \alpha(t')$, *where* $t, t' \in \mathcal{T}_{\Sigma,X}$
2. $\langle A, \alpha \rangle \models t \doteq t' \iff \alpha(t) = e = \alpha(t')$, *where e is an element of the carrier and* $t, t' \in \mathcal{T}_{\Sigma,X}$.
3. $\langle A, \alpha \rangle \models \neg\gamma \iff \langle A, \alpha \rangle \not\models \gamma$, *where* $\gamma \in \mathcal{F}_{\Sigma,X}$
4. $\langle A, \alpha \rangle \models \gamma \vee \phi \iff \langle A, \alpha \rangle \models \gamma$ *or* $\langle A, \alpha \rangle \models \phi$, *where* $\gamma, \phi \in \mathcal{F}_{\Sigma,X}$
5. $\langle A, \alpha \rangle \models \gamma \wedge \phi \iff \langle A, \alpha \rangle \models \gamma$ *and* $\langle A, \alpha \rangle \models \phi$, *where* $\gamma, \phi \in \mathcal{F}_{\Sigma,X}$

6. $A \models \gamma \iff \langle A, \alpha \rangle \models \gamma$ for all α, *where* $\gamma \in \mathcal{F}_{\Sigma,X}$

7. $A \models \Gamma \iff A \models \gamma$ for all $\gamma \in \Gamma$, *and* $\Gamma \subseteq \mathcal{F}_{\Sigma,X}$

Remark 1.1.16 *According to point 2 in definition 1.1.15, an equality may hold only if the carrier is non-empty. Given an algebra A, we have that:*

- $A \models \neg(x_s \doteq x_s) \iff s^A = \emptyset$
- $A \models x_s \doteq x_s \iff s^A \neq \emptyset$

Also, if $s^A = \emptyset$*, we have for any terms* t_s, t'_s*:*

- $A \models t_s \prec t'_s$ *and*
- $A \not\models \neg(t_s \prec t'_s)$

We introduce logical symbols, used in chapter 2 to construct reasoning systems for multialgebras, abbreviating the formulae stating that a carrier is empty or not.

Definition 1.1.17 *We define the symbols* $\mathcal{E}_s \equiv \neg(x_s \doteq x_s)$*, for any* $x_s \in X_s$*, and* $\neg\mathcal{E}_s \equiv x_s \doteq x_s$*, for any* $x_s \in X_s$*. By remark 1.1.16, for any algebra A:*

- $A \models \mathcal{E}_s \iff s^A = \emptyset$
- $A \models \neg\mathcal{E}_s \iff s^A \neq \emptyset$

Putting these definitions together, we show (in section 1.2) that the multialgebras form an institution $\mathcal{MA}$. Before that we recall the standard concepts of institution and some results which will be relevant for us in investigating the institution of multialgebras.

1.1.4 Institutions

Definition 1.1.18 *[18] An institution is a quadruple* $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$*, where:*

- **Sign** *is a category of signatures.*
- $\mathsf{Sen} : \mathbf{Sign} \to \mathbf{Set}$ *is a functor which associates a set of sentences to each signature.*
- $\mathsf{Mod} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ *is a functor which associates a category of models, whose morphisms are called* Σ*-morphisms, to each signature* Σ
- $\models$ *is a satisfaction relation – for each signature* Σ*, a relation* $\models_\Sigma \subseteq |\mathsf{Mod}(\Sigma)| \times \mathsf{Sen}(\Sigma)$*, such that the following satisfaction condition holds: for any* $M' \in \mathsf{Mod}(\Sigma'), \mu : \Sigma \to \Sigma', \phi \in \mathsf{Sen}(\Sigma)$

$$M' \models_{\Sigma'} \mathsf{Sen}(\mu)(\phi) \textit{ iff } \mathsf{Mod}(\mu)(M') \models_\Sigma \phi$$

The definition can be represented as the following diagram:

$$\begin{array}{cccc}
\Sigma & \mathsf{Mod}(\Sigma) & \models_\Sigma & \mathsf{Sen}(\Sigma) \\
\mu \big\downarrow & \big\uparrow \mathsf{Mod}(\mu) & & \big\downarrow \mathsf{Sen}(\mu) \\
\Sigma' & \mathsf{Mod}(\Sigma') & \models_{\Sigma'} & \mathsf{Sen}(\Sigma')
\end{array}$$

The following subsections review institution independent concepts and results which will be used in the later section.

Category of specifications

For $\Gamma \subseteq \mathsf{Sen}(\Sigma)$ and $\varphi \in \mathsf{Sen}(\Sigma)$ we write: $\Gamma \models_\Sigma \varphi$ iff $\forall M \in \mathsf{Mod}(\Sigma) : M \models_\Sigma \Gamma \Rightarrow M \models_\Sigma \varphi$. With this in mind we write $\Gamma^\bullet$ for the semantical consequences of Γ i.e. $\Gamma^\bullet = \{\varphi : \Gamma \models \varphi\}$

A *theory* (specification) in an institution is any pair $Th = (\Sigma, \Gamma)$ where $\Sigma \in |\mathbf{Sign}|$ and $\Gamma \subseteq \mathsf{Sen}(\Sigma)$. For a given institution $\mathcal{I}$, we have the corresponding category of theories $\mathbf{Th}$ with theories as objects and theory morphisms $\mu : (\Sigma, \Gamma) \to (\Sigma', \Gamma')$, where $\mu : \Sigma \to \Sigma'$, is a signature morphism such that: $\Gamma' \models_{\Sigma'} \mathsf{Sen}(\mu)(\Gamma)$. The models for the theory $Th = (\Sigma, \Gamma)$ is the full sub category $\mathsf{Mod}_\models(\Sigma, \Gamma)$ of $\mathsf{Mod}(\Sigma)$ where $M \in \mathsf{Mod}_\models(\Sigma, \Gamma)$ iff $M \models_\Sigma \varphi, \forall \varphi \in \Gamma$, we will write $\mathsf{Mod}(\Sigma, \Gamma)$ instead of $\mathsf{Mod}_\models(\Sigma, \Gamma)$. The satisfaction condition gives that $\mathsf{Mod}(\mu)(\mathsf{Mod}(\Sigma', \Gamma')) \subseteq \mathsf{Mod}(\Sigma, \Gamma)$, for each theory morphism $\mu : (\Sigma, \Gamma) \to (\Sigma', \Gamma') \in \mathbf{Th}$. This means that the functor Mod can be extended to a functor $\mathsf{Mod}_\models : \mathbf{Th}^{op} \to \mathbf{Cat}$. There is a canonic projection (forgetful) functor $\mathsf{Sign} : \mathbf{Th} \to \mathbf{Sign}$ and there is an embedding functor $\mathsf{th} : \mathbf{Sign} \to \mathbf{Th}$ defined by $\mathsf{th}(\Sigma) = (\Sigma, \emptyset)$. A theory morphism $\mu : (\Sigma, \Gamma) \to (\Sigma', \Gamma')$ is called axiom preserving if $\mu(\Gamma) \subseteq \Gamma'$. This defines the sub category $\mathbf{Th_0}$ with theories as objects and axiom preserving theory morphisms as morphisms.

Construction of co-limits of specifications

The following institution independent result ensures that the category of specifications has all co-limits if the signature category has. This result is used to create co-limits of specifications by first creating the co-limit for the corresponding signatures.

Theorem 1.1.19 *[18] The functor* $\mathsf{Sign} : \mathbf{Th} \to \mathbf{Sign}$ *reflects co-limits, in any institution* $\mathcal{I}$.

As a particular case, the theorem means that: Given specifications $\mathbf{X} = (\Sigma, \Phi)$, $\mathbf{X^1} = (\Sigma^1, \Phi^1)$, $\mathbf{X^2} = (\Sigma^2, \Phi^2)$, and specification morphisms $\mu_1 : \mathbf{X} \to \mathbf{X^1}$ and $\mu_2 : \mathbf{X} \to \mathbf{X^2}$. If the diagram to the left is a pushout of signatures than the diagram to the right is a pushout of specifications:

$$\begin{tikzcd} \Sigma \arrow[r, "\mu_1"] \arrow[d, "\mu_2"'] & \Sigma^1 \arrow[d, "{\mu'}_2"] \\ \Sigma^2 \arrow[r, "{\mu'}_1"'] & \Sigma' \end{tikzcd} \quad \overset{\mathsf{Sign}}{\Longleftarrow} \quad \begin{tikzcd} \mathbf{X} \arrow[r, "\mu_1"] \arrow[d, "\mu_2"'] & \mathbf{X^1} \arrow[d, "{\mu'}_2"] \\ \mathbf{X^2} \arrow[r, "{\mu'}_1"'] & \mathbf{X}' \end{tikzcd}$$

where $\mathbf{X}' = (\Sigma', \Phi')$ and $\Phi' = {\mu'}_1(\Phi^2) \cup {\mu'}_2(\Phi^1)$.

Continuity of Mod and amalgamation

Construction on specifications can be "carried over" to the respective constructions on their model classes provided that the Mod functor has some desired properties. Typical constructions on specifications are co-limits and the desired property of Mod is that it transforms co-limits in **Th** to limits in **Cat**.

Definition 1.1.20 *An institution $\mathcal{I}$ is*

1. semi exact *iff* **Sign** *has pushouts and* Mod *sends pushouts in* **Sign** *to pullbacks in* **Cat**,
2. exact *iff* **Sign** *has finite co-limits and* Mod *sends finite co-limits in* **Sign** *to limits in* **Cat**.

Of course, any exact institution is also semi exact. The importance of this notion is exemplified by the following lemma which indicates the construction for instantiation of parameterized specifications.

Lemma 1.1.21 *(Amalgamation Lemma).*
In any semi exact institution $\mathcal{I}$, for every pushout of signatures (on the left):

$$\begin{tikzcd} \Sigma \arrow[r, "\mu_1"] \arrow[d, "\mu_2"'] & \Sigma^1 \arrow[d, "{\mu'}_2"] \\ \Sigma^2 \arrow[r, "{\mu'}_1"'] & \Sigma' \end{tikzcd} \quad \overset{\mathsf{Mod}}{\Longrightarrow} \quad \begin{tikzcd} \mathsf{Mod}(\Sigma) & \mathsf{Mod}(\Sigma^1) \arrow[l, "\text{-}|_{\mu_1}"'] \\ \mathsf{Mod}(\Sigma_2) \arrow[u, "\text{-}|_{\mu_2}"] & \mathsf{Mod}(\Sigma') \arrow[l, "\text{-}|_{\mu'_1}"] \arrow[u, "\text{-}|_{\mu'_2}"'] \end{tikzcd}$$

we have that: for any two models $M_1 \in \mathsf{Mod}(\Sigma^1)$ and $M_2 \in \mathsf{Mod}(\Sigma^2)$ satisfying $M_1|_{\mu_1} = M_2|_{\mu_2}$, there exists a unique model $M' \in \mathsf{Mod}(\Sigma')$, such that $M'|_{{\mu'}_1} = M_2$ and $M'|_{{\mu'}_2} = M_1$.

The corresponding amalgamation property holds also for homomorphisms. In fact, the amalgamation lemma tells that the model class of a pushout Σ' of signatures along μ_1, μ_2 is a pullback (in **Cat**) of the respective morphisms $\text{-}|_{\mu_1}$, $\text{-}|_{\mu_2}$. The model M' is the *amalgamated union* of M_1 and M_2.

By theorem 1.1.19, the amalgamation lemma holds then also for pushouts of specifications, since these are constructed from pushouts of signatures.

1.1.5 Relations among institutions

To relate different institutions we will use *map of institutions* [36]. To define map of institutions we need the following definition:

Definition 1.1.22 *Given a functor* $\Phi : \mathbf{Th_0} \to \mathbf{Th_0'}$ *and a natural transformation* $\alpha : \mathsf{Sen} \Rightarrow \mathsf{Sen}' \circ \Phi$, Φ *is* α*-sensible iff:*

- *There is a functor* $\Phi^{\diamond} : \mathbf{Sign} \to \mathbf{Sign}'$ *such that* $\mathsf{sign}' \circ \Phi = \Phi^{\diamond} \circ \mathsf{sign}$
- $(\Gamma')^{\bullet} = (\emptyset'_{\Sigma} \cup \alpha_{\Sigma}(\Gamma))^{\bullet}$

Where we denote the set of axioms induced by $\Phi(\Sigma, \emptyset)$ *by* $\emptyset'_{\Sigma}$.

Definition 1.1.23 *Given two institutions* $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$ *and* $\mathcal{I}' = (\mathbf{Sign}', \mathsf{Sen}', \mathsf{Mod}', \models')$, *a map of institutions is a triple* $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ *where:*

- $\alpha : \mathsf{Sen} \Longrightarrow \mathsf{Sen}' \circ \Phi$ *is a natural transformation.*
- $\Phi : \mathbf{Th_0} \to \mathbf{Th_0'}$ *is an* α*-sensible functor*
- *a natural transformation* $\beta : \mathsf{Mod}' \circ \Phi^{op} \Longrightarrow \mathsf{Mod}$

such that for each $\phi \in \mathsf{Sen}(\Sigma)$ *and* $M' \in \mathsf{Mod}'(\Phi(\Sigma, \emptyset))$ *the satisfaction condition:*

$$M' \models_{sign'(\Phi(\Sigma,\emptyset))} \alpha_{\Sigma}(\phi) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') \models_{\Sigma} \phi$$

The condition from the above definition corresponds to the following commuting diagrams:

$$\begin{tikzcd}
(\Sigma,\Gamma)^{op} \arrow[r, "\Phi^{op}"] \arrow[d, "\mathsf{Mod}"'] & (\Sigma',\Gamma')^{op} \arrow[d, "\mathsf{Mod}'"] \\
\mathbf{Cat} & \mathbf{Cat} \arrow[l, Rightarrow, "\beta"]
\end{tikzcd}
\qquad
\begin{tikzcd}
(\Sigma,\Gamma) \arrow[r, "\Phi"] \arrow[d, "\mathsf{Sen}"'] & (\Sigma',\Gamma') \arrow[d, "\mathsf{Sen}'"] \\
\mathbf{Set} \arrow[r, Rightarrow, "\alpha"] & \mathbf{Set}
\end{tikzcd}$$

Definition 1.1.24 *An embedding of institutions [37] is a map of institutions* $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$, *where the functor* $\beta_T : \mathsf{Mod}'(\Phi(T)) \to \mathsf{Mod}(T)$ *is an equivalence of categories for each* $T \in \mathbf{Th}_{\mathcal{I}}$. *We will use the notation* $(\Phi, \alpha, \beta) : \mathcal{I} \hookrightarrow \mathcal{I}'$ *to denote an embedding of institutions.*

An α-extension to theories of a functor $\Phi : \mathbf{Sign} \to \mathbf{Th_0'}$ is a functor $\Phi : \mathbf{Th_0} \to \mathbf{Th_0'}$ mapping the theory $\mathbf{Th} = (\Sigma, \Gamma)$ to the theory $\Phi(\mathbf{Th})$ with signature $\Phi(\Sigma)$ and with axioms $\Phi(\Sigma) \cup \alpha_{\Sigma}(\Gamma)$, for a given natural transformation $\alpha : \mathsf{Sen} \to \mathsf{Sen}' \circ \Phi$.

Definition 1.1.25 *A map of institutions* (Φ, α, β) *is:*

- (α) simple *iff* Φ *is a* α*-extension to theories of a functor* $\mathsf{F} : \mathbf{Sign} \to \mathbf{Th_0'}$, *i.e.* Φ *maps axioms to axioms.*

- (α) plain *iff* Φ *is a* α*-extension to theories of a functor* $\mathsf{F} : \mathbf{Sign} \to \mathbf{Th_0'}$ *that maps* Σ *to* $(\Sigma', \emptyset)$*, i.e.* Φ *maps signatures to signatures.*

Definition 1.1.26 *A subinstitution [36] is a map of institutions* $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ *that is plain, with* Φ *faithful and injective on objects,* α *injective and with* β *a natural isomorphism.*

We will use *institutioin transformation* [35] as a formalisiation of model class extension.

Definition 1.1.27 *Given institutions* $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$ *and* $\mathcal{I}' = (\mathbf{Sign}', \mathsf{Sen}', \mathsf{Mod}', \models')$*, an institution transformation is a triple* $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ *where:*

- $\alpha : \mathsf{Sen} \Longrightarrow \mathsf{Sen}' \circ \Phi$ *is a natural transformation.*
- $\Phi : \mathbf{Th_0} \to \mathbf{Th_0'}$ *is an* α *plain,* α*-sensible functor*
- *a natural transformation* $\beta : \mathsf{Mod} \Longrightarrow \mathsf{Mod}' \circ \Phi^{op}$

such that for each $\phi \in \mathsf{Sen}(\Sigma)$ *and* $M \in \mathsf{Mod}(\Sigma, \emptyset)$ *the satisfaction condition:*

$$\beta_{(\Sigma,\emptyset)}(M) \models_{sign'(\Phi(\Sigma,\emptyset))} \alpha_\Sigma(\phi) \text{ iff } M \models_\Sigma \phi$$

1.2 Institution of multialgebras, $\mathcal{MA}$

We now define and prove that the multialgebras form an institution $\mathcal{MA}$ and that this institution is exact (subsection 1.2.1).

First we apply the standard concept of reduct to multialgebras.

Definition 1.2.1 *Let* $\mu : \Sigma \to \Sigma'$ *be a signature morphism.*

- *Reduct of an algebra:*
  - *The* μ*-reduct of a* Σ'*-multialgebra* A'*, is the* Σ*-multialgebra* $A'|_\mu$ *defined by:*

$$\begin{array}{ll} s^{A'|_\mu} = \mu(s)^{A'} & \text{, for all } s \in S, \\ \omega^{A'|_\mu} = \mu(\omega)^{A'} & \text{, for all } \omega \in \Omega, \end{array}$$

- *Reduct of assignment:*
  - *For a set of variables* X*,* A' *a* Σ' *algebra and* $\alpha' : \mu(X) \to A'$ *an assignment for* A'*, the* μ*-reduct of* α'*,* $\alpha'|_\mu : X \to A'|_\mu$ *is defined by:*

$$(\alpha'|_\mu)_s(x) = \alpha'_{\mu(s)}(\mu(x))$$

- *Reduct of a homomorphism:*

– *The μ reduct of a weak Σ' homomorphism $h' : A' \to B'$, is the weak Σ homomorphism $h'|_\mu : A'|_\mu \to B'|_\mu$ defined by:*

$$(h'|_\mu)_s = h'_{\mu(s)}$$

If one allows possible renaming of variables X along the signature morphisms μ, the definition would be entirely analogous – we omit this technicality.

Fact 1.2.2 *The reduct of a multialgebra is a multialgebra, the reduct of an assignment is an assignment to the corresponding reduct algebra, the reduct of a weak homomorphism is a weak homomorphism between the reduct of two algebras.*

The proof of this fact is analogous to the classical case.

We are now ready to define the model functor $\mathsf{Mod}_{\mathcal{MA}} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ which maps each signature $\Sigma \in |\mathbf{Sign}|$ to the category of all Σ-multialgebras $\mathbf{MAlg}_\Sigma$.

Definition 1.2.3 *The functor* $\mathsf{Mod}_{\mathcal{MA}} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ *is defined by:*

- *objects:* $\mathsf{Mod}_{\mathcal{MA}}(\Sigma) = \mathbf{MAlg}_\Sigma$
- *arrows:* $\mathsf{Mod}_{\mathcal{MA}}(\mu : \Sigma \to \Sigma') = \mathsf{Mod}_{\mathcal{MA}\mu} : \mathbf{MAlg}_{\Sigma'} \to \mathbf{MAlg}_\Sigma$, *where the functor* $\mathsf{Mod}_{\mathcal{MA}\mu}$ *is given by:*
  1. $\mathsf{Mod}_{\mathcal{MA}\mu}(A') = A'|_\mu$
  2. $\mathsf{Mod}_{\mathcal{MA}\mu}(h') = h'|_\mu$

Lemma 1.2.4 *(Reduct theorem) If $\mu : \Sigma \to \Sigma'$ is a signature morphism, X a set of variables, t a Σ term, A' a Σ' algebra and $\alpha' : \mu(X) \to A'$ an assignment for A' then we have that:*

$$\alpha'|_\mu(t)^{A'|_\mu} = \alpha'(\mu(t))^{A'}$$

Proof. The proof goes by induction on the complexity of the term t.

1. $t = x \in X_s$.

$$\begin{aligned}
& \alpha'|_\mu(x)^{A'|_\mu} \\
= \quad & \{\text{ assignment}\} \\
& (\alpha'|_\mu)_s(x) \\
= \quad & \{\text{ def. } \alpha'|_\mu\} \\
& (\alpha'_{\mu(s)})(\mu(x)) \\
= \quad & \{\text{ assignment}\} \\
& \alpha'(\mu(x))^{A'}
\end{aligned}$$

2. $t = c, (c :\to s)$

$\alpha'|_\mu(c)^{A'|_\mu}$
$= \quad \{$ no assignment for constants$\}$
$c^{A'|_\mu}$
$= \quad \{$ def. $A'|_\mu\}$
$\mu(c)^{A'}$
$= \quad \{$ no assignment, for constants$\}$
$\alpha'(\mu(c))^{A'}$

3. $t = \omega(t_1, \ldots, t_n), (\omega : s_1 \times \cdots \times s_n \to s)$

$\alpha'|_\mu(\omega(t_1, \ldots, t_n)^{A'|_\mu}$
$= \quad \{$ assignment on function$\}$
$\bigcup_{a_i \in \alpha'|_\mu(t_i)} \omega^{A'|_\mu}(a_1, \ldots, a_n)$
$= \quad \{$ ind. hyp. and def reduct$\}$
$\bigcup_{a_i \in \alpha'(\mu(t_i))} \mu(\omega)^{A'}(a_1, \ldots, a_n)$
$= \quad \{$ assignment on function$\}$
$\alpha'(\mu(\omega)(\mu(t_1), \ldots, \mu(t_n))^{A'}$

$\square$

The following lemma leads immediately to the satisfaction condition.

Lemma 1.2.5 *For any signature morphism $\mu : \Sigma \to \Sigma'$ and Σ' algebra A', given assignment*

- $\alpha : X \to A'|_\mu$, *define* $\alpha' : \mu(X) \to A'$ *by:* $\alpha'_{\mu(s)}(\mu(x)) = \alpha_s(x)$, *and*
- $\alpha' : \mu(X) \to A'$, *define* $\alpha : X \to A'|_\mu$ *by:* $\alpha_s(x) = \alpha'_{\mu(s)}(\mu(x))$, *i.e.* $\alpha = \alpha'|_\mu$

Then for any Σ-formula φ and for any Σ' multialgebra A' we have that:

$$(A'|_\mu) \models_\alpha \varphi \Longleftrightarrow A' \models_{\alpha'} \mu(\varphi)$$

Proof. By induction on the formulas:

1. For atomic formulae $t_1 \doteq t_2$ and $t_1 \prec t_2$, the reduct theorem gives $\alpha'|_\mu(t_i)^{A'|_\mu} = \alpha'(\mu(t_i))^{A'}$, which yields the result.

2. Composite formulae:

   - $\varphi = \neg\gamma : \gamma \in \mathcal{F}_{\Sigma,X}$

     $(A'|_\mu) \models_\alpha \neg\gamma$
     $\Longleftrightarrow \{$ satisfaction relation$\}$
     $(A'|_\mu) \not\models_\alpha \gamma$
     $\Longleftrightarrow \{$ IH$\}$
     $A' \not\models_{\alpha'} \mu(\gamma)$
     $\Longleftrightarrow \{$ satisfaction relation$\}$
     $A' \models_{\alpha'} \mu(\neg\gamma)$

- $\varphi = \gamma \vee \phi : \gamma, \phi \in \mathcal{F}_{\Sigma,X}$

$$\begin{aligned}
&(A'|_\mu) \models_\alpha \gamma \vee \phi \\
\Longleftrightarrow &\{\text{ satisfaction relation}\} \\
&(A'|_\mu) \models_\alpha \gamma \text{ or } (A'|_\mu) \models_\alpha \phi \\
\Longleftrightarrow &\{\text{ IH}\} \\
&A' \models_{\alpha'} \mu(\gamma) \text{ or } A' \models_{\alpha'} \mu(\phi) \\
\Longleftrightarrow &\{\text{ satisfaction relation}\} \\
&A' \models_{\alpha'} \mu(\gamma \vee \phi)
\end{aligned}$$

- $\varphi = \gamma \wedge \phi : \gamma, \phi \in \mathcal{F}_{\Sigma,X}$

$$\begin{aligned}
&(A'|_\mu) \models_\alpha \gamma \wedge \phi \\
\Longleftrightarrow &\{\text{ satisfaction relation}\} \\
&(A'|_\mu) \models_\alpha \gamma \text{ and } (A'|_\mu) \models_\alpha \phi \\
\Longleftrightarrow &\{\text{ IH}\} \\
&A' \models_{\alpha'} \mu(\gamma) \text{ and } A' \models_{\alpha'} \mu(\phi) \\
\Longleftrightarrow &\{\text{ satisfaction relation}\} \\
&A' \models_{\alpha'} \mu(\gamma \wedge \phi)
\end{aligned}$$

$\square$

Lemma 1.2.6 *(Satisfaction condition) The satisfaction condition is fulfilled for multialgebras, i.e. for any signature morphism $\mu : \Sigma \to \Sigma'$, for any Σ-formula φ and for any Σ' multialgebra A' we have that:*

$$(A'|_\mu) \models_\Sigma \varphi \Longleftrightarrow A' \models_{\Sigma'} \mu(\varphi)$$

Proof. Let φ be an arbitrary formula.
"$\Leftarrow$": let $\alpha : X \to A'|_\mu$ be arbitrary and let α' be as in lemma 1.2.5. Then, by assumption, $A' \models_{\Sigma'} \mu(\varphi)$ and, in particular, $A' \models_{\alpha'} \mu(\varphi)$. By lemma 1.2.5, $(A'|_\mu) \models_\alpha \varphi$. Since α was arbitrary, we obtain $(A'|_\mu) \models_\Sigma \varphi$.
"$\Rightarrow$": let $\alpha' : \mu(X) \to A'$ be arbitrary, and let α be as in lemma 1.2.5. By assumption, $(A'|_\mu) \models_\Sigma \varphi$, in particular, $(A'|_\mu) \models_\alpha \varphi$. By lemma 1.2.5, $A' \models_{\alpha'} \mu(\varphi)$. Since α' was arbitrary, we obtain $A' \models_{\Sigma'} \mu(\varphi)$. $\square$

Finally, the functor assigning to each signature the set of sentences is defined as in definition 1.1.12:

Definition 1.2.7 *The sentences functor* $\mathsf{Sen}_{\mathcal{MA}} : \mathbf{Sign} \to \mathbf{Set}$ *is given by:*

- *objects:* $\mathsf{Sen}_{\mathcal{MA}}(\Sigma) =$ *the set of all Σ formulae (def. 1.1.12)*
- $\mathsf{Sen}_{\mathcal{MA}}(\mu : \Sigma \to \Sigma') = \mathsf{Sen}_{\mathcal{MA}\mu} : \mathsf{Sen}_{\mathcal{MA}} \to \mathsf{Sen}_{\mathcal{MA}}(\Sigma')$ *defined by:*
  1. $\mathsf{Sen}_{\mathcal{MA}\mu}(t \doteq t') = \tilde{\mu}(t) \doteq \tilde{\mu}(t')$
  2. $\mathsf{Sen}_{\mathcal{MA}\mu}(t \prec t') = \tilde{\mu}(t) \prec \tilde{\mu}(t')$

3. $\mathsf{Sen}_{\mathcal{MA}_\mu}(\neg\gamma) = \neg(\tilde{\mu}(\gamma))$
4. $\mathsf{Sen}_{\mathcal{MA}_\mu}(\gamma \vee \phi) = \tilde{\mu}(\gamma) \vee \tilde{\mu}(\phi)$
5. $\mathsf{Sen}_{\mathcal{MA}_\mu}(\gamma \wedge \phi) = \tilde{\mu}(\gamma) \wedge \tilde{\mu}(\phi)$

With these definitions, lemma 1.2.6 yields the following:

Fact 1.2.8 *The multialgebras form the institution $\mathcal{MA}$ with:*

- *the category* **Sign** *as signatures (def. 1.1.1),*
- *the model functor* $\mathsf{Mod}_{\mathcal{MA}}$ *(def. 1.2.3),*
- *the sentence functor* $\mathsf{Sen}_{\mathcal{MA}}$ *(def. 1.2.7),*
- $\models$ *from def. 1.1.15 as the satisfaction relation.*

1.2.1 $\mathcal{MA}$ is an exact institution

As recalled in section 1.1.2 the category of algebraic signatures is co-complete. Using the constructions of the required co-limits of algebraic signatures from 1.1.2, we now show that $\mathcal{MA}$ is an exact institution – that the model functor for multialgebras $\mathsf{Mod}_{\mathcal{MA}} : \mathbf{Sign}^{op} \rightarrow \mathbf{Cat}$ is continuous, i.e. it maps finite co-limits in **Sign** into limits in **Cat**. (Note that this is different from showing that the category $\mathsf{Mod}_{\mathcal{MA}}(\Sigma)$ has all limits (resp. co-limits), which is claimed in [51].)

Lemma 1.2.9 *The model of the empty signature is the unit category, #, that is final in* **Cat**.

Proof. The model of the empty signature, $\Sigma_\emptyset$ is an algebra with no carrier (and no operations), i.e. $\mathsf{Mod}_{\mathcal{MA}}(\Sigma_\emptyset) = \#$. There is only one function (homomorphism) $h : \# \rightarrow \#$ – the identity homomorphism. This is obviously a final object in **Cat**. □

Lemma 1.2.10 Mod *sends sums to products:*

$$\begin{tikzcd}
\Sigma \arrow[dr, "\iota_\Sigma"'] & & \Sigma' \arrow[dl, "\iota_{\Sigma'}"] & \overset{\mathsf{Mod}}{\Longrightarrow} & \mathsf{Mod}_{\mathcal{MA}}(\Sigma) & & \mathsf{Mod}_{\mathcal{MA}}(\Sigma') \\
& \Sigma + \Sigma' & & & & \mathsf{Mod}_{\mathcal{MA}}(\Sigma + \Sigma') \arrow[ul, "|_{\iota_\Sigma}"] \arrow[ur, "|_{\iota_{\Sigma'}}"'] &
\end{tikzcd}$$

Proof. Since the reduct $\_|_\mu$ is a functor for any signature morphism μ the diagram to the right is a cone in **Cat** – we have to show that it is a product cone.

Suppose that $(\mathbf{C}, \mathsf{F} : \mathbf{C} \to \mathsf{Mod}_{\mathcal{MA}}(\Sigma), \mathsf{G} : \mathbf{C} \to \mathsf{Mod}_{\mathcal{MA}}(\Sigma'))$ is a cone in **Cat**.

$$\begin{tikzcd}
 & \mathbf{C} \arrow[d, dashed, "!u_{(\mathsf{F},\mathsf{G})}"] \arrow[ddl, bend right, "\mathsf{F}"'] \arrow[ddr, bend left, "\mathsf{G}"] & \\
 & \mathsf{Mod}_{\mathcal{MA}}(\Sigma + \Sigma') \arrow[dl, "|_{\iota_\Sigma}"] \arrow[dr, "|_{\iota_{\Sigma'}}"'] & \\
\mathsf{Mod}_{\mathcal{MA}}(\Sigma) & & \mathsf{Mod}_{\mathcal{MA}}(\Sigma')
\end{tikzcd}$$

Given two multialgebras $A \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma)$ and $A' \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma')$, we get an algebra $A \oplus A' \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma + \Sigma')$ by taking the Σ-part from A and the Σ'-part from A' – it is defined as follows: for any sort symbol $s \in \Sigma : s^{A \oplus A'} = s^A$ (and $s \in \Sigma' : s^{A \oplus A'} = s^{A'}$), and for any operation symbol $f \in \Sigma : f^{A \oplus A'} = f^A$ (and $f \in \Sigma' : f^{A \oplus A'} = f^{A'}$). This works because $\Sigma + \Sigma'$ is disjoint union.

Likewise, given a Σ-homomorphism $h : A \to B$ and a Σ'-homomorphism $h' : A' \to B'$, the $\Sigma \oplus \Sigma'$-homomorphism $h \oplus h' : A \oplus A' \to B \oplus B'$, is defined by $(h \oplus h')_s = h_s$ if $s \in \Sigma$, and h'_s otherwise (when $s \in \Sigma'$). Then $\oplus$ yields the unique objects/morphisms satisfying:

$$\begin{array}{c} (A \oplus A')|_{\iota_\Sigma} = A \quad \textit{and} \quad (A \oplus A')|_{\iota_{\Sigma'}} = A' \\ (h \oplus h')|_{\iota_\Sigma} = h \quad \textit{and} \quad (h \oplus h')|_{\iota_{\Sigma'}} = h' \end{array} \tag{1.1}$$

We define the functor $u_{(\mathsf{F},\mathsf{C})} : \mathbf{C} \to \mathsf{Mod}_{\mathcal{MA}}(\Sigma + \Sigma')$ by $u_{(\mathsf{F},\mathsf{C})}(C) = \mathsf{F}(C) \oplus \mathsf{G}(C)$ (and analogously for morphisms in $\mathbf{C}$). It is a factorization, i.e., $u_{(\mathsf{F},\mathsf{G})}; |_{\iota_\Sigma} = \mathsf{F}$, similarly for $|_{\iota_{\Sigma'}}$ and G.

$u_{(\mathsf{F},\mathsf{G})}$ is unique since each pair of algebras (resp. homomorphisms) $A \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma)$ and $A' \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma')$, has a unique pre-image $(A \oplus A') \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma + \Sigma')$, satisfying (1.1).

Thus the image of a co-product diagram from **Sign** is a product diagram in **Cat**. □

Lemma 1.2.11 Mod *sends co-equalizers to equalizers.*

Proof. Let $\mu, \nu : \Sigma \to \Sigma'$ be two morphisms in **Sign** and $\Sigma'/_\sim$, $\iota : \Sigma' \to \Sigma'/_\sim$ their co-equalizer. We have to show that $\mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim)$, $|_\iota$ is an equalizer for $|_\mu$ and $|_\nu$, in **Cat**. So assume that for any σ'' such that $\mu; \sigma'' = \nu; \sigma''$, there is a unique $u_{\sigma''}$ with $\sigma'' = \iota; u_{\sigma''}$:

$$\begin{tikzcd}
\Sigma \arrow[r, bend left, "\mu"] \arrow[r, bend right, "\nu"'] & \Sigma' \arrow[r, "\iota"] \arrow[dr, "\sigma''"'] & \Sigma'/_\sim \arrow[d, dashed, "u_{\sigma''}"] \\
 & & \Sigma''
\end{tikzcd}$$

Let $\mathbf{C}$ and $\mathsf{F} : \mathbf{C} \to \mathsf{Mod}_{\mathcal{MA}}(\Sigma')$ be arbitrary in $\mathbf{Cat}$ such that $\mathsf{F}; |_\mu = \mathsf{F}; |_\nu$. We have to show the existence of a unique $u_\mathsf{F} : \mathbf{C} \to \mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim)$ satisfying $u_\mathsf{F}; |_\iota = \mathsf{F}$, i.e. the following diagram commutes:

$$\begin{tikzcd}
\mathbf{C} \arrow[d, dashed, "!u_\mathsf{F}"'] \arrow[dr, "\mathsf{F}"] & & \\
\mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim) \arrow[r, "|_\iota"'] & \mathsf{Mod}_{\mathcal{MA}}(\Sigma') \arrow[r, shift left, "|_\mu"] \arrow[r, shift right, "|_\nu"'] & \mathsf{Mod}_{\mathcal{MA}}(\Sigma)
\end{tikzcd}$$

1. First, we show that $|_\iota; |_\mu = |_\iota; |_\nu$, i.e. $\mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim)$ is a cone.

   Let $A'' \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim)$ be arbitrary and let s be any sort symbol in Σ. We have $s^{(A''|_\iota)|_\mu} = \mu(s)^{(A''|_\iota)} = \iota(\mu(s))^{A''}$, and similarly $s^{(A''|_\iota)|_\nu} = \nu(s)^{(A''|_\iota)} = \iota(\nu(s))^{A''}$. But since ι is equalizing μ and ν, we have $\iota(\mu(s)) = \iota(\nu(s))$, so that $s^{(A''|_\iota)|_\mu} = \iota(\mu(s))^{A''} = \iota(\nu(s))^{A''} = s^{(A''|_\iota)|_\nu}$. In the same way, we show the equality $\omega^{(A''|_\iota)|_\mu} = \iota(\mu(\omega))^{A''} = \iota(\nu(\omega))^{A''} = \omega^{(A''|_\iota)|_\nu}$ for any operation symbol $\omega \in \Sigma$.

2. We now show that $\mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim)$ is a limit cone.

   (a) Given an $A' \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma')$ with $A'|_\mu = A'|_\nu$ we construct an $A^- \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim)$ satisfying $A^-|_\iota = A'$.

   Since ι is surjective, any symbol $x'' \in \Sigma'/_\sim$ is in its image, i.e., $x'' = \iota(x')$ for some $x' \in \Sigma'$. We then let $\iota(x')^{A^-} = x'^{A'}$. This is in fact well defined algebra. For suppose that there are two different symbols $s' \neq t' \in \Sigma'$ such that $\iota(s') = \iota(t') = x''$. Then, from the construction of ι and $\Sigma'/_\sim$, we know that there exists an $x \in \Sigma$ such that $s' = \mu(x)$ and $t' = \nu(x)$. But then, since $A'|_\mu = A'|_\nu$, we obtain the middle of the following equalities: $s'^{A'} = x^{A'|_\mu} = x^{A'|_\nu} = t'^{A'}$. Thus $\iota(s') = \iota(t') \Rightarrow s'^{A'} = t'^{A'}$, and A^- is well defined.

   Obviously, $A^-|_\iota = A'$, since for each symbol $x' \in \Sigma'$ we have $x'^{A^-|_\iota} = \iota(x')^{A^-} = x'^{A'}$.

   (b) In fact A^- is the unique $\Sigma'/_\sim$-algebra satisfying $A^-|_\iota = A'$.

   For if $B'' \in \mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim)$ is such that $B''|_\iota = A'$ then we must have for any symbol $\iota(x') = x'' \in \Sigma'/_\sim : x''^{B''} = \iota(x')^{B''} = x'^{B''|_\iota} = x'^{A'} = x'^{A^-|_\iota} = \iota(x')^{A^-} = x''^{A^-}$, i.e., $B'' = A^-$.

   (c) We extend the definition of A^- from 2a. to homomorphisms between the respective objects. That is, for a (homo)morphism $h' : A' \to B'$ in $\mathsf{Mod}_{\mathcal{MA}}(\Sigma')$ where both $A'|_\mu = A'|_\nu$ and $B'|_\mu = B'|_\nu$, $h^- : A^- \to B^-$ is a (homo)morphism in $\mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim)$ defined by $h^-_{\iota(s')}(a) = h'_{s'}(a)$ for all sort symbols $\iota(s') = s'' \in \Sigma'/_\sim$ and $a \in s'^{A'}$. Obviously, this is a unique h^- such that $h|_\iota = h'$.

   (d) Now, given an $\mathsf{F} : \mathbf{C} \to \mathsf{Mod}_{\mathcal{MA}}(\Sigma')$ satisfying $\mathsf{F}; |_\mu = \mathsf{F}; |_\nu$, we define $u_\mathsf{F} : \mathbf{C} \to \mathsf{Mod}_{\mathcal{MA}}(\Sigma'/_\sim)$:

- for any object $C \in \mathbf{C} : u_{\mathsf{F}}(C) = (\mathsf{F}(C))^{-}$ (as in point 2a.)
- for any morphism $h : C \to D \in \mathbf{C} : u_{\mathsf{F}}(h) = (\mathsf{F}(h))^{-}$ (as in point 2c.)

It is trivial to verify that u_{F} is a functor and, indeed, one that makes $u_{\mathsf{F}}; |_{\iota} = \mathsf{F}$. By uniqueness of A^{-} and h^{-}, this is also a unique functor satisfying this equality.

□

Summing up we get the following result.

Proposition 1.2.12 $\mathsf{Mod}_{\mathcal{MA}}$ *is a finitely continuous functor.*

Proof. We have that $\mathsf{Mod}_{\mathcal{MA}}$ is finitely continous on signatures by lemma 1.2.9, lemma 1.2.10 and lemma 1.2.11. So the result follows by teorem 1.1.19. □

Corollary 1.2.13 $\mathcal{MA}$ *is an exact institution.*

Corollary 1.2.14 *The amalgamation lemma holds for* $\mathcal{MA}$*.*

1.3 Initial models for $\mathcal{MA}$ specifications

In general, $\mathcal{MA}$ specifications do not possess initial models [22, 55]. The most powerful logic that admits initial models is Horn logic, [34], so the first restriction would be to consider a sub-institution of multialgebras with Horn clauses over atoms, $\mathcal{MAH}$. However, as showed in [22], even atomic specifications in $\mathcal{MA}$ don't always have initial models.

Using essentially the same construction as in [22], we will state sufficient semantic condition for the existence of initial models – also for specifications using disjunctive axioms (subsection 1.3.1). Then we will review Hussmann's construction in our framework obtaining conditions ensuring the existence of initial models for atomic specifications (subsection 1.3.2).

The results of this section are of relatively limited value and the problem of identifying a satisfactory general sub-institution of $\mathcal{MA}$ admitting initial models is still open.

1.3.1 The general construction of initial structure

In [22] Hussmann gives sufficient conditions to ensure the existence of initial models when multialgebras are restricted to non empty interpretation of operations and with only atomic formula as axioms. The carrier of the initial model contains all and only deterministic ground terms. We first review this construction (some lemmata and proofs are different from [22]) which leads to a general result showing that it yields an initial model *whenever* it actually belongs to the model class (proposition 1.3.12).

Definition 1.3.1 *[22] For a specification $SP = (\Sigma, \Gamma)$, the (deterministic based) term structure $T(SP)$ is defined by:*

- $s^{T(SP)} = \{t_s : t_s \in \mathcal{T}_\Sigma \wedge SP \models t_s \doteq t_s\}$, *for* $s \in S$.
- *for* $f : \overline{s} \to s$, *the operation* $f^{T(SP)} : \overline{s}^{T(SP)} \to \mathcal{P}(s^{T(SP)})$ *is defined by*

  $f^{T(SP)}(\overline{t}) = \{t'_s : t'_s \in \mathcal{T}_\Sigma \wedge SP \models t'_s \doteq t'_s \wedge SP \models t'_s \prec f(\overline{t})\}$,

The first thing is to ensure that this construction yields a multialgebra. Two conditions on specifications from [22] to this goal are: DET-completeness and DET-additivity. The former requires that for every ground term t, there exists at least one deterministic term included in t. Since we are allowing empty set interpretation of terms, this condition is not needed in our case. The second condition is as follows:

Definition 1.3.2 *[22] A specification $SP = (\Sigma, \Gamma)$ is* DET-additive iff:

$$\forall f : \overline{s} \to s \in \Omega \ : \forall \overline{t}, t \in \mathcal{T}_\Sigma : SP \models t \prec f(\overline{t}) \wedge SP \models t \doteq t \ \Rightarrow$$

$$\exists \overline{t'} \in \mathcal{T}_\Sigma : SP \models t \prec f(\overline{t'}) \wedge SP \models \overline{t'} \prec \overline{t} \wedge SP \models \overline{t'} \doteq \overline{t'}$$

The condition says: if a deterministic value t is produced by an application of an operation f to some terms $\overline{t}$, then this value t must be produced by an application of f to some deterministic elements $\overline{t'}$ of $\overline{t}$. This condition is necessary to be able to define the operations on term structure. It allows to construct the result set of an application $f(\overline{t})$ as the union of the deterministic terms returned by the applications of f to all deterministic terms (elements) $\overline{t'}$ included in $\overline{t}$.

Fact 1.3.3 *[22] If SP is a DET-additive specification then $T(SP)$ is a multialgebra.*

We now begin to turn $T(SP)$ into a model for SP.

Lemma 1.3.4 *There is a unique homomorphism $h : T(SP) \to A$ for every $A \in \mathsf{Mod}(SP)$*

Proof. Define h by $h(t) = t^A$, for all $t \in |T(SP)|$ (remember that $SP \models t \doteq t$).

- h is a homomorphism:

$$\begin{array}{ll} & h(f^{T(SP)}(\overline{x})) \\ = & \{ \ \overline{x} = \overline{t} \text{ for some } \overline{t} \in \mathcal{T}_\Sigma, \text{ and } SP \models \overline{t} \doteq \overline{t}\} \\ & h(f^{T(SP)}(\overline{t})) \\ = & \{ \text{ def. } T(SP)\} \\ & h(\{t' : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec f(\overline{t})\}) \\ = & \{ \text{ def. } h\} \end{array}$$

$$
\begin{array}{ll}
& \{t'^A : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec f(\overline{t})\} \\
\subseteq & \{\ t'^A \in f^A(\overline{t^A})\} \\
& f^A(\overline{t^A}) \\
= & \{\ \text{def. } h \text{ and } SP \models \overline{t} \doteq \overline{t}\} \\
& f^A(\overline{h(t)})
\end{array}
$$

- h is unique:
  Suppose $h' : T(SP) \to A$ is another homomorphisms with $h' \neq h$. Then:

  $$h'(t) \subseteq t^A$$

  We show this by induction on the term structure of t:

  1. $t = c$, c constant:

     $$
     \begin{array}{ll}
     & h'(t) \\
     = & \{\ \text{assumption}\} \\
     & h'(c) \\
     \subseteq & \{\ h' \text{ homomorphism}\} \\
     & c^A
     \end{array}
     $$

  2. $t = f(\overline{t'})$:

     $$
     \begin{array}{ll}
     & h'(t) \\
     = & \{\ \text{assumption}\} \\
     & h'(f^{T(SP)}(\overline{t'})) \\
     \subseteq & \{\ h' \text{ homomorphism}\} \\
     & f^A(h'(t'^*)) \\
     \subseteq & \{\ \text{ind. hyp.}\} \\
     & f^A(\overline{t'^A}) \\
     = & \{\ \text{def. interpretation}\} \\
     & (f(\overline{t'}))^A \\
     = & \{\ \text{assumption}\} \\
     & t^A
     \end{array}
     $$

For the elements $t \in |T(SP)|$ we have that $SP \models t \doteq t$, i.e. $|t^A| = 1$ and from the above we get:

$$h'(t) = t^A = h(t), \forall t \in |T(SP)|$$

$\square$

Definition 1.3.5 *[51] The* kernel of a homomorphism $h : A \to B$ is the equivalence relation $\sim_h \subset |A| \times |A|$ such that $a \sim_h b$ iff $h(a) = h(b)$

Definition 1.3.6 *[51] The* quotient of a multialgebra A wrt. an equivalence relation $\sim$ is the multialgebra $A/_{\sim}$ where:

- $|A/_{\sim}| = \{[a]_{\sim} : a \in |A|\}$
- $f^{A/\sim}(\overline{[x]}) = \{[a] : a \in f^A(\overline{x'}), \overline{x'} \in \overline{[x]}\}$

Fact 1.3.7 *[51] Given an equivalence relation $\sim$ on a multialgebra A then the mapping $h_{\sim} : A \to A/_{\sim}$, defined by $h_{\sim}(a) = [a]_{\sim}$ is a homomorphism (in fact, an epimorphism).*

Fact 1.3.8 *[51] Given a homomorphism $h : A \to B$, between two Σ multialgebras A, B, there exist homomorphisms $h_1 : A \to A/_{\sim_h}$ and $h_2 : A/_{\sim_h} \to B$, defined by $h_1(a) = [a]_{\sim_h}$ and $h_2([a]_{\sim_h}) = h(a)$ such that h_1 is epi, h_2 is mono and $h = h_1; h_2$.*

Fact 1.3.9 *[51] If $\sim_1 \subseteq \sim_2$ for two equivalences $\sim_1, \sim_2$ on A then there is a (epi) homomorphism $h : A/_{\sim_1} \to A/_{\sim_2}$.*

Definition 1.3.10 *Let $\cong$ be the equivalence relation defined on $T(SP)$ by $\cong\ = \bigcap_A \sim_{h_A}$: where h_A is the unique homomorphism $h_A : T(SP) \to A$ for each $A \in \mathsf{Mod}(SP)$.*

Lemma 1.3.11 *There is a unique homomorphism $\phi_A : T(SP)/_{\cong} \to A$, for each $A \in \mathsf{Mod}(SP)$.*

Proof. We have the following picture:

$$\begin{tikzcd}
& T(SP) \arrow[dl, "!h_A"'] \arrow[d, "\eta_A"] \arrow[dr, "\eta"] & \\
A & T(SP)/_{\sim_{h_A}} \arrow[l, "h'_A"] & T(SP)/_{\cong} \arrow[l, "h_{\sim}"]
\end{tikzcd}$$

There is a unique homomorphism $h_A : T(SP) \to A$, for each $A \in \mathsf{Mod}(SP)$, by lemma 1.3.4. Fact 1.3.8 ensures the existence of a monomorphism $h'_A : T(SP)/_{\sim_{h_A}} \to A$. By fact 1.3.9 there is, for each $A \in \mathsf{Mod}(SP)$, an epimorphism $h_{\sim} : T(SP)/_{\cong} \to T(SP)/_{\sim_{h_A}}$, since $\sim_{h_A}\ \supseteq\ \cong$. And since composition of homomorphisms is a homomorphism, we get a homomorphism $\phi_A : T(SP)/_{\cong} \to A$, defined by $\phi_A = h_{\sim}; h'_A$. By fact 1.3.7 there is an epimorphism $\eta : T(SP) \to T(SP)/_{\cong}$. This gives that ϕ_A is unique since h_A is unique and η is epi. $\square$

Lemma 1.3.11 gives us immediately the following proposition:

Proposition 1.3.12 *If $T(SP)/_{\cong} \in \mathsf{Mod}(SP)$ then $T(SP)/_{\cong}$ is an initial model for SP.*

Of course, checking whether $T(SP)/_{\cong} \in \mathsf{Mod}(SP)$ may be a hard task. Nevertheless, the above proposition gives us a general way of finding initial models irrespectively of the form of specification. For instance, we may construct an initial model for the following specification of binary nondeterministic choice $\sqcup$ which uses disjunction.

Example 1.3.13 *Initial model for nondeterministic binary choice.*

$$\begin{array}{lrll}
\textbf{spec NDChoice}^{\mathcal{MA}} = & & & \\
& \mathbf{S}: & & s \\
& \Omega: & & a :\rightarrow s \\
& & & b :\rightarrow s \\
& & & c :\rightarrow s \\
& & & d :\rightarrow s \\
& & & \sqcup : s \times s \rightarrow s \\
\textbf{axioms}: & & 1. & a \doteq a \\
& & 2. & b \doteq b \\
& & 3. & c \doteq c \\
& & 4. & d \doteq d \\
& & 5. & x \prec x \sqcup y \\
& & 6. & x \sqcup y \prec y \sqcup x \\
& & 7. & z \prec x \sqcup y \rightarrow z \prec x, z \prec y
\end{array}$$

$T(SP)/_{\cong}$ is initial for the above specification, since it is a multialgebra and satisfies all axioms, especially axiom 7.

$$\begin{array}{l}
s^{T(SP)/_{\cong}} = \{[a],[b],[c],[d]\},\ \textit{where } [a] = a, a \sqcup a, \ldots \\
a^{T(SP)/_{\cong}} = [a] \\
b^{T(SP)/_{\cong}} = [b] \\
c^{T(SP)/_{\cong}} = [c] \\
d^{T(SP)/_{\cong}} = [d] \\
(x \sqcup y)^{T(SP)/_{\cong}} = \{x, y\}\ \textit{where } x, y \in s^{T(SP)/_{\cong}}
\end{array}$$

$T(SP)/_{\cong}$ has the following property of standard initial models of deterministic specifications.

Proposition 1.3.14 *If $T(SP)/_{\cong} \in \mathsf{Mod}(SP)$ then for any ground terms $s, t \in \mathcal{T}_{\Sigma}$:*

$$T(SP)/_{\cong} \models s \doteq t \iff SP \models s \doteq t$$

Proof. $\Leftarrow$ is trivial by the assumption. So assume that $T(SP)/_{\cong} \models s \doteq t$. This means, in particular, that $[s], [t] \in |T(SP)/_{\cong}|$, i.e., that $SP \models s \doteq s$ and $SP \models t \doteq t$. For any $A \in \mathsf{Mod}(SP)$, we have the unique homomorphism $\phi_A : T(SP)/_{\cong} \rightarrow A$ by lemma 1.3.11. Thus $A \models s \doteq t$.

□

The above property does not hold for $\prec$, i.e. it's possible that $T(SP)/_{\cong} \models s \prec t$ but $SP \not\models s \prec t$

Example 1.3.15 *Consider the following specification:*

$$
\begin{array}{lll}
\textbf{spec SP} = & & \\
\quad\quad\quad \textbf{S}: & & s \\
\quad\quad\quad \Omega: & & c :\rightarrow s \\
 & & d :\rightarrow s \\
\textbf{axioms}: & 1. & c \doteq c \\
 & 2. & c \prec d
\end{array}
$$

Then $s^{T(SP)/\cong} = c = c^{T(SP)/\cong} = d^{T(SP)/\cong}$*, hence* $T(SP)/_{\cong} \models d \prec c$*, but* $SP \not\models d \prec c$*. Consider* $A \in \mathbf{M}od(SP)$ *given by:* $s^A = \{c, d\} = d^A$ *and* $c^A = c$*.*

1.3.2 Initial models for atomic specifications

We show that for an atomic DET-additive specification SP (with only atomic formulae as axioms), $T(SP)/_{\cong} \in \mathsf{Mod}(SP)$, and hence that it is an initial model. We will end up this section with a counter example demonstrating that the conditions for atomic specifications are not sufficient for ensuring the existence of initial models for specifications involving also conditional axioms.

Lemma 1.3.16 *If* $SP \models s \doteq t$*, for* $s, t \in \mathcal{T}_{\Sigma,X}$*, then we have that:* $T(SP)/_{\cong} \models s \doteq t$*.*

Proof. Note that for an $A \in \mathsf{Mod}(SP)$, each assignment $\alpha : X \to T(SP)/_{\cong}$ induces a unique assignment $\alpha' : X \to A$, defined by $\alpha' = \alpha; \phi_A$, where ϕ_A is the unique homomorphism $\phi_A : T(SP)/_{\cong} \to A$.

If $SP \models s \doteq t$, than for any given assignment $\alpha : X \to T(SP)/_{\cong}$ we have that $\phi_A(\alpha(s)) = \phi_A(\alpha(t))$ for all $A \in \mathsf{Mod}(SP)$, where ϕ_A is as above. But also $|\phi_A(\alpha(s))| = 1 = |\phi_A(\alpha(t))|$ i.e $\alpha(s) \cong \alpha(t)$, by definition 1.3.10. Thus, since α was arbitrary, $T(SP)/_{\cong} \models s \doteq t$ $\square$

Lemma 1.3.17 *[22] For an assignment* $\alpha : X \to T(SP)/_{\cong}$ *and a term* $t \in \mathcal{T}_{\Sigma,X}$ *we have that:* $\alpha(t)^{T(SP)/\cong} = \{[t'] : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(t)\}$

Proof. We have three cases to prove.

1. $t = x_s, x_s \in X$: Two cases:
   (a) If $s^{T(SP)/\cong} = \emptyset$ then:

$$
\begin{array}{ll}
 & \alpha(t)^{T(SP)/\cong} \\
= & \{\text{ assumption}\} \\
 & \alpha(x)^{T(SP)/\cong} \\
= & \{\text{ def.assignment for empty carrier}\}
\end{array}
$$

$$
\begin{aligned}
& \emptyset \\
= & \quad \{ \text{ def. } T(SP)/_{\cong}\} \\
& \{[t'] : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(x)\}
\end{aligned}
$$

(b) Else:

$$
\begin{aligned}
& \alpha(t)^{T(SP)/_{\cong}} \\
= & \quad \{ \text{ assumption}\} \\
& \alpha(x)^{T(SP)/_{\cong}} \\
= & \quad \{ \text{ def. } T(SP)/_{\cong}, \text{ lemma 1.3.16 and } SP \models \alpha(x) \doteq \alpha(x) \ \} \\
& \{[t'] : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(x)\}
\end{aligned}
$$

2. $t = c$, c constant:

$$
\begin{aligned}
& \alpha(t)^{T(SP)/_{\cong}} \\
= & \quad \{ \text{ assumption}\} \\
& \alpha(c)^{T(SP)/_{\cong}} \\
= & \quad \{ \text{ no assignment for constants}\} \\
& c^{T(SP)/_{\cong}} \\
= & \quad \{ \text{ def. } T(SP)/_{\cong}\} \\
& \{[t'] : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec c\} \\
= & \quad \{ \text{ no assignment for constants}\} \\
& \{[t'] : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(c)\}
\end{aligned}
$$

3. $t = f(t_1, \ldots, t_n)$:

$$
\begin{aligned}
& \alpha(t)^{T(SP)/_{\cong}} \\
= & \quad \{ \text{ assumption}\} \\
& \alpha(f(t_1, \ldots, t_n))^{T(SP)/_{\cong}} \\
= & \quad \{ \text{ assignment}\} \\
& \textstyle\bigcup_{a_i \in \alpha(t_i)} f(a_1, \ldots, a_n)^{T(SP)/_{\cong}} \\
= & \quad \{ \text{ def. } T(SP)/_{\cong}\} \\
& \{[t'] : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \textstyle\bigcup_{a_i \in \alpha(t_i)} f(a_1, \ldots, a_n)^{T(SP)/_{\cong}}\} \\
= & \quad \{ \text{ assignment}\} \\
& \{[t'] : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(f(t_1, \ldots, t_n))\}
\end{aligned}
$$

□

Lemma 1.3.18 *If $s, t \in \mathcal{T}_{\Sigma,X}$ and $SP \models s \prec t$ then: $T(SP)/_{\cong} \models s \prec t$:*

Proof. If $SP \models s \prec t$ and α an assignment we have that:

$$
\begin{array}{ll}
& \alpha(s)^{T(SP)/\cong} \\
= & \{ \text{ lemma 1.3.17}\} \\
& \{[t'] : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(s)\} \\
\subseteq & \{\ SP \models s \prec t\} \\
& \{[t'] : t' \in \mathcal{T}_\Sigma \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(t)\} \\
= & \{ \text{ lemma 1.3.17}\} \\
& \alpha(t)^{T(SP)/\cong}
\end{array}
$$

□

Combining lemma 1.3.11, lemma 1.3.16 and lemma 1.3.18 we get the sufficient condition for the existence of initial models for atomic specifications:

Proposition 1.3.19 *[22] If SP is an atomic DET-additive specification then $T(SP)/_{\cong}$ is an initial model for SP.*

One might hope that the conditions of proposition 1.3.19 will carry over to conditional axioms, but the following example shows that this it not the case.

Example 1.3.20 *Counter example: DET-additivity does not ensure the existence of initial models for Horn formulae.*

$$
\begin{array}{rll}
\multicolumn{3}{l}{\textbf{spec CounterEx}^{\mathcal{MAH}} =} \\
\mathbf{S}: & & s \\
\Omega: & & c :\to s \\
& & d :\to s \\
& & e :\to s \\
\textbf{axioms}: & 1. & c \doteq c \\
& 2. & d \doteq d \\
& 3. & c \prec e \\
& 4. & c \doteq e \to d \doteq e
\end{array}
$$

The following multialgebra A satisfies the specification:

$$
\begin{array}{l}
s^A = \{c, d, j\} \\
c^A = c \\
d^A = d \\
e^A = \{c, j\}
\end{array}
$$

But $T(SP)/_{\cong}$ will not satisfy the specification. It looks as follows:

$$
\begin{array}{l}
s^{T(SP)/\cong} = \{[c], [d]\},\ \textit{where } [c] = \{c, e\} \textit{ and } [d] = \{d\} \\
c^{T(SP)/\cong} = [c] \\
d^{T(SP)/\cong} = [d] \\
e^{T(SP)/\cong} = [c]
\end{array}
$$

So $T(SP)/_{\cong}$ does not satisfy axiom 4. On the other hand, if we take the least congruence $\cong'$ on $T(SP)/_{\cong}$ that satisfies the axioms, we would obtain $[c] = [d]$, but then we have no homomorphism to A – $T(SP)/_{\cong'}$ satisfies now the equation $c \doteq e$ that is not satisfied by A.

At the moment, we do not have general syntactic conditions ensuring the existence of initial models for conditional specifications and we have to live with the proposition 1.3.12 as the strongest general result.

1.4 Other specification frameworks in multialgebraic setting

Multialgebras offer a highly abstract specification formalism. The price of this abstraction is the loss of structure, some closure properties, in particular, such as the existence of initial models. In the following, we will restrict the institution $\mathcal{MA}$ in various ways to relate it to other specification frameworks and, on the other hand, to ensure the existence of initial models.

In 1.4.1 we relate multialgebras to partial algebras and in 1.4.2 to membership algebras. These two examples will hopefully illustrate the possibility to use multialgebras to compare and combine of other algebraic specification formalisms, which will be illustrated in later chapters.

Fact 1.4.1 *The multialgebras with Horn clauses is a subinstitution $\mathcal{MAH}$ of $\mathcal{MA}$. All components are as in $\mathcal{MA}$ except for the sentence functor which is $\mathsf{Sen}_{\mathcal{MA}}$ restricted to Horn clauses over atomic formulae.*

The proof of the satisfaction condition is a special case of the proof for $\mathcal{MA}$, the only difference is that now we only have to consider formulae of the form $a_1, \ldots, a_n \to b$. The subinstitution mapping is $(\Phi, \alpha, \beta) : \mathcal{MAH} \to \mathcal{MA}$, where Φ is the identity functor, each component of α is identity and each component of β is the identity functor.

Fact 1.4.2 *The multialgebras with equational Horn clauses is a subinstitution $\mathcal{MAEH}$ of $\mathcal{MAH}$. All components are as in $\mathcal{MAH}$ except for the sentence functor which is $\mathsf{Sen}_{\mathcal{MA}}$ restricted to Horn clauses over element equalities.*

The proof of the satisfaction condition is a special case of the proof for $\mathcal{MAH}$, the only difference is that now we only have to consider formulae build over element equations. The subinstitution mapping is $(\Phi, \alpha, \beta) : \mathcal{MAEH} \to \mathcal{MAH}$, where Φ is the identity functor, each component of α is identity and each component of β is the identity functor.

Fact 1.4.3 *The multialgebras with atomic sentences is a subinstitution $\mathcal{MAA}$ of $\mathcal{MAH}$. All components are as in $\mathcal{MAH}$ except for the sentence functor which is $\mathsf{Sen}_{\mathcal{MA}}$ restricted to atomic formulae.*

The proof of the satisfaction condition is a special case of the proof for $\mathcal{MAH}$, the only difference is that now we only have to consider atomic formulas. The subinstitution mapping is $(\Phi, \alpha, \beta) : \mathcal{MAA} \to \mathcal{MAH}$, where Φ is the identity functor, each component of α is identity and each component of β is the identity functor.

1.4.1 Partial algebras and multialgebras

In this subsection we will study the relationships between the institution of partial algebras, $\mathcal{PA}$ and $\mathcal{MA}$. We start with recap of partial algebras and shows an embedding of the institution of partial algebras into $\mathcal{MA}$. We also illustrates by example the intended way of reusing partial algebra specifications in our framework, the results of this subsection are used to give a methodology for partiality handling with multialgebras in chapter 3, where we in subsection 3.3.1 describes another way of relating institutions, namely *institution transformation* from [35], which formalizes the intentions of the example 1.4.8.

We recall the basic definitions from partial algebras, for a survey see [8, 10, 38].

Partial algebras uses algebraic signatures, i.e. the same signatures as for multialgebras, **Sign** from definition 1.1.1.

Definition 1.4.4 *A partial algebra A for a signature $\Sigma = (\mathbf{S}, \Omega)$ is given by:*

- *A set s^A for each sort $s \in \mathbf{S}$*
- *A partial function[1] $\omega^A : {s_1}^A \times \cdots \times {s_n}^A \to s^A$, for each symbol $\omega : s_1 \times \cdots \times s_n \to s \in \Omega$*

Given two partial algebras A and B, a (weak) homomorphism $h : A \to B$ is a set of total functions $h_s : s^A \to s^B$, for each sort $s \in \mathbf{S}$ such that:

- *$h_s(\omega^A(x_1, \ldots, x_n)) = \omega^B(h_{s_1}(x_1), \ldots, h_{s_n}(x_n))$ for each operation $\omega : s_1 \times \cdots \times s_n \to s \in \Omega$ and arguments $x_1, \ldots, x_n$, where $\omega^A(x_1, \ldots, x_n)$ is defined.*

The total functions are special cases of the partial ones. Weak homomorphism h can be equivalently described as an ordinary homomorphism such that: for each operation $\omega \in \Omega : h(\mathbf{dom}(\omega^A)) \subseteq \mathbf{dom}(\omega^B)$, where **dom** identifies ω's definition domain in a given algebra A.

Definition 1.4.5 *Formulae:*

- *Atomic $t \stackrel{e}{=} t'$ – existential equalities*
- *Formulae are universally quantified Horn clauses over existential equalities: $X; a_1 \wedge \cdots \wedge a_n \to a$ with $n \geq 0$.*

Definition 1.4.6 *Let $\alpha : X \to |A|$ be an assignment (total function):*

- *$A \models_\alpha (X; t \stackrel{e}{=} t')$ iff $\overline{\alpha}(t)$ and $\overline{\alpha}(t')$ are defined and $\overline{\alpha}(t) = \overline{\alpha}(t')$*
- *$A \models_\alpha (X; a_1 \wedge \cdots \wedge a_n \to a)$ iff $\exists i : 1 \leq i \leq n : A \not\models_\alpha a_i$ or $A \models_\alpha a$*
- *$A \models (X; \phi)$ iff $A \models_\alpha (X; \phi)$ for all α*

[1] A partial function is a function that is undefined for some arguments.

Example 1.4.7 *Following is an example of a partial algebra specification:*

$$
\begin{array}{rll}
\textbf{spec Nat}^{\mathcal{PA}} = & & \\
\mathbf{S}: & & Nat \\
\Omega: & & zero : \to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
\textbf{axioms}: & 1. & zero \stackrel{e}{=} zero \\
& 2. & \{x\}; succ(x) \stackrel{e}{=} succ(x) \\
& 3. & \{x\}; pred(succ(x)) \stackrel{e}{=} x \\
& 4. & \{x\}; pred(x) \stackrel{e}{=} pred(x) \to succ(pred(x)) \stackrel{e}{=} x
\end{array}
$$

The first two axioms make zero and succ total operations. The third axiom ensures that $pred(succ(x))$ is defined and equal to x. The last axiom means that if $pred(x)$ is defined then also $succ(pred(x))$ is defined and equal to x.

Partial algebras and multialgebras

To relate multialgebras with partial algebras we recall the similarities between the element equalities $\doteq$ from multialgebras and the existential equalities $\stackrel{e}{=}$ in partial algebras. Both equalities hold when both sides of the equality sign denote the same element. In a multialgebra, the equality does not hold if one side of the equality sign is interpreted by a set with cardinality greater than one or by the empty set. The existential equality does not hold in a partial algebra if one side of the equality sign is undefined. (Besides equality does not hold, in both multialgebra and partial algebra, when the elements are different on each side of the equality sign.) These similarities suggest the straightforward relation between multialgebras and partial algebras: replace $\stackrel{e}{=}$ by $\doteq$. Each x in the variable context X in $(X; \phi)$, will give rise to an extra condition $x \doteq x$ in the antecedent of ϕ.

Example 1.4.8 *The multialgebra specification corresponding to the specification from example 1.4.7 is as follows:*

$$
\begin{array}{rll}
\textbf{spec Nat}^{\mathcal{MA}} = & & \\
\mathbf{S}: & & Nat \\
\Omega: & & zero : \to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
\textbf{axioms}: & 1. & zero \doteq zero \\
& 2. & x \doteq x \to succ(x) \doteq succ(x) \\
& 3. & x \doteq x \to pred(succ(x)) \doteq x \\
& 4. & x \doteq x, pred(x) \doteq pred(x) \to succ(pred(x)) \doteq x
\end{array}
$$

The partial algebra models and the multialgebra models for the specifications from the above two examples will coincide on their total parts but the difference concerns the undefined parts. Among the partial algebra models, there will be ones where the term $pred(zero)$ has no interpretation – is undefined. Similarly, among the multialgebraic models there will be ones where this term denotes empty set. But we also have the possibility that this term may be nondeterministic, even completely nondeterministic.

If, instead of axiom 4., one writes the axioms $\{x\}; succ(pred(x)) \stackrel{e}{\doteq} x$, respectively $x \doteq x \rightarrow succ(pred(x)) \doteq x$, one forces $succ(pred(x))$ to be always defined (on nonempty carriers). In the partial algebra case, due to strictness of all operations, this forces also $pred(x)$ to be defined. In the multialgebra case this corresponds to a particular error recovery – even if $pred(zero)$ not return an unique element, the subsequent application of *succ* will yield *zero*. The model class of the multialgebraic specification has "more" models.

The above example shows our intended goal of allowing reuse of partial algebra specifications in the context where additional flexibility is offered by the extension of the model class. This will be our topic in subsection 3.3.1. As a preliminary step we show first a direct embedding of $\mathcal{PA}$ into $\mathcal{MA}$.

Embedding $\mathcal{PA}$ into $\mathcal{MA}$.

Any Σ-partial algebra can be trivially converted into a Σ-multialgebra by making all undefined operations return the empty set. Since operations in multialgebra are strict on the empty set, the implicit strictness assumption from partial algebras, will be enforced automatically.

Definition 1.4.9 *The functor β^- : $\mathsf{Mod}_{\mathcal{PA}}(\Sigma) \rightarrow \mathsf{Mod}_{\mathcal{MA}}(\Sigma)$ maps a partial algebra A to a multialgebra in the following way:*

- $|\beta^-(A)| = |A|$
- *for all* $\overline{x} \in |\beta^-(A)|$ *and* $f \in \Omega$: $f(\overline{x})^{\beta^-(A)} = \begin{cases} \{f(\overline{x})^A\} & \text{–if it is defined} \\ \emptyset & \text{–otherwise} \end{cases}$

For a homomorphism: $h \in \mathsf{Mod}(\Psi(\Sigma,\Gamma))$, we define $\beta^-(h) = h$.

For a multialgebra M where all undefined operations return empty set, $\beta(M)$ will denote the corresponding partial algebra, i.e., $\beta^-(\beta(M)) = M$.

Saying that a multialgebra and a partial algebra are "essentially the same", we will mean that they are obtained from each other by means of $\beta(\_)$, resp. $\beta^-(\_)$.

The embedding of $\mathcal{PA}$ into $\mathcal{MA}$ is now obtained by augmenting the partial algebra specification with additional axioms forcing all operations to return either a unique element or the empty set. This is the underlying model in partial algebras which in our, generalized, context need explicit axioms.

For an operation $f(\overline{x})$, the axiom forcing it to be empty or deterministic is of the form $y \doteq y, y \prec f(\overline{x}) \rightarrow f(\overline{x}) \doteq f(\overline{x})$, where y is a fresh variable[2]. We first show that this is the case – i.e., that multialgebras satisfying such axioms (for all operations) are "essentially" partial algebras.

[2]Note that the premiss $y \doteq y$ could be dropped if we disallow variables to be assigned to the emptyset.

Lemma 1.4.10 *Let $SP = (S, \Omega, \Gamma)$ be a specification in $\mathcal{MA}$ such that, for each operation $f : \overline{s} \to s \in \Omega : SP \models y \doteq y, y \prec f(\overline{x}) \to f(\overline{x}) \doteq f(\overline{x})$, where y is distinct from all $\overline{x}$. Then in any $M \in \mathsf{Mod}_{\mathcal{MA}}(SP)$ we have that for all $\overline{x} \in |M| : f(\overline{x})^M = \emptyset$ or $f(\overline{x})^M$ is deterministic.*

Proof. Let $M \in \mathsf{Mod}_{\mathcal{MA}}(SP)$ and $\alpha : \{y, \overline{x}\} \to M$ be an assignment, We have two cases:

1. $M \not\models_\alpha y \doteq y$, then the carrier set of s is empty so $\alpha(f(\overline{x}))^M = \emptyset$.
2. $M \models_\alpha y \doteq y$, if $M \not\models_\alpha f(\overline{x}) \doteq f(\overline{x})$ then, since $M \models_\alpha y \doteq y, y \prec f(\overline{x}) \to f(\overline{x}) \doteq f(\overline{x})$, we must have that $M \not\models_\alpha y \prec f(\overline{x})$, i.e., $\alpha(f(\overline{x}))^M = \emptyset$.

Note that $\alpha(f(\overline{x}))^M$ may be $\emptyset$ also when some of $\overline{x}$ range over empty sorts, but this case is covered by point 2. Thus we do not need additional conditions $x \doteq x$ for $x \in \overline{x}$. $\Box$

Proposition 1.4.11 *There is an embedding (Ψ, α, β) of institutions from $\mathcal{PA}$ to $\mathcal{MA}$; the embedding is a simple map.*

Proof.

- The functor $\Psi : \mathbf{Sign}_{\mathcal{PA}} \to \mathbf{Th}_{0\mathcal{MA}}$ is given by: $\Psi(S, \Omega) = ((S, \Omega), \emptyset_\Sigma)$, where $\emptyset_\Sigma$ contains an axiom $y \doteq y, y \prec f(\overline{x}) \to f(\overline{x}) \doteq f(\overline{x})$ for each $f \in \Omega$. For morphisms $\Psi(\mu_S, \mu_\Omega)$ is the identity.
- The natural transformation $\alpha : \mathsf{Sen}_{\mathcal{PA}} \to \mathsf{Sen}_{\mathcal{MA}} \circ \Psi$ is given by:
  1. $\alpha(t \stackrel{e}{=} t') \equiv t \doteq t'$ auxilary definition for atoms
  2. $\alpha(\{x_1, \ldots, x_k\}; a_1 \wedge \cdots \wedge a_n \to a) \equiv x_1 \doteq x_1, \ldots, x_k \doteq x_k, \alpha(a_1), \cdots, \alpha(a_n) \to \alpha(a)$

  Ψ is extended to a functor $\Psi : \mathbf{Th}_{0\mathcal{PA}} \to \mathbf{Th}_{0\mathcal{MA}}$ by letting $\Psi(\Sigma, \Gamma) = (\Sigma, \emptyset_\Sigma \cup \alpha_\Sigma(\Gamma))$.
- The components of the natural transformation $\beta : Mod_{\mathcal{MA}} \circ \Psi^{op} \to Mod_{\mathcal{PA}}$ are:
  β's from definition 1.4.9, i.e.:
  - $|\beta_\Sigma(M')| = |M'|$
  - $f(x_1, \ldots, x_n)^{\beta_\Sigma(M')} = \begin{cases} undefined \text{ if } f(x_1, \ldots, x_n)^{M'} = \emptyset \\ x \text{ such that } f(x_1, \ldots, x_n)^{M'} = \{x\} \text{ otherwise} \end{cases}$

  This is a well defined partial algebra by lemma 1.4.10. For a homomorphism: $h \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma))$, we define $\beta_\Sigma(h) = h$.

We verify the "truth" condition: for every $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \emptyset))$ and $\phi \in \mathsf{Sen}_{\mathcal{PA}}(\Sigma)$:

$$M' \models^{\mathcal{MA}} \alpha_\Sigma(\phi) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') \models^{\mathcal{PA}} \phi \tag{1.2}$$

By the definition of β, for any $\Psi(\Sigma, \emptyset)$-multialgebra M', the Σ-partial algebra $\beta(M')$ has the same carrier, $|M| = |\beta(M')|$. Thus for the nonempty sorts of $|M|$ is every assignment $\nu' : X \to M'$ is also an assignment $\nu' : X \to \beta(M')$, and vice versa. If $s^M = \emptyset$ is it exact one assignment for s^M and there are no assignment for $s^{\beta(M')}$. Given a partial algebra signature $\Sigma = (S, \Omega)$, $\Psi(\Sigma, \emptyset)$ adds one axiom $y \doteq y, y \prec \omega(\overline{x}) \to \omega(\overline{x}) \doteq \omega(\overline{x})$, for every $\omega \in \Omega$ – thus for any ground term $t \in \mathcal{T}_\Sigma$, $\Psi(t) = t$ will be interpreted as an element or the empty set in every $\Psi(\Sigma, \emptyset)$-multialgebra M'.

We show that for every simple formula $(X; a) \in \mathsf{Sen}_{\mathcal{PA}}(\Sigma)$ and for any assignment $\nu : X \to |M'|$:

$$M' \models_{\mathcal{MA}_\nu} \alpha_\Sigma(X; a) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') \models_{\mathcal{PA}_\nu} (X; a) \tag{1.3}$$

Let $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \emptyset)) : a_s \equiv \{x_1, \ldots, x_k\}; t \stackrel{e}{=} t'$: We have two cases:

1. No variables over empty carrier

   $M' \models_\nu \alpha(\{x_1, \ldots, x_k\}; t \stackrel{e}{=} t')$
   $\iff$ { def. α}
   $M' \models_\nu x_1 \doteq x_1, \ldots, x_k \doteq x_k \to t \doteq t'$
   $\iff$ { satisfaction (no variables over empty carrier)}
   $\nu(t)^{M'} = \nu(t')^{M'} = e, e \in |M'|$
   $\iff$ { def. β}
   $\nu(t)^{\beta(M')} = \nu(t')^{\beta(M')} = e \in |\beta(M')|$
   $\iff$ { satisfaction}
   $\beta(M') \models_\nu t \stackrel{e}{=} t'$

2. Variables over empty carrier
   $\beta(M') \models_{\mathcal{PA}_\nu} (X; a)$, since there is no assignment to a empty carrier. On the other hand is there a variable $x \in X$ over empty carrier hence $M' \not\models x \doteq x$, and $M' \models_{\mathcal{MA}_\nu} \alpha_\Sigma(X; a)$.

The general statement (1.2) follows easily from the above fact (1.3). □

We also have an immediate consequence of the above proof:

Fact 1.4.12 *For a $\mathcal{PA}$ theory (Σ, Γ), the functor $\beta_{(\Sigma,\Gamma)}$ is an equivalence (in fact, an isomorphism) of categories $\mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma))$ and $\mathsf{Mod}_{\mathcal{PA}}(\Sigma, \Gamma)$.*

Proof. The inverse functor $\beta^-_{(\Sigma,\Gamma)}$ sends a partial algebra $P \in \mathsf{Mod}_{\mathcal{PA}}(\Sigma, \Gamma)$ onto a multialgebra $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma))$ such that $\beta(M') = P$, i.e., it is β^- from definition 1.4.9. One verifies easily the isomorphism condition. □

As Mossakowski showed in [38] $\mathcal{PA}$ allows to specify exactly the finitely locally presentable categories [1], i.e. we have identified the sub-institution of $\mathcal{MA}$ allowing to specify these classes of models. Given a partial algebra specification SP, we call $\Psi(SP)$ a multialgebra specification of *partial form* and denote this sub-institution $\mathcal{MAPA}$. It's well known that the finitely locally presentable categories have initial models, see [1].

1.4.2 Membership algebras and multialgebras

We start by recalling the basic concepts from membership algebras and show, in subsection 1.4.2, that institution of membership algebras can be embedded into $\mathcal{MA}$. Membership algebras were introduced by Meseguer in [37]. We will use notation corresponding to the notation used elsewhere in the thesis, which slightly differs from the notation used in [37].

Definition 1.4.13 *(Membership signature) A (membership) signature Σ is a quadruple $\Sigma = (\mathbf{S}, \Omega, P, \pi)$, where $\mathbf{S}$ is the set of sort names and Ω the set of operation names for the $\overline{\mathbf{S}} \times \mathbf{S}$ indexed operations. P is a set of sub-sort predicate names. π is a function $\pi : P \to \mathbf{S}$ which for each sub-sort predicate $p \in P$ assigns a corresponding sort $s \in \mathbf{S}$. By P_s one denotes the sub-sort predicates of sort s i.e. $\{\pi^{-1}(s) : s \in \mathbf{S}\}$.*

The function π labels sub-sort predicate symbols by sort – its intention is to identify a predicate p with $\pi(p) = s$ as a sub-sort of sort s.

Definition 1.4.14 *A signature morphism between two membership signatures $\Sigma = (\mathbf{S}, \Omega, P, \pi)$ and $\Sigma' = (\mathbf{S}', \Omega', P', \pi')$ is a triple $\mu = (\mu_P, \mu_{\mathbf{S}}, \mu_\Omega)$, where $\mu_P : P \to P'$ and $\mu_{\mathbf{S}} : \mathbf{S} \to \mathbf{S}'$ are functions such that the following diagram*

$$\begin{tikzcd} P \arrow[r, "\mu_P"] \arrow[d, "\pi"'] & P' \arrow[d, "\pi'"] \\ \mathbf{S} \arrow[r, "\mu_{\mathbf{S}}"'] & \mathbf{S}' \end{tikzcd}$$

commutes, and there exists an ω' for each $\omega \in \Omega$ such that:

$$\mu_\Omega(\omega : \overline{s} \to s) = \omega' : \mu_{\mathbf{S}}(\overline{s}) \to \mu_{\mathbf{S}}(s).$$

Definition 1.4.15 *(Membership algebra) A membership algebra for a signature Σ is:*

- *a many sorted $(\mathbf{S}, \Omega)$ algebra A*
- *together with an assignment of a subset $p^A \subseteq \pi(p)^A$ for each predicate p.*

Definition 1.4.16 *(Homomorphism) A Homomorphism $h : A \to B$ is an ordinary total homomorphism which, in addition, satisfies: $h_{\pi(p)}(p^A) \subseteq p^B$*

Definition 1.4.17 *The axioms used for specifying classes of membership algebras are of the following form*

- *atomic formulas:*

1. *equations of the form* $t = t'$ *where* $t, t' \in \mathcal{T}_{\Sigma,X_s}$*, terms of the same sort* s.

2. *membership assertions of the form* $t : p$, $p \in P$ *and* $t \in \mathcal{T}_{\Sigma,X_{\pi(p)}}$.

- *universally quantified Horn clauses over atomic formulae, i.e. they have form:*

  1. $X; u_1 = v_1 \wedge \ldots \wedge u_n = v_n \wedge w_1 : p_1 \wedge \ldots \wedge w_k : p_k \rightarrow t = t'$ *or*
  2. $X; u_1 = v_1 \wedge \ldots \wedge u_n = v_n \wedge w_1 : p_1 \wedge \ldots \wedge w_k : p_k \rightarrow t : p$

The assignments of values to variables are as usual. Then one defines satisfaction of formulae.

Definition 1.4.18 *(Satisfaction of formulae)*

1. $A \models_\alpha (X; t = t')$ *iff* $\overline{\alpha}(t) = \overline{\alpha}(t')$
2. $A \models_\alpha (X; t : p)$ *iff* $\overline{\alpha}(t) \in p^A$
3. $A \models_\alpha (X; a_1 \wedge \ldots \wedge a_n \rightarrow a)$ *iff* $\exists i : 1 \leq i \leq n : A \not\models_\alpha a_i$ *or* $A \models_\alpha a$
4. $A \models (X; \varphi)$ *iff* $A \models_\alpha (X; \varphi)\ \forall\alpha$

It is shown in [37] that the membership algebras with above formulae and satisfaction form a liberal institution $\mathcal{MEMB}$. Moreover there exists an embedding of institutions both ways between $\mathcal{MEMB}$ and the institution of many sorted Horn logic with predicates and equalities, i.e. this two institutions can be viewed as sub-logics of each other. Note that if there are no ground terms included in a predicate then the predicate is represented by the empty set in the initial model.

Embedding $\mathcal{MEMB}$ into $\mathcal{MA}$

The embedding is based on the fact that nondeterministic constants play the same role as unary predicates. Hence the membership relation $t : p$ is naturally translated as $t \prec p$. Making, in addition, all operations deterministic, one obtains the straightforward translation of $\mathcal{MEMB}$ specifications into $\mathcal{MA}$.

Example 1.4.19 *A membership algebra specification of (a part of) natural numbers and the corresponding multialgebra specification using only nondeter-*

ministic constants:

$$
\begin{array}{ll}
\begin{array}{lll}
\multicolumn{3}{l}{\mathbf{specNat}^{\mathcal{MEMB}} =} \\
& \mathbf{S}: & Nat \\
& \Pi: & \pi(nat) = Nat \\
& & \pi(pos) = Nat \\
& \Omega: & zero :\to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
& \mathbf{ax}: & \\
& & zero : nat \\
& & \{x\}; x : pos \to x : nat \\
& & \{x\}; x : nat \to pred(succ(x)) = x \\
& & \{x\}; x : pos \to pred(x) : nat \\
& & \{x\}; x : nat \to succ(x) : pos
\end{array}
&
\begin{array}{lll}
\multicolumn{3}{l}{\mathbf{specNat}^{\mathcal{MA}} =} \\
& \mathbf{S}': & Nat \\
& \Omega': & nat :\to Nat \\
& & pos :\to Nat \\
& & zero :\to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
& \mathbf{ax}: & zero \doteq zero \\
& & succ \doteq succ \\
& & pred \doteq pred \\
& & zero \prec nat \\
& & x \doteq x \wedge x \prec pos \to x \prec nat \\
& & x \doteq x \wedge x \prec nat \to pred(succ(x)) \doteq x \\
& & x \doteq x \wedge x \prec pos \to pred(x) \prec nat \\
& & x \doteq x \wedge x \prec nat \to succ(x) \prec pos
\end{array}
\end{array}
$$

The only difference is that the multialgebra specification needs to ensure determinacy of all the operations corresponding the the operations Ω from the membership specification (but not of the constants corresponding to the predicates Π).

Both of the above specifications will have the same initial model and, as a matter of fact, the same model class.

As the above example suggests, we can translate a membership algebra specification to a multialgebra specification, which will possess the same models.

Proposition 1.4.20 *There is an embedding (Φ, α, β) of institutions from $\mathcal{MEMB}$ to $\mathcal{MA}$; the embedding is a simple map of institutions.*

Proof.

- The functor $\Phi : \mathbf{Sign}_{\mathcal{MEMB}} \to \mathbf{Th}_{\mathcal{MA}}$ is given by: $\Phi(\mathbf{S}, \Omega, \Pi)$ is the theory $(\mathbf{S}, \Omega \uplus \Pi', \emptyset_\Sigma)$, for each $p \in \Pi$ there is a corresponding constant $p :\to \pi(p)$ in Π' and where $\emptyset_\Sigma$ contains an axiom $\omega(\overline{x}) \doteq \omega(\overline{x})$, for each operation $\omega \in \Omega$. $\Phi(\mu_S, \mu_\Omega, \mu_P)$ is the signature morphism $(\mu_S, \mu_\Omega \uplus \mu_P)$
- The natural transformation $\alpha : \mathsf{Sen}_{\mathcal{MEMB}} \to \mathsf{Sen}_{\mathcal{MA}} \circ \Phi$ is given by:
  1. $\alpha(t : p) \equiv t \prec p$ auxilary definition for atoms $t : p$
  2. $\alpha(t = t') \equiv t \doteq t'$ auxilary definition for atoms $t = t'$
  3. $\alpha(\{x_1, \ldots, x_k\}; a_1 \wedge \cdots \wedge a_n \to a) \equiv x_1 \doteq x_1 \wedge \ldots \wedge x_k \doteq x_k \wedge \alpha(a_1) \wedge \cdots \wedge \alpha(a_n) \to \alpha(a)$
     for each Horn clause $a_1 \wedge \cdots \wedge a_n \to a$

  Φ is extended to a functor $\Phi : \mathbf{Th}_{0\mathcal{MEMB}} \to \mathbf{Th}_{0\mathcal{MA}}$ by:
  $\Phi(\Sigma, \Gamma) = (\Sigma, \emptyset_\Sigma \cup \alpha_\Sigma(\Gamma))$.

- The natural transformation $\beta : \mathsf{Mod}_{\mathcal{MA}} \circ \Phi^{op} \to \mathsf{Mod}_{\mathcal{MEMB}}$ is essentially the identity on models and homomorphisms. For any multialgebra $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Phi(\Sigma, \Gamma))$
  - $|\beta(M')| = |M'|$
  - $f(x_1, \ldots, x_n)^{\beta(M')} = x$ such that $f(x_1, \ldots, x_n)^{M'} = \{x\}$
  - $p^{\beta(M')} = p^{M'}$

  For any homomorphisms: $h : M' \to B' \in \mathsf{Mod}_{\mathcal{MA}}(\Phi(\Sigma, \Gamma))$, we let $\beta(h) = h$.

We verify the "truth" condition: for every $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Phi(\Sigma, \emptyset))$ and $\phi \in \mathsf{Sen}_{\mathcal{MEMB}}(\Sigma)$:

$$M' \models^{\mathcal{MA}} \alpha_\Sigma(\phi) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') \models^{\mathcal{MEMB}} \phi \tag{1.4}$$

By the definition of β, for any $\Phi(\Sigma, \emptyset)$-multialgebra M', the Σ-membership algebra $\beta(M')$ has the same carrier, $|M| = |\beta(M')|$. Thus if $s^{M'}$ is nonempty we have that every assignment $\nu' : X \to M'$ is also an assignment $\nu' : X \to \beta(M')$, and vice versa. If $s^{M'} = \emptyset$ there is only one assignment $\nu' : X \to M'_s$ for the multialgebra M', the assignment sending all x's to the emptyset, and for the membership algebra $\beta(M')$ there are no assignments since there are no nonempty functions to the emptyset, i.e. $\beta_{(\Sigma,\emptyset)}(M') \models^{\mathcal{MEMB}}_\nu (X; \phi)$, for any formula ϕ with variables ranging over empty sort, on the other hand will $M' \models^{\mathcal{MA}}_\nu \alpha_\Sigma(X; \phi)$ since there is at least one $x \in X$, ranging over a empty sort and α adds $x \doteq x$ to the premiss of ϕ.

Given a membership signature $\Sigma = (S, \Omega, \Pi)$, $\Phi(\Sigma, \emptyset)$ adds an axiom $\omega(\overline{x}) \doteq \omega(\overline{x})$, for every $\omega \in \Omega$ – thus for any ground term $t \in T(\Omega)$, $\Phi(t) = t$ will be interpreted as an element in every $\Phi(\Sigma, \emptyset)$-multialgebra M'. We show that for every atom $a \in \mathsf{Sen}_{\mathcal{MEMB}}(\Sigma)$, with variables ranging over nonempty sorts, and for any assignment $\nu : X \to |M'|$:

$$M' \models^{\mathcal{MA}}_\nu \alpha_\Sigma(X; a) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') \models^{\mathcal{MEMB}}_\nu (X; a) \tag{1.5}$$

Let $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Phi(\Sigma, \emptyset))$ – we have two possible atoms:

1. $a \equiv s = t$:

$$\begin{aligned}
&M' \models_\nu \alpha(\{x_1, \ldots, x_k\}; s = t) \\
\iff &\{\text{ def. } \alpha\} \\
&M' \models_\nu x_1 \doteq x_1 \wedge \ldots \wedge x_k \doteq x_k \to s \doteq t \\
\iff &\{\text{ satisfaction (no variables over empty sort)}\} \\
&\nu(s)^{M'} = \nu(t)^{M'} = \{e\},\ e \in |M'| \\
\iff &\{\text{ def. } \beta\} \\
&\nu(s)^{\beta(M')} = \nu(t)^{\beta(M')} = e \in |\beta(M')| \\
\iff &\{\text{ satisfaction}\} \\
&\beta(M') \models_\nu (X; s = t)
\end{aligned}$$

2. $a \equiv t : p$: We have two cases:

   (a) $\pi(p)^{M'} = \emptyset$

$$
\begin{array}{l}
M' \models_\nu \alpha(X; t : p) \\
\Longleftrightarrow \{ \; \pi(p)^{M'} = \emptyset \text{ (variables over nonempty sorts)}\} \\
t^{M'} = p^{M'} = \emptyset \\
\Longleftrightarrow \{ \text{ def. } \beta\} \\
t^{\beta(M')} = p^{\beta(M')} = \emptyset \\
\Longleftrightarrow \{ \; \pi(p)^{M'} = \emptyset \text{ (variables over nonempty sorts)}\} \\
\beta(M') \models_\nu \{X\}; t : p
\end{array}
$$

   (b) $\pi(p)^{M'}$ nonempty:

$$
\begin{array}{l}
M' \models_\nu \alpha(\{x_1, \ldots, x_k\}; t : p) \\
\Longleftrightarrow \{ \text{ def. } \alpha\} \\
M' \models_\nu x_1 \doteq x_1 \wedge \ldots \wedge x_k \doteq x_k \to t \prec p \\
\Longleftrightarrow \{ \text{ satisfaction (variables over nonempty sorts)}\} \\
\nu(t)^{M'} \subseteq \nu(p)^{M'} \\
\Longleftrightarrow \{ \; M' \in \mathsf{Mod}(\Phi(\Sigma, \emptyset)) \text{ and } t \notin \Pi\} \\
\nu(t)^{M'} \in \nu(p)^{M'} \\
\Longleftrightarrow \{ \text{ def. } \beta\} \\
\nu(t)^{\beta(M')} \in \nu(p)^{\beta(M')} \\
\Longleftrightarrow \{ \text{ satisfaction (variables over nonempty sorts)}\} \\
\beta(M') \models_\nu (\{x_1, \ldots, x_k\}; t : p)
\end{array}
$$

The general statement (1.4) follows easily from the above fact (1.5). $\square$

Fact 1.4.21 *The functor $\beta_{(\Sigma,\Gamma)}$ is an equivalence (in fact, an isomorphism) of categories $\mathsf{Mod}_{\mathcal{MA}}(\Phi(\Sigma, \Gamma))$ and $\mathsf{Mod}_{\mathcal{MA}}(\Sigma, \Gamma)$ for every $\mathcal{MEMB}$ theory (Σ, Γ).*

Proof. The inverse functor $\beta^{-1}_{(\Sigma,\Gamma)}$ sends a membership algebra $M \in \mathsf{Mod}(\Sigma, \Gamma)$ onto a multialgebra $M' \in \mathsf{Mod}(\Phi(\Sigma, \Gamma))$ such that $\beta(M') = M$ (i.e., $|M'| = |M|$, $f^{M'}(\overline{x}) = \{f^M(\overline{x})\}$ and for $p \in \Pi : p^{M'} = p^M$.) One verifies easily the isomorphism condition. $\square$

Given a membership algebra specification SP, we will call $\Phi(SP)$ a multialgebra specification *of the membership form.* Restricting the syntax of $\mathcal{MA}$ specifications in this way, we obtain a sub-institution $\mathcal{MAMB}$ of $\mathcal{MA}$ (and $\mathcal{MAH}$).

As a consequence of the above proposition (in particular, that $\beta_{(\Sigma,\Gamma)}$ is an equivalence), we get that the models of multialgebra specifications of membership form possess initial objects. We can verify this fact directly applying proposition 1.3.12.

Proposition 1.4.22 *If an $\mathcal{MA}$ specification SP is of the membership form, i.e., $SP = \Phi(SP_{\mathcal{MEMB}})$, then $T(SP)/_{\cong}$ is a model of PS.*

Proof.

1. We define the multialgebra $D(SP)$, modifying slightly the construction used earlier (definitions 1.3.1 and 1.3.10). Let (S, Ω, Π) be the signature of the membership specification $SP_{\mathcal{MEMB}}$ for which we have $SP = \Phi(SP_{\mathcal{MEMB}})$. As the carrier we take
   - $|D(SP)| = \{t \in T(\Omega)\} \subseteq \{t \in T(\Omega \uplus \Pi) : SP \models t \doteq t\}$, and we let
   - $t^{D(SP)} = \{t\}$, for $t \in T(\Omega)$ and
   - $p^{D(SP)} = \{t \in T(\Omega) : SP \models t \prec p\}$, for $p \in \Pi$
   - for $p \in \Pi$, $f \in \Omega$, $f(p)$ is defined by pointwise extension: $f(p)^{D(SP)} = \bigcup_{t \in p^{D(SP)}} f(t)^{D(SP)}$.

2. By the same argument as for $T(SP)$ we get that there is a unique homomorphism $h_A : D(SP) \to A$ for each $A \in \mathsf{Mod}(SP)$, so let $\cong$ be the equivalence relation defined on $D(SP)$ by $\cong \; = \bigcap_A \sim_{h_A}$ (cf. 1.3.10).

3. We show now that $D(SP)/_{\cong} \in \mathsf{Mod}(SP)$. Let $\alpha : X \to D(SP)/_{\cong}$ be an arbitrary assignment – for a term $t \in T(\Omega, X)$, we will denote by $\alpha(t)$ a term obtained by substituting some representative $s' \in \alpha(x) = [s']$ for each occurrence of x in t. Let $s, t \in T(\Omega, X)$, and $p \in \Pi$, and consider possible axioms:

   - $s \doteq t$: then, in particular, $SP \models \alpha(s) \doteq \alpha(t)$, and so for any $A \in \mathsf{Mod}(SP) : \alpha(s)^A = \alpha(t)^A$, i.e. $\alpha(s) \cong \alpha(t)$, and so $D(SP)/_{\cong} \models_\alpha s \doteq t$
   - $s \prec t$: from $SP \models \alpha(s) \prec \alpha(p)$ we obtain that $\alpha(s)^A \in \alpha(p)^A = p^A$ for all $A \in \mathsf{Mod}(SP)$, and so $D(SP)/_{\cong} \models_\alpha s \prec p$.
   - Horn formulae: $SP \models_\alpha a_1, \ldots a_n \to a$, where a_i, a atoms of the above form:
     (a) If $A \models \alpha(a)$ for all $A \in \mathsf{Mod}(SP)$, then $D(SP)/_{\cong} \models_\alpha a$, by the above arguments.
     (b) Otherwise, for some $A \in \mathsf{Mod}(SP) : A \not\models \alpha(a)$. Then we must also have $A \not\models \alpha(a_i)$ for some i. We show that this implies $D(SP)/_{\cong} \not\models_\alpha a_i$ – two cases:
       i. $a_i \equiv s \doteq t$. It means that $\alpha(s)^A \neq \alpha(t)^A$, hence $\alpha(s) \not\cong \alpha(t)$. By the definition of Φ, both $s' \doteq s'$ and $t' \doteq t'$ are among the axioms of SP (where s, resp. t is obtained from s', resp. t' by substituting some deterministic terms for some variables). So s' is not the same element of the carrier of A as t', and we have that $D(SP)/_{\cong} \not\models_\alpha s \doteq t$, i.e., $D(SP)/_{\cong} \not\models_\alpha a_i$.
       ii. $a_i \equiv s \prec p$. By the definition of Φ, $s' \doteq s'$ is among the axioms. Thus $\alpha(s)^A \notin \alpha(p)^A$ i.e. $SP \not\models \alpha(s) \prec \alpha(p)$, so $D(SP)/_{\cong} \not\models_\alpha s \prec p$.

This gives that $D(SP)/_{\cong} \in Mod(SP)$ and by 2) that it is initial model.

4. Finally, we show that $D(SP)/_{\cong} \simeq T(SP)/_{\cong}$. The only difference in the construction was that in $D(SP)/_{\cong}$ we excluded all $p \in \Pi$ which would be taken into account in constructing the carrier of $T(SP)/_{\cong}$, i.e., for all $p \in \Pi : [p] \notin |D(SP)/_{\cong}|$. (We started with $|D(SP)| = \{t \in T(\Omega)\}$ instead of $|T(SP)| = \{t \in T(\Omega \uplus \Pi) : SP \models t \doteq t\}$.) Since $D(SP)/_{\cong} \in \mathsf{Mod}(SP)$, if $SP \models p \doteq p$, we can conclude that there must exist a $t \in T(\Omega)$ (in particular, $SP \models t \doteq t$) such that $p^{D(SP)/_{\cong}} = \{[t]\} = t^{D(SP)/_{\cong}}$ and $D(SP)/_{\cong} \models t \prec p$. But this also means, by the construction of $D(SP)/_{\cong}$, that $SP \models t \prec p$. Hence, in $T(SP)/_{\cong}$ we would obtain $[t] = [p]$. In other words, excluding the symbols Π from the construction does not change the result and $D(SP)/_{\cong} \simeq T(SP)/_{\cong}$.

□

1.5 Concluding remarks

In this chapter we have given the formal mathematical background for the thesis. We have summarized the relevant notions concerning multialgebra specifications: the signature category, the logical sentence functor and the model functor. We have showed that multialgebras form an exact institution, $\mathcal{MA}$. This result gives us the amalgamation lemma as an immediate corollary. The amalgamation lemma is traditionally used for giving the semantics of parameter passing for parameterized specifications and parameterized datatypes. In chapter 4 we will generalize the traditional notion of semantics for parameterized datatype specifications and we will use the amalgamation lemma to define the semantics of actual parameter passing

It's accepted that institutions offers the right level of abstraction to compare and combine different specification formalisms. We have shown how the institution of partial algebras and membership algebras can be embedded into $\mathcal{MA}$, we thus suggest that the institution of multialgebras, $\mathcal{MA}$, may provide an adequate framework for both *comparing* and *combining* the advantages of earlier approaches within a unified framework. Based on the embedding we transform partial algebra specifications to $\mathcal{MA}$ specifications in chapter 3. The reason for doing this is to be able to do error recovery for partial algebra specifications. We have also summarized the known results about initial models for axiomatic classes of multialgebras from [22]. We have given a new proof for this results, and we have also managed to get a slightly stronger result, but the general problem remains still open.

Chapter 2

Logic for Multialgebras

In this chapter we give two new quantifier free logics for the institution of multialgebras, $\mathcal{MA}$, from chapter 1. There exist several logics for multialgebras but none for the institution $\mathcal{MA}$. The first reasoning system for multialgebras was presented by Hussman in [22], the system was used for term rewriting but it was not complete. Walicki gave the first sound and complete reasoning system for multialgebras in his thesis [50, 53], but he didn't allow operations to be partial, i.e. returning the emptyset. Konikowska and Białasik developed a sound and complete reasoning system for multialgebras with partial operations in [5, 6]. However, their language did not include the element equality $\doteq$, and, moreover, requires full FOL for deriving consequences of specification.

Our objective is to design a quantifier-free logic for deriving consequences of multialgebra specifications. First, using the technique of Rasiowa-Sikorski from [44], we design a sound and complete system R-S. This system is closely related to the first order logic for multialgebras given by Konikowska and Białasik in [6]. However, the element equality $\doteq$ can't be expressed in their language by a set of formulae without the use of quantifiers, and this is related to the fact that to express emptiness or non-emptiness of the carrier, quantified formulae are needed. E.g., $\exists x : x \prec x$ expresses non-emptiness of the carrier which, in our language, can be expressed by the quantifier-free formula $x \doteq x$ (with *only implicit* quantification over possible assignments, c.f. remark 1.1.16). Finally, and most significantly, the language from [6], unless extended to full first-order, is not expressive enough to state non-emptiness of any result set. Consequently, even the quantifier free tautologies have all to take into account the possibility that any involved term may yield an empty result.

This not only yields fewer and less specific tautologies, but has also more practical aspects. Writing specifications one certainly wants the possibility to state that a term is deterministic. The axiom $f(x) \doteq f(x)$ states that the operation f is a total function, and such statements figure naturally as assumptions (or consequences) in the formulae one wants to prove – preferably without the use of full first-order logic. Besides, there is the whole tradition of algebraic specifications based on equational axioms and equational reasoning. The element equality, $\doteq$, present in the institution of multialgebras, given in section 1.2, makes comparison and embedding of other institutions to the institution of multialgebras simple and straightforward, as done in section 1.4.1 and 1.4.2, without the use of quantifiers.

Although we consider the lack of a connective corresponding to $\doteq$ in [6] a serious drawback, our development and presentation owe quite a lot to this work. We utilize the technique of Rasiowa and Sikorski, [44], which was brought to our attention by [6] and which is nicely summarized in [26, 25] (and recently used also in [2]). It gives a general way for designing logics based on the semantic properties of the atomic predicates, and we apply it to our case of multialgebraic specifications with $\prec$ and $\doteq$.

In Section 2.1, we design a Rasiowa-Sikorski system, R-S, for quantifier-free logic over $\prec$ and $\doteq$, and prove it's soundness and completeness. Following the cited works, we also define a unique deduction strategy which can be used for implementing the logic. In Section 2.2, we address the issue of proving consequences of specifications. Specifications are sets of sequents and we want to derive their consequences, i.e., new sequents. We indicate the required translation schema and extend the R-S system with one rule needed for this purpose. Finally, in Section 2.3, we transform the obtained system to a sound and complete Gentzen calculus GS, which is more user-friendly than the R-S system for proving theorems by hand. In order to handle proofs of consequences of theories without any intermediary translation of the involved sequents, we replace the axiom rule (as well as various rules for the logical connectives) by the specific cut rules, originating from [41]. We thus obtain a system for direct reasoning about specification, where reasoning about sequents over atomic formulae involves only such sequents. Besides extension of the language with the useful predicate $\doteq$, we consider this result an improvement – by simplification – of the full first-order Gentzen system from [6].

2.1 The R-S calculus

We now present a quantifier free Rasiowa-Sikorski (R-S) deduction system with set inclusion and equality for multialgebras. The R-S system illustrates a powerful way of designing logical deduction systems based on semantical properties of atomic predicates of the language, which was originally introduced in [44]. Our presentation in this section is an adaptation and extension of a similar logic described in [6].

The R-S system processes sets of formulae (clauses). However, it allows one also to define a specific deduction strategy in which such sets are considered as ordered *sequences* of formulae without repetitions – this is the interpretation we will be using in this and next section. Particular sequences are singled out as *axiomatic*, in our case, sequences containing a formula or subsequence of the form:

- $x \prec x$ for a variable x

- $\phi, \neg\phi$ for a formula ϕ
- $\neg\mathcal{E}_s, t_s \prec t'_s$.

The order of occurrences of such formulae in a sequence does not matter. Given a set of formulae, we say that it satisfies the *'axiomatic sequence condition'* if the set involves formulae from which an axiomatic sequence could be formed.

Definition 2.1.1 *A structure $\langle A, \alpha \rangle$ satisfies a sequence $\Gamma = \gamma_1, \ldots, \gamma_n$, written $\langle A, \alpha \rangle \models \Gamma$, iff $\langle A, \alpha \rangle \models \gamma_i$ for some i.*

Hence the "," should be viewed as a meta-disjunction.

An R-S rule has one of the following forms, where Γ_i are sequences:

$$\frac{\Gamma_1}{\Gamma_2}, \quad \frac{\Gamma_1}{\Gamma_2 \mid \Gamma_3}, \text{ or } \quad \frac{\Gamma_1}{\Gamma_2 \mid \Gamma_3 \mid \Gamma_4}$$

Both sides of the " $|$ " sign have to hold for making an expression involving $|$ true, hence it should be viewed as a meta-conjunction.

The rules are designed so that they are invertible and one uses a strong notion of soundness.

Definition 2.1.2 *An (R-S) rule is sound when, for any structure $\langle A, \alpha \rangle$, $\langle A, \alpha \rangle$ satisfies the premise iff it satisfies the conclusion.*

The strength of this notion lies not only in the requirement of invertibility but also in the use of a structure – it says that the premise is satisfied iff the conclusion is *for any given* assignment. (The usual notion of soundness is, of course, implied by this one.)

In addition to axiomatic sequences, one also identifies the *indecomposable* sequences – in our case, these are given by the following definition.

Definition 2.1.3 *A Σ formula is indecomposable iff it has one of the following forms:*

- *$\mathcal{E}_s$ or $\neg\mathcal{E}_s$, $s \in S$*
- *$x \prec y$ or $\neg(x \prec y)$, $x, y \in X$*
- *$x \prec f(x_1, \ldots, x_n)$ or $\neg(x \prec f(x_1, \ldots, x_n))$, $x, x_i \in X$ and $f \in \Omega$ (f is possibly a constant).*

A sequence of formulae is indecomposable iff every formula in the sequence is indecomposable.

No rules can modify the indecomposable formulae and so if such a formula appears in a sequence during the proof, it will not be changed by any subsequent application of rules.

The R-S calculus has two types of rules, *replacement* rules and *expansion* rules. The goal of the replacement rules is to transform decomposable formulae leading either to axiomatic sequences or to indecomposable formulae (i.e., expressions involving only variables or function application to variables). Such rules have only one explicit formula in the premise sequence which is transformed, possibly with addition of a new formula, in the conclusion. There is exactly one decomposition rule for each case of a decomposable formula and schemata for decomposable formulae are disjoint, i.e., precisely one decomposition rule can be applied to any decomposable formula at any stage. In particular, we will have one rule for every positive decomposable formula, like $t \doteq t'$, and one rule for the corresponding negative formula, $\neg(t \doteq t')$.

The expansion rules are used to add logical consequences of the indecomposable formulae from the premise. They merely augment the premise sequence with some additional formulae without changing the formula itself.

In the notation for rules, we will use the sign "*" to indicate repetition of the active formula from the premise in the conclusion. The rule (VII–) given in the calculus below:

$$\frac{\Gamma', \neg(t \prec t'), \Gamma''}{\Gamma', x \prec t, \Gamma'', * \mid \Gamma', \neg(x \prec t'), \Gamma'', *} \quad \text{where } t \notin X \text{ and } x \in X \text{ arbitrary}$$

should be read in the following way:

$$\frac{\Gamma', \neg(t \prec t'), \Gamma''}{\Gamma', x \prec t, \Gamma'', \neg(t \prec t') \text{ and } \Gamma', \neg(x \prec t'), \Gamma'', \neg(t \prec t')}$$

where $t \notin X$ and $x \in X$ arbitrary

Such a repetition will take place only in the implicit presence of existential quantifier and will be needed to ensure the possibility of finding an adequate witness. The above rule can be thus seen as:

$$\frac{\Gamma', \neg(t \prec t'), \Gamma''}{\exists x : \Gamma', x \prec t, \Gamma'', \text{ and } \Gamma', \neg(x \prec t'), \Gamma'',} \quad \text{where } t \notin X$$

We include all relevant rules and axioms from the logic given in [6] (i.e., except the replacement rules for quantifiers and the let-construction). We extended the logic by new replacement rules for the element equality, i.e. the rules (IX+) to (XI–).

Remember that the formula $\neg\mathcal{E}_s$ is a logical symbol abbreviating the statement that the carrier of a sort s is nonempty, i.e., $x_s \doteq x_s$. Similarly, $\mathcal{E}_s$ denotes that the carrier of sort s is empty, i.e., $\neg(x_s \doteq x_s)$. The only function of the restriction $t_s \neq x_s$ in the rules (X) and (XI) is to prevent further decomposition of such formulae.

The R-S proof system

Axiomatic sequences ***(order does not matter)***

(I) $\Gamma, x \prec x, \Gamma'$, $x \in X$ (II) $\Gamma, \gamma, \Gamma', \neg\gamma, \Gamma''$, $\gamma \in \mathcal{F}_{\Sigma,X} \cup \{\mathcal{E}\}$ (III) $\Gamma, \neg\mathcal{E}_s, \Gamma', t_s \prec t'_s, \Gamma''$

Replacement rules ***(unique decomposable premise formula)***

	+	−
(IV)		$\dfrac{\Gamma', \neg\neg\gamma, \Gamma''}{\Gamma', \gamma, \Gamma''}$
(V)	$\dfrac{\Gamma', \gamma \vee \phi, \Gamma''}{\Gamma', \gamma, \phi, \Gamma''}$	$\dfrac{\Gamma', \neg(\gamma \vee \phi), \Gamma''}{\Gamma', \neg\gamma, \Gamma'' \mid \Gamma', \neg\phi, \Gamma''}$
(VI)	$\dfrac{\Gamma', \gamma \wedge \phi, \Gamma''}{\Gamma', \gamma, \Gamma'' \mid \Gamma', \phi, \Gamma''}$	$\dfrac{\Gamma', \neg(\gamma \wedge \phi), \Gamma''}{\Gamma', \neg\gamma, \neg\phi, \Gamma''}$
(VII)	$\dfrac{\Gamma', t \prec t', \Gamma''}{\Gamma', \neg(x \prec t), x \prec t', \Gamma''}$ where $t \notin X$, and $x \in X$ is fresh	$\dfrac{\Gamma', \neg(t \prec t'), \Gamma''}{\Gamma', x \prec t, \Gamma'', * \mid \Gamma', \neg(x \prec t'), \Gamma'', *}$ where $t \notin X$ and $x \in X$ arbitrary
(VIII)	$\dfrac{\Gamma', x \prec f(\ldots, t, \ldots), \Gamma''}{\Gamma', y \prec t, \Gamma'', * \mid \Gamma', x \prec f(\ldots, y, \ldots), \Gamma'', *}$ where $y \in X$ arbitrary and $t \notin X$	$\dfrac{\Gamma', \neg(x \prec f(\ldots, t, \ldots)), \Gamma''}{\Gamma', \neg(y \prec t), \neg(x \prec f(\ldots, y, \ldots)), \Gamma''}$ where $y \in X$ is fresh and $t \notin X$
(IX)	$\dfrac{\Gamma', t \doteq t', \Gamma''}{\Gamma', t \doteq x, \Gamma'', * \mid \Gamma', t' \doteq x, \Gamma'', *}$ where $t, t' \notin X$ and $x \in X$ arbitrary	$\dfrac{\Gamma', \neg(t \doteq t'), \Gamma''}{\Gamma', \neg(t \doteq x), \neg(t' \doteq x), \Gamma''}$ where $t, t' \notin X$ and $x \in X$ is fresh
(X)	$\dfrac{\Gamma', t_s \doteq x_s, \Gamma''}{\Gamma', t_s \prec x_s, \Gamma'' \mid \Gamma', x_s \prec t_s, \Gamma'' \mid \Gamma', \neg\mathcal{E}_s, \Gamma''}$ where $x_s \in X$ and $t_s \neq x_s$	$\dfrac{\Gamma', \neg(t_s \doteq x_s), \Gamma''}{\Gamma', \mathcal{E}_s, \neg(x_s \prec t_s), \neg(t_s \prec x_s), \Gamma''}$ where $x_s \in X$ and $t_s \neq x_s$
(XI)	$\dfrac{\Gamma', x_s \doteq t_s, \Gamma''}{\Gamma', t_s \prec x_s, \Gamma'' \mid \Gamma', x_s \prec t_s, \Gamma'' \mid \Gamma', \neg\mathcal{E}_s, \Gamma''}$ where $x_s \in X$ and $t_s \neq x_s$	$\dfrac{\Gamma', \neg(x_s \doteq t_s), \Gamma''}{\Gamma', \mathcal{E}_s, \neg(x_s \prec t_s), \neg(t_s \prec x_s), \Gamma''}$ where $x_s \in X$ and $t_s \neq x_s$

Expansion rules ***(indecomposable premise formulae)***

(XII) $$\frac{\Gamma', \neg(x \prec y), \Gamma''}{\Gamma', \neg(x \prec y), \neg(y \prec x), \Gamma''}$$

(XIII) $$\frac{\Gamma', \neg(y \prec x), \Gamma'', \neg(x \prec z), \Gamma'''}{\Gamma', \neg(y \prec x), \Gamma'', \neg(x \prec z), \neg(y \prec z), \Gamma'''}$$

$$\text{(XIV)} \quad \frac{\Gamma', \neg(y \prec x), \Gamma'', \neg(x \prec f(\overline{z})), \Gamma'''}{\Gamma', \neg(y \prec x), \Gamma'', \neg(x \prec f(\overline{z})), \neg(y \prec f(\overline{z})), \Gamma'''} \quad f \text{ is possibly a constant}$$

$$\text{(XV)} \quad \frac{\Gamma', \neg(y \prec z), \Gamma'', \neg(x \prec f(\ldots, z, \ldots)), \Gamma'''}{\Gamma', \neg(y \prec z), \Gamma'', \neg(x \prec f(\ldots, z, \ldots)), \neg(x \prec f(\ldots, y, \ldots)), \Gamma'''}$$

f is possibly a constant

$$\text{(XVI)} \quad \frac{\Gamma', \neg\mathcal{E}_s, \Gamma'', \neg(x_{s'} \prec f(\ldots, y_s, \ldots)), \Gamma'''}{\Gamma', \neg\mathcal{E}_s, \Gamma'', \neg(x_{s'} \prec f(\ldots, y_s, \ldots)), \neg\mathcal{E}_{s'}, \Gamma'''} \quad f \text{ is possibly a constant}$$

Example 2.1.4 *We illustrate use of the logic by proving the tautology: $c \doteq c, c \prec x \to x \prec c$, i.e: $\neg(c \doteq c), \neg(c \prec x), x \prec c$. We mark the active formulae by boldface. Similarly, the variable introduced in the conclusion is in boldface. If a branch terminates, the axiomatic subsequences are underlined. We drop sort subscripts.*

$$\frac{\neg(\mathbf{c} \doteq \mathbf{c}), \neg(c \prec x), x \prec c}{\neg(c \doteq \mathbf{y}), \neg(c \prec x), x \prec c} \quad \textit{(IX–)}$$

$$\frac{\neg(\mathbf{c} \doteq \mathbf{y}), \neg(c \prec x), x \prec c}{\mathcal{E}, \neg(y \prec c), \neg(c \prec y), \neg(c \prec x), x \prec c} \quad \textit{(X–)}$$

$$\frac{\mathcal{E}, \neg(y \prec c), \neg(\mathbf{c} \prec \mathbf{y}), \neg(c \prec x), x \prec c}{\mathcal{E}, \underline{\neg(y \prec c)}, \underline{(\mathbf{y} \prec c)}, \neg(c \prec x), x \prec c, * \mid i)} \quad \textit{(VII–)}$$

$$\frac{i) = \mathcal{E}, \neg(y \prec c), \neg(\mathbf{y} \prec y), \neg(\mathbf{c} \prec \mathbf{x}), x \prec c, \neg(c \prec y)}{\mathcal{E}, \underline{\neg(y \prec c)}, \neg(y \prec y), \underline{\mathbf{y} \prec c}, x \prec c, \neg(c \prec y), * \mid ii)} \quad \textit{(VII–)}$$

$$\frac{ii) = \mathcal{E}, \neg(y \prec c), \neg(y \prec y), \neg(\mathbf{y} \prec \mathbf{x}), x \prec c, \neg(c \prec y), \neg(c \prec x)}{\neg(y \prec c), \mathcal{E}, \neg(y \prec y), \neg(y \prec x), \neg(x \prec y), x \prec c, \neg(c \prec y), \neg(c \prec x)} \quad \textit{(XII)}$$

$$\frac{\mathcal{E}, \neg(\mathbf{y} \prec \mathbf{c}), \neg(y \prec y), \neg(y \prec x), \neg(\mathbf{x} \prec \mathbf{y}), x \prec c, \neg(c \prec y), \neg(c \prec x)}{\mathcal{E}, \neg(y \prec c), \neg(y \prec y), \neg(y \prec x), \neg(x \prec y), \underline{\neg(x \prec c)}, \underline{x \prec c}, \neg(c \prec y), \neg(c \prec x)} \quad \textit{(XIV)}$$

2.1.1 Construction of a unique deduction tree

Before proving soundness and completeness of the calculus, we show first that for a given sequence $\Gamma = \gamma_1, \ldots, \gamma_n$ one can choose a unique, canonical deduction tree. This fact will be used in the proof of completeness, but it is also of independent importance since it suggests the way of possible implementation of the logic.

The strategy was illustrated in the above example 2.1.4. We start with the first formula γ_1. If it is decomposable, we apply the appropriate rule, (IX–). We now check whether the obtained indecomposable formulae ("to the left" of the "active position" in the obtained sequence[1]) can be used in any expansion rule and if they can, we apply the rule. This was not possible in the example, so we repeated the application of a decomposition rule in the second step. After this step, still no expansion rule was applicable to the obtained formulae $\mathcal{E}, \neg(y \prec c)$ – so we consider the next formula to the right to which we could apply a decomposition rule, (VII–). Left branch gets closed and in the right we consider all indecomposable formulae obtained so far ("to the left" of the "active position"). Transitivity rule (XIV) applied to $\neg(y \prec c), \neg(y \prec y)$ yields a repetition, so it is not applied. The first possibility is application of the symmetry rule (XII) to $\neg(x \prec y)$, after which an application of (XIV) yields an axiomatic sequence. The following definition captures the above strategy.

Definition 2.1.5 *An R-S rule ρ is correctly applicable to a sequence Γ iff one of the following conditions is satisfied:*

1. *ρ is an R-S rule which augments Γ by some new formula or*
2. *there is no rule with the above property that can be applied to a formula or pair of formulae that lies to the left of the (active) formula or pair of formulae to which ρ is applicable.*

The first point refers exclusively to the expansion rules. In the second point, ρ may be a replacement rule in which case it is applied to the leftmost decomposable formula, so that no expansion rule can be applied "to the left" of it. If, in this second case, ρ turns out to be an expansion rule, we see that first we have to apply the rule with one premise formula, i.e., (XII) or else the rule with two premise formulae which, together, lie as far "to the left" as possible. Since we only can use one replacement rule for a formula at any time and point 2 in 2.1.5 uniquely defines the expansion rule that is correctly applicable, we get that there is at most one R-S rule that is correctly applicable to any sequence Γ at any time.

By a deduction tree for a sequence Γ we mean a tree with Γ labelling the root, where the number and labelling of the children of each node originates from the application of some rule to the (sequence labelling the) node itself. Such a tree is a proof if all leaves are labelled by axiomatic sequences. The above definition and remarks allow us to define a unique decomposition tree for any sequence.[2] We identify vertex v with its label Π.

Definition 2.1.6 *A decomposition tree $DT(\Gamma)$ for a sequence Γ is a ternary tree with vertices being (or labelled by) sequences of formulae defined inductively by:*

[1]Indexing formulae in a sequence as $\gamma_1, \gamma_2, \ldots, \gamma_n$, by "$\gamma_i$ lying to the left of γ_j" we mean simply that $i \leq j$. The "active position" is the index of the explicit formula from the premise – the one which has been processed by the rule.

[2]We assume, in general, the the set X of all variables is countable. More generally, here we only need the assumption that it is well ordered, so that we can choose the first variable not present in a sequence for a fresh variable and we choose "the next variable in the ordering" for an arbitrary variable (for each formula which is processed repeatedly, i.e., inherited as indicated by *).

1. *The root of $DT(\Gamma)$ is Γ.*
2. *If a vertex Π is either:*
   (a) *an axiomatic sequence or*
   (b) *an indecomposable sequence to which no expansion rule is correctly applicable*

   then Π is a leaf.
3. *Otherwise the vertex Π has:*
   (a) *A single child Π', if the unique rule correctly applicable to Π has a single conclusion.*
   (b) *Two children Π', Π'', if the unique rule correctly applicable to Π has two conclusions.*
   (c) *Three children Π', Π'', Π''', if the unique rule correctly applicable to Π has three conclusions.*

We thus obtain

Lemma 2.1.7 *For any sequence Γ one can choose a unique decomposition tree, $DT(\Gamma)$.*

This fact can be used in implementation of the logic. We will use it in the proof of completeness, where also the following, obvious property will be of importance.

Lemma 2.1.8 *Assume that $B = \Pi_1, \Pi_2, \Pi_3, \ldots$ is an infinite branch in $DT(\Gamma)$ and let Γ_B be the set of all formulae occurring on the vertices of B. Then:*

1. *Γ_B is closed under all expansion rules.*
2. *If $\gamma_i \in \Gamma_B$ is decomposable, then there exists a vertex $\Pi_i \in B$ such that $\Pi_i = \Gamma', \gamma_i, \Gamma''$, where Γ' is indecomposable (possibly empty) and closed under all expansion rules. The vertex Π_{i+1}, following Π_i in B, is (one of) the conclusion(s) $\Gamma', \gamma_{i+1}, \Gamma''$ obtained by the correct application of the appropriate decomposition rule to Π_i.*

2.1.2 Soundness

Theorem 2.1.9 *The R-S system is sound.*

Proof. We only have to prove that the replacement rules involving $\doteq$ are sound, i.e. the rules (IX+) to (XI–). The axiomatic sequences and the remaining rules were proved sound in [6]. We consider an arbitrary structure $\langle A, \alpha \rangle$. We drop sort subscript assuming that we always address only the relevant sort. We consider only the cases when a sequence is satisfied because the explicit (active) formulae are satisfied, since the other cases are trivial.

1. (IX+)

$$\frac{\Gamma', t \doteq t', \Gamma''}{\Gamma', t \doteq x, \Gamma'', * \mid \Gamma', t' \doteq x, \Gamma'', *} \quad \text{where } t, t' \notin X \text{ and } x \in X \text{ arbitrary}$$

$\Downarrow$ If $\langle A, \alpha \rangle \models t \doteq t'$, then $\langle A, \alpha \rangle$ trivially satisfies both conclusions since $t \doteq t'$ is repeated on each side of the conjunction sign.
$\Uparrow$ If $\langle A, \alpha \rangle \models t \doteq x$ and $\langle A, \alpha \rangle \models t' \doteq x$, then $\alpha(t) = \alpha(x) = e = \alpha(x) = \alpha(t')$, where $e \in |A|$, so $\langle A, \alpha \rangle \models t \doteq t'$.

2. (IX–)

$$\frac{\Gamma', \neg(t \doteq t'), \Gamma''}{\Gamma', \neg(t \doteq x), \neg(t' \doteq x), \Gamma''} \quad \text{where } t, t' \notin X \text{ and } x \in X \text{ is a new variable}$$

$\Downarrow$ When $\langle A, \alpha \rangle \models \neg(t \doteq t')$, we have two cases:

(a) $|A| = \emptyset$ then $\alpha(x) = \emptyset = \alpha(t)$, so $\langle A, \alpha \rangle \models \neg(t \doteq x)$ and the conclusion holds.

(b) $|A| \neq \emptyset$, we have the following subcases:

- $\alpha(t) = \emptyset$, then $\langle A, \alpha \rangle \models \neg(t \doteq x)$ and the conclusion holds. Similarly if $\alpha(t') = \emptyset$.
- $|\alpha(t)| > 1$, then $\langle A, \alpha \rangle \models \neg(t \doteq x)$ and the conclusion holds. Similarly if $|\alpha(t')| > 1$.
- $\alpha(t) = e \neq e' = \alpha(t')$, then we must have either $\langle A, \alpha \rangle \models \neg(t' \doteq x)$ or $\langle A, \alpha \rangle \models \neg(t' \doteq x)$ since $\alpha(x)$ can't be equal to both e and e'.

$\Uparrow$ Suppose that the conclusion holds. We have two cases:

(a) $|A| = \emptyset$ then $\alpha(t) = \emptyset = \alpha(t')$, i.e., $\langle A, \alpha \rangle \models \neg(t \doteq t')$ and the premise holds.

(b) $|A| \neq \emptyset$, so if $\langle A, \alpha \rangle \models t \doteq x$, then for the conclusion to hold we must have $\langle A, \alpha \rangle \models \neg(t' \doteq x)$, i.e, $\langle A, \alpha \rangle \models \neg(t \doteq t')$. Similarly if $\langle A, \alpha \rangle \models t \doteq x$.

3. (X+)

$$\frac{\Gamma', t \doteq x, \Gamma''}{\Gamma', t \prec x, \Gamma'' \mid \Gamma', x \prec t, \Gamma'' \mid \Gamma', \neg\mathcal{E}, \Gamma''} \quad \text{where } x \in X \text{ and } t \neq x$$

$\Downarrow$ Suppose $\langle A, \alpha \rangle \models t \doteq x$ then:

$\alpha(x) = \alpha(t) = e \in |A|$, so $|A|$ is nonempty i.e $\langle A, \alpha \rangle \models \neg\mathcal{E}$ and $\langle A, \alpha \rangle \models t \prec x$ and $\langle A, \alpha \rangle \models x \prec t$.
$\Uparrow$ Suppose $\langle A, \alpha \rangle \models t \prec x$ and $\langle A, \alpha \rangle \models x \prec t$ and $\langle A, \alpha \rangle \models \neg\mathcal{E}$ then: $e = \alpha(x) \subseteq \alpha(t) \subseteq \alpha(x) = e$, i.e $e = \alpha(x) = \alpha(t)$, so $\langle A, \alpha \rangle \models t \doteq x$.

4. (X–)
$$\frac{\Gamma', \neg(t \doteq x), \Gamma''}{\Gamma', \mathcal{E}, \neg(x \prec t), \neg(t \prec x), \Gamma''} \quad \text{where } x \in X \text{ and } t \neq x$$

$\Downarrow$ Suppose that $\langle A, \alpha \rangle \models \neg(t \doteq x)$, two cases:

(a) $|A| = \emptyset$ then $\langle A, \alpha \rangle \models \mathcal{E}$.

(b) $|A| \neq \emptyset$, two possibilities:

i. There exists an element $e \in \alpha(t)$ such that $e \neq \alpha(x)$, then $\langle A, \alpha \rangle \models \neg(t \prec x)$, or

ii. $\alpha(x) = e'$ and $e' \notin \alpha(t)$, so $\langle A, \alpha \rangle \models \neg(x \prec t)$

$\Uparrow$ Three cases:

(a) $\langle A, \alpha \rangle \models \mathcal{E}$, then $\langle A, \alpha \rangle \models \neg(t \doteq x)$

(b) $\langle A, \alpha \rangle \models \neg(t \prec x)$, then there exists an element e such that $e \in \alpha(t)$ and $e \notin \alpha(x)$ so $\langle A, \alpha \rangle \models \neg(t \doteq x)$

(c) $\langle A, \alpha \rangle \models \neg(x \prec t)$, then there exists an element e such that $\alpha(x) = e$ and $e \notin \alpha(t)$ so $\langle A, \alpha \rangle \models \neg(t \doteq x)$

5. (XI+)
$$\frac{\Gamma', x_s \doteq t_s, \Gamma''}{\Gamma', t_s \prec x_s, \Gamma'' \mid \Gamma', x_s \prec t_s, \Gamma'' \mid \Gamma', \neg\mathcal{E}_s, \Gamma''} \quad \text{where } x \in X \text{ and } t \neq x$$

The proof of this rule is analogous to the proof of $(X+)$.

6. (XI–)
$$\frac{\Gamma', \neg(x_s \doteq t_s), \Gamma''}{\Gamma', \mathcal{E}_s, \neg(x_s \prec t_s), \neg(t_s \prec x_s), \Gamma''} \quad \text{where } x \in X \text{ and } t \neq x$$

— The proof of this rule is analogous to the proof of (X–).

$\square$

2.1.3 Completeness

We first show a lemma which gives the main part of the proof of completeness.

Lemma 2.1.10 *Given a set of indecomposable formulae, Γ_{ind}, that is closed under all expansion rules and that does not satisfy the axiomatic sequence condition (from section 2.1), there exist a structure $M_C = \langle A_C, \alpha_C \rangle$, such that $M_C \not\models \gamma$, for every formula $\gamma \in \Gamma_{ind}$, i.e. M_C is a counter-model for Γ_{ind}.*

Proof. Given such a Γ_{ind}, we define the relation $\sim$ on the set X of variables by:[3]

$$x \sim y \iff \neg(y \prec x) \in \Gamma_{ind}$$

Closure under expansion rules implies that $\sim$ is symmetric, rule (XII), and transitive, rule (XIII).

The relation $\asymp$ is the reflexive closure of $\sim$, i.e.:

$$x \asymp y = \sim \ \cup \{(x,x) : x \in X\}$$

Again, closure under expansion rules implies that $\asymp$ is a congruence wrt. function applications: given $x \asymp y$ then if $\neg(x \prec f(z)) \in \Gamma_{ind}$, then also $\neg(y \prec f(z)) \in \Gamma_{ind}$, by rule (XIV), and if $\neg(z \prec f(x)) \in \Gamma_{ind}$, then also $\neg(z \prec f(y)) \in \Gamma_{ind}$, by rule (XV).

We now define the counter-model $M_C = \langle A_C, \alpha_C \rangle$ for Γ_{ind} as follows:

1. Carrier sets:
   (a) $|A_C|_s = \emptyset$ iff $\neg\mathcal{E}_s \in \Gamma_{ind}$
   (b) $|A_C|_s = X_s/_{\asymp}$ – otherwise
2. Operations: for $f : s_1 \times \ldots \times s_n \to s_{n+1}$, we define:
   (a) $f([\overline{x}])^{A_C} = \emptyset$, if $|A_C|_{s_i} = \emptyset$ for some $1 \leq i \leq n+1$
   (b) $f([\overline{x}])^{A_C} = \{[y] : \neg(y \prec f(\overline{x})) \in \Gamma_{ind}\}$ – otherwise
3. Assignment:
   (a) $\alpha_C(x) = \emptyset$ iff $\neg\mathcal{E}_s \in \Gamma_{ind}$
   (b) $\alpha_C(x) = [x]$ – otherwise

We prove that M_C is indeed a counter-model for Γ_{ind}, i.e., $M_C \not\models \gamma$, for every formula $\gamma \in \Gamma_{ind}$. We prove the statement for each type of indecomposable formula.

1. $\gamma = \mathcal{E}_s \in \Gamma_{ind}$, since Γ_{ind} is non-axiomatic it means that $\neg\mathcal{E}_s \notin \Gamma_{ind}$, hence: $|A_C|_s = X/_{\asymp} \neq \emptyset$, so $M_C \not\models \mathcal{E}_s$.
2. $\gamma = \neg\mathcal{E}_s \in \Gamma_{ind}$, so $|A_C|_s = \emptyset$, and we have that $M_C \not\models \neg\mathcal{E}_s$
3. $\gamma = x_s \prec y_s \in \Gamma_{ind}$, then $\neg\mathcal{E}_s \notin \Gamma_{ind}$ (otherwise Γ_{ind} would be axiomatic) and so $|A_C|_s = X_s/_{\asymp}$ and $\alpha_C(x) = [x] \neq \emptyset \neq [y] = \alpha_C(y)$. Then $M_C \models x \prec y \iff [x] \subseteq [y]$, but this holds only if $[x] = [y]$, i.e., only if $\neg(x_s \prec y_s) \in \Gamma_{ind}$, which is not the case since Γ_{ind} is not axiomatic.
4. $\gamma = \neg(x_s \prec y_s) \in \Gamma_{ind}$, then we have the following subcases:
   (a) $|A_C|_s = \emptyset$ then $\alpha_C(x) = \emptyset = \alpha_C(y)$, so $M_C \not\models \neg(x_s \prec y_s)$

[3] We assume, in general, that the set X of all variables is countable. If it is not, we choose here only the variables which occur in some formula in Γ_{ind}.

(b) $|A_C|_s = X/_{\asymp}$, then $\alpha_C(x) = [x] \neq \emptyset \neq [y] = \alpha_C(y)$. Then $M_C \models x \prec y \iff [x] \subseteq [y]$, but this holds since $\neg(x_s \prec y_s) \in \Gamma_{ind}$, i.e. $M_C \not\models \neg(x_s \prec y_s)$.

5. $\gamma = x_s \prec f_s(x_1, \ldots, x_n) \in \Gamma_{ind}$, since Γ_{ind} is non-axiomatic we have $\neg\mathcal{E}_s \notin \Gamma_{ind}$, and $|A_C|_s = X/_{\asymp}$. We have the following subcases:

   (a) $\alpha_C(x_i) = \emptyset$ for some $1 \leq i \leq n$, then $|A_C|_{s_i} = \emptyset$ by definition of assignment, (and also $\neg\mathcal{E}_{s_i} \in \Gamma_{ind}$ by definition of M_C 1a). But then $f^{M_C}(x_1, \ldots, x_n) = \emptyset$ by 2a, while $\alpha_C(x) = [x] \neq \emptyset$, so $M_C \not\models x_s \prec f_s(x_1, \ldots, x_n)$.

   (b) For all $1 \leq i \leq n : |A_C|_{s_i} \neq \emptyset$. Then $[x] \in f([x_1], \ldots, [x_n])$ iff $\neg(x \prec f(x_1, \ldots, x_n)) \in \Gamma_{ind}$, but this is a contradiction since Γ_{ind} is non-axiomatic, so $M_C \not\models x_s \prec f_s(x_1, \ldots, x_n)$.

6. $\gamma = \neg(x_s \prec f_s(x_1, \ldots, x_n)) \in \Gamma_{ind}$, we have the following subcases:

   (a) $|A_C|_s = \emptyset$, then $f^{A_C}(x_1, \ldots, x_n) = \emptyset = \alpha_C(x)$, so $M_C \not\models \neg(x_s \prec f_s(x_1, \ldots, x_n))$.

   (b) $|A_C|_s = X_s/_{\asymp}$, we have two subcases:

      i. $\alpha_C(x_i) = \emptyset$ for some $1 \leq i \leq n$, then we have for the sort s_i of x_i that $\neg\mathcal{E}_{s_i} \in \Gamma_{ind}$. Since $\neg(x_s \prec f_s(x_1, \ldots, x_n)) \in \Gamma_{ind}$ and Γ_{ind} is closed under expansion rules we get by the expansion rule (XVI) that $\neg\mathcal{E}_s \in \Gamma_{ind}$, this is a contradiction since $\neg\mathcal{E}_s \in \Gamma_{ind}$ implies that $|A_C|_s = \emptyset$.

      ii. All the carriers of the sorts of the variables x_i are nonempty. So we have $[x] \in f([x_1], \ldots, [x_n])$ iff $\neg(x \prec f(x_1, \ldots, x_n)) \in \Gamma_{ind}$, and since the latter holds we get that $M_C \not\models \neg(x \prec f(x_1, \ldots, x_n))$.

$\square$

Using this lemma, we obtain the main completeness theorem.

Theorem 2.1.11 *The R-S system is complete: if* $\models \Gamma$ *(i.e.,* $\forall\langle A, \alpha\rangle : \langle A, \alpha\rangle \models \Gamma$*), then* $\vdash \Gamma$.

Proof. We show that if $\not\vdash \Gamma$, then $\not\models \Gamma$, i.e., there exists a structure M_C with $M_C \not\models \Gamma$. Let $DT(\Gamma)$ be the unique decomposition tree for Γ as defined in def. 2.1.6. There are two situations when $DT(\Gamma)$ is not a proof:

I. Some leaves are labelled by non-axiomatic sequences – then such leaves have labels containing only indecomposable formulae, or

II. The tree is infinite, which implies (by the König lemma) that there exists an infinite branch.

In either case we can find a set Γ_{ind} of indecomposable formulae closed under all expansion rules which is valid if Γ is valid. Thus, a counter-model M_C for Γ_{ind}, which exists by lemma 2.1.10, is also a counter-model for Γ.

I. The non-axiomatic sequence labelling one of the leaves can be taken as Γ_{ind} – by definition of $DT(\Gamma)$, Γ_{ind} is closed under all expansion rules. Since $M \models \Gamma$ implies $M \models \Gamma_{ind}$, lemma 2.1.10, giving a counter-model $M_C \not\models \Gamma_{ind}$, implies that $M_C \not\models \Gamma$.

II. Select an infinite branch B from $DT(\Gamma)$. If an indecomposable formula appears at some vertex of B, then it appears also at all subsequent vertices. Let Γ_{ind} be the union (infinite set) of all indecomposable formulae appearing in the labels of the vertices on the branch B. Γ_{ind} does not satisfy the axiomatic sequence condition, for if it does, then there exists a vertex at which this axiom occurs and which would terminate B. Also, by lemma 2.1.8, Γ_{ind} is closed under all expansion rules. Thus Γ_{ind} satisfies the conditions of lemma 2.1.10, so let M_C be the counter-model as it was defined in the proof of this lemma.[4] We show that M_C is a counter-model for all the formulae occurring in the labels of the vertices of B, and since the root vertex of B is the root of $DT(\Gamma)$, i.e., is labelled with Γ, we have that M_C is a counter-model for Γ. The proof goes by induction on the rank of a formula γ, $\mathrm{ord}(\gamma)$, which we define so that the applications of rules never increase the rank of the formulae in the sequence and, eventually, decrease it.

- $\mathrm{ord}(\gamma) = 0$, if γ is indecomposable;
- otherwise:
  - $\mathrm{ord}(x \prec t) = \mathrm{ord}(\neg(x \prec t)) = 1$ (where $t \notin X$, $x \in X$)
  - $\mathrm{ord}(t \prec t') = \mathrm{ord}(\neg(t \prec t')) = 2$ (where $t \notin X$)
  - $\mathrm{ord}(x \doteq t) = \mathrm{ord}(t \doteq x)) = \mathrm{ord}(\neg(x \doteq t)) = \mathrm{ord}(\neg(t \doteq x)) = 2$ (where t may be a variable, even x)
  - $\mathrm{ord}(t \doteq t') = \mathrm{ord}(\neg(t \doteq t')) = 3$ (where both $t \notin X$ and $t' \notin X$)
  - $\mathrm{ord}(\gamma' \vee \gamma'') = \mathrm{ord}(\gamma' \wedge \gamma'') = \max(\mathrm{ord}(\gamma'), \mathrm{ord}(\gamma'')) + 1$.
  - $\mathrm{ord}(\neg\gamma') = \mathrm{ord}(\gamma') + 1$, if γ' is not any of the above cases.

Now, if $M_C \models \Gamma$, then the set Γ_{sat} of all formulae γ', appearing in one of the vertices of B and such that $M_C \models \gamma'$, is nonempty, since $\Gamma \cap \Gamma_{sat} \neq \emptyset$. Let $\gamma_i \in \Gamma_{sat}$ be such that $\mathrm{ord}(\gamma_i) \leq \mathrm{ord}(\gamma')$, for every $\gamma' \in \Gamma_{sat}$. We show, by induction on the rank of γ_i, that it must be indecomposable.

Suppose that γ_i is decomposable. By point 2. of lemma 2.1.8, there exists a vertex $\Pi_i \in B$ such that $\Pi_i = \Gamma', \gamma_i, \Gamma''$, where Γ' is indecomposable (possibly empty) and closed under all expansion rules, and Π_{i+1} is the vertex following Π_i in B, with the label $\Gamma', \gamma_{i+1}, \Gamma'''$ obtained by the correct application of a decomposition rule. Considering the possible cases for γ_i, we show that there

[4]Here, as well as in the point I., the carrier of the counter-model M_C can be constructed only from the variables occurring in Γ_{ind}.

exists a $\gamma' \in \Gamma_{sat}$ with $\mathrm{ord}(\gamma') < \mathrm{ord}(\gamma_i)$, which contradicts the assumption about γ_i:

1. $\phi \vee \mu, \neg(\phi \vee \mu), \phi \wedge \mu, \neg(\phi \wedge \mu), \neg\neg\phi$. If γ_i has one of these forms, then $\mathrm{ord}(\gamma_{i+1}) < \mathrm{ord}(\gamma_i)$ and $M_C \models \gamma_{i+1}$, which contradicts the definition of γ_i.

2. $\neg(t \doteq t')$.

   (a) If neither $t, t' \notin X$, then by rule (IX–), $\Pi_{i+1} = \Gamma', \neg(t \doteq x), \neg(t' \doteq x), \Gamma''$. We then have that $M_C \models \gamma'$, where γ' is either $\neg(t \doteq x)$ or $\neg(t' \doteq x)$. In either case $\mathrm{ord}(\gamma') = 2 < 3 = \mathrm{ord}(\gamma_i)$ which contradicts the definition of γ_i.

   (b) If $t = x \in X$ or $t' = x \in X$ then, by rule (X–) or (XI–), $\Pi_{i+1} = \Gamma', \neg(t \prec x), \neg(x \prec t), \Gamma''$, and if $M_C \models \gamma_i$ then $M_C \models \gamma'$ where γ' is either $\neg(t \prec x)$ or $\neg(x \prec t)$. Hence $\mathrm{ord}(\gamma') \leq 1 < 2 = \mathrm{ord}(\gamma_i)$, which contradicts the definition of γ_i. ($\mathrm{ord}(\gamma') < 1$ when both $t, t' \in X$.)

3. $t \doteq t'$.

   (a) If $t \notin X$ and $t' \notin X$ then, by rule (IX+), γ_{i+1} is $t \doteq x$ (if B follows $DT(\Gamma)$ along the left conclusion), or $t' \doteq x$ (if B proceeds along the right conclusion – both cases are entirely analogous). If $M_C \models \gamma_{i+1}$ then we are done, since in either case $\mathrm{ord}(\gamma_{i+1}) = 2 < 3 = \mathrm{ord}(\gamma_i)$. However, it may happen that $M_C \not\models \gamma_{i+1}$ (because of a wrong choice of the variable x). Then Π_{i+1}, as well as all other vertices along B, inherit $t \doteq t'$ as $\Gamma', t \doteq x, \Gamma'', t \doteq t'$. [5] At some point, the trailing $t \doteq t'$ will be processed anew according to rule (IX+), introducing a new variable: $t \doteq y$ (or in $t' \doteq y$). Eventually, since $M_C \models t \doteq t'$, we will get the appropriate variable, say z, such that $M_C \models t \doteq z$ (or $M_C \models t' \doteq z$). [6] Satisfaction of this formula contradicts the assumption about $\gamma_i = (t \doteq t')$, since $\mathrm{ord}(t \doteq z) = 2 < 3 = \mathrm{ord}(t \doteq t')$.

   (b) If $t \in X$ or $t' \in X$ then γ_{i+1} has the form $t \prec x$ or $x \prec t$ (or t' instead t) or $\neg\mathcal{E}$ (by rule (X+) or (XI+)) and hence:
   $\mathrm{ord}(\gamma_{i+1}) \leq 1 < 2 = \mathrm{ord}(\gamma_i)$, which contradicts the definition of γ_i. ($\mathrm{ord}(\gamma_{i+1}) = 0$ if $\gamma_{i+1} = \neg\mathcal{E}$.)

4. $\neg(t \prec t')$.

   (a) If $t \notin X$ then $\Pi_{i+1} = \Gamma', \gamma_{i+1}, \Gamma'', \neg(t \prec t')$. The argument is entirely analogous to that in point 3a. B follows $DT(\Gamma)$ either along the left conclusion of the rule (VII–) with $\gamma_{i+1} = x \prec t$, or along the

[5] If $M_C \models \gamma''$ for some formula $\gamma'' \in \Gamma''$ with $\mathrm{ord}(\gamma'') = 3 = \mathrm{ord}(t \doteq t')$, then γ'' is either of the form $s \doteq s'$ or $\neg(s \doteq s')$. The latter case was covered in point 2 and the former is the same as the current case. (It may also be of the form $\phi \wedge \psi, \phi \vee \psi$ or $\neg\phi$ but all these cases have been covered in point 1.)

[6] Since the carrier $|A_C|$ of M_C is constructed as a quotient of the variable set X, the assignment α_C is surjective.

right one with $\gamma_{i+1} = \neg(x \prec t')$. In either case $\mathrm{ord}(\gamma_{i+1}) = 1 < 2 = \mathrm{ord}(\gamma_i)$, so if $M_C \models \gamma_{i+1}$, we are done.[7] Eventually, the trailing $\neg(t \prec t')$, inherited in Π_{i+1} and all following vertices in B, will be processed again and again along B according to the rule (VII–) providing, eventually, a witness variable, say y, such that $M_C \models y \prec t$ – if B happens to proceed along a left conclusion, or $M_C \models \neg(y \prec t')$ – if B proceeds along the right conclusion. In either case, the satisfied formula has lower rank than γ_i which contradicts its definition.

(b) If $t \in X$ then, as γ_{i+1} is decomposable, $t' = f(\ldots, t'', \ldots)$ with $t'' \notin X$. By rule (VIII–), $\Pi_{i+1} = \Gamma', \neg(y \prec t''), \neg(t \prec f(\ldots, y, \ldots), \Gamma''$, and the assumption $M_C \models \neg(t \prec t')$ implies $M_C \models \neg(y \prec t''), \neg(t \prec f(\ldots, y, \ldots)$. Before any branching of $DT(\Gamma)$, i.e., still along the branch B, the rule (VIII–) (and possibly some expansion rules) will be applied to both these formulae until we get only indecomposable formulae. Since $M_C \models \gamma_i = \neg(t \prec t')$, we must have $M_C \models \gamma'$ for one of these indecomposable formulae, which contradicts the definition of γ_i.

5. $t \prec t'$.

(a) If $t \notin X$ then, by rule (VII+), $\Pi_{i+1} = \Gamma', \neg(x \prec t), x \prec t', \Gamma''$. Then $M_C \models t \prec t'$ implies $M_C \models \neg(x \prec t), x \prec t'$. But each of these two formulae has rank lower than γ_i, which contradicts its definition.

(b) If $t = x \in X$ then $t' = f(\ldots, t'', \ldots)$ and B follows $DT(\Gamma)$ either along the left or along the right conclusion of rule (VIII+), with $\gamma_i = x \prec f(\ldots, t'', \ldots)$ repeated at the end of the sequence. The argument is analogous to those in points 3a and 4a. γ_{i+1} has the form $y \prec t''$ (if B follows the left branch), or $x \prec f(\ldots, y, \ldots)$ (if B follows the right branch), and in either case the rule (VIII+) is applied until γ_{i+1} becomes indecomposable formula. If M_C satisfies it, we get a contradiction with the definition of γ_i. Otherwise, the trailing repetition of γ_i has to be processed again and again along B, and the above argument leads to an indecomposable formula which has to be satisfied, thus yielding a contradiction with the definition of γ_i.

Thus we have shown that a formula γ_i which appears on an infinite branch B of $DT(\Gamma)$, which is satisfied, $M_C \models \gamma_i$, and which is of lowest rank among the formulae satisfying these two conditions, has to have rank 0, i.e., must be indecomposable, which means that $\gamma_i \in \Gamma_{ind}$. But then $M_C \not\models \gamma_i$, by the construction of M_C in lemma 2.1.10, which contradicts the assumption that $M_C \models \gamma_i$. □

[7] If $M_C \models \gamma''$ for some $\gamma'' \in \Gamma''$ with $\mathrm{ord}(\gamma'') = 2 = \mathrm{ord}(\gamma_i)$, then γ'' has either the form $s \prec s'$ or $\neg(s \prec s')$. The former case is treated in point 5 below, while the latter is the present case.

Corollary 2.1.12 *A sequence Γ has a proof in the R-S system iff $DT(\Gamma)$ is a proof.*

Proof. The 'if' part is trivial and the 'only if' part follows from the proof of the above theorem. If $DT(\Gamma)$ is not a proof, then we have a counter-model for Γ. Since R-S is sound, we conclude that Γ is not provable. $\square$

2.2 Specifications and system R-S*

The system R-S can be used to derive only tautologies – valid sequences. But we are really interested in proving logical consequences of specifications. (For the purposes of this chapter, we can safely ignore the signature assuming that all expressions are well-formed, and identify a specification with the set of its axioms.) We are interested in proving sequents which follow logically from specifications. In this section we extend the R-S logic to fulfill this function. (In the following section we will return to the sequent form and transform the R-S* logic into a sound and complete Gentzen system.)

2.2.1 Specifications

Specifications are sets of sequents from which one may derive other sequents.

Definition 2.2.1 *A Σ sequent is a pair (Γ, Δ) of finite sets of formulae from $\mathcal{F}_{\Sigma,X}$, written $\Gamma \to \Delta$.*

The notation $\Gamma \to \Delta$ is implicitly assumed to mean the same as: $\gamma_1, \ldots, \gamma_n \to \delta_1, \ldots, \delta_m$. As a matter of fact, following earlier works, e.g. [22, 55], our specifications involve only sequents of atomic formulae (i.e., each γ_i, δ_j is terms combined by either equality or inclusion), but we may occasionally need this more general definition, allowing γ_i, δ_j to be formulae as given in definition 1.1.12. Keep also in mind that all formulae in a sequent are quantifier free.

Definition 2.2.2 *A Σ sequent $\Gamma \to \Delta$ is valid iff for every Σ-structure $\langle A, \alpha \rangle$ such that $\langle A, \alpha \rangle \models \gamma$, for all $\gamma \in \Gamma$, there exists a $\delta \in \Delta$ such that $\langle A, \alpha \rangle \models \delta$.*

Lemma 2.2.3 *A sequent $\Gamma \to \Delta$ is valid iff the sequence $\neg\Gamma, \Delta$ is valid.*

The latter notation stands for the sequence $\neg\gamma_1, \ldots, \neg\gamma_n, \delta_1, \ldots, \delta_m$, where $\gamma_1, \ldots, \gamma_n \to \delta_1, \ldots, \delta_m$ is the respective sequent.

Definition 2.2.4 *The function* tr *translates sequents to (quantifier free) formulae in $\mathcal{F}_{\Sigma,X}$:*

- $\mathsf{tr}(\gamma_1, \ldots, \gamma_n \to \delta_1, \ldots, \delta_m) = \neg\gamma_1 \vee \ldots \vee \neg\gamma_n \vee \delta_1 \vee \ldots \vee \delta_m$.
- *for* $\Psi = \{\psi_1, ..., \psi_n\} : \mathsf{tr}(\Psi) = \{\mathsf{tr}(\psi_1), ..., \mathsf{tr}(\psi_n)\}$

With the above notation, lemma 2.2.3 can be stated as: for any structure $\langle A, \alpha\rangle$ and sequent $\psi : \langle A, \alpha\rangle \models \psi \iff \langle A, \alpha\rangle \models \mathsf{tr}(\psi)$

The models for a specification are not structures with an assignment, but algebras satisfying the axioms for all possible assignments:

Definition 2.2.5 *Given a specification $\mathcal{SP} = (\Sigma, \Psi)$, a Σ-algebra A, and a sequent (formula, sequence) ψ, we define the satisfaction relation for sequents and algebras by:*

1. $A \models \psi \iff A \models \mathsf{tr}(\psi)$ *i.e* $\forall \alpha . \langle A, \alpha\rangle \models \mathsf{tr}(\psi)$
2. $A \models \Psi \iff \forall \psi_i \in \Psi . A \models \mathsf{tr}(\psi_i)$
3. $\Psi \models \psi \iff \forall A . (A \models \Psi \Rightarrow A \models \mathsf{tr}(\psi))$.

Notice that in the case of tautologies the two notions of satisfiability coincide.The above definition may be applied also when ψ's are arbitrary formulae, in which case we simply drop the applications of $\mathsf{tr}(\psi)$. This convention will be applied below – ψ stands, in general, for arbitrary formula, while the notation $\mathsf{tr}(\psi)$ indicates that ψ is a sequent.

2.2.2 System R-S*

We introduce the syntax for indicating nonlogical axioms, define their semantics, and extend the system R-S with a new rule to handle such sequents.

Definition 2.2.6 *An axiom ψ is written* $![\psi]$

The procedure for extending the R-S system is quite standard – in order to prove a sequent ψ from a specification (set of sequents) $\Psi = \{\psi_1, \ldots, \psi_n\}$, we perform a translation, tr, of ψ and all the sequents from Ψ into formulae, form a sequence corresponding to $(\bigwedge_{\psi_i \in \Psi} ![\mathsf{tr}(\psi_i)]) \to \mathsf{tr}(\psi)$, and try to prove it in the system R-S augmented with the appropriate rule for treating axioms on the left of '$\to$'. The standard notion of satisfaction of such a formula is equivalent to the satisfaction of a sequence

$$\neg ![\mathsf{tr}(\psi_1)], \ldots, \neg ![\mathsf{tr}(\psi_n)], \mathsf{tr}(\psi) \tag{2.1}$$

The details concerning the corresponding Gentzen system will be given in Section 2.3. For the time being we merely observe that in order to reason about specifications we have to extend the R-S proof system by a new rule to handle axiomatic formulae, i.e., the formulae with the form $\neg ![\phi]$. (Notice that in (2.1) we do not nest axiomatic formulae, and they always occur under the negation $\neg$. Since specifications will only involve sequents over atomic formulae, we do not need the full power of universal and/or existential quantifiers.) Therefore we introduce $![\_]$, resp. $\neg ![\_]$ as new logical connectives which, however, are used only at the outermost level of formulae.

Definition 2.2.7 *For a structure $\langle A, \alpha \rangle$ and a formula ψ, we define:*

- $\langle A, \alpha \rangle \models\ ![\psi] \iff A \models \psi$ *(i.e., iff $\forall \alpha'.\langle A, \alpha' \rangle \models \psi$). Consequently:*
- $\langle A, \alpha \rangle \models \neg ![\psi] \iff \langle A, \alpha \rangle \not\models\ ![\psi] \iff A \not\models \psi$
(i.e., iff $\exists \alpha'.\langle A, \alpha' \rangle \models \neg\psi$).

As usual, α' matters only in so far as it differs from α on the variables occurring in ψ.

Notice that we quantify over *assignments* α' – according to definition 1.1.13 such an assignment may exist even if the carrier A is empty, in which case all variables are assigned $\emptyset$. $![\_]$ does play the role of the universal closure but over assignments and not only elements of the carrier. Similarly $\neg ![\_]$ corresponds to existential closure over assignments.

For a formula ϕ, we write $\phi[\overline{y}/\overline{x}]$ for ϕ with all occurrences of the variables from the sequence $\overline{x}$ replaced by the respective variables from the sequence $\overline{y}$.

The R-S* system is obtained by augmenting the R-S system with the following rule:

$$\text{(AX)}\quad \frac{\Gamma', \neg ![\gamma], \Gamma''}{\Gamma', \neg\gamma[\overline{y}/\overline{x}], \Gamma'', *} \quad \text{where } \overline{x} \text{ are all variables in } \gamma \text{, and } \overline{y} \text{ are arbitrary}$$

Lemma 2.2.8 *The R-S* system is sound:*

Proof. The R-S system is sound so it remains to prove soundness of the new rule. We let $\langle A, \alpha \rangle$ be an arbitrary structure. $M = \langle A, \alpha \rangle$

$\Downarrow$ If $\langle A, \alpha \rangle \models \neg ![\gamma]$, then $\langle A, \alpha \rangle$ obviously satisfies the conclusion of the rule since this formula is repeated there.

$\Uparrow$ If $\langle A, \alpha \rangle \models \neg\gamma[\overline{y}/\overline{x}]$, we let $\alpha'(\overline{x}) = \alpha(\overline{y}) \in |A| \uplus \{\emptyset\}$, so $\langle A, \alpha' \rangle \models \neg\gamma$. By the definitions 2.2.7 and 2.2.5 we have that:

$$\langle A, \alpha' \rangle \models \neg\gamma \Rightarrow \langle A, \alpha' \rangle \not\models \gamma \Rightarrow A \not\models \gamma \Rightarrow \langle A, \alpha \rangle \models \neg ![\gamma]$$

Every other formula from the conclusion appears also in the premise, so all these cases are trivial.

$\square$

Remark 2.2.9 *As one could expect, the definition of satisfaction 2.2.7 makes $\neg ![\gamma]$ equivalent to its standard counterpart $\exists \alpha : \langle A, \alpha \rangle \not\models \gamma$. The non-standard aspect is that such quantification over assignments does not coincide with the quantification over elements of the carrier, since in case of empty carrier we still admit the assignment $\alpha(x) = \emptyset$.*

Consider the following special cases, with $\Gamma' = \emptyset = \Gamma''$, of the application of rule (AX):

1. *If γ is $\neg(x_s \doteq x_s)$, we get:*

$$(AX) \quad \frac{\neg![\neg(x_s \doteq x_s)]}{\neg\neg(y_s \doteq y_s), *} \; x_s \in X$$

Applying (IV–) to the conclusion, we obtain $y_s \doteq y_s$, i.e., $\neg\mathcal{E}_s$. Thus the formulae $\neg![\neg(x_s \doteq x_s)]$ and $\neg\neg(x_s \doteq x_s) \equiv \neg\mathcal{E}_s$, are really equivalent, i.e. $\exists\alpha : x \doteq x$ is equivalent to $\exists x : x \doteq x$. (If the carrier is empty, there is not only no element but also no assignment making $x \doteq x$, since $\emptyset$ does not satisfy this equality.)

2. *If γ is $x_s \doteq x_s$, we get:*

$$(AX) \quad \frac{\neg![x_s \doteq x_s]}{\mathcal{E}_s, *} \; x_s \in X$$

where $\mathcal{E}_s$ in the conclusion corresponds to $\neg(y_s \doteq y_s)$, for some variable y_s substituted for x_s.

Thus $\neg![x_s \doteq x_s]$ and $\neg(x_s \doteq x_s) \equiv \mathcal{E}_s$, are equivalent, and correspond to $\exists\alpha : \neg(x \doteq x)$ which is satisfied only by the structures with empty carrier. Note, however, that this is not equivalent to $\exists x : \neg(x \doteq x)$ – this last formula is actually a contradiction.

In earlier logics see e.g. [50, 54], one did not admit empty carrier and then $x \doteq x$ was axiomatic. The generalization with this respect amounts to having made this formula valid if and only if carrier is non-empty. The significant difference with respect to [6](where empty carriers were allowed), is that our treatment of (non-)empty carrier is essentially quantifier-free – it amounts merely to the treatment of the formulae $x \doteq x$ (resp. $\neg(x \doteq x)$) which is carried over to the respective axioms as shown in the remark above. In [6], this required formulae of the form $\exists x.x \prec x$ (resp. $\neg\exists x.x \prec x$).

Lemma 2.2.10 *The R-S* proof system is complete: for any sequence Γ (of formulae from $\mathcal{F}_{\Sigma,X}$ or, possibly, of the form (2.1)), if $\models \Gamma$ then $\vdash \Gamma$.*

Proof. The only difference from the R-S proof system is the presence of the new kind of formulae and the new rule (AX). Note that the R-S* system has the same axiomatic sequences and the same indecomposable formulae as the R-S system. The proof is therefore the same as before, based on the counter-model M_C from lemma 2.1.10. We only have to consider a new possible case of a formula γ_i which, occuring on the selected infinite branch B (from which we constructed the counter-model M_C), has the lowest rank such that $M_C \models \gamma_i$. (Lemmata 2.1.7 and 2.1.8 remain trivially true for the R-S* system. To obtain unique decomposition tree, we would have to extend the well-ordering of variables from footnote 2 to finite sequences of variables since rule (2.2.2) performs uniform substitution for *all* variables in the processed formula.) We define the rank of the formula $\neg![\gamma]$ by:

- $\mathrm{ord}(\neg ![\gamma]) = \mathrm{ord}(\neg\gamma) + 1$

As in the proof of theorem 2.1.11, let $\Pi_i \in B$ be such that $\Pi_i = \Gamma', \gamma_i, \Gamma''$, where Γ' is indecomposable (possibly empty), $\gamma_i = \neg ![\gamma]$ and Π_{i+1} be the vertex following Π_i in B with the label $\Gamma', \gamma_{i+1}, \Gamma'''$.

6. If B follows the conclusion, then, eventually, there must be a vertex $\Pi_j \in B$, $j > i$, including the formula $\neg\gamma[\overline{y}/\overline{x}]$ such that $M_C \models \neg\gamma[\overline{y}/\overline{x}]$, since $M_C \models \gamma_i$ (and since the carrier of M_C is obtained as a quotient of the variable set X, with $\alpha_C(y) = [y]$). Whether this happens already for $j = i + 1$ or later does not matter since in either case we have: $\mathrm{ord}(\neg\gamma[\overline{y}/\overline{x}]) = \mathrm{ord}(\neg\gamma) < \mathrm{ord}(\neg\gamma) + 1 = \mathrm{ord}(\gamma_i)$.

□

We introduce the following notational abbreviations:

Definition 2.2.11 *Given a set of formulae $\Psi = \{\psi_1, \ldots, \psi_n\} \subseteq \mathcal{F}_{\Sigma,X}$ and a formula $\psi \in \mathcal{F}_{\Sigma,X}$, we write:*

- $\Psi \vdash \psi \iff \vdash \neg ![\psi_1], \ldots, \neg ![\psi_n], \psi$
- $\Psi \models \psi \iff \models \neg ![\psi_1], \ldots, \neg ![\psi_n], \psi$

The above lemmata 2.2.8, 2.2.10 give us:

Theorem 2.2.12 *For any formula ϕ and set of formulae $\Phi = \{\phi_1, \ldots, \phi_n\}$:*

$$\Phi \vdash \phi \iff \Phi \models \phi$$

Corollary 2.2.13 *For any specification Ψ and sequent ψ:*

$$\Psi \vdash \psi \iff \Psi \models \psi$$

Proof. By the above theorem 2.2.12, we only have to show, for any specification Ψ and sequent $\psi : \Psi \models \psi \iff \Psi \models \psi$.

We have:

$$\begin{aligned}
\Psi \models \psi \quad & \overset{2.2.3}{\iff} \quad \mathsf{tr}(\Psi) \models \mathsf{tr}(\psi) \\
& \overset{2.2.11}{\iff} \quad \models \neg ![\mathsf{tr}(\psi_1)] \vee ... \vee \neg ![\mathsf{tr}(\psi_1)] \vee \mathsf{tr}(\psi) \\
& \overset{2.2.5}{\iff} \quad \models \neg ![\mathsf{tr}(\psi_1)] \vee ... \vee \neg ![\mathsf{tr}(\psi_1)] \vee \mathsf{tr}(\psi) \\
& \iff \quad \forall A.\ A \models \bigvee_{\psi_i \in \Psi} \neg ![\mathsf{tr}(\psi_i)] \vee \mathsf{tr}(\psi)
\end{aligned}$$

And:

$$\begin{aligned}
\Psi \models \psi \quad & \overset{2.2.5}{\iff} \quad \forall A.\ (A \models \Psi \Rightarrow A \models \mathsf{tr}(\psi)) \\
& \iff \quad \forall A.\ (A \models \bigwedge_{\psi_i \in \Psi} \mathsf{tr}(\psi_i) \Rightarrow A \models \mathsf{tr}(\psi))
\end{aligned}$$

We thus have to show the following equivalence:

$$\forall A.\big(A \models \bigwedge_{\psi_i \in \Psi} \mathsf{tr}(\psi_i) \quad \Longrightarrow \quad A \models \mathsf{tr}(\psi)\big)$$
$$\Longleftrightarrow \quad \forall A.\big(A \models \bigvee_{\psi_i \in \Psi} \neg ![\mathsf{tr}(\psi_i)] \vee \mathsf{tr}(\psi)\big)$$

$\Leftarrow$) Assume the RHS, and let A be arbitrary:

$$\begin{aligned} A \models \bigwedge \mathsf{tr}(\psi_i) \quad &\stackrel{2.2.7}{\Longleftrightarrow} \quad \forall\alpha.(\ \langle A,\alpha\rangle \models \bigwedge ![\mathsf{tr}(\psi_i)]\ ) \\ &\Longleftrightarrow \quad \forall\alpha.(\ \langle A,\alpha\rangle \not\models \bigvee \neg ![\mathsf{tr}(\psi_i)]\ ) \\ &\stackrel{RHS}{\Longrightarrow} \quad \forall\alpha.(\ \langle A,\alpha\rangle \models \mathsf{tr}(\psi)\ ) \\ &\stackrel{2.2.5}{\Longleftrightarrow} \quad A \models \mathsf{tr}(\psi) \end{aligned}$$

$\Rightarrow$) Assume LHS and choose arbitrary A and α.
If $\langle A,\alpha\rangle \models \bigvee \neg ![\mathsf{tr}(\psi_i)]$, then RHS is satisfied. So assume the opposite, i.e.:

$$\begin{aligned} \langle A,\alpha\rangle \not\models \bigvee \neg ![\mathsf{tr}(\psi_i)] \quad &\Longleftrightarrow \quad \langle A,\alpha\rangle \models \bigwedge ![\mathsf{tr}(\psi_i)] \\ &\stackrel{2.2.7}{\Longleftrightarrow} \quad A \models \bigwedge \mathsf{tr}(\psi_i) \\ &\stackrel{LHS}{\Longrightarrow} \quad A \models \mathsf{tr}(\psi) \\ &\stackrel{2.2.5}{\Longrightarrow} \quad \langle A,\alpha\rangle \models \mathsf{tr}(\psi) \end{aligned}$$

□

We have thus obtained the sound and complete system for proving consequences of specifications. As remarked, the system R-S*, with the unique proof strategy described in Section 2.1.1, is well suited for implementation. It is, however, less convenient for doing proofs by hand. In the following section we make the last step and design a Gentzen system which provides simpler means for performing proofs by hand – it works directly with sequents and does not require any translation of sequents into formulae.

2.3 Gentzen calculus

We will first describe a trivial translation of the R-S* system into a Gentzen system, GS', and then simplify it to the system GS", which we show to be equivalent to GS'. The final Gentzen system GS, given in Section 2.3.1, will be obtained by some further simplifications of GS".

Definition 2.3.1 *We say that a formula γ, in $\mathcal{F}_{\Sigma,X}$, or of the form, $\neg ![\phi], \phi \in \mathcal{F}_{\Sigma,X}$, is negative if it has the form $\neg\gamma'$, and non-negative else. For any sequence Γ we define:*

- $\Gamma^+ = \{\gamma \in \mathcal{F}_\Sigma : \gamma$ *is non-negative and* $\gamma \in \Gamma\}$
- $\Gamma^- = \{\gamma \in \mathcal{F}_\Sigma : \neg\gamma \in \Gamma\}$

We can now rephrase lemma 2.2.3 in the following way:

Corollary 2.3.2 *A sequence of formulae* Γ *is valid iff the sequent* $\Gamma^- \to \Gamma^+$ *is valid.*

The Genzen system we will design has three fundamental differences from the R-S system:

- The rules in Gentzen system are applied "bottom up" (hence the inversion of the R-S rules).
- The rules in Gentzen system are not invertible – they are sound "top down", i.e., if the premise (above the stroke) is valid then so is the conclusion.
- Sequents are pairs of *sets* of formulae, where sequence ordering is ignored.

We use the corollary 2.3.2 on the different types of R-S rules and get the following lemma. Π stands for the active (sub)sequence of an R-S rule and Λ for the resulting (sub)sequence. Γ' and Γ'' in the R-S rules are arbitrary, so they are replaced in Gentzen rules by arbitrary Γ's and Δ's.

Lemma 2.3.3 *For any sound R-S rule (in the left column), the corresponding sequent (in the right column) is sound:*

R-s rule	*Gentzen rule*
$\dfrac{\Gamma', \Pi, \Gamma''}{\Gamma', \Lambda, \Gamma''}$	$\dfrac{\Gamma, \Lambda^- \to \Delta, \Lambda^+}{\Gamma, \Pi^- \to \Delta, \Pi^+}$
$\dfrac{\Gamma', \Pi, \Gamma''}{\Gamma', \Lambda_1, \Gamma'' \mid \Gamma', \Lambda_2, \Gamma''}$	$\dfrac{\Gamma, \Lambda_1^- \to \Delta, \Lambda_1^+ \mid \Gamma, \Lambda_2^- \to \Delta, \Lambda_2^+}{\Gamma, \Pi^- \to \Delta, \Pi^+}$
$\dfrac{\Gamma', \Pi, \Gamma''}{\Gamma', \Lambda_1, \Gamma'' \mid \Gamma', \Lambda_2, \Gamma'' \mid \Gamma', \Lambda_3, \Gamma''}$	$\dfrac{\Gamma, \Lambda_1^- \to \Delta, \Lambda_1^+ \mid \Gamma, \Lambda_2^- \to \Delta, \Lambda_2^+ \mid \Gamma, \Lambda_3^- \to \Delta, \Lambda_3^+}{\Gamma, \Pi^- \to \Delta, \Pi^+}$

Applying the translation schema from lemma 2.3.3 to all the rules and axioms of the R-S system yields a Gentzen system GS', to which we add one rule:

$$(IV+)\ \frac{\Gamma, \gamma \to \Delta}{\Gamma \to \Delta, \neg\gamma}$$

Theorem 2.3.4 *The system GS' is sound and complete, for any sequent* ψ*:*

$$\vdash_{GS'} \psi \iff \models \psi$$

Proof. The system is sound by lemma 2.3.3 – soundness (and invertibility) of the rule (IV+) is obvious and so, by completeness of the R-S system, this rule is admissible there.

If a sequent $\Gamma \to \Delta$ is valid, then the corresponding R-S sequence $\neg\Gamma, \Delta$ is also valid. Since the R-S system is complete it means that $\neg\Gamma, \Delta$ has a proof in the R-S system, i.e. a finite decomposition tree T with leaves labelled by axiomatic sequences. We can then construct a Gentzen proof of the sequent $\Gamma \rightarrow \Delta$ in GS', by mimicking the structure of this decomposition tree T.

The construction starts at the leaves of the deduction tree T, they are labelled by axiomatic sequences, and the corresponding sequents are axiomatic too. We proceed upwards in T by replacing each downward application of an R-S rule by an upwards application of the corresponding GS' rule.

The induction is finished at the root of T, which is labelled by the sequence $\neg\Gamma, \Delta$. Thus the sequent: $(\neg\Gamma, \Delta)^- \rightarrow (\neg\Gamma, \Delta)^+$ can be derived in GS', but this sequent need not be the original sequent $\Gamma \rightarrow \Delta$. The possible difference concerns the negative formulae: a formula $\neg\phi \in \neg\Gamma, \Delta$, may figure in the original sequent as ϕ on the left of '$\rightarrow$' or as $\neg\phi$ on the right. To obtain the original sequent from the above one, the required transformations can be performed using the added swapping rule (IV+) (and, possibly, (IV–)). $\square$

The system GS" given below is a slightly simplified version of GS'. Each rule has the same number as the respective rule in the R-S system from which it was obtained.

Axioms

(I) $\Gamma \to x \prec x, \Delta \;\; : x \in X$ (II) $\Gamma, \gamma \to \gamma, \Delta$ (III) $\Gamma, \mathcal{E}_s \to t_s \prec t'_s, \Delta$

Replacement rules

	+	−
(IV)	$\dfrac{\Gamma, \gamma \to \Delta}{\Gamma \to \Delta, \neg\gamma}$	$\dfrac{\Gamma \to \Delta, \gamma}{\Gamma, \neg\gamma \to \Delta}$
(V)	$\dfrac{\Gamma \to \Delta, \gamma, \phi}{\Gamma \to \Delta, \gamma \vee \phi}$	$\dfrac{\Gamma, \gamma \to \Delta \mid \Gamma, \phi \to \Delta}{\Gamma, \gamma \vee \phi \to \Delta}$
(VI)	$\dfrac{\Gamma \to \Delta, \gamma \mid \Gamma \to \Delta, \phi}{\Gamma \to \Delta, \gamma \wedge \phi}$	$\dfrac{\Gamma, \gamma, \phi \to \Delta}{\Gamma, \gamma \wedge \phi \to \Delta}$
(VII)	$\dfrac{\Gamma, x \prec t \to \Delta, x \prec t'}{\Gamma \to \Delta, t \prec t'}$ $t \notin X$, and $x \in X$ is fresh	$\dfrac{\Gamma \to \Delta, x \prec t \mid \Gamma, x \prec t' \to \Delta}{\Gamma, t \prec t' \to \Delta}$ $t \notin X$ and $x \in X$ arbitrary
(VIII)	$\dfrac{\Gamma \to \Delta, y \prec t \mid \Gamma \to \Delta, x \prec f(\ldots, y, \ldots)}{\Gamma \to \Delta, x \prec f(\ldots, t, \ldots)}$ where $y \in X$ arbitrary and $t \notin X$	$\dfrac{\Gamma, y \prec t, x \prec f(\ldots, y, \ldots) \to \Delta}{\Gamma, x \prec f(\ldots, t, \ldots) \to \Delta}$ where $y \in X$ is fresh and $t \notin X$
(IX)	$\dfrac{\Gamma \to \Delta, t \doteq x \mid \Gamma \to \Delta, t' \doteq x}{\Gamma \to \Delta, t \doteq t'}$ $t, t' \notin X$ and $x \in X$ arbitrary	$\dfrac{\Gamma, t_s \doteq x_s, t'_s \doteq x_s \to \Delta}{\Gamma, t_s \doteq t'_s \to \Delta}$ $t_s, t'_s \notin X$ and $x_s \in X$ is fresh
(X)	$\dfrac{\Gamma \to \Delta, t_s \prec x_s \mid \Gamma \to \Delta, x_s \prec t_s \mid \Gamma \to \Delta, \neg\mathcal{E}_s}{\Gamma \to \Delta, t_s \doteq x_s}$ where $x_s \in X$ and $t_s \neq x_s$	$\dfrac{\Gamma, t_s \prec x_s, x_s \prec t_s, \neg\mathcal{E}_s \to \Delta}{\Gamma, t_s \doteq x_s \to \Delta}$ where $x_s \in X$ and $t_s \neq x_s$
(XI)	$\dfrac{\Gamma \to \Delta, t_s \prec x_s \mid \Gamma \to \Delta, x_s \prec t_s \mid \Gamma \to \Delta, \neg\mathcal{E}_s}{\Gamma \to \Delta, x_s \doteq t_s}$ where $x_s \in X$ and $t_s \neq x_s$	$\dfrac{\Gamma, t_s \prec x_s, x_s \prec t_s, \neg\mathcal{E}_s \to \Delta}{\Gamma, x_s \doteq t_s \to \Delta}$ where $x_s \in X$ and $t_s \neq x_s$
(AX)		$\dfrac{\Gamma, \gamma[\overline{y}_s/\overline{x}_s] \to \Delta}{\Gamma, ![\gamma] \to \Delta}$ where $\overline{y}_s$ arbitrary

Expansion rules

$$\text{(XIV)} \quad \frac{\Gamma, y \prec f(\overline{z}) \to \Delta}{\Gamma, y \prec x, x \prec f(\overline{z}) \to \Delta} \qquad \text{(XII)} \quad \frac{\Gamma, x \prec y \to \Delta}{\Gamma, y \prec x \to \Delta}$$

(sound for arbitrary $x \in X$)

$$\text{(XV)} \quad \frac{\Gamma, x \prec f(\ldots, y, \ldots) \to \Delta}{\Gamma, y \prec z, x \prec f(\ldots, z, \ldots) \to \Delta} \qquad \text{(XIII)} \quad \frac{\Gamma, y \prec z \to \Delta}{\Gamma, y \prec x, x \prec z \to \Delta}$$

(sound for arbitrary $z \in X$)

$$\text{(XVI)} \quad \frac{\Gamma, x_{s'} \prec f(\ldots, y_s, \ldots) \to \Delta, \neg\mathcal{E}_s, \neg\mathcal{E}_{s'}}{\Gamma, x_{s'} \prec f(\ldots, y_s, \ldots) \to \Delta, \neg\mathcal{E}_s}$$

Theorem 2.3.5 *The system GS" given above is equivalent to the system GS', in particular, GS" is sound and complete, i.e., for any sequent ψ :*

$$\vdash_{GS''} \psi \iff \vdash_{GS'} \psi$$

Proof. The replacement rules in GS" are essentially the same as in GS', i.e., are obtained directly by the translation of the respective R-S* rules. The only difference is that the GS" rules (VII–), (VIII+), (IX+), and the axiom rule (AX), i.e., the rules involving a choice of an arbitrary variable, do not repeat the active formulae (from the conclusion in the premise(s)). This does not change the power of the system since the repetition of the active formulae (in R-S* and GS') can be now simulated by an immediate choice ("guessing") of the appropriate variable.

The new replacement rule, (IV–), was commented in the proof of completeness theorem 2.3.4, and is present in both systems GS' and GS".

The remaining expansion rules are simplified slightly in GS" by dropping some of the formulae from the premisses. (Thus, they are not invertible, though obviously sound.) For instance, following the translation schema from lemma 2.3.3, the symmetry rule in GS' looks as:

$$(XII') \ \frac{\Gamma, x \prec y, y \prec x \to \Delta}{\Gamma, y \prec x \to \Delta}$$

while in GS" it became:

$$(XII) \ \frac{\Gamma, x \prec y \to \Delta}{\Gamma, y \prec x \to \Delta}$$

However, each is admissible given the other, given admissibility of weakening rules (W) which follows by standard argument.

(W)+(XII') ⇒ (XII)	(XII) ⇒ (XII')
$$\begin{array}{r} \text{(W)} \\ \text{(XII')} \end{array} \begin{array}{c} \Gamma, x \prec y \to \Delta \\ \hline \Gamma, y \prec x, x \prec y \to \Delta \\ \hline \Gamma, y \prec x \to \Delta \end{array}$$	$$\begin{array}{c} \Gamma, y \prec x, x \prec y \to \Delta \\ \hline \Gamma, y \prec x, y \prec x \to \Delta \\ = \Gamma, y \prec x \to \Delta \end{array} \text{(XII)}$$

Equivalence of the other expansion rules from GS' and GS" is shown by similarly simple and entirely analogous derivations.

Weakening rules (W) are admissible in GS' (and in R-S*) by the standard argument. Consider the rule

$$(W)\ \frac{\Gamma \to \Delta}{\Gamma, \Lambda \to \Delta}$$

, and assume that $\Gamma \to \Delta$ is derivable in GS'. Each leaf of its proof tree T, which is an axiomatic sequent $\Gamma' \to \Delta'$, can be extended with Λ to $\Gamma', \Lambda \to \Delta'$, yielding again an axiomatic sequent. Propagating Λ's across the whole tree T will give a proof for $\Gamma, \Lambda \to \Delta$. (Possibly, the names of fresh variables at some nodes may need to be chosen differently in order not to clash with the names of variables in Λ.) By completeness of GS', the rule is admissible. $\square$

Summarizing our results (the above theorem 2.3.5, soundness and completeness of GS' from theorem 2.3.4, we obtain a counterpart of the corollary 2.2.13 for the system GS":

Corollary 2.3.6 *For any specification $\Psi = \{\psi_1, \ldots, \psi_n\}$ and sequent ψ we have that:*

$$\Psi \models \psi \iff \vdash_{GS''} ![\mathsf{tr}(\psi_1)], \ldots, ![\mathsf{tr}(\psi_n)] \to \mathsf{tr}(\psi)$$

2.3.1 The final Gentzen system GS

Assuming that all our sequents are as indicated in the specifications, i.e., contain only atomic formulae, and observing that function tr (def. 2.2.4) introduces only disjunctions, the above corollary 2.3.6 holds also when we remove from GS" both rules (VI). We now perform a final transformation to obtain a "pure" sequent calculus for specifications, i.e., one operating only on sequents of atomic formulae and allowing to derive such sequents from specifications without any translation nor axiom rules.

For the sake of example, let our specification contain only one sequent, $\Psi = \{\gamma \to \delta\}$. To derive from it $\Gamma \to \Delta$, we would try to prove $![\neg\gamma \vee \delta], \Gamma \to \Delta$ which, applying the rule (AX), amounts to:

$$\frac{(\neg\gamma \vee \delta)[\overline{y}/\overline{x}], \Gamma \to \Delta}{![\neg\gamma \vee \delta], \Gamma \to \Delta}\ \overline{y} \text{ arbitrary} \tag{2.2}$$

where $\overline{y}$'s match the respective variables $\overline{x}$ from $\gamma \to \delta$. Applying the rules for disjunction (V) and for negation (IV–) in the antecedent of a sequent, we will end up with the assumptions as indicated in the following:

$$\frac{\delta[\overline{y}/\overline{x}], \Gamma \to \Delta \mid \Gamma \to \Delta, \gamma[\overline{y}/\overline{x}]}{![\neg\gamma \vee \delta], \Gamma \to \Delta}\ \overline{y} \text{ arbitrary} \tag{2.3}$$

All the assumptions are now sequents and this illustrates the idea of the final step. We are interested in proving statements of the form $\Psi \vdash \psi$, where all involved sequents are of the simple form $\gamma_1, \ldots, \gamma_n \to \delta_1, \ldots, \delta_m$, with all γ_i, δ_j being atomic inclusions or equalities. We can thus remove the rules for treating connectives, (IV), (V) and (VI), as well as the axiom rule (AX). We precede all assumptions and conclusions of the rules by $\Psi \vdash \ldots$ and add the rules of *specific cut*, [41], for each non-logical axiom $\gamma_1, \ldots, \gamma_n \to \delta_1, \ldots, \delta_m \in \Psi$:

$$\text{(SPC)}\ \frac{\Psi \vdash \Gamma \to \Delta, \gamma_1' \mid \ \ldots \ \mid \Psi \vdash \Gamma \to \Delta, \gamma_n' \mid \Psi \vdash \Gamma, \delta_1' \to \Delta \mid \ \ldots \ \mid \Psi \vdash \Gamma, \delta_m' \to \Delta}{\Psi \vdash \Gamma \to \Delta}$$

where the primed versions denote uniform, arbitrary renaming of variables occurring in the involved axiom $\gamma_1, \ldots, \gamma_n \to \delta_1, \ldots, \delta_m \in \Psi$.

The possible simplification for the proofs by hand comes from the fact that we now do not have to write and carry around (the translations of) all the axioms of the specification, but can apply the rule (SPC) only for the needed axioms. In addition, of course, we no longer need to consider any other kinds of formulae or sequents and their translations, but only those consisting only of atomic formulae, as prescribed by the format of specifications.

Observe that, as argued in [41], the specific cut rules are significantly more manageable than the general cut. In fact, the "undecidability" of such rules (applied bottom-up) is essentially of the same kind as that of the axiom rules (AX) and concerns only the choice of the appropriate variable names.

The rules of the resulting system GS are given below. (Since we now consider only atomic formulae in the sequents, we have moved $\neg\mathcal{E}$ along the "$\to$" and replaced it with $\mathcal{E}$.) We can not claim the equivalence of GS" and GS, since the latter does not allow any formulae with axioms. However, taking into account the restrictions on such formulae we have put in GS" (only $\neg ![\ldots]$ occurring only at the outermost level, with the exception of one formula, corresponding to the sequent we are proving), the above remarks make it obvious that

$$\vdash_{GS''} ![\mathsf{tr}(\psi_1)], \ldots, ![\mathsf{tr}(\psi_n)] \to \mathsf{tr}(\psi) \iff \{\psi_1, \ldots, \psi_n\} \vdash_{GS} \psi,$$

for any sequents $\psi_1, \ldots, \psi_n, \psi$ over atomic formulae. Indeed, if there is a proof in GS" involving an application of (AX), as in (2.2), then, moving "bottom-up", it must split the tree into branches for separate disjuncts (of each $\mathsf{tr}(\psi_i)[\overline{y}/\overline{x}]$) before processing the involved disjuncts themselves. Hence it must pass through nodes as given in the assumptions of (2.3). Except for the superficial differences of syntax, the rule (SPC) mimics exactly transition to such nodes. On the other hand, the rule is obviously sound (with the interpretation of $\{\psi_1 \ldots \psi_n\} \vdash \psi$ as $\models ![\mathsf{tr}(\psi_1)], \ldots, ![\mathsf{tr}(\psi_n)] \to \mathsf{tr}(\psi)$), and hence it is admissible in GS".

We thus obtain the calculus GS for deriving consequences of specifications, which does not require any transformation of the involved sequents, and the following theorem follows.

Theorem 2.3.7 *The system GS given below is sound and complete, i.e., for any specification Ψ and sequent ψ (involving only atomic formulae):*

$$\Psi \models \psi \iff \Psi \vdash_{GS} \psi$$

Axioms

(I) $\Psi \vdash\ \Gamma \rightarrow x \prec x, \Delta\ : x \in X$ (II) $\Psi \vdash \Gamma, \gamma \rightarrow \gamma, \Delta$ (III) $\Psi \vdash\ \Gamma, \mathcal{E}_s \rightarrow t_s \prec t'_s, \Delta$

Replacement rules

	+	−
(VII)	$\dfrac{\Psi \vdash \Gamma, x \prec t \rightarrow \Delta, x \prec t'}{\Psi \vdash \Gamma \rightarrow \Delta, t \prec t'}$ $t \notin X$, and $x \in X$ is fresh	$\dfrac{\Psi \vdash \Gamma \rightarrow \Delta, x \prec t \mid \Psi \vdash \Gamma, x \prec t' \rightarrow \Delta}{\Psi \vdash \Gamma, t \prec t' \rightarrow \Delta}$ $t \notin X$ and $x \in X$ arbitrary
(VIII)	$\dfrac{\Psi \vdash \Gamma \rightarrow \Delta, y \prec t \mid \Psi \vdash \Gamma \rightarrow \Delta, x \prec f(\ldots, y, \ldots)}{\Psi \vdash \Gamma \rightarrow \Delta, x \prec f(\ldots, t, \ldots)}$ where $y \in X$ arbitrary and $t \notin X$	$\dfrac{\Psi \vdash \Gamma, y \prec t, x \prec f(\ldots, y, \ldots) \rightarrow \Delta}{\Psi \vdash \Gamma, x \prec f(\ldots, t, \ldots) \rightarrow \Delta}$ where $y \in X$ is fresh and $t \notin X$
(IX)	$\dfrac{\Psi \vdash \Gamma \rightarrow \Delta, t \doteq x \mid \Psi \vdash \Gamma \rightarrow \Delta, t' \doteq x}{\Psi \vdash \Gamma \rightarrow \Delta, t \doteq t'}$ $t, t' \notin X$ and $x \in X$ arbitrary	$\dfrac{\Psi \vdash \Gamma, t_s \doteq x_s, t'_s \doteq x_s \rightarrow \Delta}{\Psi \vdash \Gamma, t_s \doteq t'_s \rightarrow \Delta}$ $t_s, t'_s \notin X$ and $x_s \in X$ is fresh
(X)	$\dfrac{\Psi \vdash \Gamma \rightarrow \Delta, t_s \prec x_s \mid \Psi \vdash \Gamma \rightarrow \Delta, x_s \prec t_s \mid \Psi \vdash \Gamma, \mathcal{E}_s \rightarrow \Delta}{\Psi \vdash \Gamma \rightarrow \Delta, t_s \doteq x_s}$ where $x_s \in X$ and $t_s \neq x_s$	$\dfrac{\Psi \vdash \Gamma, t_s \prec x_s, x_s \prec t_s \rightarrow \Delta, \mathcal{E}_s}{\Psi \vdash \Gamma, t_s \doteq x_s \rightarrow \Delta}$ where $x_s \in X$ and $t_s \neq x_s$
(XI)	$\dfrac{\Psi \vdash \Gamma \rightarrow \Delta, t_s \prec x_s \mid \Psi \vdash \Gamma \rightarrow \Delta, x_s \prec t_s \mid \Psi \vdash \Gamma, \mathcal{E}_s \rightarrow \Delta}{\Psi \vdash \Gamma \rightarrow \Delta, x_s \doteq t_s}$ where $x_s \in X$ and $t_s \neq x_s$	$\dfrac{\Psi \vdash \Gamma, t_s \prec x_s, x_s \prec t_s \rightarrow \Delta, \mathcal{E}_s}{\Psi \vdash \Gamma, x_s \doteq t_s \rightarrow \Delta}$ where $x_s \in X$ and $t_s \neq x_s$

Specific cut rules

(SPC) $$\frac{\Psi \vdash \Gamma \rightarrow \Delta, \gamma'_1 \ldots \Psi \vdash \Gamma \rightarrow \Delta, \gamma'_n \mid \Psi \vdash \Gamma, \delta'_1 \rightarrow \Delta \ldots \Psi \vdash \Gamma, \delta'_m \rightarrow \Delta}{\Psi \vdash \Gamma \rightarrow \Delta}$$

for each axiom $\gamma_1, \ldots, \gamma_n \rightarrow \delta_1, \ldots, \delta_m \in \Psi$, with arbitrary renaming $'$ of variables

Expansion rules

(XIV) $$\frac{\Psi \vdash \Gamma, y \prec f(\overline{z}) \rightarrow \Delta}{\Psi \vdash \Gamma, y \prec x, x \prec f(\overline{z}) \rightarrow \Delta}$$ (sound for arbitrary $x \in X$)

(XII) $$\frac{\Psi \vdash \Gamma, x \prec y \rightarrow \Delta}{\Psi \vdash \Gamma, y \prec x \rightarrow \Delta}$$

(XV) $$\frac{\Psi \vdash \Gamma, x \prec f(\ldots, y, \ldots) \rightarrow \Delta}{\Psi \vdash \Gamma, y \prec z, x \prec f(\ldots, z, \ldots) \rightarrow \Delta}$$ (sound for arbitrary $z \in X$)

(XIII) $$\frac{\Psi \vdash \Gamma, y \prec z \rightarrow \Delta}{\Psi \vdash \Gamma, y \prec x, x \prec z \rightarrow \Delta}$$

(XVI) $$\frac{\Psi \vdash \Gamma, \mathcal{E}_s, x_{s'} \prec f(\ldots, y_s, \ldots), \mathcal{E}_{s'} \rightarrow \Delta}{\Psi \vdash \Gamma, \mathcal{E}_s, x_{s'} \prec f(\ldots, y_s, \ldots) \rightarrow \Delta}$$

We finish with an example showing the simplification in proofs in GS as compared to R-S*.

Example 2.3.8 *Suppose that the specification Ψ has two axioms*

1. $f(x) \prec c \rightarrow g(x) \doteq d$ *and*
2. $g(x) \doteq d \rightarrow h(x) \prec a$.

We want to prove $\Psi \vdash f(x) \prec c \rightarrow h(x) \prec a$. We first give the proof in the Gentzen calculus:

If a branch terminates, the axiomatic subsequences are underlined. We drop sort subscripts.

$$\frac{\Psi \vdash f(x) \prec c, \underline{g(x) \doteq d} \rightarrow h(x) \prec a, \underline{g(x) \doteq d} \mid \Psi \vdash f(x) \prec c, g(x) \doteq d, \underline{h(x) \prec a} \rightarrow \underline{h(x) \prec a}}{\Psi \vdash f(x) \prec c, g(x) \doteq d \rightarrow h(x) \prec a} \quad \textit{(SPC) ax.2}$$

$$\frac{\Psi \vdash \underline{f(x) \prec c} \rightarrow h(x) \prec a, \underline{f(x) \prec c} \quad \mid \quad \Psi \vdash f(x) \prec c, g(x) \doteq d \rightarrow h(x) \prec a}{\Psi \vdash f(x) \prec c \rightarrow h(x) \prec a} \quad \textit{(SPC) ax.1}$$

And the proof in the R-S calculus – the active formulae are in boldface.*

$$\dfrac{![(\neg(\mathbf{f}(\mathbf{x}) \prec \mathbf{c}) \vee \mathbf{g}(\mathbf{x}) \doteq \mathbf{d})], ![(\neg(g(x) \doteq d) \vee h(x) \prec a)], \neg(f(x) \prec c) \vee h(x) \prec a}{\neg(\neg(f(x) \prec c) \vee g(x) \doteq d), ![(\neg(g(x) \doteq d) \vee h(x) \prec a)], \neg(f(x) \prec c) \vee h(x) \prec a}\ \textit{(AX)}$$

$$\dfrac{\neg(\neg(f(x) \prec c) \vee g(x) \doteq d), ![(\neg(\mathbf{g}(\mathbf{x}) \doteq \mathbf{d}) \vee \mathbf{h}(\mathbf{x}) \prec \mathbf{a})], \neg(f(x) \prec c) \vee h(x) \prec a}{\neg(\neg(f(x) \prec c) \vee g(x) \doteq d), \neg(\neg(g(x) \doteq d) \vee h(x) \prec a), \neg(f(x) \prec c) \vee h(x) \prec a}\ \textit{(AX)}$$

$$\dfrac{\neg(\neg(f(x) \prec c) \vee g(x) \doteq d), \neg(\neg(g(x) \doteq d) \vee h(x) \prec a), \neg(\mathbf{f}(\mathbf{x}) \prec \mathbf{c}) \vee \mathbf{h}(\mathbf{x}) \prec \mathbf{a}}{\neg(\neg(f(x) \prec c) \vee g(x) \doteq d), \neg(\neg(g(x) \doteq d) \vee h(x) \prec a), \neg(f(x) \prec c), h(x) \prec a}\ \textit{(V+)}$$

$$\dfrac{\neg(\neg(\mathbf{f}(\mathbf{x}) \prec \mathbf{c}) \vee \mathbf{g}(\mathbf{x}) \doteq \mathbf{d}), \neg(\neg(g(x) \doteq d) \vee h(x) \prec a), \neg(f(x) \prec c), h(x) \prec a}{\dfrac{\neg\neg(\mathbf{f}(\mathbf{x}) \prec \mathbf{c}), \neg(\neg(g(x) \doteq d) \vee h(x) \prec a), \neg(f(x) \prec c), h(x) \prec a}{\underline{f(x) \prec c}, \neg(\neg(g(x) \doteq d) \vee h(x) \prec a), \underline{\neg(f(x) \prec c)}, h(x) \prec a}\ \textit{(IV–)}\quad \mid\ i}\ \textit{(V–)}$$

$$\dfrac{i = \quad \neg g(x) \doteq d, \neg(\neg(\mathbf{g}(\mathbf{x}) \doteq \mathbf{d}) \vee \mathbf{h}(\mathbf{x}) \prec \mathbf{a}), \neg(f(x) \prec c), h(x) \prec a}{j \quad \mid \quad \neg g(x) \doteq d, \underline{\neg h(x) \prec a}, \neg(f(x) \prec c), \underline{h(x) \prec a}}\ \textit{(V–)}$$

$$\dfrac{j = \quad \neg g(x) \doteq d, \neg\neg(\mathbf{g}(\mathbf{x}) \doteq \mathbf{d}), \neg(f(x) \prec c), h(x) \prec a}{\underline{\neg g(x) \doteq d}, \underline{g(x) \doteq d}, \neg(f(x) \prec c), h(x) \prec a}\ \textit{(IV–)}$$

2.4 Concluding remarks

We have applied the technique of Rasiowa-Sikorski [44] for designing sound, complete and cut-free logics for reasoning about multialgebras. More details on and applications of this technique can be found in [2, 25, 26].

As compared to the most closely related work which also used this technique, [6], the main difference is the presence of the predicate $\doteq$, which was not included in the language of [6]. We have argued why this predicate is relevant and useful, especially, for writing specifications of nondeterministic data types, and we have shown how (non-)empty carriers can be treated using this predicate instead of quantifiers needed in [6]. Furthermore, the logic from [6] allows one to derive only tautologies but not logical consequences of sets of given, non-logical axioms. We have elaborated the possibility (only implicit in [6]) of extending logic for such purpose, by providing the required translation schema. Then, we have shown how this translation schema (as well as rules for connectives and axioms), needed to handle non-logical axioms in the R-S* system (and in [6]), can be removed and replaced by the specific cut rules, inspired by [41]. The resulting system can be used directly, without any intermediary transformations, for deriving consequences from specifications, that is, sequents from sets of sequents, and the obtained simplifications were illustrated by an example.

The unique decomposition tree which provides a proof strategy and has been identified for the introduced logics R-S and R-S*, following [44], is a natural candidate for a possible implementation and we expect that such an implementation will become available in not too far future.

Chapter 3

Partiality handling with Multialgebras

This chapter presents a new way to model partial operations by use of nondeterminism: a partial operation is modelled as a nondeterministic operation returning, possibly, any value of the carrier. We illustrate the flexibility of the institution of multialgebras $\mathcal{MA}$ by examples showing uniform treatment of strictness, non-strictness and various error handling. We present a methodology for specification development from an abstract specification to low level error handling. Finally we apply institution transformations to illustrate the possibility of reusing partial algebra specifications in the proposed framework – a partial algebra specification can be conservatively (preserving the models) imported to $\mathcal{MA}$ while the extension of the model class allows for further development towards explicit error treatment.

The problem of partial operations is of fundamental importance in specifying and deriving programs. At the abstract level, one would like to be able to ignore many details related to the fact that some operations may happen to be undefined for some particular arguments. However, at the level close to actual implementation, it will often be mandatory to address explicitly possible error situations and to describe the program's behavior in such situations.

In the tradition of algebraic specifications, there are two main approaches to describing partially defined operations: the partial algebra approach on the one hand [8, 45, 10], and total algebras with explicit definition domains or even error elements, on the other. This later approach comprises manifold variations, including error-algebras [15], labelled algebras [4], order-sorted algebras [16], techniques using predicates [37] or functions [21] to specify definition domains of operations. A closer discussion of the traditional solutions can be found e.g. in [39].

In partial algebras, an undefined term has no interpretation in the carrier. During the refinement process, one makes gradually more terms defined, by adding values for undefined terms. Terms which remain undefined until the very end of the specification process are then understood as errors whose handling is left for the implementation. Partial algebras offer an abstract and user-friendly model for partiality due to strictness assumption which releases the specifier from the need to explicitly treat error situations On the other hand, however, there are non-strict operations (like if_then_else) which then require special treatment [9, 10]. This, however, turns out to be a drawback when the specification approaches the implementation level and explicit error handling becomes desirable. Strictness of all operations makes explicit error handling very difficult, if at all possible. Possible extensions of partial algebras to handle this level of specification are indicated in [10, 9]. Often, they suggest some form of translation of a partial algebra specification into some total framework [10].

The approaches based on total algebras explore the possibilities of explicit error handling, by allowing for values which can be used as error values.

The error elements solution is to add distinct elements to the carrier which will be returned by partial operations. The problem with this solution is that operations applied to error elements need to be explicitly specified and thus specifications tend to include – even at an early, abstract stage – a lot of axioms for such situations. In order-sorted algebras, the sorts are arranged in a hierarchy of subsorts and domains of operations are restricted to appropriate subsorts. To some extent, static type checking can then take care of excluding partial terms. On the other hand, static type checking may give unintended consequences [10], e.g. intuitively legal terms may become ill-typed. To handle this one introduces the dynamical concepts like retracts [20]. A retract is much like a predicate telling if some term is defined/well-typed or not. Finally, instead of subsorts one may use the predicates to specify the definition domains of operations [37]. A partial term may remain under specified by guarding the axioms to hold only for terms satisfying appropriate predicates.

We thus have, on the one hand, an abstract framework of partial algebras and, on the other hand, the total algebra approaches which allow one to specify errors explicitly at a level closer to implementation but for the price of the abstract character of partial algebras. We are proposing a single framework capable of addressing both these aspects. We view undefinedness as nondeterminism – a term without a well-defined value may result in any value. At the early stages of development, this nondeterminism expresses our ignorance or disinterest in what exactly will happen in a given – error – situation. At the later stages, such nondeterminism may be narrowed leading, eventually, to explicit error values.

The fact that an operation is well-defined is represented by this operation having a unique value. Such a unique value may be an intended "proper" value, or else it may represent an error value after one started to introduce such error values explicitly into specification.

The opposite, i.e., undefined situation may be represented in two ways in multialgebras. A term may be undefined in the sense that it returns no value, i.e. denotes the empty set. This turns out to be the exact analogue of undefinedness in partial algebras. Other operations applied to the empty set return empty set, and thus operations' behavior is strict. The empty set result can also be understood as delegating further error treatment to the level of implementation.

On the other hand, a term may be undefined in the sense undeterminate, we don't know the value yet:- in this situation the term denotes a non-empty set of values. Initially, such a set may be thought of as the possibilities of later error recovery, or else it may model abnormal behavior of the system. In the refinement process, the set may be narrowed to include only the relevant values to be used in error situation. However, one may also *force* a term to return a set. Such "error sets" will then include, on the one hand, some "proper" values to be used by other operations for the recovery purposes. On the other hand, the sets may contain error values functioning as labels and indicating particular error situation. These error values may be then propagated by other operations or else caught and removed – thus we obtain a model of throwing and catching exceptions.

We will present various scenarios for the development process. The one which we favor most, begins with a partial algebra specification which, however, in our context has a richer class of models than the standard partial algebra models – it contains also models where operations are non-strict. Through a simple process of refinement, one can arrive at the level of explicit error handling, either by introducing unique error values, or else error sets which are amenable to further specification of errors and exceptions. On the other hand, one may use the language of multialgebras to introduce nondeterministic constants which are essentially the same as predicates in many total algebra approaches. Such predicates can be used for the specification of domain definitions and specifications can be refined in the way standard total algebra specifications are. At the technical level, we show that institutions of partial algebras and membership alegbras can be transformed into the proposed institution of multialgebras, thus justifying our claims about both flexibility of our approach and the possibility of reusing the existing specifications.

In section 3.1 we show how the basic partiality problems can be addressed in $\mathcal{MA}$. Since multialgebras offer a nonstrict (or rather, not necessarily strict) framework, we also illustrate how to deal with strictness, if it is desirable for some reasons. In section 3.2 we present a methodology for developing specifications in $\mathcal{MA}$, illustrated by examples.

The rest of the chapter presents the technical results. In section 3.3 we compare $\mathcal{MA}$ with partial algebras, showing, firstly, a transformation of institution of partial algebras, $\mathcal{PA}$, into $\mathcal{MA}$. This gives us the possibility to reuse such specifications in a later development toward explicit error handling. To complete the picture we show that membership algebra specifications also can be transformed to multialgebras.

3.1 Partiality handling with multialgebras

The straightforward way to model partiality in a multialgebra A is to let an operation undefined on some arguments, $\omega^A(a_1, \ldots, a_n)$, be represented by a multi-function returning the empty set when applied to these arguments,

$\omega^A(a_1, \ldots, a_n) = \emptyset$. This was done, for instance, in [51] where one showed that modelled this way, partial algebras form a full subcategory of multialgebras.

It is our contention that multialgebras can provide a framework unifying the two extremes of partial algebras and total algebras in a single formalism. As indicated in the introduction, besides the above solution (where partial operation returns empty set), we have the possibility to model a partial operation by proper nondeterminism. Thus an undefined operation $\omega^A(a_1, \ldots, a_n)$ may return various results, potentially, any available element of the carrier. At the very beginning of specification process, this nondeterminism may represent our complete ignorance and disinterest in what particularly will happen when error situation occur. Later it may be used to represent all potential error elements without discriminating against any one. Finally, it may also capture possible "totalization" when, at a later stage, one refines the earlier specification by choosing a particular (error or not) value to be returned in an error situation.

3.1.1 Definedness and Undefinedness

Definedness

As noted above, definedness is the same as determinism in our setting. To specify definedness we use element equalities. The axiom $t \doteq t$ holds only when t is defined i.e. when t is deterministic. Likewise the axiom $t \doteq t'$ specifies both terms t and t' to be equal and deterministic, i.e., defined. Specially, the axiom $f(\overline{x}) \doteq f(\overline{x})$ specifies the operation f to be defined (deterministic) on all arguments. Thus, since operations may be nondeterministic, to make an ordinary total specification in $\mathcal{MA}$ one has to add this type of axioms for every constant and operation.

Example 3.1.1 *Here is a simple example of a specification of the natural numbers, with successor and predecessor, using multialgebras:*

$$\begin{array}{rll}
\textbf{spec Nat} = & & \\
\textbf{S}: & & Nat \\
\Omega: & & zero :\to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
\textbf{axioms}: & 1. & zero \doteq zero \\
& 2. & succ(x) \doteq succ(x) \\
& 3. & pred(succ(x)) \doteq x
\end{array}$$

This specification will have all the expected classical (total, as well as partial algebras) among its models. The first two axioms make zero and succ deterministic operations. The last one ensures that pred is deterministic and returns x when applied to $succ(x)$.

However, $pred(zero)$ remains unspecified. In the traditional setting, this amounts to underspecification. So does it here, only that here it opens the possibility for this term to be nondeterministic. $pred(zero)$ may return anything.

Consequently, although $succ(x)$ will always return a unique value, $succ(pred(0))$ will be, in general, nondeterministic since it denotes all the values returned by $succ(y)$ where y is returned by $pred(0)$.

Note that in this specification the succ is always defined as a consequence of axiom 2.

Undefinedness

Since element equality, $\doteq$, holds only if both sides of the equality sign denote same element, we have two possible situations for this equality to fail – (at least) one side may be a set with more than one element or else it may be the empty set. We interpret these two possibilities as two types of undefinedness in multialgebras: nondeterminism and partiality.

Partiality corresponding to empty result set leads to strictness: if a term t denotes an empty set, any operation applied to it will also yield empty set. In fact, such partiality can be forced by the specification by adding a negated axiom:-'$x \prec t \rightarrow$' requires that no x be included in t. In terms of program development, we may interpret such axioms as requirements to handle the respective situation, caused by t, at the implementation level. It excludes any further possibility of a more refined treatment of this situation at the specification level.

Undefinedness represented by nondeterminism, e.g., by an axiom '$t \doteq t \rightarrow$', on the other hand, allows further refinement and will be the main issue in the following. Delegation of the responsibility for error treatment to implementation may occur at the very last steps of specification.

Using element equality $\doteq$ we can only force a term to contain exactly one value. Thus, the intuition behind our element equality $\doteq$ corresponds exactly to the existential equality $\stackrel{e}{=}$ from partial algebras: forcing a term to have a uniquely defined value. The difference concerns the negation: if existential equation $t \stackrel{e}{=} t$ fails, the term t is undefined and other operations are strict; if the element equality $t \doteq t$ fails, the term t may be undefined in the sense of being empty set (leading to strictness of other operations), or else it may be nondeterministic i.e. a non-empty set. The choice between these two possibilities is left for the specifier.

Unless stated otherwise, we will in the sequel mean by "undefinedness" proper nondeterminism and by "partiality" empty result set.

3.1.2 Predicates, order-sortedness and strictness

In the most abstract way one may say that an operation f is strict in an argument if, whenever this argument is undefined, then so is the result of the operation. This is how it is viewed in partial algebras. However, when undefinedness is described more explicitly, typically by an explicit indication of the definition domain, then this needs a reformulation: whenever the argument falls outside the definition domain, then so does the result of the operation. Here "definition domain" is understood simply as a set of designated elements of the carrier considered as well-defined. Even more specific notion is obtained when each operation obtains a specific definition domain. Then strictness means: when the argument of the operation comes from outside of the operation's definition domain, then the result is undefined.

As noted in the introduction multialgebras offer a non-strict framework. Although we believe this to be its advantage, we will now make some remarks on how also strict operations can be specified in this framework.

Non-strictness and non-injectivity

We now start with an example of a non-strict function and then proceed to a more detailed treatment of explicit definition domains.

Example 3.1.2 *Let us extend the specification from example 3.1.1 as follows:*

$$
\begin{array}{rll}
\textbf{spec Nat1} = & & \\
\textbf{S}: & & Nat \\
\Omega: & & zero :\to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
& & f : Nat \to Nat \\
\textbf{axioms}: & 1. & zero \doteq zero \\
& 2. & succ \doteq succ \\
& 3. & pred(succ(x)) \doteq x \\
& 4. & f(x) \doteq succ(zero)
\end{array}
$$

The last axiom makes f non-strict - and non-injective. No matter what argument or set of arguments it receives, its result will always be $succ(zero)$.

All operations are strict on the partial terms – applied to empty set, they return empty set. Unless some axioms of the form 4. are given for some operation, the operation will be allowed to be strict also with respect to nondeterministic undefinedness: applied to an error (nondeterministic term) it will, in some models, result in error (nondeterminate result).

In other models, however, it may be non-strict in that it returns unique result on undefined (nondeterministic) arguments. We think that this generosity is a virtue rather than a sign of indefinite abstractness. One should not be forced to make this kinds of decisions at an early stage of development – forcing strictness damages the possibility for later error recovery, as error recovery is non strict by nature.

Definedness predicates

Since nondeterministic constants denote a set, we can use them as predicates. For instance, we can add definedness predicates (constants) to each sort, e.g. $d_{Nat} :\to Nat$. This gives a possibility to write axioms like $succ(x) \prec d_{Nat}$ with the intended meaning as $succ(x) \doteq succ(x)$. This meaning, however, remains merely "intended" since the constant d_{Nat} functions merely as a label unless the range of its elements is explicitly specified.

The use of definedness predicates implies closer classification of the elements of the carrier, so it is less abstract than the mere use of element equalities.

Example 3.1.3 *This following specification intends to use definedness predicates to make "the same" specification of the natural numbers, with successor and predecessor, as example 3.1.1.*

$$
\begin{array}{rll}
\textbf{spec Nat2} = & & \\
\mathbf{S}: & & Nat \\
\Omega: & & zero :\to Nat \\
 & & succ : Nat \to Nat \\
 & & pred : Nat \to Nat \\
 & & d_{Nat} :\to Nat \\
\textbf{axioms}: & 1. & zero \doteq zero \\
 & 2. & succ \doteq succ \\
 & 3. & zero \prec d_{Nat} \\
 & 4. & x \prec d_{Nat} \to succ(x) \prec d_{Nat} \\
 & 5. & x \prec d_{Nat} \to pred(succ(x)) \doteq x \\
 & 6. & succ(x) \prec d_{Nat} \to x \prec d_{Nat} \\
 & 7. & pred(succ(x)) \doteq x \to x \prec d_{Nat}
\end{array}
$$

The constant d_{Nat} is intended to comprise all "defined" values of sort Nat. Axiom 3. and 4. specify its minimal range. Axiom 5. needs then a guard, since we do not know what may happen when operations are applied to elements outside d_{Nat}.

Axiom 6. makes succ strict in the more specific sense than mere preservation of nondeterminism: when its argument falls outside d_{Nat}, so does its result. Similar effect is achieved for pred by axiom 7.

Order-sorting and predicates

The previous example illustrated the possibility to identify a subset of the carrier as a set of "defined" elements. A more detailed treatment of undefinedness is achieved by an explicit introduction of predicates identifying definition domains for various operations.

Example 3.1.4 *We extend the specification from example 3.1.1 with a new constants representing subsort pos of Nat.*

$$
\begin{array}{lll}
\textbf{spec Nat3} = & & \\
\quad\quad \mathbf{S}: & & Nat \\
\quad\quad \Omega: & & zero :\to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
& & pos :\to Nat \\
\textbf{axioms}: & 1. & zero \doteq zero \\
& 2. & succ \doteq succ \\
& 3. & succ(zero) \prec pos \\
& 4. & x \prec pos \to succ(x) \prec pos \\
& 5. & zero \prec pos \to \\
& 6. & x \prec pos \to pred(x) \doteq pred(x) \\
& 7. & pred(succ(x)) \doteq x
\end{array}
$$

We have here a more detailed description of definition domain for pred: the first two axioms include there all positive numbers and the third one excludes from it zero. Then, according to axiom 6., if the argument of pred comes from its definition domain pos, then the result is defined in the sense that it is deterministic.

A more detailed notion of strictness than in example 3.1.3 follows now from this more detailed specification of the definition domain pos. The above specification is consistent with the axiom

$$8.\ \ pred(x) \doteq errorNat, x \prec pos$$

i.e., the requirement that the result of $pred(x) \doteq errorNat$ *for some new (error) constant, when applied to an* $x \not\prec pos$ *(i.e., an* $x \notin pos$*).*

Alternative axiom

$$8'.\ \ pred(x) \doteq pred(x) \to x \prec pos$$

would make pred strict with respect to pos: if the argument does not belong to the domain pos, then the result is undefined.

If this example reminds you of order-sorted algebras [14], then it was also our intention. Recasting order-sorted formalism within the present framework should be straightforward to do. The additional power can be used, for example, to specify exactly the relations between subsorts (similarly to axiom 3.) Notice also, that we don't run into the known problem with typing intuitively correct terms: $pred(pred(succ(succ(zero))))$ is a correct term. The role played by retracts in order-sorted approach is here overtaken by the proof obligation: to decide that it is defined, one has to show that the argument $pred(succ(succ(zero)))$ actually yields something belonging to pos.

Definedness predicates from example 3.1.3 and subsorts from the above example 3.1.4 reflect only methodological aspects - formally they are just constants in the specification. A particular consequence of this is that they can be introduced in arbitrary order during development, and their relations can be specified quite tightly. For instance, one will typically introduce first definedness predicates, like d_{Nat} and later subsorts, like *pos*, for a tighter specification of definition domains. Extending axioms 1.-3. from example 3.1.3 with axioms 1.-2. from example 3.1.4 and adding the axiom

$$x \prec d_{Nat} \to x \doteq zero, x \prec pos$$

will enforce the expected relation between the three constants in every model A, namely: $d^A_{Nat} = \{zero^A\} \cup pos^A$ and $zero^A \notin pos^A$.

Strictness

The axioms 4. in example 3.1.3 and 6. in example 3.1.4 exemplify the general way of enforcing strictness of required operations.

Schema 3.1.5 *Enforcing strictness:*

1. *partial function* $f : s_1 \times \cdots \times s_n \to s$
2. *definedness constants for each argument* $1 \leq i \leq n$ *(relative to the function):* $d_{f_{s_i}} :\to s_i$
3. *definedness constant for the target sort (relative to the function):* $d_{f_s} :\to s$
4. *strictness axioms:* $f(x_1, \ldots, x_n) \prec d_{f_s} \to x_i \prec d_{f s_i}$*, for each argument* x_i *in which* f *is to be strict.*

If no definition domains are specified but one merely has the definedness constants for the subsets of the well-defined elements of each sort (like d_{Nat}), these later constants will be used instead. In particular, the definedness constant 3. will, typically, be of this kind.

This solution works in many cases but is not fully general. A function with two (or more arguments), $f : s_1 \times s_2 \to s$ may have a definition domain which cannot be described as a product of some subsets of s_1 and s_2. (E.g. $f(1, 2)$ and $f(2, 3)$ may be defined while $f(1, 3)$ is undefined.) In such cases, one needs a more general technique, similar to that used in guarded algebras [21]. It introduces, for each partial function f, an explicit definition domain operation dom_f as follows:

Schema 3.1.6 *General way of enforcing strictness:*

1. *partial function* $f : s_1 \times \cdots \times s_n \to s$
2. *definition domain function:* $dom_f : s_1 \times \cdots \times s_n \to s$
3. *definedness constant for the target sort:* $d_s :\to s$
4. *definedness axiom:* $dom_f(x_1, \ldots, x_n) \prec d_s \to f(x_1, \ldots, x_n) \prec d_s$.

It seems like we have added a lot of constants and functions to treat strictness. Indeed, we have. To defend this wastage of ink we can say two things. Firstly, the above schemata can be added purely mechanically having a mere indication of a function being strict in some arguments. More importantly, one of the schemata is used only when a function is to be strict. We believe that, in practice, most functions are not or, in any case, should not be. Eventually, one wants to describe explicitly error handling. For this purpose, we find it more natural to allow the operations to be, by default, non-strict. And in such a case, our formalism does not force one to do any additional specification work.

Specifying evaluation strategies

We finish this section by giving a few simple examples of specifying various evaluation strategies for boolean operations.

Example 3.1.7 *Basic specification of booleans.*

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textbf{spec Bool}^{\mathcal{MA}} =} \\
\multicolumn{2}{l}{\textbf{include: Nat}} \\
\quad\mathbf{S}: & Bool \\
\quad\Omega: & d_{Bool} \to Bool \\
& true :\to Bool \\
& false :\to Bool \\
& not : Bool \to Bool \\
& and\colon Bool \times Bool \to Bool \\
& if\ \_\ then\ \_\ else\ \_ : Bool \times Nat \times Nat \to Nat \\
\textbf{axioms}: & not \doteq not \\
& true \doteq true \\
& false \doteq false \\
& true \prec d_{Bool} \\
& false \prec d_{Bool} \\
& not(true) \doteq false \\
& not(false) \doteq true \\
& (if\ true\ then\ x\ else\ y) \doteq x \\
& (if\ false\ then\ x\ else\ y) \doteq y
\end{array}
$$

Adding axiom '$x \prec d_{Bool} \to x \doteq true, x \doteq false$' would make 'true' and '$false$' the only defined boolean values in all models. In the above specification, the same will be the case in the initial model. The operation 'if _ then _ else _' is non-strict in the second and third argument.

Example 3.1.8 *Strict evaluation of 'and' operation.*

$$
\begin{array}{lll}
\multicolumn{3}{l}{\textbf{spec StrictBool}^{\mathcal{MA}} =} \\
\multicolumn{3}{l}{\textbf{enrich: Bool by:}} \\
\textbf{axioms}: & a. & x \prec d_{Bool} \to x\ and\ true \doteq x \\
& b. & x \prec d_{Bool} \to x\ and\ false \doteq false \\
& c. & y \prec d_{Bool} \to true\ and\ y \doteq y \\
& d. & y \prec d_{Bool} \to false\ and\ y \doteq false
\end{array}
$$

Note that 'and' is under-specified outside d_{Bool}. The axioms b. and d. force 'and' to have the expected result when the arguments come from d_{Bool} but do *not exclude the possibility that this happens also when they do not. Strictness – in the sense that the result of and is included in d_{Bool} only if the arguments are – is obtained by adding two further axioms:*

$$\begin{array}{ll} e. & x\ and\ y \prec d_{Bool} \rightarrow x \prec d_{Bool} \\ f. & x\ and\ y \prec d_{Bool} \rightarrow y \prec d_{Bool} \end{array}$$

Example 3.1.9 *Left to right evaluation of 'and':*

$$\begin{array}{lll} \multicolumn{3}{l}{\textbf{spec LtoRBool}^{\mathcal{MA}} =} \\ \multicolumn{3}{l}{\textbf{enrich: Bool by:}} \\ \textbf{axioms}: & a. & true\ and\ y \doteq y \\ & b. & false\ and\ y \doteq false \end{array}$$

This evaluation may be non-strict in the first argument (it is here under-specified for first argument from outside of d_{Bool}). Left to right evaluation, strict in the the first argument, is obtained by adding the axiom:

$$c. \quad x\ and\ y \prec d_{Bool} \rightarrow x \prec d_{Bool}$$

Example 3.1.10 *Possibly non-strict parallel evaluation of 'and'.*

$$\begin{array}{lll} \multicolumn{3}{l}{\textbf{spec NSPBool}^{\mathcal{MA}} =} \\ \multicolumn{3}{l}{\textbf{extend: Bool by:}} \\ \textbf{axioms}: & a. & true\ and\ y \doteq y \\ & b. & false\ and\ y \doteq false \\ & c. & x\ and\ true \doteq x \\ & d. & x\ and\ false \doteq false \end{array}$$

We have merely indicated the possibilities of obtaining non-strict evaluations. Possible errors and error recovery are left for a more detailed specification of the possible values outside d_{Bool}.

3.2 Developing partial specifications in $\mathcal{MA}$

We illustrate the possibilities of development in $\mathcal{MA}$ from abstract specifications analogous to partial-algebra specifications, through a series of refinement steps, to specifications with explicit error handling. At the initial, most abstract level, error situations are not addressed at all. Operations known to be total are specified as deterministic, while others remain under-specified which allows, in particular, for their nondeterministic interpretation. At the next stage, error situations are identified and we indicate several possible ways to do that. Then one can begin explicit error handling, first by specifying the behavior of other operations in error situations and, eventually, by introducing explicit error values. A great flexibility of error treatment is offered enabling one to introduce error values, exceptions and various ways of reacting to them. It is to be remarked that the whole process involves merely a gradual refinement of initial specification by extending its signature and the set of axioms.

Subsection 3.2.1 to 3.2.4 gives a simple example and discusses various alternatives and subsection 3.2.5 summarizes the methodological observations.

As an example of the possibilities offered by multialgebras, we discuss various ways of developing a specification of stacks of natural numbers. The process starts from an abstract specification quite similar to partial algebra specification and ends with a specification with explicit error treatment.

3.2.1 Initial specification

We start with a standard development of specification not addressing any error situations explicitly.

$$
\begin{array}{rll}
\multicolumn{3}{l}{\textbf{spec Stack}^{\mathcal{MA}} =} \\
\textbf{include} & & \textbf{Nat} \\
\textbf{S}: & & \mathit{Stack} \\
\Omega: & & \mathit{empty} :\to \mathit{Stack} \\
& & \mathit{top} : \mathit{Stack} \to \mathit{Nat} \\
& & \mathit{pop} : \mathit{Stack} \to \mathit{Stack} \\
& & \mathit{push} : \mathit{Nat} \times \mathit{Stack} \to \mathit{Stack} \\
\textbf{axioms}: & 1. & \mathit{push}(x,s) \doteq \mathit{push}(x,s) \to \mathit{top}(\mathit{push}(x,s)) \doteq x \\
& 2. & \mathit{push}(x,s) \doteq \mathit{push}(x,s) \to \mathit{pop}(\mathit{push}(x,s)) \doteq s \\
& 3. & \mathit{empty} \doteq \mathit{empty}
\end{array}
$$

At this abstract level, only *empty* is explicitly specified to be deterministic. Axioms 1. and 2. are the "usual" stack axioms, guarded to yield well-defined (deterministic) results only on arguments which are well-defined (deterministic). This is the way we will do it in general – the outermost operation in a composition of functions will be guarded by a deterministic assertion. The way axiom 2. should be understood is: if a stack s' is constructed from $\mathit{push}(x,s)$ and s' is defined, then $\mathit{pop}(s')$ is defined and equal to s. Note that since variables only range over individual elements, we need no additional guards of the form $s \doteq s$. Also, the models for the specification may display flexible behavior on nondeterministic values resulting from *push*.

Notice that this specification is essentially the same as a partial algebra specification – just replace the sign $\doteq$ by $\stackrel{e}{=}$ and for any variable in the formula add a guard $x \doteq x$. The possibility to reuse partial algebra specifications in our framework is discussed in section 3.3.1.

3.2.2 Error situations

Error situations may be identified and marked by appropriate error constants:

1. **spec ErrStack1**$^{\mathcal{MA}}$ =
   enrich Stack by:

$$\begin{array}{rll} \Omega: & & errStack :\rightarrow Stack \\ & & errNat :\rightarrow Nat \\ \textbf{axioms}: & 1. & pop(empty) \prec errStack \\ & 2. & top(empty) \prec errNat \end{array}$$

Such error constants do not have any direct influence on the semantics; they function mainly as labels visualizing the special situations: *pop*(*empty*) may still be deterministic or not – it has only been marked as a special term "of type" $errStack$.

Instead of introducing explicit error constants, one may indicate error situations by forcing respective terms to be nondeterministic:

2. **spec ErrStack2**$^{\mathcal{MA}}$ =
   enrich Stack by:

$$\begin{array}{rll} \textbf{axioms}: & 1. & pop(empty) \doteq pop(empty) \rightarrow \\ & 2. & top(empty) \doteq top(empty) \rightarrow \end{array}$$

The inequalities ensure that the error terms cannot be interpreted as individual elements of the carrier. They must be sets – possibly empty. The associated logic from chapter 2 prevents then one from substituting such terms for variables. This solution precludes later treatment of particular error situations by means of deterministic error constants. Nevertheless, we will illustrate the flexibility offered by modelling errors by sets.[1]

Finally, one can follow the order-sorted approach by introducing constants for appropriate subsorts:

3. **spec OsStack**$^{\mathcal{MA}}$ =
   enrich Stack by:

$$\begin{array}{rll} \Omega: & & nonempty :\rightarrow Stack \\ \textbf{axioms}: & 1. & s \prec nonempty \rightarrow top(s) \doteq top(s) \\ & 2. & s \prec nonempty \rightarrow pop(s) \doteq pop(s) \\ & 3. & empty \prec nonempty \rightarrow \\ & 4. & push(x,s) \doteq push(x,s) \rightarrow push(x,s) \prec nonempty \end{array}$$

The new constant *nonempty* is used here as a subsort of non-empty stacks (by axiom 3. *empty* does not belong to this subsort) for which *pop* and *top* are defined. Note that adding "strictness' axioms like:

[1] Notice that here we are interpreting the result *set* not as a nondeterministic result but as an actual union of the elements of the set. Every refinement of this specification – eventually, also the implementation – must conform to this. The only difference between further specification refinement and eventual implementation is that the former may further restrict the range of the respective sets, while the latter may interpret elements of the set as "simultaneously present", e.g., as a recovery value and an error label/message. This point is illustrated further in 3.2.4.

$pop(s) \doteq pop(s) \rightarrow s \prec nonempty$, would preclude the possibility of deterministic error recovery at a later stage – the axiom forces the result of $pop(empty)$ to be nondeterministic (cf. **ErrStack2** above).

The order-sorted approach may be naturally combined with the explicit error constants, e.g., this specification may extend **ErrStack1**.

3.2.3 Behavior on errors

Error constants, like those introduced in **ErrStack1**, may be used for a uniform treatment of all errors which they include:

1. **spec ErrBehStack1**$^{\mathcal{MA}}$ =
   enrich ErrStack1 by:
   axioms : 1. $pop(push(x, errStack)) \doteq empty$
   2. $top(push(x, errStack)) \doteq x$
   3. $pop(push(errNat, s)) \prec pop(s), s \prec errStack$
   4. $top(push(errNat, s)) \prec top(s), s \prec errStack$

In the first two axioms, the prescribed results are always well-defined. If the first argument happens to be an element from $errNat$, axiom 1. will give $empty$ and axiom 2. will result in the same element $errNat$. The last two axioms use $\prec$ and not $\doteq$. This is so because the terms on the right-hand-side of the first atom may, possibly, be error terms (when s is $empty$). The alternatives give precedence to $errStack$ over $errNat$. If both arguments are err, axiom 1., respectively 2., will be applied – instead of axiom 3., axiom 1. will yield $empty$ as the result of pop. Similarly in axiom 4.

Another possibility is to treat each error situation separately. The following specification refines **ErrStack2** but it might as well be a refinement of **ErrStack1**:

2. **spec ErrBehStack2**$^{\mathcal{MA}}$ =
   enrich ErrStack2 by:
   axioms : 1. $pop(push(x, pop(empty))) \doteq empty$
   2. $top(push(x, pop(empty))) \doteq x$
   3. $pop(push(top(empty), s)) \prec pop(s), s \prec pop(empty)$
   4. $top(push(top(empty), s)) \prec top(s), s \prec pop(empty)$

These two possibilities take full advantage of non-strict semantics allowing one to describe actions to be performed in error situations. Notice that they do not specify explicitly the value of error terms: ($pop(empty)$ is merely labelled in the first case and made nondeterministic in the second) – they only specify the behavior of other operations applied to such error terms. $errStack$ may later be specified to be a particular value or else remain nondeterministic.

The most dramatic possibility is to delegate all responsibility for error treatment to the implementation. Explicit specification which excludes further description of error makes the result of respective error terms empty (this can be read as a requirement of raising a run-time exception):

3. **spec ErrBehStack3**$^{\mathcal{MA}}$ =
   enrich ErrStack2 by:
   axioms : 1. $s \prec pop(empty) \rightarrow$
   2. $x \prec top(empty) \rightarrow$
   3. $s \prec push(errNat, s') \rightarrow$

Notice that this implies strictness of other operations applied to these arguments, since empty argument will always lead to empty result. (For instance, $push(x, pop(empty))$ will return the empty set.) Making analogous extension of **ErrStack1** is possible, though it does not seem quite purposeful – labelling error situations will, typically, involve later their explicit treatment.

3.2.4 Error values

1. **spec ErrValStack1**$^{\mathcal{MA}}$ =
   enrich ErrBehStack1 by:
   axioms : 1. $pop(empty) \doteq empty$
   2. $top(empty) \doteq errNat$
   3. $push(errNat, s) \doteq s$

This is apparently consistent with the intention of the specification of the behavior in **ErrBehStack1** (according to first two axioms from **ErrBehStack1**, *errStack* behaves as *empty*, and according to the last two, pushing $errNat$ on s behaves then as s). However, there are serious problems with this refinement.

A counter-intuitive consequence of this enrichment is that $empty \prec errStack$ by axiom 1. from **ErrStack1**. Axiom 1. here says that, no matter what we have previously said about the error situation $pop(empty)$, it can be disregarded and that we, instead, do immediate error recovery. Since $pop(empty)$ has been earlier identified as an error *errStack*, a more plausible refinement would be to identify this error value which can take care of a possible indication of the error situation.

Another problem might occur from making $errNat$ deterministic. If it is a constant introduced in this specification (as it happened in our example), this might be ok. However, if this error constant comes from another specification **Nat**, we shouldn't force additional restrictions since these may interfere with its specification elsewhere.

The really serious problem is caused by the last axiom. It makes $push(errNat, s)$ deterministic. Thus, since $errNat$ is deterministic, we can substitute it into axioms from **Stack** and may conclude that: $top(push(errNat, s)) \doteq errNat$. However, according to **ErrBehStack1**, if: $s \not\prec errStack$ we have that $top(push(errNat, s)) \prec top(s)$. Thus, this may lead to collapsing the sort of elements (here Nat).

In general, forcing some error terms to be deterministic, requires revisiting earlier developed specifications and checking for such unintended coincidences. The uniform way of introducing errors, which is safe, is to force errors to be sets. The following specification makes $pop(empty)$ a new deterministic constants (this is ok in this example), but illustrates this generally recommended way by axiom 3.

2. **spec ErrValStack2**$^{\mathcal{MA}}$ =
   enrich ErrBehStack1 by:

$$\begin{array}{lll} & \Omega: & popEmpty :\rightarrow Stack \\ \textbf{axioms}: & 1. & pop(empty) \doteq popEmpty \\ & 2. & top(empty) \prec errNat \\ & 3. & push(errNat, s) \doteq push(errNat, s) \rightarrow \end{array}$$

We are forcing $push(errNat, s)$ to be nondeterministic in order to avoid interference with the axioms of **Stack**. The axioms from **ErrBehStack1** prescribe recovery from this situation.

The last remaining question concerns now the actual values to be returned by $push(errNat, s)$. We are doing it by specifying the values to be included in the set to be denoted by this term. On the one hand, we include there some error value – it does not have any influence on the other operations. On the other hand, we include the value which is to be used for the recovery purpose:

3. **spec ErrValStack3**$^{\mathcal{MA}}$ =
   enrich ErrValStack2 by:

$$\begin{array}{lll} \Omega: & & pushErr :\rightarrow Stack \\ \textbf{ax}: & 1. & s \prec push(errNat, s) \\ & 2. & pushErr \doteq pushErr \\ & 3. & pushErr \prec push(errNat, s) \\ & 4. & s \prec pop(pushErr) \rightarrow \\ & 5. & x \prec top(pushErr) \rightarrow \\ & 6. & s \prec push(x, pushErr) \rightarrow \end{array}$$

Together, these axioms ensure the desired behavior (as specified in **ErrBehStack1**). The first axiom includes the recovery value s, the second axiom ensures that $pushErr$ is an error stack and the third axiom includes this error value into pushing an error element to a stack. Thus, the results of other operations applied to $push(errNat, s)$ will be obtained by summing up the results applied to s and to $pushErr$.

The last three axioms make this later value insignificant for other operations – it merely marks the error situation. It can be naturally interpreted as a side effect of $push(errNat, s)$, and can be implemented as, for instance, sending an error message to the user. On the other hand, the presence of such an error value in the result set can be interpreted as an exception. The axioms 3.-5. correspond here to immediate catching this exception. Replacing, e.g., the axiom 3. with $pushErr \prec pop(pushErr)$ will then correspond to throwing this exception also from an application of *pop* until one arrives at a situation where other operations ignore this error value (specified as above by the axioms 3.-5. which correspond to catching the exception and performing the further action only on the remaining values in the set – here on s).

At any previous development step we could have extended the stack specification to a specification of bounded stack. This can be also done now as a refinement of the last specification:

4. **spec BoundedStack**$^{\mathcal{MA}}$ =
   enrich ErrValStack3 by:

$$
\begin{array}{lll}
\Omega: & & max :\to Nat \\
 & & hgh : Stack \to Nat \\
\textbf{axioms}: & 1. & max \doteq max \\
 & 2. & hgh \doteq hgh \\
 & 3. & hgh(empty) \doteq 0 \\
 & 4. & push(x,s) \doteq push(x,s) \to hgh(push(x,s)) \doteq succ(hgh(s)) \\
 & 5. & hgh(s) \doteq max \to hgh(push(x,s)) \doteq succ(max) \\
 & 6. & hgh(s) < max \to push(x,s) \doteq push(x,s) \\
 & 7. & hgh(s) \geq max \to push(x,s) \prec errStack
\end{array}
$$

Notice that the last axiom makes the error resulting from exceeding the bound behave as *empty* according to **ErrBehStack1**. If this is not desirable, one would have to identify it as a new error type with appropriate axioms.

Also, when $hgh(s) \doteq max$, the result of $push(x,s)$ will yield a possibly nondeterministic $errStack$. Nevertheless, its height is defined uniquely and uniformly for all such possible results by axiom 5.

3.2.5 The methodology for developing partial specifications

The above examples show the wide range of possible error treatments offered by multialgebras and suggest the following fixed methodology for the development of partial specifications in $\mathcal{MA}$.

1. **Initial specification**
   The first step involves the usual abstract specification which need not address any partiality issues. The only requirement on the axioms – needed for later error handling – is that the arguments of the outermost function applications should be guarded by a definedness assertion, i.e., a determinacy clause (like $push(x,s) \doteq push(x,s)$ in axioms 1., 2. of **Stack**. There is no need to guard the variables, since variables are always deterministic. Only operations that can never possibly lead to error situations should be explicitly specified deterministic.

   At this level of abstraction the form of the specification is like a partial algebra specification, with $\doteq$ instead of $\stackrel{e}{=}$. In fact, any partial-algebra specification can be reused in this way (see section 3.3.1).

2. **Error situations**
   The second step identifies error situations and, possibly, introduces order-sorting constants. Among the possibilities here, forcing the error terms to be a set is the solution offering most flexibility in later development, see (**ErrStack2**).

Alternatively, one may introduce labels in form of error constants, like in (**ErrStack1**). We do not specify that operations are strict on error constants at this level. The error constants work merely like syntactic marking until we specify the errors closer at a lower level. They are not specified here to be deterministic (later they may be) and serve merely as negative predicates telling that some terms may yield ill-defined results.

3. **Behavior on errors**
   The third step should specify the behavior of other significant operations when applied to the error constants or terms – this is the stage where we begin the error handling. Error constants may be used for a uniform error handling; alternatively, each error term may obtain separate treatment.

   Notice that here we do not introduce explicit error elements but only specify behavior in error situations. Error recovery means here that we specify an error term to behave as an ordinary term after use of some operations. Strictness means that errors remain (or become other) errors after application of operations. The most extreme case of strictness is to delegate error handling to the implementation by explicitly specifying the result set to be empty (as in **ErrBehStack3**).

4. **Error values**
   The fourth and last step is to identify the errors with some particular error sets/elements. In some cases, immediate error recovery can be specified (as was the case with *pop*(*empty*), which could be made equal *empty* in **ErrValStack1** or *popEmpty* in **ErrValStack2**). The general error recovery, however, is that exemplified by $push(errNat, s)$ in **ErrValStack3**: en error situation results in a set comprising the recovery value and an explicit error value which does not affect other operations unless it is passed as an exception.

The first two steps are flexible with respect to error handling. After step 2 we have not taken any decision regarding (non-)strictness. So working at this abstract level it should be possible to have compositional specification building operations, treating the nondeterministic error constants as predicates. From step three one actually specifies consequences of errors so strictness and error handling may interfere with compositionality.

3.3 Reuse of specifications from other institutions

In this section we formally prove the claim that we can reuse and refine specifications from other formalisms within the multialgebra setting. We start in 3.3.1 by formalizing the transformation of partial algebras specifications to multialgebras, for the sake of partiality handling, as argued in section 3.2.5, we mean that this is the preferred strategy. For other reasons one may need syntax to express predicates, so we extend the models of the institution of membership algebras,

$\mathcal{MEMB}$, allowing partial functions, and illustrates how one may transform the obtained institution to multialgebras in 3.3.2.

3.3.1 Extending the model class of $\mathcal{PA}$ specifications

The intention of passing from partial algebra to multilagebra specification is, on the one hand, to be able to reuse specifications which have been written in the former framework and, on the other, to allow for their further development with explicit error handling. The embedding of institutions from proposition 1.4.11 does not address this latter issue since it merely yields essentially the same model class.

What we will do now is to merely import the partial algebra specifications *without* augmenting them with the additional axioms (that prevents nondeterminism). This will allow us to make the desired transition: given a partial algebra specification, we translate it trivially into a multialgebraic specification. This transition results in a larger model class, where in addition to essentially the same partial models, we also have the models where operations are non-strict. Further development can now take place in the multialgebraic framework, allowing one to refine the specification to the level of explicit error treatment.

The isomorphism of respective model classes stated in fact 1.4.12 gives us the following relation called *institution transformation* in [35].

Proposition 3.3.1 *There is an institution transformation* $(\Psi^*, \alpha, \beta^-)$ *from* $\mathcal{PA}$ *to* $\mathcal{MA}$.

Proof.

- The functor $\Psi^* : \mathbf{Sign}_{\mathcal{PA}} \to \mathbf{Th}_{0\mathcal{MA}}$ is given by:

  Specifications: $\Psi^*(S, \Omega) = (S, \Omega, \emptyset)$

  Morphisms: $\Psi^*(\mu_S, \mu_\Omega)$ is the identity

- The natural transformation $\alpha : \mathsf{Sen}_{\mathcal{PA}} \to \mathsf{Sen}_{\mathcal{MA}} \circ \Psi^*$ is given by:

  Atoms: $\alpha(t \overset{e}{=} t') \equiv t \doteq t'$ auxilary definition for atoms

  Formulae: $\alpha(\{x_1, \ldots, x_k\}; a_1 \wedge \cdots \wedge a_n \to a) \equiv x_1 \doteq x_1 \wedge \ldots \wedge x_k \doteq x_k \wedge \alpha(a_1) \wedge \cdots \wedge \alpha(a_n) \to \alpha(a)$, for each clause $\{x_1, \ldots, x_k\}; a_1 \wedge \cdots \wedge a_n \to a$

  Ψ^* is extended to a functor $\Psi^* : \mathbf{Th}_{0\mathcal{PA}} \to \mathbf{Th}_{0\mathcal{MA}}$ by letting:
  $\Psi^*(\Sigma, \Gamma) = (\Sigma, \alpha_\Sigma(\Gamma))$.

- The components of the natural transformation:
  $\beta^- : \mathsf{Mod}_{\mathcal{PA}} \to \mathsf{Mod}_{\mathcal{MA}} \circ (\Psi^*)^{op}$ are β^- from definition 1.4.9:

From the embedding in proposition 1.4.11 above we have the "truth" condition: for every $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \emptyset))$ and $\phi \in \mathsf{Sen}_{\mathcal{PA}}(\Sigma)$ (Ψ is as in proposition 1.4.11):

$$M' \models^{\mathcal{MA}} \alpha_\Sigma(X; \phi) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') \models^{\mathcal{PA}} (X; \phi) \tag{3.1}$$

Since $\beta_{(\Sigma,\Gamma)}$ is an isomorphism of categories (fact 1.4.12) we get that:

1. for each $P \in \mathsf{Mod}_{\mathcal{PA}}(\Sigma, \emptyset)$ there exists an $M \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \emptyset))$, with $P = \beta(M)$ and $M = \beta^{-}(P)$.

Now, the extension of the model class is "conservative": for a $\mathcal{PA}$ specification (Σ, Γ), $\Psi^*(\Sigma, \Gamma) \subset \Psi(\Sigma, \Gamma)$ and hence $\mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma)) \subseteq \mathsf{Mod}_{\mathcal{MA}}(\Psi^*(\Sigma, \Gamma))$. Thus, the M existing by 1. above is actually an element of $\mathsf{Mod}_{\mathcal{MA}}(\Psi^*(\Sigma, \Gamma))$, and we obtain the condition for the institution transformation:
for all $P \in \mathsf{Mod}_{\mathcal{PA}}(\Sigma, \emptyset)$:

$$\beta^{-}_{(\Sigma,\emptyset)}(P) \models^{\mathcal{MA}} \alpha_{\Sigma}(\phi) \text{ iff } P \models^{\mathcal{PA}} \phi \tag{3.2}$$

$\square$

Notice that this proposition does not imply the existence of initial model for $\Psi^*(SP)$. For instance, let $f(t) \overset{e}{=} s$ be the only axiom of SP. The initial partial algebra model will have two elements $[s] = \{s, f(t)\}$ and $[t]$. The latter appears because of the underlying strictness assumption, $f(t) \overset{e}{=} s \to t \overset{e}{=} t$. The corresponding assumption is absent in $\mathcal{MA}$ and the construction of the term structure $T(\Psi^*(SP))/_{\cong}$, as given in section 1.3, will give only one element in the carrier $[s] = \{s, f(t)\}$, which does not yield a multialgebra[2].Although this is, perhaps, bad news for the adherents of initial semantics, it seems to be the obvious price for dropping the strictness assumption. And getting rid of this assumption is necessary for a more detailed specification of error situations.

3.3.2 Extending the model class of $\mathcal{MEMB}$ specifications

We now extend the membership algebras (see section 1.4.2), by allowing functions to be partial.

Definition 3.3.2 *The institution $\mathcal{PMEMB}$ is as $\mathcal{MEMB}$ except that the equalities are changed to existential equalities, $\overset{e}{=}$, and operations are interpreted as partial functions; to relate algebras we use weak homomorphism, i.e. the usual weak homomorphisms from definition 1.4.4 for operations and a homomorphism $h : A \to B$ satisfies the condition $x \in \pi^A_s \Rightarrow h(x) \in \pi^B_s$, for every predicate π.*

This is an (obvious) sub-institution of what is called $\mathcal{RP}(\mathbf{R} \overset{\mathbf{e}}{=}\!\!\to \mathbf{R} \overset{\mathbf{e}}{=})$ in [38] – an institution where signatures, in addition to sort and function symbols, may contain n-ary predicates and have a distinguished set of total function symbols. There are embeddings both ways between $\mathcal{RP}(\mathbf{R} \overset{\mathbf{e}}{=}\!\!\to \mathbf{R} \overset{\mathbf{e}}{=})$ and $\mathcal{PA}$, and $\mathcal{PA}$ is an (obvious) sub-institution of $\mathcal{PMEMB}$. (All these institutions (and many more, see [38]) specify the same categories of models, the finitely locally presentable categories). Composition of the above embeddings is illustrated in

[2]In this trivial example, an initial multialgebra could be obtained by adding the axiom $t \doteq t$. This, however, is an option left to the specifier.

the following diagram:

$$\begin{tikzcd}
 & & \mathcal{RP}(\mathbf{R} \stackrel{\mathbf{e}}{=}\!\!\rightarrow \mathbf{R} \stackrel{\mathbf{e}}{=}) \arrow[dd, hook] \\
\mathcal{PMEMB} \arrow[urr, hook] & & \\
 & & \mathcal{PA} \arrow[ull, hook]
\end{tikzcd}$$

Fact 3.3.3 *There is a simple embedding* (Φ^*, α, β) *from* $\mathcal{PMEMB}$ *to* $\mathcal{MA}$. *Moreover, each component of* β *is an isomorphism of model categories.*

- $\Phi^* : \mathbf{Sign}_{\mathcal{PMEMB}} \to \mathbf{Th}_{\mathcal{MA}}$ *is given by:* $\Phi^*(S, \Omega, \Pi) = (S, \Omega \uplus \Pi', \emptyset_\Sigma)$, *where*
  - *for each* $p \in \Pi$ *there is a corresponding constant* $p :\to \pi(p)$ *in* Π'
  - $\emptyset_\Sigma$ *contains an axiom* $y \doteq y, y \prec \omega(\overline{x}) \to \omega(\overline{x}) \doteq \omega(\overline{x})$, *for each operation* $\omega \in \Omega$
  - $\Phi^*(\mu_S, \mu_\Omega, \mu_P)$ *is the signature morphism* $(\mu_S, \mu_\Omega \uplus \mu_P)$
- *The natural transformation* $\alpha : \mathsf{Sen}_{\mathcal{PMEMB}} \to \mathsf{Sen}_{\mathcal{MA}} \circ \Phi^*$ *is given by:*
  1. $\alpha(t : c) \equiv t \prec c$ *auxiliary definition for atoms*
  2. $\alpha(t \stackrel{e}{=} t') \equiv t \doteq t'$ *auxiliary definition for atoms*
  3. $\alpha(\{x_1, \ldots, x_k\}; a_1 \wedge \cdots \wedge a_n \to a) \equiv x_1 \doteq x_1 \wedge \ldots \wedge x_k \doteq, x_k \wedge \alpha(a_1) \wedge \cdots \wedge \alpha(a_n) \to \alpha(a)$, *for Horn clause* $\{x_1, \ldots, x_k\}; a_1 \wedge \cdots \wedge a_n \to a$

  Φ^* *is extended to a functor* Φ^* :
  $\mathbf{Th}_{0\mathcal{PMEMB}} \to \mathbf{Th}_{0\mathcal{MA}}$ *by* $\Phi^*(\Sigma, \Gamma) = (\Sigma, \emptyset_\Sigma \cup \alpha_\Sigma(\Gamma))$.
- *The components of natural transformation* $\beta : \mathsf{Mod}_{\mathcal{MA}} \circ \Phi^{*op} \to \mathsf{Mod}_{\mathcal{PMEMB}}$ *are essentially the identities on models and homomorphisms (cf.* β *in definition 1.4.9). For an* $M \in \mathsf{Mod}_{\mathcal{MA}}(\Phi^*(\Sigma, \Gamma))$
  - $|\beta_\Sigma(M)| = |M|$
  - $f(x_1, \ldots, x_n)^{\beta_\Sigma(M)} = \begin{cases} x & - \text{ such that } f(x_1, \ldots, x_n)^M = \{x\} \text{ if it exists} \\ undefined & - \text{ otherwise} \end{cases}$
  - $p^{\beta_\Sigma(M)} = p^M$ *for all* $p \in \Pi$

  For any homomorphisms: $h : M \to B \in \mathsf{Mod}(\Phi^*(\Sigma, \Gamma))$, *we let* $\beta(h) = h$.

Using this fact we can construct an institution transformation from $\mathcal{PMEMB}$ to $\mathcal{MA}$ conservatively extending the model-class of $\mathcal{PMEMB}$ in the same way as we did in proposition 3.3.1 for $\mathcal{PA}$. This means that we can use the syntax of membership algebra specifications directly in a multialgebra specification and introduce additional "partiality" axioms (corresponding to $\emptyset_\Sigma$ above) at an appropriate stage. Notice also that a "totality" axiom $\omega(\overline{x}) \doteq \omega(\overline{x})$ is a strengthening of the respective "partiality" axiom – thus a "totalization" of partial operation is always possible as a simple refinement step. In particular, one can obtain a membership algebra from a partial (membership) algebra by simply adding such a totality axiom (for each operation). As the embedding between partial algebras (as in the diagram above and in [38]) show, predicates do not extend the specification power of partial algebras. Using predicates leads, however, to a more natural presentation.

3.4 Concluding remarks

We have presented a multialgebra based approach to developing specifications with partial operations. The novelty of the proposed framework lies in thinking about and modelling undefined operations by nondeterministic ones – an operation applied to an argument outside its definition domain may result in an unexpected and initially unknown value. This view leads actually to the combination of various features of several earlier approaches. It allows one to start with high level specifications, where error situations can be dealt with at the same level of abstraction as in partial algebras. Narrowing the range of nondeterminism modelling undefinedness, one can refine such specifications to a low level error handling. We have illustrated by examples a wide range of possibilities for error handling admitted by the proposed framework. In particular, utilizing sets to model error situations allows a function to return both a marking that such a situation occurred and relevant recovery values.

From the methodological perspective, based on institution transformation we have shown the possibility of reusing partial algebra specifications without the necessity to perform any translation (except for the trivial replacement of $\stackrel{e}{=}$ by $\doteq$ and adding guards for variables in the variable context of a formula.) Thus, we believe the proposed framework may be more useful, and in any case easier to apply, when extending partial algebra specifications to explicit error handling, than the frameworks based on translation of such specifications into deterministic specifications with predicates.

Chapter 4

Parameterized datatypes

The need for modularization techniques in software development is well motivated by large software projects. At the implementation stage of such a large project, it is possible to identify subtasks of the whole system as parameter programs. By stepwise identification of subtasks, the whole system can be implemented by composition of parameterized programs.

The important distinction between parameterized specifications and specifications of parameterized programs has been originally pointed out in [46]. The major difference concerns the objects which are reused.

Parameterized specification, "PSP", offers means to combine and reuse specification *texts*. This makes PSPs applicable for structuring the problem domain at the analysis stage of a software project.

A specification of a parameterized data type, "PDT" [47, 46], on the other hand, requires a reusable *implementation* of a program. Thus PDTs offer formalism to reuse program pieces, i.e. to structure programs in a modular way. E.g., specification of a data type stack parameterized by elements, **Stack[El]** requires an *implementation of a data type with a parameter*, i.e., one capable of taking any implementation of **El** and resulting in an implementation of **Stack[El]**. The model class of such a specification is seen as consisting of some – perhaps all – functors sending models of the formal parameter specification $\mathbf{X}$ to models of the parameterized specification $\mathbf{P}[\mathbf{X}]$:

$$\mathsf{FMod}(\mathbf{P}[\mathbf{X}]) \subseteq \{F : \mathsf{Mod}(\mathbf{X}) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}])\}. \tag{4.1}$$

Our semantic requirements on such functors is motivated by preservation of the parameter program combined with the possibility to extend the parameter program with new functionality and new data.

Study of PSPs has long tended in the direction of PDTs [11, 49, 12, 13, 42, 43, 19, 24]. One of the problems is that, while PSPs continued the tradition of working with classes axiomatized by (possibly conditional) equations, the PDTs require a precise grasp on individual algebras (which, for modelling purposes, are identified with programs). Now, a program P taking as a parameter another program X cannot change X - X functions in the context of P, that is in P[X], in the same way as it would in isolation. This intuition of "preserving actual parameter" has been identified as one of the semantic requirements, namely, persistency of the functors from 4.1, see e.g. [13, 49, 12]. This requirement is, however, very restrictive forbidding, in general, extension of the data types from the parameter program with new elements. However, in the purely equational context, there was hardly any syntactic counterpart of this semantic requirement. Thus, no syntactic/logical means were available for reasoning about correctness of such implementations.

Even worse, persistency turns out to be all too strong a requirement eliminating many interesting examples of PDTs as illegitimate. For instance, a functor which takes an **El**-algebra of elements and adds a new element (intended, e.g., as the "error value" resulting from inspecting the top of an empty stack in the specification **Stack[El]**) is *not* persistent. In general, most free functors are not persistent, since these involve, typically, generation of new elements. Partial algebras admit more free functors than the total ones, since "error" elements remain simply undefined, but they still exclude the possibility of adding new elements and, in particular, as pointed out in chapter 3 of explicit error treatment.

We introduce a more adequate framework for specifying PDTs. The first thing is a generalization of the classical concept of persistent functors, so that our semantic functors can add new elements to the parameter algebras. This idea, which is the main aspect of our approach, goes actually back to [42, 43]. However, it was there used only in the limited context of order-sorted algebras and was restricted to admitting only new errors elements. Our approach captures these aspects but, in addition, allows one also to extend parameter algebra with "regular" elements (e.g. extend a monoid to a grup by adding inverse elements), and to choose whether the axioms from the parameter specification shall apply to the new elements or not.

We also introduce the notion of refinement and present results about composition of PDT's. We discuss refinement of PDTs in relation to the classical concept of implementation as model class inclusion. The main difference concerns our view of PDTs as design specifications, i.e., rather low level specifications which prescribe not only some desired functionality but also a specific *structure* to the program. Refinement amounts then to introduction of additional structure and exemplifies the idea of "constructor specifications" from [47]. We also present results about vertical and horizontal composition of PDTs which, in fact, provide examples of refinement. Their relation to the respective classical concepts is also discussed.

Although the definition of the desirable semantic functors for PDTs is simple, a lot of book-keeping is required on the side of the syntax. Section 4.1 begins by introducing the syntactic preliminaries needed for specification of PDTs, and then defines the syntax and semantics of PDTs. Section 4.2 discusses syntax and semantics of actual parameter passing. Section 4.3 shows the counterparts of the classical, vertical and horizontal composition theorems. In this connection, we also encounter the concept of refinement of PDTs. Since PDTs correspond more to the design-, and not only to the requirement-specifications, their refinement reflects more specific design decisions. Unlike the classical concept of model class inclusion, refinement of PDTs amounts to introduction of additional structure. For instance, identifying a part of a flat specification as a parameter, amounts to requiring a structured, i.e., parameterized rather than a flat implementation. This section leads to a general concept of such a refinement, exemplifying the idea of "constructor specifications" from [47], which is summarized in section 4.4. Section 4.5 contains some concluding remarks.

4.1 Specifications of parameterized data types

To specify parameterized data types we will use a restricted syntax for specifications. All specifications can be naturally viewed as standard multialgebraic specifications. Also, all semantic constructions take place in the category of standard multialgebras. Subsection 4.1.3 introduces merely convenient syntactic abbreviations.

4.1.1 Signatures with sort constants

We start by modifying the concept of signature and specification. The idea is that each signature may have, in addition to the standard set of sort and operation symbols, a (possibly empty) set of distinguished constant symbols $S^\star = \bigstar \cup C^\star$. The set $\bigstar$ contains constant symbols $\star_s$ for various sort symbols s – the intention of $\star_s$ is to denote all the elements of the respective sort s. The set $C^\star$ may contain additional constants which will represent various subsorts – the constants from this set are called "subsort constants". [1]

Definition 4.1.1 *A signature with sort constants, $\Sigma_\star$, is a triple $\Sigma_\star = (\mathbf{S}, \Omega, S^\star)$, where $\Sigma = (\mathbf{S}, \Omega)$ is an ordinary signature and $S^\star = \bigstar \cup C^\star$ is a (possibly empty) set of additional constants, satisfying: $S^\star \cap \Omega = \emptyset$ and $\bigstar \cap C^\star = \emptyset$.*

Signatures with sort constants will be used merely as a syntactic representation of ordinary signatures. This is possible in multialgebraic setting since constants may denote sets of elements. [2]

We allow the set $S^\star$ to be empty. Also, we allow the set $\bigstar$ to contain several distinct constants of the same sort although their intended meaning will be the same. The technical reasons for that come up in relating construction of co-limits (proposition 4.1.6, especially, lemma 4.1.9) and, in particular, pushouts (subsection 4.2.1). For the most, we think of the set $\bigstar$ as containing one constant for each sort, i.e., as $\bigstar = \{\star_s :\to s : s \in \mathbf{S}\}$. Most relevant constructions will involve and yield such signatures. [3]

[1] Usually, the distinction between $\bigstar$ and $C^\star$ does not matter and then we will write "(sub)sort constants".

[2] In a more traditional setting, one would have to represent the sort constants, for instance, by predicates or (sub)sort symbols.

[3] In general, we use the symbol $\star_s$ for an arbitrary constant from $\bigstar$ of sort s, e.g., for a signature morphism μ, $\mu(c) \neq \star_s$ means the same as $\mu(c) \notin \bigstar$.

We will use the following operations relating the signatures with sort constants to ordinary signatures.

Definition 4.1.2 *Given a signature with sort constants* $\Sigma_\star = (\mathbf{S}, \Omega, S^\star)$ *we let:*

1. $\underline{\Sigma_\star} = (\mathbf{S}, \Omega \cup S^\star)$ *i.e.* $(\mathbf{S}, \Omega \cup \bigstar \cup C^\star)$ *– the underlying signature*
2. $\Sigma_- = (\mathbf{S}, \Omega, C^\star)$ *i.e.* $\Sigma_\star \setminus \bigstar$ *– the reduced signature*
3. $\Sigma = (\mathbf{S}, \Omega)$ *i.e.* $(\Sigma_\star \setminus S^\star)$ *– the standard (part of the) signature*

Conversely, given an ordinary signature $\Sigma = (\mathbf{S}, \Omega)$*, we let*

4. $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar)$ *, where* $\bigstar = \{\star_s :\to s : s \in \mathbf{S}\}$ *and* $\bigstar \cap \Omega = \emptyset$ *– the corresponding signature with sort constants.*
5. $\Sigma^\uparrow = (\mathbf{S}, \Omega, \emptyset)$ *, – the included signature (with sort constants).*

Unless stated otherwise, the signatures considered in this chapter will always be signatures with sort constants.

Definition 4.1.3 *A morphism between signatures with sort constants:* $\mu : \Sigma_\star \to \Sigma'_\star$ *is a signature morphism between the underlying signatures:* $\mu : \underline{\Sigma_\star} \to \underline{\Sigma'_\star}$*, sending* $S^\star$ *to* $S^{\star\prime}$ *and* Σ *to* Σ'*.*

In other words, the $\bigstar$-constants need not be sent to $\bigstar'$-constants but may be mapped to subsort constants $C^{\star\prime}$, as well.

Fact 4.1.4 *The signatures with sort constants form a category* $\mathbf{Sign}_\star$*, with the identity function as identity and function composition as composition.*

Since the signatures with sort constants essentially use underlying signature morphism, the transformation from the former to the latter (point 4 of def. 4.1.2) can be extended to a functor:

$$\underline{\ \ } : \mathbf{Sign}_\star \to \mathbf{Sign} \tag{4.2}$$

, which is the pointwise identity on the signature morphisms.

The other way around the inclusion of signatures (point 5 of def. 4.1.2) can also be extended to a functor:

$$^\uparrow : \mathbf{Sign} \to \mathbf{Sign}_\star \tag{4.3}$$

, which sends Σ to $\Sigma^\uparrow$ and is the identity on morphisms. The following fact says that **Sign** can be treated as a full subcategory of $\mathbf{Sign}_\star$.

Fact 4.1.5 $^\uparrow : \mathbf{Sign} \to \mathbf{Sign}_\star$ *is full and faithful.*

We have that for any $\Sigma \in \mathbf{Sign} : \Sigma = \underline{(\Sigma^{\uparrow})}$ but, in general, for $\Sigma_\star \in \mathbf{Sign}_\star : \Sigma_\star \not\simeq (\underline{\Sigma_\star})^{\uparrow}$. This is because for an isomorphism in $\mathbf{Sign}_\star$ we must have isomorphism between the respective sets of (sub)sort constants, but while $S^\star$ in $\Sigma_\star$ may be non-empty, it is always empty in $(\underline{\Sigma_\star})^{\uparrow}$.

The following proposition shows that finite co-limits in $\mathbf{Sign}_\star$ can be obtained from the respective co-limits in $\mathbf{Sign}$.

Proposition 4.1.6 *The functor* $\underline{\ } : \mathbf{Sign}_\star \to \mathbf{Sign}$ *reflects and preserves finite co-limits.*

To prove the proposition we show that the functor reflects and preserves initial object, sums and co-equalizers.

Lemma 4.1.7 *The functor* $\underline{\ }$ *reflects and preserves initial objects.*

Proof. The empty signature $\Sigma_\emptyset = (\emptyset, \emptyset)$ is *the* initial object in $\mathbf{Sign}$, and the empty signature $(\Sigma_\emptyset)^{\uparrow} = (\emptyset, \emptyset, \emptyset)$ is *the* initial object in $\mathbf{Sign}_\star$. But $\Sigma_\emptyset = \underline{\Sigma_\emptyset^{\uparrow}}$, so initial object is both reflected and preserved. $\square$

Lemma 4.1.8 *The functor* $\underline{\ }$ *reflects and preserves sums (binary co-products).*

Proof. Let $\Sigma_{\star 1} = (\mathbf{S}_1, \Omega_1, S_1^\star)$ and $\Sigma_{\star 2} = (\mathbf{S}_2, \Omega_2, S_2^\star)$ be $\mathbf{Sign}_\star$ objects and T be a co-cone in $\mathbf{Sign}_\star$ as in the left diagram. Assume that its image $\underline{T}$ (as in the right diagram) is a co-limit (sum) in $\mathbf{Sign}$. We have to show that T is a sum in $\mathbf{Sign}_\star$.

$$\begin{tikzcd}
T & \Sigma_\star & \\
\Sigma_{\star 1} \arrow[ur, "\iota_1"] & & \Sigma_{\star 2} \arrow[ul, "\iota_2"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\underline{T} & \underline{\Sigma_\star} \simeq \underline{\Sigma_{\star 1}} + \underline{\Sigma_{\star 2}} & \\
\underline{\Sigma_{\star 1}} \arrow[ur, "\underline{\iota_1}"] & & \underline{\Sigma_{\star 2}} \arrow[ul, "\underline{\iota_2}"']
\end{tikzcd}$$

By the standard construction in $\mathbf{Sign}$: $\underline{\Sigma_\star} \simeq \underline{\Sigma_{\star 1}} + \underline{\Sigma_{\star 2}}$, where the latter denotes disjoint union (of sort and operation symbols from both signatures). To simplify the notation, let us assume, without loss of generality, that we have equality here. Then both $\underline{\iota_i}$'s are injections,

Let C be any co-cone $\mu_1 : \Sigma_{\star 1} \to \Sigma'_\star$, $\mu_2 : \Sigma_{\star 2} \to \Sigma'_\star$ in $\mathbf{Sign}_\star$. Since $\underline{T}$ is a sum, we have a unique mediator to $\underline{C}$, $u_{\underline{\mu_1},\underline{\mu_2}} : \underline{\Sigma_\star} \to \underline{\Sigma'_\star}$, such that $\underline{\mu_i} = \underline{\iota_i}; u_{\underline{\mu_1},\underline{\mu_2}}$ for $i \in \{1, 2\}$. It is given by: for any symbol $x \in \underline{\Sigma_\star}$:

$$u_{\underline{\mu_1},\underline{\mu_2}}(x) = \begin{cases} \underline{\mu_1}(x) & if\ x \in \underline{\Sigma_{\star 1}} \\ \underline{\mu_2}(x) & if\ x \in \underline{\Sigma_{\star 2}} \end{cases} .$$

$$\begin{tikzcd}
C & \Sigma'_\star & \\
 & \Sigma_\star \arrow[u, dotted, "u_{\mu_1,\mu_2}"] & \\
\Sigma_{\star 1} \arrow[ur, "\iota_1"] \arrow[uur, bend left, "\mu_1"] & & \Sigma_{\star 2} \arrow[ul, "\iota_2"'] \arrow[uul, bend right, "\mu_2"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\underline{C} & \underline{\Sigma'_\star} & \\
 & \underline{\Sigma_{\star 1}} + \underline{\Sigma_{\star 2}} \arrow[u, dotted, "u_{\underline{\mu_1},\underline{\mu_2}}"] & \\
\underline{\Sigma_{\star 1}} \arrow[ur, "\underline{\iota_1}"] \arrow[uur, bend left, "\underline{\mu_1}"] & & \underline{\Sigma_{\star 2}} \arrow[ul, "\underline{\iota_2}"'] \arrow[uul, bend right, "\underline{\mu_2}"']
\end{tikzcd}$$

The claim is that there is also a unique mediator $u_{\mu_1,\mu_2} : \Sigma_\star \to \Sigma'_\star$. Indeed, let it be given by $u_{\mu_1,\mu_2}(x) = u_{\underline{\mu_1},\underline{\mu_2}}(x)$ for all $x \in \Sigma_\star$ (then $\underline{u_{\mu_1,\mu_2}} = u_{\underline{\mu_1},\underline{\mu_2}}$). It obviously makes $\mu_i = \iota_i; u_{\mu_1,\mu_2}$, for $i \in \{1,2\}$.

It is also a $\mathbf{Sign}_\star$ morphism because all ι_i's and μ_i's are: the (sub)sort constants $S_1^\star, S_2^\star$ from $\Sigma_{\star 1}, \Sigma_{\star 2}$, respectively, are mapped by ι_1, ι_2 to (sub)sort constants $S^\star$ in $\Sigma_\star$. Their ι_i images are also *all* the (sub)sort constants $S^\star$, since the other symbols in $\Sigma_\star$ are images of Σ_1, resp., Σ_2 symbols (i.e., of those symbols from $\Sigma_{\star 1}, \Sigma_{\star 2}$ which are *not* (sub)sort constants). Then, since also all the images under μ_i of (sub)sort constants from $\Sigma_{\star i}$ are (sub)sort constants in $\Sigma'_\star$, it follows from the definition of u_{μ_1,μ_2} that it, too, maps (sub)sort constants – of the form $c = \iota_i(c)$ – to (sub)sort constants, since $u_{\mu_1,\mu_2}(c) = \mu_i(c)$, for respective i's.

Finally, this u_{μ_1,μ_2} is unique making $\mu_i = \iota_i; u_{\mu_1,\mu_2}$. For if there is another $u \neq u_{\mu_1,\mu_2}$, such that $\mu_i = \iota_i; u$, then it would also be the case that $\underline{u} \neq u_{\underline{\mu_1},\underline{\mu_2}}$ and, furthermore, that $\underline{\mu_i} = \underline{\iota_i}; \underline{u}$, contradicting the uniqueness of $u_{\underline{\mu_1},\underline{\mu_2}}$.

Preservation of sums follows now easily. A sum $\Sigma_{\star 1} + \Sigma_{\star 2}$ in $\mathbf{Sign}_\star$ must be isomorphic to a sum as given above, i.e., a disjoint union of all the symbols from both signatures: $\Sigma_{\star 1} + \Sigma_{\star 2} \simeq (\mathbf{S}_1 \uplus \mathbf{S}_2, \Omega_1 \uplus \Omega_2, S_1^\star \uplus S_2^\star)$ where also $(\Omega_1 \uplus \Omega_2) \cap (S_1^\star \uplus S_2^\star) = \emptyset$.
But then $\underline{\Sigma_{\star 1} + \Sigma_{\star 2}} \simeq (\mathbf{S}_1 \uplus \mathbf{S}_2, (\Omega_1 \cup S_1^\star) \uplus (\Omega_2 \cup S_2^\star)) \simeq \underline{\Sigma_{\star 1}} + \underline{\Sigma_{\star 2}}$. $\square$

Lemma 4.1.9 *The functor _ reflects and preserves co-equalizers.*

Proof. Let $\Sigma_\star = (\mathbf{S}, \Omega, S^\star)$, $\Sigma'_\star = (\mathbf{S}', \Omega', S^{\star\prime})$, $\Sigma''_\star = (\mathbf{S}'', \Omega'', S^{\star\prime\prime})$ be objects in $\mathbf{Sign}_\star$ and suppose that we have a co-cone in $\mathbf{Sign}_\star$ as on the left diagram (with $\mu_1;\sigma = \mu_2;\sigma$), and that its image (on the right diagram) is a co-limit (co-equalizer) in $\mathbf{Sign}$. We have to show that the original co-cone (on the left) is a co-limit in $\mathbf{Sign}_\star$.

$$\Sigma_\star \overset{\mu_1}{\underset{\mu_1}{\rightrightarrows}} \Sigma'_\star \xrightarrow{\ \sigma\ } \Sigma''_\star \qquad\qquad \underline{\Sigma_\star} \overset{\underline{\mu_1}}{\underset{\underline{\mu_1}}{\rightrightarrows}} \underline{\Sigma'_\star} \xrightarrow{\ \underline{\sigma}\ } \underline{\Sigma''_\star} \simeq \underline{\Sigma'_\star}/_{\approx}$$

By the standard construction in $\mathbf{Sign}$, we have an isomorphism $\underline{\Sigma''_\star} \simeq \underline{\Sigma'_\star}/_{\approx}$, where $\underline{\Sigma'_\star}/_{\approx} = (\mathbf{S}'/_{\approx}, \Omega'/_{\approx})$, is the standard choice of co-equalizer i.e. the quotient by the least equivalence $\approx$ on $\underline{\Sigma'_\star}$ induced by the relation with the following components:

1. Sorts: $\approx_{\mathbf{S}'} = \{\langle \underline{\mu_1}(s), \underline{\mu_2}(s)\rangle : s \in \mathbf{S}\}$,
2. Operations: $\approx_{\Omega' \cup S^{\star\prime}} = \{\langle \underline{\mu_1}(\omega), \underline{\mu_2}(\omega)\rangle : \omega \in \Omega \cup S^\star\}$

To simplify the notation we will assume, without loss of generality, that, in fact, $\underline{\Sigma''_\star} = \underline{\Sigma'_\star}/_{\approx}$.

Since μ_1, μ_2 are $\mathbf{Sign}_\star$-morphisms, the equivalence $\approx$ above can be viewed (is the same) as the least equivalence $\sim$ on $\Sigma'_\star$ induced by the following components:

3. Sorts: $\sim_{\mathbf{S}'} = \{\langle \mu_1(s), \mu_2(s)\rangle : s \in \mathbf{S}\}$,
4. Operations: $\sim_{\Omega'} = \{\langle \mu_1(\omega), \mu_2(\omega)\rangle : \omega \in \Omega\}$

5. (sub)sort constants: $\sim_{S^{\star\prime}} = \{\langle \mu_1(c), \mu_2(c)\rangle : c \in S^\star\}$.

We then have $\underline{\Sigma'_\star/_\sim} = \underline{\Sigma'_\star}/_\approx$, so we let $\Sigma''_\star = \Sigma'_\star/_\sim$.

Let μ_1, μ_2, γ be an arbitrary co-cone as shown on the left diagram.

$$\begin{tikzcd}
\Sigma_\star \arrow[r, shift left, "\mu_1"] \arrow[r, shift right, "\mu_1"'] & \Sigma'_\star \arrow[r, "\sigma"] \arrow[dr, "\gamma"'] & \Sigma''_\star = \Sigma'_\star/_\sim \arrow[d, dotted, "u_\gamma"] \\
& & \Sigma'''_\star
\end{tikzcd}
\qquad
\begin{tikzcd}
\underline{\Sigma_\star} \arrow[r, shift left, "\underline{\mu_1}"] \arrow[r, shift right, "\underline{\mu_1}"'] & \underline{\Sigma'_\star} \arrow[r, "\underline{\sigma}"] \arrow[dr, "\underline{\gamma}"'] & \underline{\Sigma'_\star}/_\approx \arrow[d, dotted, "u_{\underline{\gamma}}"] \\
& & \underline{\Sigma'''_\star}
\end{tikzcd}$$

Since the image $\underline{\sigma}$, $\underline{\Sigma''_\star} = \underline{\Sigma'_\star/_\sim} = \underline{\Sigma'_\star}/_\approx$ is co-equalizer, we have a unique mediator $u_{\underline{\gamma}}$ making $\underline{\gamma} = \underline{\sigma}; u_{\underline{\gamma}}$. We show that $u_\gamma : \Sigma''_\star \to \Sigma'''_\star$, given by $u_\gamma(x) = u_{\underline{\gamma}}(x)$ for all symbols $x \in \Sigma''_\star$ (in particular, $\underline{u_\gamma} = u_{\underline{\gamma}}$) is a unique mediator in $\mathbf{Sign}_\star$. It obviously makes $\gamma = \sigma; u_\gamma$.

It is also a morphism in $\mathbf{Sign}_\star$. Since both σ, γ are morphisms in $\mathbf{Sign}_\star$ they send (sub)sort constants $S^{\star\prime}$ to the (sub)sort constants $S^{\star\prime\prime}$, respectively, $S^{\star\prime\prime\prime}$. Since $\underline{\sigma}$ is surjective, then so is σ, and thus the (sub)sort constants in $\Sigma''_\star$ are exactly the σ-images of (sub)sort constants $S^{\star\prime}$ from $\Sigma'_\star$. By definition of u_γ and the fact that $\gamma = \sigma; u_\gamma$, this means that for every (sub)sort constant $\sigma(c) = [c] \in S^{\star\prime\prime}$, $u_\gamma([c]) = \gamma(c) \in S^{\star\prime\prime\prime}$.

Finally, u_γ is a unique mediator. For if there was another $u \neq u_\gamma$ making $\gamma = \sigma; u$, then we would also have $\underline{u} \neq u_{\underline{\gamma}}$ and $\underline{\gamma} = \underline{\sigma}; \underline{u}$, contradicting the uniqueness of $u_{\underline{\gamma}}$.

The fact that co-equalizers are also preserved by the functor _ follows now easily. A co-equalizer of μ_1, μ_2 must be isomorphic to the quotient $\Sigma'_\star/_\sim$, with $\sim$ defined by the points 3.-5. above. But then its image $\underline{\Sigma'_\star/_\sim}$ is trivially isomorphic to the co-equalizer $\underline{\Sigma'_\star}/_\approx$ in $\mathbf{Sign}$. □

Since we can create all finite co-limits by initial objects, sums and co-equalizers the proposition 4.1.6 follows from the above lemmata. In the following we will be interested in constructing co-limits in $\mathbf{Sign}_\star$. The concrete way of doing this is to construct a co-limit in $\mathbf{Sign}$ and then make an appropriate choice of the (sub)sort constants, according to the prescriptions given in the proofs above.

4.1.2 Guarded specifications

To write specifications of parameterized data types we will use guarded axioms. In general, one only requires that the axioms from the parameter specification hold only for the "old" elements (from the parameter algebras), and guards are needed to mark these elements.

Definition 4.1.10 *Given a signature $\Sigma_\star$ with sort constants $S^\star$:*

1. *a* guard *γ is an atom of the form $x \prec c$, where x is a variable and $c \in S^\star$;*
2. *a* (fully) guarded formula *is of the form $\phi_\star = \gamma^*, \overline{a} \to \overline{b}$ where:*
   - *$\overline{a}, \overline{b}$ are sequences of Σ_- atoms and*

- γ^* *is sequence of guards* $\gamma_i = x_i \prec c_i$ *for each (and only) variable* x_i *occurring in the atoms* $\overline{a}, \overline{b}$
  γ_i*'s in such conditional formulae are called* local guards*;*

The only places where $\bigstar$ may occur are in guards – for conditional axioms in the premises. We will also allow unconditional axioms of form 1.Note that guards with sort constants, $x \prec \star$ are only special cases of guards – in general, $x \prec c$, where $c \in S^\star$, is a guard.

A formula containing only ground $\Sigma_\star$ terms (a ground $\Sigma_\star$ formula) is a fully guarded formula, according to point 2.

We will use only a restricted form of the specifications, namely, guarded specifications.

Definition 4.1.11 *A* guarded specification *is a triple* $\mathbf{SP}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$ *where:*

- $\Sigma_\star$ *is a signature with sort constants*
- *Each* $\phi_\star \in \Phi_\star$ *is a (fully) guarded formula*
- $\Gamma_\Sigma = \{x \prec \star_s : \star_s \in \bigstar\}$ *is the set of axioms called* global guards *(with appropriately sorted variables as indicated by the subscript).*

Note that all the local guards of the form $x \prec \star \rightarrow ...$ in a guarded specification are trivially satisfied due to the presence of the global guards Γ. Yet, this apparently redundant syntactic form will be of importance for defining the constructions on specifications. All such constructions will assume that the involved specifications are guarded.

Example 4.1.12 *Guarded specification of the natural numbers.*

$$
\begin{array}{rllll}
\multicolumn{5}{l}{\mathbf{spec\ Nat}_\star =} \\
\mathbf{S}: & Nat & & & \\
\Omega: & zero: & & \rightarrow Nat & \\
 & succ: & Nat & \rightarrow Nat & \\
 & pred: & Nat & \rightarrow Nat & \\
S^\star: & \star_{Nat}: & & \rightarrow Nat & \\
\Phi: & 1. & & & zero \doteq zero \\
 & 2. & x \prec \star_{Nat} & \rightarrow & succ(x) \doteq succ(x) \\
 & 3. & x \prec \star_{Nat} & \rightarrow & pred(succ(x)) \doteq x \\
\Gamma: & 4. & & & x \prec \star_{Nat}
\end{array}
$$

Obviously, there is no real difference between this and the usual specification of **Nat** (except for the constant $\star_{Nat}$ which comprises all the elements of sort Nat.)

Definition 4.1.13 *Given a guarded specification* $\mathbf{SP}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$*,*

1. *its* weakening *is a specification* $\mathbf{SP}_- = (\Sigma_\star, \Phi_\star)$.
2. *its* underlying *specification is* $\underline{\mathbf{SP}_\star} = (\underline{\Sigma_\star}, \Phi_\star \cup \Gamma_\Sigma)$.

Conversely, for an ordinary specification $\mathbf{SP} = ((\mathbf{S}, \Omega), \Phi)$, $\mathbf{SP}_\star$ *denotes the guarded specification* $(\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$, *where* $\bigstar$ *is as in def. 4.1.2,* $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar)$ *and* $\Gamma_\Sigma = \{x \prec \star_s : \star_s \in \bigstar\}$, *and* Φ_cons *contains fully guarded versions of all axioms* Φ, *i.e. with the premise* $x \prec \star_s$ *for each variable occurring in an axiom.*

As for formulae, $\mathbf{SP}_\star$ denotes, in general, a guarded specification.

Keep also in mind that, given a specification $\mathbf{SP}_\star$ with signature $\Sigma_\star$, the signature of its weakened version $\mathbf{SP}_-$ is still $\Sigma_\star$ and not Σ_-.

As models for guarded specifications we use ordinary multialgebras.

Definition 4.1.14 *The model class of a guarded specification* $\mathbf{SP}_\star$ *is the model class of its underlying specification:* $\mathsf{Mod}(\mathbf{SP}_\star) = \mathsf{Mod}(\underline{\mathbf{SP}_\star})$.

In particular, for a given ordinary specification $\mathbf{SP}$, there is obvious equivalence of model categories between the unguarded $\mathsf{Mod}(\mathbf{SP})$ and guarded $\mathsf{Mod}(\mathbf{SP}_\star)$. Also, for a given guarded specification $\mathbf{SP}_\star$ there is the obvious inclusion functor:

$$\mathsf{id}_- : \mathsf{Mod}(\mathbf{SP}_\star) \to \mathsf{Mod}(\mathbf{SP}_-) \tag{4.4}$$

which sends each algebra in the first class to itself.

4.1.3 Specification of parameterized data types

Definition 4.1.15 *A parameterized data type specification (a PDT) is a quadruple* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$, *where*

1. $\mathbf{X}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$, $\mathbf{P}[\mathbf{X}]_\star = (\Sigma'_\star, \Phi'_\star, \Gamma'_{\Sigma'})$ *are guarded specs,*
2. $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar, C^\star) \subseteq (\mathbf{S}', \Omega', \bigstar', C^{\star\prime}) = \Sigma'_\star$ *are signatures,*
3. $\mu : \bigstar \to \bigstar' \cup C^{\star\prime}$ *is called the* parameterization morphism
4. $\delta : \bigstar \to \bigstar' \cup C^{\star\prime}$ *is called the* local guard mapping
5. *the two mappings,* μ *and* δ *are such that:*
   (a) *for every axiom* $\phi_\star \in \Phi_\star : \delta(\phi_\star) \in \Phi'_\star$.
   (b) *if* $\mu(\star_s) \neq \delta(\star_s) \neq \star_s$ *then* $\mu(\star_s) \prec \delta(\star_s) \ \in \Phi'_\star$

For convenience, we treat μ and δ as signature morphisms $\Sigma_\star \to \Sigma'_\star$ which are identities on all symbols except (possibly) some of $\bigstar$ – this is reflected in point 5a, which means that if the respective axioms do not involove $\bigstar$, they are simply included in $\mathbf{P}[\mathbf{X}]_\star$. We write δ at the end of the tuple because in many constructions it plays no role, and then it may be dropped from the notation. For all practical purposes we can think of the syntax as given by μ and δ with $\delta(\star_s) = \mu(\star_s)$ or else $\delta(\star_s) = \star_s$ (see below). This covers most natural situations and will be the case in all our examples.

5a stated in more detail says: for each guarded axiom $\phi_\star \in \Phi_\star$ ::

$$\phi_\star = x_1 \prec \star_1, \ldots, x_m \prec \star_m, x_{m+1} \prec p_{m+1}, \ldots, x_z \prec p_z, \overline{a} \Rightarrow \overline{b}$$

with all the local guards explicitly listed and $\star_i \in \bigstar$ (and $\overline{a}, \overline{b}$ sequences of Σ_- atoms), the corresponding axiom $\delta(\phi_\star) \in \Phi'_\star$, where:

$$\delta(\phi_\star) = x_1 \prec \delta(\star_1), \ldots, x_m \prec \delta(\star_m), x_{m+1} \prec p_{m+1}, \ldots, x_z \prec p_z, \overline{a} \Rightarrow \overline{b}$$

Axioms of the form 5b are needed to ensure that the guarded axioms from the parameter specification will still apply, at least, to the elements originating from the parameter algebras. (We do not include the axiom $\mu(\star) \prec \delta(\star)$ when $\delta(\star) = \star$ to conform to the format from Def. 4.1.11 (and 4.1.10), but then it will be satisfied due to the global guards in $\mathbf{P}[\mathbf{X}]_\star$).

The image under $\mu : \bigstar \to S^{\star\prime}$ can be twofold and could be marked with the keywords:

1) **non-extending the carrier** s:
   This is the case when $\mu(\star_s) = \star_s$, and it corresponds to the classical case of persistency.
2) **extending carrier** s:
   Is the case when $\mu(\star_s) \neq \star_s$, we have introduced a distinction between the elements of sort s originating from the parameter, $\mu(\star_s)$, and the possibly new ones $\star_s$.

The mapping δ allows more flexibility in PDTs. If the carrier of sort s is not extended, case 1) above, δ has no effect – according to 5b, it has to be $\delta(\star_s) = \mu(\star_s) = \star_s$.[4] But if the carrier of s is extended, case 2) above, δ allows to either

2a) **restrict the local guards** from the formal parameter, when $\delta(\star_s) = \mu(\star_s)$ – in this case the axioms from the parameter specification are required to hold only for the elements from the parameter algebra,

2b) or else **extend the local guards** – in which case the axioms from the parameter specification have to hold also for possibly new elements from $\delta(\star_s)$; the presence of axioms 5b, $\mu(\star_s) \prec \delta(\star_s)$, ensures that the old axioms still hold at least for the old elements.

Again this keywords may be added to the specification language, as the respective effect can be then obtained automatically.

The case 2a) applies typically in situations when the carrier of a data type is extended with special kind of elements (like "error" values), the typical example being stacks parameterized by elements, where *pop*(*empty*) requires a new "error" element. 2b) applies in situations when the added elements are "essentially" of the same kind (e.g., group parameterized by monoid may require adding new, but "standard", inverse elements).

Example 4.1.16 *Specification of groups parameterized by monoids.*

First, we take a standard deterministic multialgebraic specification of groups, with multiplication $\cdot$*, unit* e *and inverse* $(\_)^-$*:*

[4] In general, $\delta(\star_s)$ may be equal to another subsort constant p. But then 5b forces $\mathbf{P}[\mathbf{X}]_\star \models \star_s \prec p$ which, together with the global guard $x \prec \star_s$ says that the two constants, $\star_s$ and p, denote the same set, i.e., the whole carrier of sort s.

$$
\begin{array}{lrcl}
\multicolumn{4}{l}{\textbf{spec Group} =} \\
\mathbf{S}': & S & & \\
\Omega': & \cdot: & S \times S \to S & \\
 & e: & \to S & \\
 & (\_)^{-}: & S \to S & \\
\Phi': & 1. \quad e \cdot x & \doteq & x \cdot e \\
 & 2. \quad e \cdot x & \doteq & x \\
 & 3. \quad x \cdot (y \cdot z) & \doteq & (x \cdot y) \cdot z \\
 & 4. \quad e & \doteq & e \\
 & 5. \quad x \cdot y & \doteq & x \cdot y \\
 & 6. \quad x \cdot x^{-} & \doteq & e \\
 & 7. \quad x^{-} & \doteq & x^{-}
\end{array}
$$

This does not conform to the required format – we add the sort constant $\star_S$*, a global guard* $x \prec \star_S$*, and then a new constant mono to mark the elements originating from the monoid specification. The result is the PDT on the right, with the formal parameter on the left:*

$$\boxed{\begin{array}{lllcl}
\multicolumn{5}{l}{\textbf{spec Monoid}_\star =} \\
\mathbf{S}: & S & & & \\
\Omega: & \cdot: & S \times S \to S & & \\
 & e: & \to S & & \\
S^\star: & \star_S: & \to S & & \\
\Phi: & 1. & e \cdot x & \doteq & x \cdot e \\
 & 2. & e \cdot x & \doteq & x \\
 & 3. & x \cdot (y \cdot z) & \doteq & (x \cdot y) \cdot z \\
 & 4. & e & \doteq & e \\
 & 5. & x \cdot y & \doteq & x \cdot y \\
\Gamma: & 6. & x & \prec & \star_S
\end{array}}$$

$$\Big\downarrow \; {}^{\mu(\star_S) = mono}_{\delta(\star_S) = \star_S}$$

$$\boxed{\begin{array}{lllcl}
\multicolumn{5}{l}{\textbf{spec Group[Monoid]}_\star =} \\
\mathbf{S}': & S & & & \\
\Omega': & \cdot: & S \times S \to S & & \\
 & e: & \to S & & \\
 & (_)^-: & S \to S & & \\
S^{\star\prime}: & \star_S: & \to S & & \\
 & mono: & \to S & & \\
\Phi': & 1. & e \cdot x & \doteq & x \cdot e \\
 & 2. & e \cdot x & \doteq & x \\
 & 3. & x \cdot (y \cdot z) & \doteq & (x \cdot y) \cdot z \\
 & 4. & e & \doteq & e \\
 & 5. & x \cdot y & \doteq & x \cdot y \\
 & 6. & x \cdot x^- & \doteq & e \\
 & 7. & x^- & \doteq & x^- \\
\Gamma': & 8. & x & \prec & \star_S
\end{array}}$$

The parameterization morphism sends $\mu(\star_S) = mono$. This is so because constructing a **Group***-algebra out of a given* **Monoid***-algebra, one may need to add new elements,* **extending carrier***, of sort S. The constant mono keeps the track of the old elements.*

On the other hand, even if new elements appear in the resulting algebra, we certainly want the **Monoid***-axioms to hold not only for the old but also for these new elements. Therefore, we let $\delta(\star_S) = \star_S$.*

In the presentation above, we have dropped all the local guards $x \prec \star_S \to ...$ in both specifications, since they will be trivially satisfied due to the presence of the global guards Γ, resp. Γ'. We have also dropped the axiom $mono \prec \star_S$ (required by definition 4.1.15, point 5b), since it follows from the global guard Γ'. The respective specifications are equivalent (i.e., isomorphic in $\mathbf{Th}_{\mathcal{MA}}$). We will often use such abbreviations in order to simplify the examples. (The syntactic presence of all guards will be of significance first in considering composition in section 4.3.)

It should be observed that the flexibility offered by δ is highly desirable – in many other situations, one would like to restrict the local guards from the formal parameter to hold only for the elements originating from it but not for the new ones added by the parameterized specification. The classical example of such a situation is introduction of error elements.

Example 4.1.17 *As in the previous example, we drop the local guards* $s \prec \star_{Stack}$ *and the axiom* $ok \prec \star_{El}$ *from* $\mathbf{Stack[El]}_\star$.

$$\boxed{\begin{array}{ll} \multicolumn{2}{l}{\mathbf{spec\ El}_\star =} \\ \mathbf{S}: & El \\ S^\star: & \star_{El} :\to El \\ \Gamma: & x \prec \star_{EL} \end{array}}$$

$\mu(\star_{El}) = ok$

$\delta(\star_{El}) = ok$

$$\boxed{\begin{array}{lrrcl} \multicolumn{5}{l}{\mathbf{spec\ Stack[El]}_\star =} \\ \mathbf{S}': & \multicolumn{4}{l}{Stack, El} \\ \Omega': & empty: & & & \to Stack \\ & top: & & Stack & \to El \\ & pop: & & Stack & \to Stack \\ & push: & & El \times Stack & \to Stack \\ S^{\star\prime}: & \star_{El}: & \to El & & \\ & \star_{Stack}: & \to Stack & & \\ & ok: & \to El & & \\ \Phi': & 1. & & & empty \doteq empty \\ & 2. & x \prec ok & \to & top(push(x,s)) \doteq x \\ & 3. & x \prec ok & \to & pop(push(x,s)) \doteq s \\ \Gamma': & 4. & & & x \prec \star_{EL} \\ & 5. & & & s \prec \star_{Stack} \end{array}}$$

The parameterization morphism with $\mu(\star_{El}) = ok$*, allows* **extending carrier** *of sort El. The local guard mapping* δ *coincides here with* μ*, i.e.,* $\delta(\star_{El}) = ok$*, thus* **restricting (local) guards** *of sort El – the potentially new elements of sort El arising in* **Stack***, like top(empty), are not intended to behave as the "ordinary" elements from the actual parameter sort but to function merely as, say, "error" elements.*

The effect of δ may be little visible in the above example since the parameter $\mathbf{El}_\star$ does not contain any proper axioms. However, it should be clear that a PDT where the local guards in $\mathbf{Stack[El]}_\star$ axioms 2. and 3. were replaced by $x \prec \star_{El}$ would be very different. Also, it should be easy to imagine replacing the formal parameter $\mathbf{El}_\star$ with, say $\mathbf{Nat}_\star$ from example 4.1.12, in which axioms should be guarded. For instance, the axiom $x \prec \star_{Nat} \to pred(succ(x)) \doteq x$

from $\mathbf{Nat}_\star$ would correspond to $x \prec ok \to pred(succ(x)) \doteq x$ in $\mathbf{Stack}[\mathbf{Nat}]_\star$, since we do not necessarily want this axiom to apply to the result of $top(empty)$.

We register a simple fact about the parameterization morphisms:

Fact 4.1.18 *If* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ *is a PDT then:*

1. $\mu : \mathbf{X}_\star \to \mathbf{P}[\mathbf{X}]_\star$ *may not be a specification morphism, but*
2. $\mu : \mathbf{X}_- \to \mathbf{P}[\mathbf{X}]_-$ *is a specification morphism, and hence also*
3. $\mu : \mathbf{X}_- \to \mathbf{P}[\mathbf{X}]_\star$ *is a specification morphism.*

Proof.

1. A trivial counter-example is given by $\mathbf{X}_\star$ with the only global guard axiom $x \prec \star$, $\mathbf{P}[\mathbf{X}]_\star$ with a subsort constant c and the only global guard axiom $x \prec \star$, and $\mu(\star) = \delta(\star) = c$. Obviously $\mathbf{P}[\mathbf{X}]_\star \not\models \mu(x \prec \star) = x \prec c$.
2. follows from definition 4.1.15. μ-translations of all unguarded axioms from $\mathbf{X}_-$ are included in $\mathbf{P}[\mathbf{X}]_\star$ by point 5a. For any guarded axiom $x_i \prec \star_i, \overline{a} \to \overline{b}$ from $\mathbf{X}_-$, the corresponding axiom $x_i \prec \delta(\star_i), \overline{a} \to \overline{b}$ is in $\mathbf{P}[\mathbf{X}]_\star$. But then we also have the axioms $\mu(\star_i) \prec \delta(\star_i)$ in $\mathbf{P}[\mathbf{X}]_\star$, and these together imply that $\mathbf{P}[\mathbf{X}]_\star \models x_i \prec \mu(\star_i), \overline{a} \to \overline{b}$, i.e., $\mathbf{P}[\mathbf{X}]_\star \models \mu(x_i \prec \star_i, \overline{a} \to \overline{b})$.
3. follows from 2., since $\mathbf{P}[\mathbf{X}]_\star \models \mathbf{P}[\mathbf{X}]_-$.

□

4.1.4 Semantics of parameterized data type specification

It is, of course, possible to use ordinary (loose) semantics for our PDTs, i.e., to treat them as simple parameterized specifications. But the trouble we have taken with the syntactic operations and restrictions on the specifications was meant to provide the possibility to define the semantics as parameterized data types, i.e., data types consisting of algebras parameterized by algebras. It seems to us satisfying that PDTs can be seen as more specific, special cases of parameterized specifications.

To define the semantics for PDTs we will use a special case of the general (weak) homomorphisms of multialgebras.

Definition 4.1.19 *A* tight *homomorphism* $h : A \to B$ *is a homomorphism satisfying:*

$$h(\omega^A(x_1, \ldots, x_n)) = \omega^B(h(x_1), \ldots, h(x_n))$$

The following results only appear as technical arguments in some proofs later on, hence the reader may move forward and continue reading directly after proposition 4.1.22. The tight homomorphisms and, in particular, tight monomorphisms have a logical counterpart, which will be useful in some proofs and which we now establish in proposition 4.1.22. Note that for multialgebras corresponds the tight homomorphisms exactly to the injective homomorphism. To prove the result we need the following lemma:

Lemma 4.1.20 *Let $A, B \in \mathsf{Mod}(\Sigma_\star)$, $\iota : A \to B$ be a tight $\Sigma_\star$-homomorphism, $t(x_1, \ldots, x_n)$ be a $\Sigma_\star$ term, and $\alpha : \{x_1, \ldots, x_n\} \to |A|$ an assignment. Then:*

$$\iota(t^A(\alpha(x_1), \ldots, \alpha(x_n))) = t^B(\iota(\alpha(x_1)), \ldots, \iota(\alpha(x_n)))$$

Proof. For any constant c we have $\iota(c^A) = c^B$ by definition. We drop multiple arguments to simplify notation. For any operation ω, we have $\iota(\omega^A(\alpha(x))) = \omega^B(\iota(\alpha(x)))$. The result for any term t follows trivially by induction. $\square$

Corollary 4.1.21 *With the notation from the previous lemma, let ι be a tight monomorphism.*

- *For any assignment $\alpha : X \to |A|$, let $\beta : X \to |B|$ be given by $\beta = \alpha; \iota$.*
- *Conversely, for any assignment $\beta : X \to |B|$, such that:*
  $\forall x \in X : \beta(x) \in \iota[A]$, let $\alpha : X \to |A|$ be given by $\alpha = \beta; \iota^-$.

Then for any atomic formula a : $A \models_\alpha a \iff B \models_\beta a$.

Proof. To simplify the notation, we write only a single argument/variable $X = \{x\}$.

1. If $A \models_\alpha s(x) \preceq t(x)$ (where $\preceq$ stands for $\prec$ or $\doteq$), then by 4.1.20:

   $$s^B(\iota(\alpha(x))) = \iota(s^A(\alpha(x))) \preceq \iota(t^A(\alpha(x))) = t^B(\iota(\alpha(x)))$$

   , i.e. $B \models_\beta s(x) \preceq t(x)$.

   Conversely, if $B \models_\beta s(x) \doteq t(x)$, i.e., if $s^B(\iota(\alpha(x))) = t^B(\iota(\alpha(x))) = e \in |B|$, then by 4.1.20, $\iota(s^A(\alpha(x))) = \iota(t^A(\alpha(x))) = e \in |B|$. But since ι is mono (i.e., injective, [51]), this means that $s^A(\alpha(x)) = t^A(\alpha(x)) = e' \in |A|$, i.e., $A \models_\alpha s(x) \doteq t(x)$.

   If $B \models_\beta s(x) \prec t(x)$, i.e., $s^B(\iota(\alpha(x))) \subseteq t^B(\iota(\alpha(x)))$ then, by the same argument (in particular, injectivity of ι), we get $s^A(\alpha(x))) \subseteq t^A(\alpha(x)))$, i.e., $A \models_\alpha s(x) \prec t(x)$.

2. Notice, that since ι is a monomorphism, $\alpha = \beta; \iota^-$ in point 2. is well defined. But then, $\alpha; \iota = \beta$ and the result follows by point 1.

$\square$

Proposition 4.1.22 *With the notation from lemma 4.1.20 and the above corollary, let ι be a tight monomorphism and let ϕ be an arbitrary, fully guarded formula. Then:*

$$A \models \phi \iff B \models \phi$$

Proof. Again, to simplify the notation, we write only a single argument/variable $X = \{x\}$. Let ϕ be $x \prec c, a_1, \ldots, a_m \to a_{m+1}, \ldots, a_n$.

Assume that $A \models \phi$ and let $\beta : X \to |B|$ be arbitrary assignment. If $\beta(x) \notin c^B$, then $B \models_\beta \phi$. If, on the other hand, $\beta(x) \in c^B$, then we can define $\alpha : X \to |A|$, as in point 2. of corollary 4.1.21. But then we get that for all $a_i : A \models_\alpha a_i \iff B \models_\beta a_i$, i.e., : $A \models_\alpha \phi \iff B \models_\beta \phi$. Since β was arbitrary, it follows that $B \models \phi$.

So assume that $A \not\models \phi$, and let $\alpha : X \to |A|$ be an assignment falsifying ϕ. Defining β as in point 1. of corollary 4.1.21, we get $B \not\models_\beta \phi$, i.e., $B \not\models \phi$. $\square$

Now, given a PDT $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ and a functor $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$, we obtain two functors:

- $\mathsf{id}_- : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ defined in equation (4.4) at the end of section 4.1.2, and
- the composition $\mathsf{F}; \_|_\mu : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$.

The latter has the target $\mathsf{Mod}(\mathbf{X}_-)$ and not $\mathsf{Mod}(\mathbf{X}_\star)$ because, in the case when $\mu(\star_s) = c \neq \star_s$ for some s, the reduct $A|_\mu$ of an algebra $A \in \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ may contain more elements in the sort $s^{A|_\mu}$ than those in $\star_s^{A|_\mu}$, i.e., it may fail to satisfy the global guard $x \prec \star_s$. This captures the intention that the parameterized algebra may actually add new elements to the sorts of the parameter algebra. We define (loose) semantics of parameterized data type specifications by putting some restrictions on the functors $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$. Notice that the effects of δ are present in the actual axioms of both specifications, so that we do not have to consider δ explicitly here.

Definition 4.1.23 *The semantics of the PDT $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ is the class of all functors $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$, such that there exists a natural transformation $\iota : \mathsf{id}_- \Longrightarrow \mathsf{F}; \_|_\mu$, where for each $A \in \mathsf{Mod}(\mathbf{X}_\star)$ the component ι_A is a tight $\Sigma(\mathbf{X}_\star)$-monomorphism.*

A functor with this property is called a *semantic functor* for the PDT $\mathbf{P}$, and $\mathsf{PMod}(\mathbf{P})$ will denote the class of all semantic functors for a parameterized data type.

First, let us observe that although this semantics $\mathsf{PMod}(\mathbf{P})$ is rather liberal, it does capture the structuring aspect at least in the sense that the algebras which may result from it are not all possible models of the flat specification. More precisely, a PDT $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ can be seen as defining two classes of algebras:

1. the one is simply the model class $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ with $\mathbf{P}[\mathbf{X}]_\star$ viewed as a flat specification, and
2. the other is obtained from the semantic functors of the PDT, namely, the class $\mathsf{FMod}(\mathbf{P}) = \{\mathsf{F}(X) : X \in \mathsf{Mod}(\mathbf{X}_\star), \mathsf{F} \in \mathsf{PMod}(\mathbf{P})\}$ of all $\mathbf{P}[\mathbf{X}]_\star$ algebras which can be obtained as an image of some $\mathbf{X}_\star$ algebra under some semantic functor F.

Obviously, the latter class is contained in the former.

Fact 4.1.24 *In general* $\mathsf{FMod}(\mathbf{P}) \neq \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$.

Proof. Consider following specification $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$:

$$\begin{array}{lllll} \multicolumn{5}{l}{\textbf{spec } \mathbf{X}_\star =} \\ \mathbf{S}: & s & & & \\ \Omega: & c: & \to s & & \\ S^\star: & \star_s: & \to s & & \\ \Gamma: & 1. & x \prec \star_s & & \end{array} \xrightarrow{\mu(\star_s)=d} \begin{array}{lllll} \multicolumn{5}{l}{\textbf{spec } \mathbf{P}[\mathbf{X}]_\star =} \\ \mathbf{S}': & s & & & \\ \Omega': & c: & \to s & & \\ S^{\star\prime}: & d, \star_s: & \to s & & \\ \Gamma': & 1. & x \prec \star_s & & \end{array}$$

The algebra A given by: carrier $|A| = \{c\}$ and $c^A = \star_s^A = c, d^A = \emptyset$, is in $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$. However, the reduct $A|_\mu$ has no tight $\Sigma_\star$-subalgebra in $\mathsf{Mod}(\mathbf{X}_\star)$, because $\star_s^{A|_\mu} = \emptyset$ while $c^{A|_\mu} = c$. Hence $\mathsf{FMod}(\mathbf{X}) \neq \mathsf{Mod}(\mathbf{P}[\mathbf{X}])$. $\square$

The following fact is an alternative formulation of the above definition 4.1.23.

Fact 4.1.25 *A functor* $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ *is a semantic functor of a parameterized data type specification* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ *iff:*

1. *there exists a functor* $\iota : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ *such that for every algebra* $A \in \mathsf{Mod}(\mathbf{X}_\star)$ *there is a tight* $\Sigma_\star$*-monomorphism* $\iota_A : A \to \iota(A)$.
2. *For every* $A \in \mathsf{Mod}(\mathbf{X}_\star) : \iota(A) = (\mathsf{F}(A))|_\mu$*, i.e., the following diagram commutes:* [5]

$$\begin{tikzcd} \mathsf{Mod}(\mathbf{X}_\star) \arrow[rr, "\mathsf{F}"] \arrow[dr, "\iota"'] & & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[dl, "|_\mu"] \\ & \mathsf{Mod}(\mathbf{X}_-) & \end{tikzcd}$$

3. *For any* $\Sigma_\star$*-homomorphism* $h : A \to B$*,* F *satisfies* $\mathsf{F};|_\mu(h) = h$.

The requirement of ι_A being a monomorphism implies that ι_A must be injective [51]. The tightness requirement ensures that $\iota_A(\star^A) = \mu(\star)^{\mathsf{F}(A)|_\mu}$, i.e., the carrier $s^A = \star_s^A$ is injectively embedded into the carrier $s^{\mathsf{F}(A)}$ as the subset $\mu(\star_s)^{\mathsf{F}(A)}$. Together, the requirements mean that A is a (tight) subalgebra of $\mathsf{F}(A)|_\mu$ and the carrier of this subobject corresponds bijectively in $\mathsf{F}(A)$ to $\mu(\star_s)^{\mathsf{F}(A)}$ – thus ensuring protection of the parameter algebra. This is a generalization of the requirement that F has to be a persistent functor. The classical case of persistency is obtained as the special case when $\mu(\star_s) = \star_s$, for all s.

Example 4.1.26 *For the specification of stacks from example 4.1.17 we may, for instance, define the following semantic functor:*
$\mathsf{F} : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\mathbf{Stack}[\mathbf{El}]_\star)$*:*

[5] Although the ι here is not the same as the ι in definition 4.1.23, it's role is essentially the same and there is no real danger in confusing the two.

- *objects:* $A \in \mathsf{Mod}(\mathbf{El}_\star)$ *is mapped to* $\mathsf{F}(A) \in \mathsf{Mod}(\mathbf{Stack}[\mathbf{El}]_\star)$ *given by:*
  - $El^{\mathsf{F}(A)} = El^A \cup \{\bot\}$, *where* $\bot \notin El^A$, *and*
  - $Stack^{\mathsf{F}(A)} = (El^A)^*$ *– finite strings of elements from* A
  - $empty^{\mathsf{F}(A)} = \varepsilon$ *– the empty string*
  - $push^{\mathsf{F}(A)}(x,s) = \begin{cases} s & if\ x = \bot \\ xs & otherwise \end{cases}$
  - $pop^{\mathsf{F}(A)}(xs) = s$ *and* $pop^{\mathsf{F}(A)}(\varepsilon) = \varepsilon$
  - $top^{\mathsf{F}(A)}(xs) = x$ *and* $top^{\mathsf{F}(A)}(\varepsilon) = \bot$
  - $ok^{\mathsf{F}(A)} = El^A$, $\star_{El}^{\mathsf{F}(A)} = El^{\mathsf{F}(A)}$ *and* $\star_{Stack}^{\mathsf{F}(A)} = Stack^{\mathsf{F}(A)}$.
- *morphisms:* $h : A \to B$ *is mapped to* $\mathsf{F}(h) : \mathsf{F}(A) \to \mathsf{F}(B)$ *given by:*
  - *for* $x \in El^{\mathsf{F}(A)} : \mathsf{F}(h)(x) = \begin{cases} h(x) & \text{if } x \neq \bot \\ \bot & \text{otherwise} \end{cases}$
  - *for* $s \in Stack^{\mathsf{F}(A)} : \mathsf{F}(h)(s)$ *is the pointwise application of* $\mathsf{F}(h)(x)$ *to successive* x*'s in* s.

Of course, one might attempt a more specific error treatment but we are here merely illustrating the basic idea. We check that F *satisfies definition 4.1.23 by verifying the conditions of fact 4.1.25.*

The functor $\iota : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\mathbf{El}_-)$ *is given by* $\iota(A) = \mathsf{F}(A)|_\mu$, *which obviously makes the diagram in point 2. (fact 4.1.25) commute. For each* $A \in \mathsf{Mod}(\mathbf{El}_\star)$, ι_A *is the embedding of* A *into* $\mathsf{F}(A)|_\mu$ *(*$\forall x \in |A| : \iota_A(x) = x$*), i.e., it is monomorphism. It is tight because we have that:* $\iota_A(\star^A) = El^A = (ok^{\mathsf{F}(A)})|_\mu = \mu(\star)^{\mathsf{F}(A)|_\mu}$. *To obtain this last equality it is essential that the target is required merely to be a model of* $\mathbf{EL}_-$ *and not of* $\mathbf{El}_\star$. *Verification of tightness for other operations is easy. Thus, although the reduct* $\mathsf{F}(A)|_\mu$ *contains the additional element* $\bot$ *in its carrier, this element does not interfere with the requirement of* A *being a tight subalgebra of this reduct. (*F *trivially satisfies point 3 of Fact 4.1.25.)*

Notice that, thanks to the local guards $x \prec ok \to \ldots$ in axioms 2. and 3. from $\mathbf{Stack}[\mathbf{El}]_\star$, we can actually obtain extension of the carrier with a new *El*ement $\bot$, and yet ignore it when *push*ing on stacks. The above functor illustrates just one possibility of the functor semantics from definition 4.1.23. There are, of course, more.

A functor sending each $\mathbf{El}_\star$ algebra A to the standard stack algebra where *top*(*empty*) returns the empty set would be another possibility. This would be, in fact, the solution analogous to the free-persistent functor semantics with partial algebras (the local guards $x \prec ok$ could then be dropped in axioms 2. and 3. in $\mathbf{Stack}[\mathbf{El}]_\star$). It is known that partial algebras admit more free persistent functors than the total algebras do, and we can capture this extensions since undefinedness of partial algebra terms can be modelled using empty set (as done in chapter 3).

But our extension is much more powerful since, in general, it will admit free functor semantics whenever $\mathbf{P}[\mathbf{X}]_\star$ does not force any new identifications of $\mathbf{X}_\star$ elements that aren't already forced by $\mathbf{X}_\star$. The example illustrates the possibility of extending the carrier of parameter algebra. If our specifications are deterministic (or, more generally, allow free extensions), we can always choose the free functor semantics (as long as no new identifications are implied). Such a functor can be chosen as the semantics of the specification from example 4.1.16. This cannot be done with partial algebras alone as long as one uses the classical definition of persistent functor.

Of course, the definition 4.1.15 of the syntax of PDT does not guarantee the existence of a semantic functor. As in the case of persistency, we would have to show that the parameterized theory $\mathbf{P}[\mathbf{X}]_\star$ is a conservative extension of the parameter $\mathbf{X}_-$, and this problem is undecidable. It is a possible topic for further work to identify syntactic restrictions on the specifications ensuring the existence of a semantic functor.

4.2 Actual parameter passing

Since we specify PDTs, it might seem that passing an actual parameter amounts merely to applying a semantic functor to the actual (parameter) algebra in order to obtain a new algebra. This, however, is only what happens at the level of programs implementing our specifications. Remaining still at the specification level, we want to allow passing other specifications as actual parameters to a specification of parameterized data type – such an operation should yield a specification of a *new* PDT. That is, instantiating the formal parameter of a PDT by a specification allows one to reuse the PDTs. This will be addressed mostly by the syntactic considerations below and in subsection 4.2.1. In subsection 4.2.2 we will show that instantiation at the level of specifications can be reflected at the level of semantic functors. In fact, given a semantic functor F for a PDT $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ and an instantiation of the formal parameter $\mathbf{X}_\star$ by an actual parameter $\mathbf{Y}_\star$, we can actually construct a semantic functor for the resulting specification in a canonical way. As in the classical approach to PDTs based on free-persistent functor semantics, this implies the possibility of reusing not only specifications of PDTs but also their actual implementations.

We start with an example of the intended construction. We use the specification of stacks parameterized by elements from example 4.1.17. As the actual parameter we want to pass the specification of natural numbers from example 4.1.12. (Keep in mind that all our constructions presuppose fully guarded specifications and that, given a standard specification **Nat**, we can obtain the desired form $\mathbf{Nat}_\star$ by applying the syntactic operation $\_{}_\star$ from definition 4.1.13.)

Let the actual parameter passing morphism – a specification morphism $\nu : \mathbf{El}_\star \to \mathbf{Nat}_\star$ – be given by $\nu(El) = Nat$ and $\nu(\star_{El}) = \star_{Nat}$. The result of instantiation is the following specification

$$
\begin{array}{rrrl}
\multicolumn{4}{l}{\textbf{spec Stack[Nat]}_\star =} \\
\mathbf{S}: & \multicolumn{3}{l}{Stack, Nat} \\
\Omega: & empty: & & \rightarrow Stack \\
 & top: & Stack & \rightarrow Nat \\
 & pop: & Stack & \rightarrow Stack \\
 & push: & Nat \times Stack & \rightarrow Stack \\
 & zero: & & \rightarrow Nat \\
 & succ: & Nat & \rightarrow Nat \\
S^\star: & \star_{Nat}: & \multicolumn{2}{l}{\rightarrow Nat} \\
 & \star_{Stack}: & \multicolumn{2}{l}{\rightarrow Stack} \\
 & ok: & \multicolumn{2}{l}{\rightarrow Nat}
\end{array}
$$

$$
\begin{array}{rlrcl}
\Phi: & 1. & & & zero \doteq zero \\
 & 2. & x \prec ok & \rightarrow & succ(x) \doteq succ(x) \\
 & 3. & x \prec ok & \rightarrow & pred(succ(x)) \doteq x \\
 & 4. & & & empty \doteq empty \\
 & 5. & x \prec ok & \rightarrow & top(push(x,s)) \doteq x \\
 & 6. & x \prec ok & \rightarrow & pop(push(x,s)) \doteq s \\
\Gamma: & 7. & & & x \prec \star_{Nat} \\
 & 8. & & & s \prec \star_{Stack}
\end{array}
$$

The first three axioms come from $\mathbf{Nat}_\star$. The thing to observe is that the guards $x \prec \star_{Nat}$ in axioms 2. and 3. from $\mathbf{Nat}_\star$ have changed to $x \prec ok$. This happens because the parameterization morphism was defined by $\mu(\star_{El}) = ok$, and $\delta(\star_{El}) = ok$ prescribed **restricting (local) guards**. The above specification is obtained as a pushout (in the category of multialgebraic specifications $\mathbf{Th}_{\mathcal{MA}}$ – see section 1.1.4) of μ and ν:

$$
\begin{tikzcd}
\mathbf{El}_- \arrow[r, "\mu"] \arrow[d, "\nu"'] & \mathbf{Stack}[\mathbf{El}]_\star \arrow[d, "\nu'"] \\
\mathbf{Nat}_- \arrow[r, "\mu'"'] & \mathbf{Stack}[\mathbf{Nat}]_\star
\end{tikzcd}
\tag{4.5}
$$

Notice that the formal and actual parameter specifications in this diagram are weakened (to $\mathbf{El}_-$, resp. $\mathbf{Nat}_-$) by removing their global guards. The intention of this weakening is to remove the translation $x \prec ok$ of the global guard: $x \prec \star_{Nat}$ along μ' from the resulting specification – we want to keep this global guard there, and not the stronger (but not intended) $x \prec \mu'(\star_{Nat}) = ok$. The global guard $x \prec \star_{Nat}$ enters the result along ν' from $\mathbf{Stack}[\mathbf{El}]_\star$.

The result is a (new) PDT $(\mu', \mathbf{Nat}_\star, \mathbf{Stack}[\mathbf{Nat}]_\star, \delta')$, where μ' and δ' are identities except for:

- $\mu'(\star_{Nat}) = ok$

- $\delta'(\star_{Nat}) = ok$.

The latter reflects **restricting (local) guards** in the axioms from $\mathbf{Nat}_{-}$ (2., 3.) according to μ.

4.2.1 Syntax of actual parameter passing

In order to ensure the existence of the pushout as illustrated in diagram (4.5) above, we have to ensure that the involved morphisms μ and ν from $\mathbf{El}_{-}$ are actually specification morphisms. The former is so by fact 4.1.18.3. As to the latter, we have to take into account a more general situation than in diagram (4.5), namely, the possibility that the actual parameter passing morphism is not surjective on the sorts, i.e., the actual parameter contains sorts which are not in the image of ν.

The global guards of sorts s' which are in the image of ν should be dropped (and not translated by μ'!). But if the actual parameter $\mathbf{Y}_\star$ contains a sort s which is not in the image of ν, its global guard $x \prec \star_s$ should be preserved in $\mathbf{P}[\mathbf{Y}]_\star$, while the local ones left unchanged (since parameterization does not affect these sorts at all). The treatment of global guards in these two cases calls for the definition 4.2.1 of weakening a specification $\mathbf{Y}_\star$ not to $\mathbf{Y}_{-}$ but only to $\mathbf{Y}_{\nu(-)}$ by removing only the global guards which are in the image of a given (signature) morphism ν.

Definition 4.2.1 *Given signatures* $\Sigma_\star = (\mathbf{S}, \Omega, \star)$ *and* $\Sigma' = (\mathbf{S}', \Omega', \star')$*, guarded specifications* $\mathbf{SP}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$*,* $\mathbf{SP}'_\star = (\Sigma'_\star, \Phi'_\star, \Gamma'_{\Sigma'})$ *and a signature morphism* $\nu : \Sigma_\star \to \Sigma'_\star$*, the* weakening of $\mathbf{SP}'_\star$ along ν *is the specification:*
$\mathbf{SP}'_{\nu(-)} = (\Sigma'_\star, \Phi'_\star, \Gamma'_{\Sigma'} \setminus \{x \prec \nu(\star_s) : s \in S\})$.

This weakening removes only these global guards from the target specification which are in the image of global guards from the source specification. In particular, if $\nu(\star_s) \neq \star_{\nu(s)}$, the global guard $x \prec \star_{\nu(s)}$ will not be removed, but $x \prec \nu(\star_s)$ will be. This definition will be used only in conjunction with the next one.

Definition 4.2.2 *An* actual parameter passing *is a specification morphism:* $\nu : \mathbf{X}_{-} \to \mathbf{Y}_{\nu(-)}$ *satisfying* $\nu(\star_s) = \star_{\nu(s)}$*, for all* $s \in \Sigma(\mathbf{X}_{-})$.

The intuition behind the extra requirement $\nu(\star_s) = \star_{\nu(s)}$ should be clear: $\star_s$ in the formal parameter stands for all elements of the carrier of sort s and $\star_{\nu(s)}$ does the same with respect to the elements of the carrier $\nu(s)$ in the actual parameter. Thus ν should identify the two – the restrictions induced on and expressed using $\star_s$ should now be transferred to $\star_{\nu(s)}$.

We register the following simple fact to be of relevance for defining semantics of instantiation and composition.

Lemma 4.2.3 *Given specifications* $\mathbf{X}_\star$ *and* $\mathbf{Y}_\star$*, if* $\nu : \mathbf{X}_{-} \to \mathbf{Y}_{\nu(-)}$ *is an actual parameter passing, then* $\nu : \mathbf{X}_\star \to \mathbf{Y}_\star$ *is a specification morphism.*

Proof. Let $\mathbf{X}_- = (\Sigma, \Phi_\star)$, $\mathbf{Y}_{\nu(-)} = (\Sigma', \Phi'_\star, \Gamma'_-)$, and $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ be an actual parameter passing. Suppose that $A \in \mathsf{Mod}(\mathbf{Y}_\star)$. Then

1. First, $A \in \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ since $\mathbf{Y}_{\nu(-)} \subseteq \mathbf{Y}_\star$. Then, since $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ is a specification morphism we get that $A|_\nu \in \mathsf{Mod}(\mathbf{X}_-)$, so $A|_\nu \models \Phi_\star$.
2. Since $A \in \mathsf{Mod}(\mathbf{Y}_\star)$, we have that $A \models x \prec \star_{s'}$ for all $s' \in \Sigma'$, in particular, $A \models x \prec \star_{\nu(s)}$ for all $s \in \Sigma$. Since $\nu(\star_s) = \star_{\nu(s)}$, we obtain $A|_\nu \models x \prec \star_s$ for all $s \in \Sigma$, i.e., $A|_\nu \models \Gamma_\Sigma$.

1. and 2. mean that $A|_\nu \in \mathsf{Mod}(\mathbf{X}_\star)$, and since A was arbitrary, the claim follows. $\square$

Definition 4.2.4 *Given a PDT $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ and an actual parameter passing $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$, the result of instantiation (the actual parameter passing) is a specification $\mathbf{P}[\mathbf{Y}]_\star$ obtained by pushout in $\mathbf{Th}_{\mathcal{MA}}$ of ν and μ:*

$$\begin{tikzcd}
\mathbf{X}_- \arrow[r, "\mu", "\delta"'] \arrow[d, "\nu"'] & \mathbf{P}[\mathbf{X}]_\star \arrow[d, "\nu'"] \\
\mathbf{Y}_{\nu(-)} \arrow[r, "\mu'", "\delta'"'] & \mathbf{P}[\mathbf{Y}]_\star
\end{tikzcd}$$

Now, it is not clear that the result will be a PDT. In fact, since pushout is defined only up to isomorphism, this need not be the case. To ensure that it is, we will make a canonical choice of the pushout object.

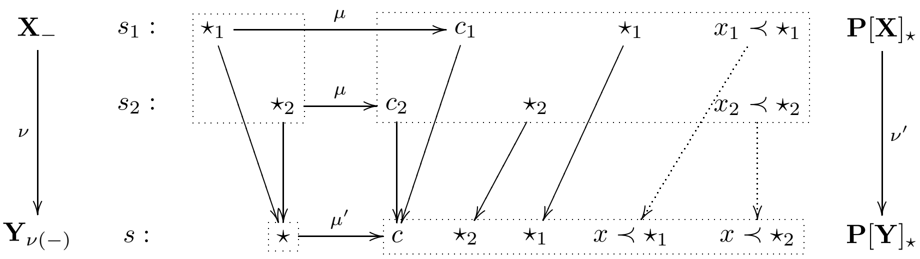
The canonical pushout object

The examples and technicalities in this subsection culminate in lemma 4.2.8 and proposition 4.2.9. Impatient reader may only register these facts and continue reading at section 4.2.2.

In the definition 4.1.15 of PDT we demand that the parameterization morphism be an inclusion on the reduced signature of the formal parameter. Thus μ' has to be the identity on $\Sigma_-(\mathbf{Y}_{\nu(-)})$, so the names in the pushout object $\mathbf{P}[\mathbf{Y}]_\star$ should be chosen appropriately.

A more intricate question concerns the choice of the sets $\bigstar$ and $C^\star$ in the resulting specification. Since morphisms in $\mathbf{Sign}_\star$ are the same as in $\mathbf{Sign}$, we may have two different signatures $\Sigma_\star \neq \Sigma'_\star$, with $\underline{\Sigma_\star} = \Sigma = \underline{\Sigma'_\star}$, which differ merely by the fact that a constant c in $\Sigma_\star$ belongs to $C^\star$ while in $\Sigma'_\star$ to $\bigstar'$. We want the resulting specification to possess sort constant $\star_s$ for each sort symbol s (assuming that the parameterized specification and the actual parameter do), and also to be fully guarded (assuming that the parameterized specification and the actual parameter are). The idea here is to include among the sort constants $\bigstar_{\mathbf{P}[\mathbf{Y}]_\star}$ of the resulting specification all the (sub)sort constants c, such that the specification contains also the respective global guard $x \prec c$. We illustrate it by the following examples.

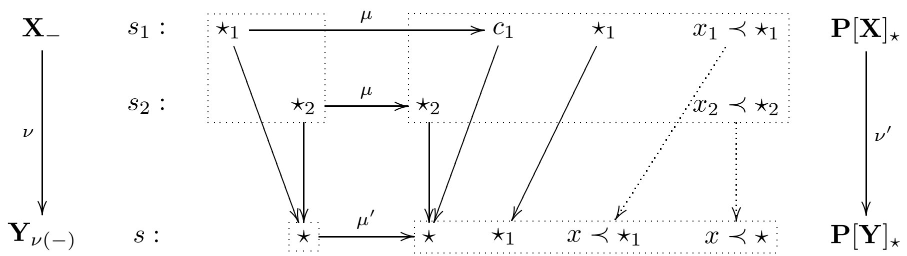
Example 4.2.5 *Let* $\mathbf{X}_\star$ *contain two sorts* s_1, s_2 *and* μ *send the respective* $\mu(\star_i) = c_i$ *in* $\mathbf{P}[\mathbf{X}]_\star$. *Let* ν *identify these two sorts, i.e.,* $\nu(s_1) = \nu(s_2) = s$. *The resulting pushout* $\mathbf{P}[\mathbf{Y}]_\star$ *is shown in the rightmost bottom corner:*



The resulting specification $\mathbf{P}[\mathbf{Y}]_\star$ *contains two constants* $\star_1$ *and* $\star_2$*, both of the same sort* s *and it also contains both global guards* $x \prec \star_1$ *and* $x \prec \star_2$ *originating from the global guards in* $\mathbf{P}[\mathbf{X}]_\star$.

The example indicates the reason for allowing the set $\bigstar$ to contain more than one constant $\star$ of each sort. Of course, trivial and automatic manipulation may be performed to remove such redundant duplicates, but its result, although in a strong sense equivalent, [6] won't be isomorphic to the pushout specification due to the difference in the signatures.

Example 4.2.6 *Let* $\mathbf{X}_\star$ *contain again two sorts* s_1, s_2 *which are identified by* ν*, i.e.,* $\nu(s_1) = \nu(s_2) = s$. *But now let* μ *send* $\mu(\star_1) = c_1$ *while* $\mu(\star_2) = \star_2$ *in* $\mathbf{P}[\mathbf{X}]_\star$. *The resulting pushout* $\mathbf{P}[\mathbf{Y}]_\star$ *is shown in the rightmost bottom corner:*



The resulting specification $\mathbf{P}[\mathbf{Y}]_\star$ *contains again two constants* $\star_1$ *and* $\star$*, both of the same sort* s*, with the respective global guards* $x \prec \star_1$ *and* $x \prec \star$ *originating from the global guards in* $\mathbf{P}[\mathbf{X}]_\star$. *The point now is that* ν' *is sending* $\nu'(c_1) = \star$ *or, in other words, that* $\mu'(\star)$ *is actually a sort constant* $\star \in \bigstar_{\mathbf{P}[\mathbf{Y}]_\star}$ *and not merely a subsort constant* $c \in C^\star_{\mathbf{P}[\mathbf{Y}]_\star}$. *This is because* $\mu(\star_2) = \star_2$ *which "overrides" the fact that* $\mu(\star_1) = c_1$.

[6] due to the global guards, the respective model categories are not only equivalent but contain "essentially" the same objects

The examples illustrate the motivation for the following definition of the canonical choice of the pushout specification and, in particular, its signature.

Definition 4.2.7 *In definition 4.2.4 we choose the (signature for the) pushout object* $\mathbf{P}[\mathbf{Y}]_\star$ *in the following canonical way.*

Given a pushout signature $\underline{\Sigma(\mathbf{P}[\mathbf{Y}]_\star)}$ *of* μ *and* ν *in* **Sign***, we choose as the names in* $\Sigma(\mathbf{P}[\mathbf{Y}]_\star)$ *all the names coming from* $\Sigma(\mathbf{Y}_\star)$ *– the rest of the names are "inherited" from* $\mathbf{P}[\mathbf{X}]_\star$*. (When several subsort constants from* $\mathbf{P}[\mathbf{X}]_\star$ *get identified (like in example 4.2.5), just choose a fresh name for the resulting subsort constant.)*

As the resulting set of sort constants, $\bigstar_{\mathbf{P}[\mathbf{Y}]_\star}$*, we take the images of all constants* $c \in S^\star_{\mathbf{Y}_\star} \cup S^\star_{\mathbf{P}[\mathbf{X}]_\star}$ *for which we also have a global guard* $x \prec c$*. These are the images under* ν' *of all* $\bigstar_{\mathbf{P}[\mathbf{X}]_\star}$ *and those* $\bigstar_{\mathbf{Y}_\star}$ *which are* not *in the image of* ν *– cf. example 4.2.6.*

The subsort constants are all the remaining images of subsort constants from $\mathbf{Y}_\star$ *and* $\mathbf{P}[\mathbf{X}]_\star$*, i.e.,*

$$C^\star_{\mathbf{P}[\mathbf{Y}]_\star} = (\nu'[S^\star_{\mathbf{P}[\mathbf{X}]_\star}] \cup \mu'[S^\star_{\mathbf{Y}_\star}]) \setminus \bigstar_{\mathbf{P}[\mathbf{Y}]_\star}.$$

Finally, as the axioms we take the union of the axioms from $\mathbf{Y}_{\nu(-)}$ *translated along* μ' *and* $\mathbf{P}[\mathbf{X}]_\star$ *translated along* ν'*.*

With this choice, we can now state two facts which will be used in the sequel.

Lemma 4.2.8 *Given a PDT* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ *with an actual parameter passing* $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$*, the pushout diagram as in the definition 4.2.4 and the canonical pushout object from definition 4.2.7: then* $\nu' : \mathbf{P}[\mathbf{X}]_- \to \mathbf{P}[\mathbf{Y}]_{\nu(-)}$ *is an actual parameter passing.*

Proof. Let $\mathbf{P}[\mathbf{X}]_\star = (\Sigma'_\star, \Phi'_\star, \Gamma'_{\Sigma'_\star})$. That $\nu'(\star_{s'}) = \star_{\nu'(s')}$ for all $s' \in \Sigma'_\star$ follows from the pushout properties, since if:
$\star_{s'} = \mu(\star_s)$, then $\nu'(\star_{s'}) = \nu(\star_s) = \star_{\nu(s)}$ (since ν is an actual parameter passing), and otherwise $\nu'(\star_s) = \star_{\nu'(s')}$. Furthermore, ν' is a specification morphism, i.e., $\mathbf{P}[\mathbf{Y}]_{\nu(-)} \models \mathbf{P}[\mathbf{X}]_-$, since for every formula:
$\phi_\star \in \mathbf{P}[\mathbf{X}]_- : \nu'(\phi_\star) \in \mathbf{P}[\mathbf{Y}]_{\nu(-)}$, by the canonical pushout construction. □

Proposition 4.2.9 *With* $\mathbf{P}[\mathbf{Y}]_\star$ *and* μ' *from definition 4.2.7,* $(\mu', \mathbf{Y}_\star, \mathbf{P}[\mathbf{Y}]_\star, \mu')$ *is a PDT.*

Proof. All global guards from $\mathbf{Y}_{\nu(-)}$ get included in $\mathbf{P}[\mathbf{Y}]_\star$. Translations along μ (and then ν') of global guards $\Gamma_{\Sigma(\mathbf{X}_\star)}$ originating from $\mathbf{X}_\star$ are "forgotten" (by starting from $\mathbf{X}_-$), but the global guards themselves are passed from $\mathbf{P}[\mathbf{X}]_\star$ along ν'. Thus $\mathbf{P}[\mathbf{Y}]_\star$ is globally guarded (if $\mathbf{Y}_\star$ and $\mathbf{P}[\mathbf{X}]_\star$ are). Also, if $\mathbf{Y}_\star$ and $\mathbf{P}[\mathbf{X}]_\star$ are fully guarded, then so is $\mathbf{P}[\mathbf{Y}]_\star$.

μ' is identity by the choice of the names in $\mathbf{P}[\mathbf{Y}]_\star$, possibly with the exception of some $\star \in \bigstar_{\mathbf{Y}_\star}$, which are in the image of $\bigstar_{\mathbf{X}_\star}$ and are mapped along μ (example 4.2.5). But these are then mapped to fresh constants, either as by μ, or by the choice of the canonical $\mathbf{P}[\mathbf{Y}]_\star$.

The local guard mapping $\delta' = \mu'$ makes inclusion of the axioms: $\mu'(\star_{s'}) \prec \delta'(\star_{s'})$ unnecessary.

Finally, by the canonical pushout construction for all the axioms $\phi_\star \in \mathbf{Y}_{\nu(-)}$ (i.e., except some global guards), we have $\mu'(\phi_\star) \in \mathbf{P}[\mathbf{Y}]_\star$, so that the point 5 of definition 4.1.15 is satisfied. $\square$

The importance of this fact is that we *always* can see the result of instantiation as a PDT. However, in some cases, it might seem more natural to choose δ' in a more specific way.

Example 4.2.10 *Assume that $\mu(\star) = c \neq \star = \delta(\star)$ and ν is an actual parameter passing in virtue of being simply an inclusion (translation) of the axioms:*

$$\begin{array}{cccc}
\mathbf{X}_- & x \prec \star, \ldots \to \ldots & \xrightarrow[\delta(\star)=\star]{\mu(\star)=c} & x \prec \star, \ldots \to \ldots & \mathbf{P}[\mathbf{X}]_\star \\
& \Big\downarrow {\scriptstyle \nu(\star)=\star} & & \Big\downarrow {\scriptstyle \nu'(\star)=\star} & \\
\mathbf{Y}_{\nu(-)} & x \prec \star, \ldots \to \ldots & \xrightarrow[\delta'(\star)=?]{\mu'(\star)=c} & \begin{array}{l} x \prec \star, \ldots \to \ldots \\ x \prec c, \ldots \to \ldots \end{array} & \mathbf{P}[\mathbf{Y}]_\star
\end{array}$$

Since the PDT allows extension of the carrier ($\mu(\star) = c$) and, at the same time, stipulates the old axioms to remain valid for the new elements ($\delta(\star) = \star$), we might in this case expect the δ' to do the same. Indeed, in this special case, it may be natural to define δ' corresponding to μ' by

$$\delta'(\star_{s'}) = \begin{cases} \nu'(\delta(\star_s)) & \text{if for some } s \in \Sigma(\mathbf{X}_\star) : \nu(s) = s' \\ \star_{s'} & \text{otherwise} \end{cases}$$

Remember that both specifications on the right ($\mathbf{P}[\mathbf{X}]_\star$ and $\mathbf{P}[\mathbf{Y}]_\star$) have all global guards, i.e., $x \prec \star$. The μ' translation of the axiom yields a weaker guard ($x \prec c \ldots$) than the ν' inclusion of the respective axiom from $\mathbf{P}[\mathbf{X}]_\star$. Indeed, the former is redundant in the presence of the latter – an isomorphic specification would be obtained by just dropping the μ' translation. Thus, in this case, we could safely use the above definition of δ' instead of the general one from proposition 4.2.9, since pushout is defined up to isomorphism.

However, the above example illustrates only a special case. This definition of δ' would not work in a more general situation.

Example 4.2.11 *Let $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ be a PDT and $\nu : \mathbf{X}_- \to \mathbf{Y}_-$ be an actual parameter passing (signature inclusion) as shown below (we only write* *relevant axioms):*

$$
\begin{tikzcd}[column sep=large, row sep=huge]
\mathbf{X}_- & x \prec \star \to f(x) \doteq g(x) \arrow[r, "\mu(\star)=c", "\delta(\star)=\star"'] \arrow[d, "\nu(\star)=\star"'] & x \prec \star \to f(x) \doteq g(x) \arrow[d, "\nu'(\star)=\star"] & \mathbf{P}[\mathbf{X}]_\star \\
\mathbf{Y}_- & \begin{array}{c} x \prec \star \to f(x) \doteq a \\ x \prec \star \to a \doteq g(x) \end{array} \arrow[r, "\mu'(\star)=c", "\delta'(\star)=?"'] & \begin{array}{c} x \prec \star \to f(x) \doteq g(x) \\ x \prec c \to f(x) \doteq a \\ x \prec c \to a \doteq g(x) \end{array} & \mathbf{P}[\mathbf{Y}]_\star
\end{tikzcd}
$$

If we replaced the local guards $x \prec c, \ldots$ in the resulting $\mathbf{P}[\mathbf{Y}]_\star$ by $x \prec \star, \ldots$, i.e., applied δ' as defined in the above example 4.2.10, we would obtain a PDT $\mathbf{P}[\mathbf{Y}]'_\star$ but it would not be a pushout object in $\mathbf{Th}_{\mathcal{MA}}$!

There is, of course, a canonical construction which replaces the local guards $x \prec c \to \ldots$ in the specification $\mathbf{P}[\mathbf{Y}]_\star$ resulting from the pushout construction, by $x \prec \star \to \ldots$, leading to another specification $\mathbf{P}[\mathbf{Y}]'_\star$. There is also an obvious specification morphism from the former to the latter (since, in the presence of global guard $x \prec \star$, we have $(x \prec \star, \overline{a} \to \overline{b}) \models (x \prec c, \overline{a} \to \overline{b})$, for any (sub)sort constant c). And finally, the specification $(\mu', \mathbf{Y}_\star, \mathbf{P}[\mathbf{Y}]'_\star, \delta')$, where δ' is as in example 4.2.10, is a PDT.

Thus, we expect that one can obtain more flexibility in passing actual parameters, but the details of that need to be postponed to a future work. For the moment, we are satisfied with the proposition 4.2.9, and ignore the details and possibility of more specific choices of the PDTs resulting from instantiation.

4.2.2 Semantics of actual parameter passing

The first aspect of the semantics of instantiation is expressed in proposition 4.2.9 – it gives a specification of a new parameterized data type. That is, we can reuse PDTs for constructing new PDTs by instantiating the formal parameter.

There is, however, another semantic aspect which will be called "actualization". Given a functor $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ defining semantics of a parameterized data type specification according to definition 4.1.23 and an actual parameter passing $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ we want to define the way of transforming $\mathbf{Y}_\star$ algebras into $\mathbf{P}[\mathbf{Y}]_\star$ algebras, i.e., a functor $\mathsf{F}' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star)$ which is *induced* by F. In addition, we want this functor to satisfy the conditions corresponding to those imposed on the semantics of parameterized data type specifications (fact 4.1.25).

Let us return to the example of parameterized specification of stacks from example 4.1.17, the chosen semantic functor $\mathsf{F} : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\mathbf{Stack}[\mathbf{El}]_\star$ from example 4.1.26 and the actual parameter passing $\nu : \mathbf{El}_- \to \mathbf{Nat}_-$ from the beginning of this section 4.2. Let N be the standard $\mathbf{Nat}_\star$ algebra (i.e., the standard $\mathbf{Nat}$ algebra with $\star^N_{Nat}$ being all the natural numbers). The functor

F' will simply embed N into $\mathsf{F}'(N)$ and mimic the action of F with respect to constructing the rest of $\mathbf{Stack}[\mathbf{Nat}]_\star$ algebra $\mathsf{F}'(N)$.

The important point is that such an actualization of functor F to a functor F' can be done canonically given a semantic functor F with the corresponding $\iota : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$. This corresponds to the classical case of free-persistent functor semantics of parameterization in the presence of amalgamation lemma. Notice, however, that our result is far more general, since we show it for any functor satisfying definition 4.1.23. Thus we do not require persistency (but allow extension of the carrier) and, furthermore, the extension need not be free, i.e., free functors are only special cases. To show this, we will need the following definition.

Definition 4.2.12 *Given a PDT* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$*, the semantic functor:* $\mathsf{F} : \mathbf{X}_\star \to \mathbf{P}[\mathbf{X}]_\star$*, with corresponding* ι *and an actual parameter passing morphism* $\nu_- : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$*. A functor* $\iota' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ *is* induced by ι *iff for all* $Y \in \mathsf{Mod}(\mathbf{Y}_\star)$*:*

1. *there is a tight monomorphism* $\iota'_Y : Y \to \iota'(Y)$*, and*
2. $\iota(Y|_\nu) = \iota'(Y)|_\nu$.

The second point uses overloaded notion of ν which is admissible by lemma 4.2.3. It means the commutativity of the leftmost square (2.) in the diagram below. The rest of the diagram is referred to in the following definition.

$$\begin{tikzcd}
 & & \mathsf{Mod}(\mathbf{X}_\star) \arrow[d, "\iota"] \arrow[drr, "\mathsf{F}"] & & \\
\mathsf{Mod}(\mathbf{Y}_\star) \arrow[urr, "|_\nu"] \arrow[d, "\iota'"'] \arrow[drrr, "\mathsf{F}'"] \arrow[rr, phantom, "2."] & & \mathsf{Mod}(\mathbf{X}_-) \arrow[rr, phantom, "1."] \arrow[dr, phantom, "3."] & & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[ll, "|_\mu"'] \\
\mathsf{Mod}(\mathbf{Y}_{\nu(-)}) \arrow[urr, "|_\nu"'] & & & \mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star) \arrow[ur, "|_{\nu'}"'] \arrow[lll, "|_{\mu'}"] &
\end{tikzcd}$$

Notice that the lower triangle of this diagram is actually a more specific case of the general requirement, namely, that ι' is a functor $\iota' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{Y}_-)$, since $\mathsf{Mod}(\mathbf{Y}_{\nu(-)}) \subseteq \mathsf{Mod}(\mathbf{Y}_-)$. It corresponds to the fact that carrier of any sort s from $\mathbf{Y}_\star$, which is not in the image of ν, is not extended in the models of $\mathbf{P}[\mathbf{Y}]_\star$ (or else, that μ' maps its $\star_s$ and the respective global guard to the same $\star_s$ and global guard in $\mathbf{P}[\mathbf{Y}]_\star$).

Definition 4.2.13 *Let* F, ι *be as in the previous definition (4.2.12) and* ι' *be induced by* ι*. The induced actualization functor* $\mathsf{F}' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star)$ *is then defined by:*

- *objects:* $\mathsf{F}'(Y) = \iota'(Y) \oplus_{\iota'(Y)|_\nu} \mathsf{F}(Y|_\nu)$

- *homomorphisms:* $\mathsf{F}'(h) = \iota'(h) \oplus_{\iota'(h)|_\nu} \mathsf{F}(h|_\nu)$

The notation $Y \oplus_X Z$ denotes the amalgamated sum of Y and Z with respect to the common reduct X (cf. section 1.1.4).

F' is well-defined since, by commutativity of 1. and 2. in the diagram above, $\iota'(Y)|_\nu = (\mathsf{F}(Y|_\nu))|_\mu$ and $\iota'(h)|_\nu = (\mathsf{F}(h|_\nu))|_\mu$. With this definition of F', all the loops in the diagram above commute.

By the continuity of the Mod-functor from $\mathcal{MA}$, proposition 1.2.12 in section 1.2.1, the square 3. is a pullback diagram, since the corresponding specification was constructed as pushout according to definition 4.2.4. Hence, by amalgamation property, $\mathsf{F}'(Y)$ is indeed guaranteed to belong to $\mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star)$.

We thus have the construction of the desired induced actualization functor provided that we have a functor ι' induced by ι. The following proposition shows that such an induced functor can always be obtained providing also a way to construct it.

Proposition 4.2.14 *Given a functor* $\iota : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ *associated with the semantic functor for the parameterized data type specification and an actual parameter passing morphism* $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$, *there exists a functor* $\iota' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ *induced by* ι.

Proof. Let the formal parameter specification be $\mathbf{X}_\star = ((\mathbf{S}, \Omega), \Phi, \Gamma_\mathbf{S})$ and the actual parameter specification:

$$\mathbf{Y}_\star = ((\nu(\mathbf{S}) \cup \mathbf{S}', \nu(\Omega) \cup \Omega'), \nu(\Phi) \cup \Phi', \nu(\Gamma_\mathbf{S}) \cup \Gamma'_{\mathbf{S}'})$$

(where $\nu(\mathbf{S}) \cap \mathbf{S}' = \nu(\Omega) \cap \Omega' = \nu(\Gamma_\mathbf{S}) \cap \Gamma'_{\mathbf{S}'} = \emptyset$). We let $\star$'s to be included in respective Ω's, since they do not require separate treatment here. s/ω range over symbols from $\mathbf{S}/\Omega$ and s'/ω' from $\mathbf{S}'/\Omega'$. The algebra $V = \iota'(Y)$ is constructed by inheriting the sorts and operations which are not in the image of ν directly from Y, while those which are in the image of ν from $\iota(Y|_\nu)$. We define ι' on:

- algebras: for every $Y \in \mathsf{Mod}(\mathbf{Y}_\star)$ we let $\iota'(Y) = V \in \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ be the following algebra:
  1. sorts:
     (a) for $s \in \mathbf{S} : \nu(s)^V = s^{\iota(Y|_\nu)}$
     (b) for $s' \in \mathbf{S}' : {s'}^V = {s'}^Y$
  2. operations:
     (a) for $\omega \in \Omega : \nu(\omega)^V = \omega^{\iota(Y|_\nu)}$,
     (b) for $\omega' \in \Omega' : \omega'(\overline{y})^V = \begin{cases} \omega'(\overline{y})^Y & \text{if all } \overline{y} \in |Y| \\ \emptyset & \text{otherwise (if some } \overline{y} \notin |Y|) \end{cases}$
- homomorphisms, essentially as identity: $\iota'(h) = h'$, where:
  $h'_{s'} = \begin{cases} h_{\nu(s)} & \text{if } s' = \nu(s) \\ h_{s'} & \text{if } s' \in \mathbf{S}' \end{cases}$

We ignore possible renaming (i.e. let $\iota'(y) = y$) and show that ι' satisfies definition 4.2.12. This defining equation makes it trivially a monomorphism. It is, indeed, a tight $\Sigma(\mathbf{Y}_\star)$-homomorphism because: for all $\omega \in \Omega$, i.e., $\nu(\omega) \in \nu(\Omega)$, and all $\overline{y} \in |Y| : \nu(\omega)(\overline{y})^V \stackrel{2a}{=} \omega(\overline{y})^{\iota(Y|_\nu)} = \omega(\overline{y})^{Y|_\nu} = \omega(\overline{y})^Y$, where the middle equality holds since ι is tight homomorphism, and the last one by the definition of reduct (1.2.1). For the remaining operations $\omega' \in \Omega'$ and $\overline{y} \in |Y| : \omega'(\overline{y})^V \stackrel{2b}{=} \omega'(\overline{y})^Y$. Furthermore, we have that $V|_\nu = \iota(Y|_\nu)$, by construction.

By construction, i.e., by 1b, V satisfies all the global guards in $\Gamma_{\mathbf{S}'}$. By proposition 4.1.22, it also satisfies all the fully guarded axioms of $\mathbf{Y}_\star$, since ι' is a tight mono. Thus $V \in \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$. $\square$

We have thus shown the possibility of reusing specifications of parameterized data types not only by syntactic instantiation (def. 4.2.4) but also by providing the above actualisation construction where we obtain a canonical canditate for the semantics of parameter instantiation.

In the final example of the whole setting, we use also the possibility offered by multialgebras to model nondeterminism.

Example 4.2.15 *We specify a generic extension of a deterministic data type with a (binary) nondeterministic choice: we give a generic PDT and instantiate it for a more specific data type.*

We do not extend carrier, i.e., $\mu(\star_{El}) = \star_{El}$.

$$
\boxed{\begin{array}{lll}
\textbf{spec } \mathbf{El}_\star = & & \\
\mathbf{S}: & El & \\
\Omega: & \star_{El}: & \to El \\
 & & \\
\Gamma: & 1. x \prec \star_{El} &
\end{array}}
\xrightarrow[\delta(\star_{El}) = \star_{El}]{\mu(\star_{El}) = \star_{El}}
\boxed{\begin{array}{llccl}
\multicolumn{5}{l}{\textbf{spec } \sqcup[\mathbf{El}]_\star =} \\
\mathbf{S}': & El & & & \\
\Omega': & & \sqcup: & El \times El & \to El \\
 & \star_{El}: & & & \to El \\
\Phi': & 1. & & & x \prec x \sqcup y \\
 & 2. & & & y \prec x \sqcup y \\
 & 3. & z \prec x \sqcup y & \to & z \doteq x, z \doteq y \\
\Gamma': & 4. & & & x \prec \star_{El}
\end{array}}
$$

As the semantic functor we take the free functor $\mathsf{F} : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\sqcup[\mathbf{El}]_\star)$, *i.e. for an* $\mathbf{El}_\star$ *algebra* A, $\mathsf{F}(A)$ *is given by:*

- $|\mathsf{F}(A)| = |A|$
- $x \sqcup^{\mathsf{F}(A)} y = \{x, y\}$ *(and generally, for subsets* $S, T \in \mathcal{P}(|A|) : S \sqcup^{\mathsf{F}(A)} T = S \cup T$*)*

In particular, when applied to a deterministic algebra A, F *will build a nondeterministic multialgebra structure on top of it, by assigning minimal set-semantics to* $\sqcup$. *Any instantiation can now be associated with an actualization of this functor. Using a specification of natural numbers as actual parameter we get the specification of natural numbers with binary choice as the result. The corresponding semantic functor (induced by* F *and the obvious parameter passing* $\nu(El) = Nat$*),* $\mathsf{F}' : \mathsf{Mod}(\mathbf{Nat}_\star) \to \mathsf{Mod}(\sqcup[\mathbf{Nat}]_\star)$, *will embed an arbitrary* $\mathbf{Nat}_\star$ *algebra* A *into* $\mathsf{F}'(A)$ *by*

- $|\mathsf{F}(A)| = |A|$
- $x \sqcup^{\mathsf{F}(A)} y = \{x, y\}$
- $\omega^{\mathsf{F}(A)} = \omega^A$ *– for all operation symbols* $\omega \in \Sigma(\mathbf{Nat}_\star)$.

One should keep in mind that although syntax and semantics of the instantiated specification are obtained from the parameterized specification itself, the two represent specifications of two distinct – and, as a mater of fact, unrelated – (parameterized) data types.

One could probably think of a more general means of specifying instantiation mechanisms at the algebra (program) level, that is, mechanisms of taking a PDT and matching an actual parameter *algebra* in order to obtain a new data type (and not merely, as we have now, a PDT which can be applied to actual parameter algebras coming only from the model class of the parameter specification). The actualization mechanism described above would be a special case of such a more general instantiation in that the "matching" of actual parameter algebras here is expressed by the reduct functor from $\mathsf{Mod}(\mathbf{Y}_\star)$ to $\mathsf{Mod}(\mathbf{X}_\star)$. This would require a closer look at the possibilities of describing the semantic functors and we have to leave such considerations for future work.

4.3 Composition of PDTs

We will now review various ways of composing specifications of parameterized data types. We will discuss the classical vertical and horizontal composition, showing the counterparts of the standard compositionality theorems. The main difference will concern the fact that, in general, stepwise application of constructions will not yield the same result as a direct construction along the respective composition, but a refinement of the latter. Subsections 4.3.1 and 4.3.2 discuss vertical, and 4.3.3 and 4.3.4 horizontal composition. Section 4.4 will summarize the concept of refinement which emerges from this section.

We recall that, given a parameter passing diagram (like 1. below in Figure 4.1), by proposition 4.2.9, $\mu' : \mathbf{Y}_{\nu(-)} \to \mathbf{P}[\mathbf{Y}]_\star$ is a parameterization morphism, and hence, in particular (by fact 4.1.18), $\mu' : \mathbf{Y}_- \to \mathbf{P}[\mathbf{Y}]_\star$ is a specification morphism.

4.3.1 Vertical composition

Given two actual parameter passing morphisms:

$$\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)} \text{ and } \rho : \mathbf{Y}_- \to \mathbf{Z}_{\rho(-)}$$

, (as indicated in the diagrams 1. and 2. in Figure 4.1), we would like to compose them vertically, i.e., we want to show that also: $(\nu;\rho) : \mathbf{X}_- \to \mathbf{Z}_{(\nu;\rho)(-)}$ is an actual parameter passing.
The notation from this figure will be used throughout this and next subsections (4.3.1, 4.3.2).

$$
\begin{tikzcd}
\mathbf{X}_{-} \arrow[r, "\mu"] \arrow[d, "\nu"'] \arrow[dr, phantom, "1."] \arrow[ddddd, bend right, "\nu;\rho"'] & \mathbf{P}[\mathbf{X}]_{\star} \arrow[d, "\nu'"] \arrow[ddddd, bend left, "(\nu;\rho)'"] \\
\mathbf{Y}_{\nu(-)} \arrow[r, "\mu'"] & \mathbf{P}[\mathbf{Y}]_{\star} \\
\mathbf{Y}_{-} \arrow[r, "\mu'"] \arrow[d, "\rho"'] \arrow[dr, phantom, "2."] & \mathbf{P}[\mathbf{Y}]_{\star} \arrow[d, "\rho'"] \\
\mathbf{Z}_{\rho(-)} \arrow[r, "\mu''"] & \mathbf{P}[\mathbf{Z}]_{\star} \\
& \\
\mathbf{Z}_{(\nu;\rho)(-)} \arrow[r, "\mu'''"] & \mathbf{P}[\mathbf{Z}]'_{\star}
\end{tikzcd}
$$

Figure 4.1:

In general, the specifications $\mathbf{Z}_{\rho(-)}$ and $\mathbf{Z}_{(\nu;\rho)(-)}$ need not be the same – the latter may have more global guards than the former.

Fact 4.3.1 *Given a PDT* $(\mu, \mathbf{X}_{\star}, \mathbf{P}[\mathbf{X}]_{\star})$ *and actual parameter passing* ν *and* ρ *as in the Figure 4.1:*

1. $\mathbf{Z}_{(\nu;\rho)(-)} \models \mathbf{Z}_{\rho(-)}$.
2. *If* ν *is surjective on the sorts, then* $\mathbf{Z}_{\rho(-)} \models \mathbf{Z}_{(\nu;\rho)(-)}$.
3. *If* ν *is surjective on the sorts, then* $\mathbf{Z}_{(\nu;\rho)(-)} \simeq \mathbf{Z}_{\rho(-)}$.

Proof. Direct from definition 4.2.1. Obviously, both specifications have isomorphic signatures (so, for simplicity, we assume that they are equal). Also, all the axioms except, possibly, some global guards, are involved in both pushout constructions and will be satisfied by both specifications. The only difference may concern absence in $\mathbf{Z}_{\rho(-)}$ of some global guards which are present in $\mathbf{Z}_{(\nu;\rho)(-)}$.

1. All sorts which are in the image of $(\nu;\rho)$ are also in the image of ρ, so the global guards dropped in $\mathbf{Z}_{(\nu;\rho)(-)}$ are also dropped in $\mathbf{Z}_{\rho(-)}$.
2. If ν is surjective on the sorts then if a sort is in the image of ρ it will also be in the image of $(\nu;\rho)$. Hence all global guards from $\mathbf{Z}_{(\nu;\rho)(-)}$ will also be present in $\mathbf{Z}_{\rho(-)}$.
3. If ν is surjective on the sorts, then the isomorphism follows from the two points above.

$\square$

Notice that, in points 2. and 3., ν's surjectivity on sorts is sufficient but not necessary condition. It is sufficient and necessary that for any sort $s \in \Sigma(\mathbf{Y})$ which is not in the image of ν, there is a sort $s' \in \Sigma(\mathbf{Y})$ which is in the image of ν and such that $\rho(s) = \rho(s')$.

Proposition 4.3.2 *If $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ and $\rho : \mathbf{Y}_- \to \mathbf{Z}_{\rho(-)}$ are actual parameter passing morphisms, then so is $(\nu;\rho) : \mathbf{X}_- \to \mathbf{Z}_{(\nu;\rho)(-)}$ (see the diagram in Figure 4.1).*

Proof. We have both $\nu(\star_s) = \star_{\nu(s)}$ and $\rho(\star_{s'}) = \star_{\rho(s')}$ for all sort symbols $s \in \Sigma(\mathbf{X})$ and $s' \in \Sigma(\mathbf{Y})$, and thus $(\nu;\rho)(\star_s) = \star_{(\nu;\rho)(s)}$. We show that:
$(\nu;\rho) : \mathbf{X}_- \to \mathbf{Z}_{(\nu;\rho)(-)}$ is a specification morphism, i.e., $\mathbf{Z}_{(\nu;\rho)(-)} \models (\nu;\rho)(\mathbf{X}_-)$.

$$\begin{array}{rcl} \text{fact 4.3.1}: & \Rightarrow & \mathbf{Z}_{(\nu;\rho)(-)} \models \mathbf{Z}_{\rho(-)} \\ \rho \text{ is a specification morphism}: & \Rightarrow & \mathbf{Z}_{\rho(-)} \models \rho(\mathbf{Y}_-) \\ & \Rightarrow & \mathbf{Z}_{(\nu;\rho)(-)} \models \rho(\mathbf{Y}_-) \end{array}$$

Now, the axioms of $\mathbf{Y}_{\nu(-)} = (\Phi, \Gamma')$, where Γ' is the subset of global guards from $\mathbf{Y}_\star$ which (whose sort symbols) are not in the image of ν. To complete the proof we have to show that $\mathbf{Z}_{(\nu;\rho)(-)} \models \rho(\Gamma')$. But this follows directly from definition 4.2.1. For any global guard $\gamma \in \Gamma'$ is *not* in the image of ν and hence it will *not* be in the image of $\nu;\rho$. Consequently, if $\gamma \in \Gamma'$ then $\rho(\gamma) \in \mathbf{Z}_{(\nu;\rho)(-)}$ (though not necessarily $\rho(\gamma) \in \mathbf{Z}_{\rho(-)}$!!).

Thus $\mathbf{Z}_{(\nu;\rho)(-)} \models \rho(\Gamma')$ which together with (4.3.1) yields

$$\mathbf{Z}_{(\nu;\rho)(-)} \models \rho(\mathbf{Y}_{\nu(-)}). \tag{4.6}$$

In diagram 1. ν is a parameter passing, so $\mathbf{Y}_{\nu(-)} \models \nu(\mathbf{X}_-)$ which implies $\rho(\mathbf{Y}_{\nu(-)}) \models (\nu;\rho)(\mathbf{X}_-)$. This, together with (4.6) give the conclusion:
$\mathbf{Z}_{(\nu;\rho)(-)} \models (\nu;\rho)(\mathbf{X}_-))$. □

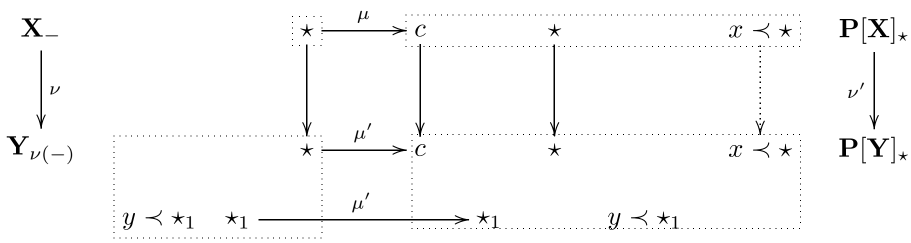
In general, the specifications $\mathbf{P}[\mathbf{Z}]_\star$ and $\mathbf{P}[\mathbf{Z}]'_\star$, in figure 4.1, may be different. In the classical case, this is merely a consequence of their definition by pushout (which is unique only up to isomorphism). In our case, however, the difference may be more significant, since we also may drop and/or add some global guards on the way. As in fact 4.3.1, the only difference may concern the presence/absence of global guards (since all other axioms are involved in the pushout construction), so these are the only axioms we mention in the following example.

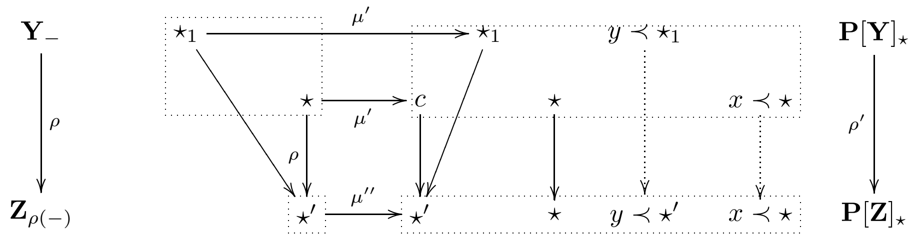
Example 4.3.3 *Consider first two instantiations:*

$$\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)} \text{ and } \rho : \mathbf{Y}_- \to \mathbf{Z}_{\rho(-)}$$

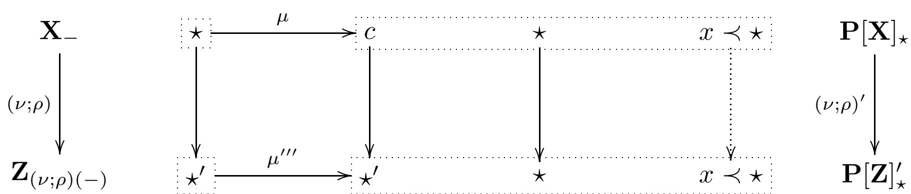
(Two lines in $\mathbf{Y}_{\nu(-)}$, $\mathbf{Y}_-$, etc. represent two distinct sorts which are identified

by the second instantiation ρ.)





And now a direct instantiation along $\nu;\rho$:



The significant difference consists in that $\mathbf{P}[\mathbf{Z}]_\star$ has the global guard for $\mu''(\star)$, namely $y \prec \star'$ originating from $\mathbf{P}[\mathbf{Y}]_\star$. Thus here $\bigstar = \{\star, \star'\}$ and $C^\star = \emptyset$. In $\mathbf{P}[\mathbf{Z}]'_\star$, on the other hand, this guard is not present, so here $\bigstar' = \{\star\}$, while $C^{\star'} = \{\star'\}$.

Thus, the PDT $(\mu'', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]_\star)$ would forbid extending the carrier of $\star'$, while $(\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star)$ would not.

So, in general, $\mathbf{P}[\mathbf{Z}]_\star$ and $\mathbf{P}[\mathbf{Z}]'_\star$ are not isomorphic. We have the following fact:

Fact 4.3.4 *With the notation from Figure 4.1 and example 4.3.3:*

1. $\mathbf{P}[\mathbf{Z}]_\star \models \mathbf{P}[\mathbf{Z}]'_\star$.
2. *if* $\mathbf{P}[\mathbf{Z}]'_\star \not\models \mathbf{P}[\mathbf{Z}]_\star$, *then it is only because for some sort constant(s)* c : $\mathbf{P}[\mathbf{Z}]_\star \models x \prec c$ *and* $\mathbf{P}[\mathbf{Z}]'_\star \not\models x \prec c$.

Proof. The signatures of both specifications will be isomorphic, so we assume that they are identical. All axioms except global guards are involved in the pushout constructions, so their presence (or satisfaction) follows from the standard isomorphism of pushout objects. The difference may concern only some constants which are in $\bigstar$ but not in $\bigstar'$ (only in $C^{\star\prime}$, as in the example 4.3.3). This justifies point 2. For point 1. we show that if $\star \in \bigstar'$ then $\star \in \bigstar$, that is if $\mathbf{P}[\mathbf{Z}]'_\star \models x \prec \star$ then $\mathbf{P}[\mathbf{Z}]_\star \models x \prec \star$, which will yield the conclusion.

This follows trivially. Any global guard $x \prec \star$ in $\mathbf{P}[\mathbf{Z}]'_\star$ is an image of a respective global guard either from $\mathbf{Z}_{(\nu;\rho)(-)}$ or from $\mathbf{P}[\mathbf{X}]_\star$. In the latter case, it will also be present in $\mathbf{P}[\mathbf{Y}]_\star$ and hence also in $\mathbf{P}[\mathbf{Z}]_\star$.

In the former case, if this guard is also in $\mathbf{Z}_{\rho(-)}$ it will be present in $\mathbf{P}[\mathbf{Z}]_\star$. If it does not belong to $\mathbf{Z}_{\rho(-)}$, this means that it (its sort) is in the image of ρ (and therefore was dropped). But then, its ρ pre-image must be in $\mathbf{Y}_\star$, that is, must be present in $\mathbf{P}[\mathbf{Y}]_\star$. But then it is also present in $\mathbf{P}[\mathbf{Z}]_\star$ as the result of pushout construction. □

This fact that stepwise instantiation along ν and then along ρ leading to $\mathbf{P}[\mathbf{Z}]_\star$ yields a different result than the direct instantiation along $\nu;\rho$ leading to $\mathbf{P}[\mathbf{Z}]'_\star$) may look like a severe weakness of our setting. After all, equality of these two indicates the desirable compositionality which would be expected by anybody familiar with the traditional, pushout based theory of parameterized specifications.

However, we are not developing a theory of parameterized specifications but of specification of parameterized data types. This means, we are interested in constructions allowing us to obtain new data types (algebras) from others. In this setting, performing different series of constructions or, as in the case of vertical composition, performing constructions in different ways, may be expected to yield different results.

Our point is that stepwise instantiation, first along ν and then along ρ represents a slightly different construction than the direct instantiation along $\eta = \nu;\rho$. In fact, we suggest to think of the former as a refinement of the latter. The latter is a one step construction along η. In this sense, splitting this construction in two steps, first along ν and then ρ, is a more detailed, refined construction which may introduce new aspects. We certainly want the result of this refined construction to be "compatible" with the results prescribed by the more rough, one step construction. This is the meaning of one construction refining another which corresponds to the classical concept of refinement by model class inclusion. This is indicated by 1. in fact 4.3.4 and we now proceed to illustrate the semantic aspect of this refinement.

4.3.2 Vertical composition – semantics

As noted in section 4.2.2, we can view the semantics of instantiation from two angles: on the one hand, as a new PDT with a class of its semantic functors and, on the other hand, as an actualization: a functor for the resulting PDT induced by a particular functor for the instantiated PDT. We now apply this distinction in the discussion of the semantics of vertical composition.

Vertical composition as a refinement of PDT

We postpone the general definition of refinement to section 4.4 and for the moment take it intuitively to mean: a PDT $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ is a *refinement* of a PDT $\mathbf{P}' = (\mu', \mathbf{X}'_\star, \mathbf{P}[\mathbf{X}]'_\star, \delta')$, $\mathbf{P}' \rightsquigarrow \mathbf{P}$, if any semantic functor for $\mathbf{P}$ can be used for obtaining a semantic functor for $\mathbf{P}'$.

A trivial, though by no means only, example of such a refinement is when $\mathbf{P}[\mathbf{X}]'_\star \rightsquigarrow \mathbf{P}[\mathbf{X}]_\star$, i.e., $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \supseteq \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$, while other components are equal. This is, in fact, the case with the results of vertical composition. If we view $\mathbf{P}[\mathbf{Z}]_\star$ and $\mathbf{P}[\mathbf{Z}]'_\star$ as two independent PDTs (i.e., "forget" that they both originate from instantiation of the same PDT), we see that, by fact 4.3.4, $\mathbf{P}[\mathbf{Z}]_\star \models \mathbf{P}[\mathbf{Z}]'_\star$, i.e., we have an inclusion (functor):

$$\mathsf{i} : \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]_\star) \subseteq \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]'_\star)$$

Thus any semantic functor F for $\mathbf{P} = (\mu'', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]_\star, \delta'')$ gives a semantic functor for $\mathbf{P}' = (\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star, \delta''')$, simply by composing $\mathsf{F};\mathsf{i}$. The other components of both PDTs are (essentially) the same, and so we get

Fact 4.3.5 *Given* $\mathbf{P} = (\mu'', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]_\star, \delta'')$ *and* $\mathbf{P}' = (\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star, \delta''')$ *(as in Figure 4.1),* $\mathbf{P}' \rightsquigarrow \mathbf{P}$.

Refinement amounts in this case to the situation illustrated in example 4.3.3, namely, that while $\mathbf{P}'$ may allow extension of some carriers (corresponding to $\star'$ in the example), $\mathbf{P}$ may forbid it by introducing additional global guards. Thus, in general, all semantic functors for $\mathbf{P}$ are also semantic functors for $\mathbf{P}'$, but there may be some functors for $\mathbf{P}'$ which are not valid semantic functors for $\mathbf{P}$.

The classical concept of vertical composition is different but, nevertheless, follows from the above. With reference to Fig. 4.1, one considers there $\mathbf{P_Y} = (\mu', \mathbf{Y}_\star, \mathbf{P}[\mathbf{Y}]_\star)$ to be an implementation of: $\mathbf{P_X} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ and also $\mathbf{P_Z} = (\mu'', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]_\star)$ to be an implementation of $\mathbf{P_Y}$. The statement is that then $\mathbf{P_Z}$ is also an implementation of $\mathbf{P_X}$. The concept of implementation, however, does not coincide with our notion of refinement of PDTs because it allows restrictions of the source as well as target categories. This will be a special case of the semantic counterpart of the diagram from Fig. 4.1, when ν and ρ induce the respective reduct functors which are inclusions. We then get that any semantic functor for the resulting PDT $\mathbf{P_Z}$ has a source and target included in, respectively, the source and target of the semantic functors for $\mathbf{P_X}$.

Vertical composition as an actualization of a particular semantic functor

There is, however, a more specific relation between the stepwise instantiation and the direct one. According to proposition 4.2.14, any semantic functor F_X for $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ induces a semantic functor F_Y for any instantiation of formal parameter $\mathbf{X}_\star$ by an actual parameter $\mathbf{Y}_\star$. If we now consider the results of respective actualizations, i.e., functors F_Z (obtained by stepwise actualization through $\mathbf{Y}_\star$ first along ν and then ρ) and F'_Z (obtained by direct actualization along $(\nu;\rho)$) which are both induced starting from the same, given F_X, then it turns out that the semantics is fully compositional, i.e., both functors are equal.

We discuss it in more detail. The semantic counterpart of the diagram from Figure 4.1 is shown below.

$$
\begin{tikzcd}
 & & & & \mathsf{Mod}(\mathbf{X}_\star) \arrow[d, "\iota_X"] \arrow[dr, "\mathsf{F}_X"] & \\
 & & & \mathsf{Mod}(\mathbf{Y}_\star) \arrow[ur, "|_\nu"] \arrow[d, "\iota_{Y1}"] \arrow[dr, "\mathsf{F}_Y"] \arrow[dl, dotted, dash, "="] & \mathsf{Mod}(\mathbf{X}_-) & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[l, "|_\mu"'] \\
 & & \mathsf{Mod}(\mathbf{Y}_\star) \arrow[d, "\iota_Y"] \arrow[dr, "\mathsf{F}'_Y"] & \mathsf{Mod}(\mathbf{Y}_{\nu(-)}) \arrow[ur, "|_\nu"] \arrow[dl, dotted, hook] & \mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star) \arrow[l, "|_{\mu'}"] \arrow[ur, "|_{\nu'}"] \arrow[dl, dotted, dash, "="] & \\
 & \mathsf{Mod}(\mathbf{Z}_\star) \arrow[ur, "|_\rho"] \arrow[d, "\iota_Z"] \arrow[dr, "\mathsf{F}_Z"] & \mathsf{Mod}(\mathbf{Y}_-) & \mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star) \arrow[l, "|_{\mu'}"'] & & \\
 \mathsf{Mod}(\mathbf{Z}_\star) \arrow[uuuurrrr, bend left, "|_{(\nu;\rho)}"] \arrow[ur, dotted, dash, "="] \arrow[d, "\iota_{Z1}"] \arrow[dr, "\mathsf{F}'_Z"] & \mathsf{Mod}(\mathbf{Z}_{\rho(-)}) \arrow[ur, "|_\rho"] & \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]_\star) \arrow[l, "|_{\mu''}"] \arrow[ur, "|_{\rho'}"] \arrow[dl, dotted, hook, "\mathsf{i}"] & & & \\
 \mathsf{Mod}(\mathbf{Z}_{(\nu;\rho)(-)}) \arrow[ur, dotted, hook] & \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]'_\star) \arrow[l, "|_{\mu'''}"] \arrow[uuuurrrr, bend right, "|_{(\nu;\rho)'}"'] & & & &
\end{tikzcd}
$$

Given a semantic functor F_X in the uppermost diagram, proposition 4.2.14 allows us to construct a functor F_Y, and similarly, an F_Z can be constructed given an arbitrary F'_Y. Thus, using F_Y obtained from the actualization along ν for F'_Y, we can construct an F_Z from a given F_X. Notice that the associated ι_Z guarantees the image of $\mathsf{Mod}(\mathbf{Z}_\star)$ to be included in $\mathsf{Mod}(\mathbf{Z}_{\rho(-)})$.

For the direct actualization, we can obtain F'_Z from a given F_X by proposition 4.2.14. On the other hand, by fact 4.3.4, we also have the inclusion functor $\mathsf{i} : \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]_\star) \subseteq \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]'_\star)$. Hence, composing we obtain:

$$\mathsf{F}_Z; \mathsf{i} : \mathsf{Mod}(\mathbf{Z}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]'_\star)$$

, which gives a possible semantic functor F'_Z for the PDT $\mathbf{P}' = (\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star, \delta''')$. Compositionality of actualization is expressed in the following proposition.

Proposition 4.3.6 *With the notation from the diagram above, where all functors are induced by* F_X *(in particular,* $\mathsf{F}_Y = \mathsf{F}'_Y$ *and* $\iota_Y = \iota_{Y1}$*), then:* $\mathsf{F}_Z; \mathsf{i} = \mathsf{F}'_Z$.

Proof. The discussion above shows that $\mathsf{F}_Z;\mathsf{i}$ is a possible semantic functor for $\mathbf{P}'$. To show the equality to F'_Z induced by a direct actualization, there remains a couple of tedious details.

1. Firstly, ι_{Z1}, associated with the functor F'_Z obtained from the direct actualization, will include $\mathsf{Mod}(\mathbf{Z}_\star)$ in $\mathsf{Mod}(\mathbf{Z}_{(\nu;\rho)(-)})$, while ι_Z associated with F_Z obtained through the stepwise actualization guarantees only inclusion in $\mathsf{Mod}(\mathbf{Z}_{\rho(-)})$. We show that F_Z actually is a special case of a functor obtained from a direct actualization, i.e., that actually:
$\mathsf{F}_Z;|_{\mu''} : \mathsf{Mod}(\mathbf{Z}_\star) \rightarrow \mathsf{Mod}(\mathbf{Z}_{(\nu;\rho)(-)})$, i.e., the lowest square (with two inclusions and reducts) commutes.

   (a) This is unproblematic when $\mathbf{Z}_{\rho(-)} = \mathbf{Z}_{(\nu;\rho)(-)}$, so let us consider the case when they are not equal. Then there is a global guard $x \prec c_s$ (of sort s) which is included in $\mathbf{Z}_{(\nu;\rho)(-)}$ but not in $\mathbf{Z}_{\rho(-)}$. This means that for any algebra $A \in \mathsf{Mod}(\mathbf{Z}_\star)$, $(\mathsf{F}'_Z(A))|_{\mu'''} \models x \prec c$ – in other words, F'_Z does not extend the carrier of s.

   (b) In principle, from the diagram, it might look that F_Z might extend this carrier since we may have $x \prec c_s \notin \mathbf{Z}_{\rho(-)}$. However, this last fact holds only if s is in the image of ρ (which makes the respective global guards disappear from $\mathbf{Z}_{\rho(-)}$). At the same time, since the guard is present in $\mathbf{Z}_{(\nu;\rho)(-)}$, it means that s is not in the image of $(\nu;\rho)$ – hence its ρ pre-image s' must not be in the image of ν.

   (c) This means that the respective guard $x \prec c_{s'} \in \mathbf{Y}_{\nu(-)}$ and, by the pushout construction, $x \prec c_{s'} \in \mathbf{P}[\mathbf{Y}]_\star$. But then the respective guard $x \prec \rho'(c_{s'}) = c_s$ will also appear in $\mathbf{P}[\mathbf{Z}]_\star$. Finally, since s' is not in the image of ν, we get $\mu'(c_{s'}) = c_{s'}$, which implies that also $\mu''(c_s) = c_s$. In short F_Z will not, after all, extend the carrier of sort s, and hence $(\mathsf{F}_Z(A))|_{\mu''} \in \mathsf{Mod}(\mathbf{Z}_{(\nu;\rho)(-)})$.

2. To prove the main claim, we need to look at the details of definitions of induced functors. What we have to show is that the following two are equal for any $A \in \mathsf{Mod}(\mathbf{Z}_\star)$ (cf. definition 4.2.13):

   (a) $\mathsf{F}'_Z(A) = \iota_{Z1}(A) \oplus_{\iota_{Z1}(A)|_{(\nu;\rho)}} \mathsf{F}_X(A|_{(\nu;\rho)})$ – direct actualization, and

   (b) $\mathsf{F}_Z(A) = \iota_Z(A) \oplus_{\iota_Z(A)|_\rho} \mathsf{F}_Y(A|_\rho)$, where:
   $\mathsf{F}_Y(A|_\rho) = \iota_{Y1}(A|_\rho) \oplus_{\iota_{Y1}(A|_\rho)|_\nu} \mathsf{F}_X((A|_\rho)|_\nu)$.

   The problem here might possibly originate from the situation as in point 2 in fact 4.3.4 which was illustrated in example 4.3.3, i.e., that $\mathsf{F}'_Z(A)$ yields an algebra which does not satisfy the global guard $x \prec c$ satisfied by all algebras in $\mathsf{Mod}(\mathbf{P}[\mathbf{Z}]_\star)$. Showing equality of 1. and 2. we show, in particular, that such a situation does not occur.

   Actually it will suffice to show that $\iota_{Z1} = \iota_Z$, because the induced functor is constructed from F_X and ι (cf. def. 4.2.12, 4.2.13). This will, in particular, imply that $(\iota_{Z1}(A))|_{(\nu;\rho)} = ((\iota_Z(A))|_\rho)|_\nu$, for all algebras

$A \in \mathsf{Mod}(\mathbf{Z}_\star)$ – the fact which is sufficient for concluding the equality of two functors according to def. 4.2.13. The table below lists the definitions of (the relevant) ι's induced by ι_X :

	$a.\quad \iota_{Z1}(A)$	$b.\quad \iota_Z(A)$	$c.\quad \iota_Y(A\|_\rho)$
$\mathbf{S}: 1.$	$if\ s \in \mathbf{S}_X :$ $(\nu;\rho)(s)^{\iota_{Z1}(A)} = s^{\iota_X(A\|_{(\nu;\rho)})}$	$if\ s \in \mathbf{S}_Y :$ $\rho(s)^{\iota_Z(A)} = s^{\iota_Y(A\|_\rho)}$	$if\ s \in \mathbf{S}_X :$ $\nu(s)^{\iota_Y(A\|_\rho)} = s^{\iota_x((A\|_\rho)\|_\nu)}$
$2.$	$otherwise : s^{\iota_{Z1}(A)} = s^A$	$otherwise : s^{\iota_Z(A)} = s^A$	$otherwise : s^{\iota_Y(A\|_\rho)} = s^{A\|_\rho}$
$\Omega: 3.$	$if\ \omega \in \Omega_X :$ $(\nu;\rho)(\omega)^{\iota_{Z1}(A)} = \omega^{\iota_X(A\|_{(\nu;\rho)})}$	$if\ \omega \in \Omega_Y :$ $\rho(\omega)^{\iota_Z(A)} = \omega^{\iota_Y(A\|_\rho)}$	$if\ \omega \in \Omega_X :$ $\nu(\omega)^{\iota_Y(A\|_\rho)} = \omega^{\iota_X((A\|_\rho)\|_\nu)}$
$4.$	$if\ \omega \notin (\nu;\rho)[\Omega_X] : \omega(\overline{x})^{\iota_{Z1}(A)}$ $= \omega(\overline{x})^A,\ if\ all\ \overline{x} \in \|A\|$ $= \emptyset,\ otherwise$	$if\ \omega \notin \rho[\Omega_Y] : \omega(\overline{x})^{\iota_Z(A)}$ $= \omega(\overline{x})^A,\ if\ all\ \overline{x} \in \|A\|$ $= \emptyset,\ otherwise$	$if\ \omega \notin \nu[\Omega_X] : \omega(\overline{x})^{\iota_Y(A\|_\rho)}$ $= \omega(\overline{x})^{A\|_\rho},\ if\ all\ \overline{x} \in \|A\|_\rho\|$ $= \emptyset,\ otherwise$

For any symbol $s \in \Sigma(\mathbf{Z}_\star)$ we have three possibilities:

(a) it is not in the image of ρ (and hence not in the image of $(\nu;\rho)$ either), or

(b) it is in the image of ρ but not of $(\nu;\rho)$, or

(c) it is in the image of $(\nu;\rho)$.

To simplify the notation, we will ignore possible renaming, e.g., in case 2) we will assume that $s = \rho(s)$, and similarly, in case 3) that: $s = \rho(s) = \rho(\nu(s))$. Justification of equations, written $\stackrel{Rc}{=}$, refers to row R, column c in the table above and we use $\stackrel{Red}{=}$ if something follows from the reduct definition. We use functoriality of the reduct, i.e., the fact that $(A|_\rho)|_\nu = A|_{(\nu;\rho)}$, without mentioning it. Let us first consider the sorts.

(a) $s^{\iota_{Z1}(A)} \stackrel{2a}{=} s^A \stackrel{2b}{=} s^{\iota_Z(A)}$

(b) $s^{\iota_{Z1}(A)} \stackrel{2a}{=} s^A \stackrel{Red}{=} s^{A|_\rho} \stackrel{2c}{=} s^{\iota_Y(A|_\rho)} \stackrel{2b}{=} s^{\iota_Z(A)}$

(c) $s^{\iota_{Z1}(A)} \stackrel{1a}{=} s^{\iota_X(A|_{(\nu;\rho)})} = s^{\iota_X((A|_\rho)|_\nu)} \stackrel{1c}{=} s^{\iota_Y(A|_\rho)} \stackrel{1b}{=} s^{\iota_Z(A)}$

So, operations:

(a) $\omega^{\iota_{Z1}(A)}(\overline{x}) \stackrel{4a}{=} \left\{ \begin{array}{ll} \omega^A(\overline{x}) & if\ \overline{x} \in |A| \\ \emptyset & otherwise \end{array} \right\} \stackrel{4b}{=} \omega^{\iota_Z(A)}(\overline{x}).$

(b) By 4a. we have $\omega^{\iota_{Z1}(A)}$ as in the previous point. Since ω is not in the image of ν, we have $\omega^{\iota_Z(A)} \stackrel{3b}{=} \omega^{\iota_Y(A|_\rho)}(\overline{x}) \stackrel{4c}{=} \left\{ \begin{array}{ll} \omega^{A|_\rho}(\overline{x}) & if\ \overline{x} \in |A|_\rho| \\ \emptyset & otherwise \end{array} \right.$. But for all $\overline{x} : \overline{x} \in |A| \iff \overline{x} \in |A|_\rho|$, since ω, and hence all its sorts, are in the image of ρ. Then also $\omega^A(\overline{x}) = \omega^{A|_\rho}(\overline{x})$ which proves the equality $\omega^{\iota_{Z1}(A)} = \omega^{\iota_Z(A)}$ in this case.

(c) $\omega^{\iota_{Z1}(A)} \stackrel{3a}{=} \omega^{\iota_X(A|_{(\nu;\rho)})} = \omega^{\iota_X((A|_\nu)|_\rho)} \stackrel{3c}{=} \omega^{\iota_Y(A|_\rho)} \stackrel{3b}{=} \omega^{\iota_Z(A)}$

$\square$

4.3.3 Horizontal composition

Definition 4.3.7 *For PDTs* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ *and* $(\mu', \mathbf{P}[\mathbf{X}]_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, \delta')$*, we define their horizontal composition to be* $((\mu; \mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, (\delta; \delta'))$*.*

Proposition 4.3.8 *The composition as defined in 4.3.7 is (isomorphic to) a PDT. (In the sense that there exists a* $\mathbf{W}[\mathbf{P}[\mathbf{X}]]'_\star \simeq \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$ *such that:* $((\mu; \mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]'_\star, (\delta; \delta'))$ *is a PDT).*

Proof. The first 4 points of definition 4.1.15 are trivially satisfied. We have to verify point 5.

5b If, for some $\star : \mu'(\mu(\star)) \neq \delta'(\delta(\star))$ then either:

1. $\mu(\star) \neq \delta(\star) \neq \star$ or
2. $\mu'(\star) \neq \delta'(\star)$.

Case 1: By 5b of definition 4.1.15, $\mathbf{P}[\mathbf{X}]_\star$ contains the axiom $\mu(\star) \prec \delta(\star)$, we have 3 subcases:

- $c_1 = \mu(\star) \neq \star \neq \delta(\star) = c_2$, then this axiom is actually $c_1 \prec c_2$ and, by point 5a, $\mu'(c_1) \prec \mu'(c_2) \in \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$. But then also both δ' and μ' are identities on c_1, c_2, i.e., this last axiom is in fact $\mu'(\mu(\star)) \prec \delta'(\delta(\star))$.
- $\mu(\star) = \star$, then we must have that $\delta(\star) = \star$ since, if $\mu(\star) = \star$, then by the presence of $\mu(\star) \prec \delta(\star)$ in $\mathbf{P}[\mathbf{X}]_\star$, we would have to have also $\delta(\star) = \star$, i.e. this contradicts 1.
- $c = \mu(\star) \neq \delta(\star) = \star$, i.e. we have $c \prec \star \in \mathbf{P}[\mathbf{X}]_\star$. By point 5a, we have then $\mu'(\mu(\star)) = \mu'(c) \prec \mu'(\star) = \mu'(\delta(\star)) \in \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$, while by 5b, $\mu'(\star) \prec \delta'(\star)$ (or $\mu'(\star) = \delta'(\star)$. But then, adding the axiom $\mu'(\mu(\star)) \prec \delta'(\delta(\star))$ yields an isomorphic specification.

Case 2: Having verified case 1), we can now assume that $\mu(\star) = \delta(\star)$, since if $\mu(\star) = \delta(\star) = c \neq \star$, then μ', δ' are identities on c, which contradicts the assumption that $\mu'(\mu(\star)) \neq \delta'(\delta(\star))$, I.e., $\delta(\star) = \mu(\star) = \star$. But then by 5b, we have $\mu'(\star) \prec \delta'(\star) \in \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$, which is the required axiom $\mu'(\mu(\star)) \prec \delta'(\delta(\star))$.

5a Follows equally easily. Let $x_1 \prec \star_1, \ldots, x_m \prec \star_m, \overline{a} \to \overline{b}$ be a fully guarded axiom from $\mathbf{X}_\star$. Then, by 5a, $x_1 \prec \delta(\star_1), \ldots, x_m \prec \delta(\star_m), \overline{a} \to \overline{b}$ is in $\mathbf{P}[\mathbf{X}]_\star$. But then, by the same point:
$x_1 \prec \delta'(\delta(\star_1)), \ldots, x_m \prec \delta'(\delta(\star_m)), \overline{a} \to \overline{b}$ is in $\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$.

$\square$

4.3.4 Horizontal composition – semantics

As was the case with vertical composition, horizontal composition of PDTs gives us a more structured specification. According to proposition 4.3.8, composing horizontally two PDTs, we obtain a new PDT with the associated class of semantic functors. We show that semantics of a PDT obtained by a stepwise, horizontal composition of the PDT $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ and the PDT $\mathbf{P}' = (\mu', \mathbf{P}[\mathbf{X}]_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, \delta')$, which can be written as $\mathbf{P}; \mathbf{P}'$, is a refinement of the semantics of the respective composed PDT $((\mu; \mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, (\delta; \delta'))$ – the former, possessing more structure in the form of the intermediary stage $\mathbf{P}[\mathbf{X}]_\star$, may put additional restrictions on the admissible functors. Yet, composition of such functors will always yield a functor for the composed PDT. We show this fact first.

Proposition 4.3.9 *Given PDTs* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ *and* $(\mu', \mathbf{P}[\mathbf{X}]_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, \delta')$, *with the semantic functors:*

$$\mathsf{F}_{\mathbf{X}} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$$

and

$$\mathsf{F}_{\mathbf{P}[\mathbf{X}]} : \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \to \mathsf{Mod}(\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star)$$

The composition:

$$\mathsf{F}_{\mathbf{X}}; \mathsf{F}_{\mathbf{P}[\mathbf{X}]} : \mathbf{X}_\star \to \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$$

, is a semantic functor for: $((\mu; \mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, (\delta; \delta'))$.

The proposition means that all the loops in the following diagram commute:

$$\begin{tikzcd}
\mathsf{Mod}(\mathbf{X}_\star) \arrow[d, "\iota_{\mathbf{X}}"] \arrow[dr, "\mathsf{F}_{\mathbf{X}}"] \arrow[drr, bend left, "\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}"] \arrow[dd, bend right, "\iota'"'] & & \\
\mathsf{Mod}(\mathbf{X}_-) \arrow[d, dotted, "\iota_{\mathbf{P}[\mathbf{X}]}|_\mu"] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[l, "|_\mu"'] \arrow[d, "\iota_{\mathbf{P}[\mathbf{X}]}"] \arrow[dr, "\mathsf{F}_{\mathbf{P}[\mathbf{X}]}"] & \\
\mathsf{Mod}(\mathbf{X}_-) & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_-) \arrow[l, dotted, "|_\mu"'] & \mathsf{Mod}(\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star) \arrow[l, "|_{\mu'}"'] \arrow[ll, bend left, "|_{\mu;\mu'}"]
\end{tikzcd}$$

Proof. First notice that, by fact 4.1.18, μ is also a specification morphism $\mathbf{X}_- \to \mathbf{P}[\mathbf{X}]_-$, and so $|_\mu$ is also a functor $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]_-) \to \mathsf{Mod}(\mathbf{X}_-)$. Thus $|_{\mu;\mu'} = |_{\mu'}; |_\mu$ is a functor $\mathsf{Mod}(\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ as indicated on the diagram.

The functor $\iota' : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ corresponding to $\mathsf{F}_{\mathbf{X}}; \mathsf{F}_{\mathbf{P}[\mathbf{X}]}$ is defined by $\iota' = \iota_{\mathbf{X}}; (\iota_{\mathbf{P}[\mathbf{X}]}|_\mu)$, i.e., $\iota'(A)$ is given by:

- $s^{\iota'(A)} = s^{\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A)}$, for sorts $s \in \Sigma(\mathbf{X}_\star)$

- $\omega^{\iota'(A)}(\overline{x}) = \omega^{\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A)}(\overline{x})$, for operations $\omega \in \Sigma_{-}(\mathbf{X}_{\star})$
- $\star_s^{\iota'(A)} = (\mu;\mu')(\star_s)^{\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A)}$.

Thus, the outermost triangle commutes, i.e., for any $A \in \mathsf{Mod}(\mathbf{X}_{\star})$: $(\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A))|_{\mu;\mu'} = \iota'(A)$:

- sorts: $s^{(\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A))|_{\mu;\mu'}} = s^{\iota'(A)}$ since $\mu;\mu'(s) = s$
- operations: $\omega^{(\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A))|_{\mu;\mu'}} = \omega^{\iota'(A)}$ since $\mu;\mu'(\omega) = \omega$
- subsorts: $\star_s^{(\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A))|_{\mu;\mu'}} = (\mu;\mu')(\star_s)^{\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A)} = \star_s^{\iota'(A)}$

The tight monomorphism $\iota'_A : A \to \iota'(A)$ is defined $\forall A \in \mathsf{Mod}(\mathbf{X}_{\star})$ by:

$$\iota'_A = \iota_A; (\iota_{P[A]}|_{\mu})$$

, where ι_A and $\iota_{P[A]}$ are tight monomorphisms associated with $\iota_{\mathbf{X}}$ (and A) and $\iota_{\mathbf{P}[\mathbf{X}]}$ (and $\mathsf{F}_{\mathbf{X}}(A)$), respectively. Since $\iota_{P[A]}$ is a tight monomorphism, then so is its reduct $\iota_{P[A]}|_{\mu'}$. Then, ι'_A, being a composition of two tight monomorphisms, is a tight monomorphism [51].

The conditions of fact 4.1.25 are satisfied, so we conclude that $\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}$ is indeed a semantic functor for the composed PDT. $\square$

Horizontal composition as a refinement of PDTs.

Again, horizontal composition gives more structure. According to Prop. 4.3.8, composing horizontally two PDTs, we obtain a new PDT with the associated class of semantic functors. However, the semantics of a PDT obtained by a stepwise, horizontal composition of PDTs $\mathbf{P}$ and $\mathbf{P}'$ is a refinement of the semantics of the respective composed PDT $\mathbf{P};\mathbf{P}'$.

The following example illustrates that horizontal composition, introducing an intermediary parameter, can actually be a strict refinement of the composed PDT, i.e., that some functors admissible as a semantics for the composed PDT may no longer be obtained as a composition of the semantic functors for the component PDTs.

Example 4.3.10 *The following PDT $\mathbf{P} = (\mu, \mathbf{X}_{\star}, \mathbf{P}[\mathbf{X}]_{\star}, \delta)$ requires extension of the parameter algebra A with a new function f and allows extending A's carrier with new elements (one of which may be d).*

$$\boxed{\begin{array}{ll} \multicolumn{2}{l}{\mathbf{X}_{\star} =} \\ \mathbf{S}: & El \\ S^{\star}: & \star :\to El \\ \Gamma: & x \prec \star \end{array}} \xrightarrow[\delta(\star)=\star]{\mu(\star)=ok} \boxed{\begin{array}{llccc} \multicolumn{5}{l}{\mathbf{P}[\mathbf{X}]_{\star} =} \\ \mathbf{S}': & El & & & \\ \Omega': & d: & & \to El & \\ & f: & El & \to El & \\ S^{\star\prime}: & \star, ok: & & \to El & \\ \Phi': & 1. & & & f(d) \doteq d \\ & 2. & x \prec \star & \to & f(x) \doteq f(x) \\ & 3. & & & ok \prec \star \\ \Gamma': & 4. & & & x \prec \star \end{array}}$$

Let the semantic functor $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ *send an* $A \in \mathsf{Mod}(\mathbf{X}_\star)$ *to* $\mathsf{F}(A)$ *given by:*

- $|\mathsf{F}(A)| = |A| \uplus d$, *i.e.* d *is a new element added to the carrier of* A,
- $f^{\mathsf{F}(A)}(x) = d$, *for all* $x \in |\mathsf{F}(A)|$,
- $ok^{\mathsf{F}(A)} = |A|$, *by the semantic functor requirement,*
- $\star^{\mathsf{F}(A)} = |\mathsf{F}(A)|$, *by default.*

Let's now introduce $\mathbf{W}[\mathbf{X}]_\star$ *as an intermediary parameter, i.e., we now have two PDTs* $\mathbf{P}' = (\mu', \mathbf{X}_\star, \mathbf{W}[\mathbf{X}]_\star, \delta')$ *and* $\mathbf{P}'' = (\mu'', \mathbf{W}[\mathbf{X}]_\star, \mathbf{P}[\mathbf{W}[\mathbf{X}]]_\star, \delta'')$*:*

$$\mathbf{X}_\star \xrightarrow[\delta'(\star)=\star]{\mu'(\star)=\star} \boxed{\begin{array}{lllll} \mathbf{W}[\mathbf{X}]_\star = & & & & \\ \;\mathbf{S}'' : & El & & & \\ \;\Omega'' : & \;\; f: & El \to El & & \\ S^{\star\prime\prime} : & \;\; \star: & \;\to El & & \\ \;\Phi'' : & 2. & x \prec \star & \to & f(x) \doteq f(x) \\ \;\Gamma'' : & 4. & & & x \prec \star_{EL} \end{array}} \xrightarrow[\delta''(\star)=\star]{\mu''(\star)=ok} \begin{array}{c} \mathbf{P}[\mathbf{X}]_\star \\ = \\ \mathbf{P}[\mathbf{W}[\mathbf{X}]]_\star \end{array}$$

Obviously, we have that $\mathbf{P} = ((\mu';\mu''), \mathbf{X}_\star, \mathbf{P}[\mathbf{W}[\mathbf{X}]]_\star, (\delta';\delta''))$. *But the refinement* $\mathbf{P} \rightsquigarrow \mathbf{P}';\mathbf{P}''$ *is strict – e.g., the functor* F *for the former cannot be obtained by composing any two functors for the latter two.*

For any semantic functor $\mathsf{F}' : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{W}[\mathbf{X}]_\star)$ *can't extend the carrier of any* $A \in \mathsf{Mod}(\mathbf{X}_\star)$, *but merely adds a deterministic function* f. *Furthermore, any semantic functor* $\mathsf{F}'' : \mathsf{Mod}(\mathbf{W}[\mathbf{X}]_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{W}[\mathbf{X}]]_\star)$ *may add a new element* d *to the carrier of a parameter algebra* $B \in \mathsf{Mod}(\mathbf{W}[\mathbf{X}]_\star)$ *and force* $f(d) \doteq d$. *However,* F'' *has to "preserve" the parameter algebra* B, *i.e.,* B *must be a tight subalgebra of* $\mathsf{F}''(\mathsf{B})$. *This means that function* $f^{\mathsf{F}''(B)}$ *applied to the elements from the carrier of* B *(i.e., from* $ok^{\mathsf{F}''(B)}$*) has to return elements from the same carrier. If* d *is a new element, it will never be "reachable by* f*" from these old elements. This illustrates the impossibility of obtaining the original functor* F *as a composition of any* F' *and* F''.

The following gives a more detailed and concrete example of the same idea of refining the structure of PDT by introducing an intermediary parameter.

Example 4.3.11 *We start with a specification of stacks from example 4.1.17 and refine it by introducing the intermediary parameter specification of constructed stacks. To do that we first refine the specification* $\mathbf{Stack}[\mathbf{El}]_\star \rightsquigarrow \mathbf{Stack}[\mathbf{El}]'_\star$, *by adding the subsort constant* ok_{stack}, *and error constants for stacks and elements:*

$$
\boxed{\begin{array}{ll}
\multicolumn{2}{l}{\textbf{spec El}_\star =}\\
\mathbf{S}: & El\\
S^\star: & \star_{El} :\to El\\
\Gamma: & x \prec \star_{EL}
\end{array}}
\xrightarrow[\delta(\star_{El}) = ok]{\mu(\star_{El}) = ok}
$$

$$
\boxed{\begin{array}{llrcl}
\multicolumn{5}{l}{\textbf{spec Stack[El]'}_\star =}\\
\mathbf{S}': & \multicolumn{4}{l}{Stack, El}\\
\Omega': & empty: & & \to & Stack\\
 & top: & Stack & \to & El\\
 & pop: & Stack & \to & Stack\\
 & push: & El \times Stack & \to & Stack\\
 & errStack & & \to & Stack\\
 & errEl & & \to & El\\
S^{\star\prime}: & \star_{El}, ok_{El}: & \to El & & \\
 & \star_{St}, ok_{St}: & \to Stack & & \\
\Phi': & 1. & & & empty \doteq empty\\
 & 2. & x \prec ok_{El}, s \prec \star_{St} & \to & top(push(x,s)) \doteq x\\
 & 3. & x \prec ok_{El}, s \prec \star_{St} & \to & pop(push(x,s)) \doteq s\\
 & 4. & & & pop(empty) \doteq errStack\\
 & 5. & & & top(empty) \doteq errEl\\
\Gamma': & 6. & & & x \prec \star_{EL}\\
 & 7. & & & s \prec \star_{St}
\end{array}}
$$

The intermediary parameter $\textbf{ConstrStack[El]}_\star$ *contains only the intended constructors for stacks::*

$$
\begin{array}{c}
\textbf{El}_\star\\
\Big\downarrow{\scriptstyle \mu'(\star_{El}) = \star_{El},\ \delta'(\star_{El}) = \star_{El}}
\end{array}
$$

$$
\boxed{\begin{array}{llrcl}
\multicolumn{5}{l}{\textbf{spec ConstrStack[El]}_\star =}\\
\mathbf{S}': & \multicolumn{4}{l}{Stack, El}\\
\Omega': & empty: & & \to & Stack\\
 & push: & El \times Stack & \to & Stack\\
S^{\star\prime}: & \star_{El}: & \to El & & \\
 & \star_{St}: & \to Stack & & \\
\Phi': & 1. & \multicolumn{3}{l}{empty \doteq empty}\\
\Gamma': & 6. & x \prec \star_{EL} & & \\
 & 7. & s \prec \star_{St} & &
\end{array}}
\xrightarrow[\mu''(\star_{St}) = ok_{St},\ \delta''(\star_{St}) = ok_{St}]{\mu''(\star_{El}) = ok_{El},\ \delta''(\star_{El}) = ok_{El}}
\textbf{Stack[El]}'_\star
$$

Let the functors: $\mathsf{F}' : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\textbf{ConstrStack[El]}_\star)$ *and* $\mathsf{F}'' : \mathsf{Mod}(\textbf{ConstrStack[El]}_\star) \to \mathsf{Mod}(\textbf{Stack[El]}'_\star)$ *be semantic functors for the* *respective PDTs. The effect of the intermediary parameter is that, in any algebra* $C = \mathsf{F}''(\mathsf{F}'(A))$, *both* $empty^C$ *and all stacks obtained by* $push^C$ *must belong to* ok^C_{St}.

This reflects the more structured design which need not apply to a semantic functor of the original PDT $(\mu, \mathbf{El}_\star, \mathbf{Stack}[\mathbf{El}]'_\star, \delta)$, *where empty could be any element of the sort Stack, possibly errStack which could (and should) be outside* ok_{St}.

The specification $\mathbf{Stack}[\mathbf{El}]'_\star$ *does not say anything about errStack or errEl but these can be new elements added to the carrier of* $B = \mathsf{F}'(A)$. *(This effect could be forced by the specification by adding the axioms:* $errStack \prec ok_{St} \rightarrow$, *and* $errEl \prec ok_{El} \rightarrow$. *In general, the new error elements may be treated as illustrated in chapter 3.)*

The classical concept of horizontal composition states that: if $\mathbf{X_1} \rightsquigarrow \mathbf{X_2}$ and $\mathbf{P_1}[\mathbf{X_1}] \rightsquigarrow \mathbf{P_2}[\mathbf{X_1}]$ then $\mathbf{P_1}[\mathbf{X_1}] \rightsquigarrow \mathbf{P_2}[\mathbf{X_2}]$, where all $\rightsquigarrow$ represent model class inclusions (in the opposite direction). This means that a functor for $\mathbf{P_2}[\mathbf{X_2}]$ may have a source $\mathsf{Mod}(\mathbf{X_2}) \subset \mathsf{Mod}(\mathbf{X_1})$, which makes it too specific to be used for obtaining a semantic functor for the original PDT with the parameter $\mathbf{X_1}$. Yet, this fact of "implementation commuting with parameterization" allows to perform independent refinements on various components ensuring that their composition will yield an implementation of the composition of the original components.

Although this property does not reflect our notion of refinement, it still obtains in our setting. If $\mathbf{P_1}[\mathbf{X_1}]_\star \rightsquigarrow \mathbf{P_2}[\mathbf{X_1}]_\star$ then, as remarked in section 4.3.2, we obtain the refinement of the respective PDTs. If, in addition, we have $\mathbf{X_1}_\star \rightsquigarrow \mathbf{X_2}_\star$, then any semantic functor for $(\mu_{21}, \mathbf{X_2}_\star, \mathbf{P_2}[\mathbf{X_2}]_\star)$ is a restriction of the semantics of $(\mu_1, \mathbf{X_1}_\star, \mathbf{P_1}[\mathbf{X_1}]_\star)$, that is, an implementation in the classical sense in that the source and target categories of the semantic functors for the former are subcategories of, respectively, source and target categories of the semantic functors for the latter.

$$\begin{tikzcd}
 & \mathsf{Mod}(\mathbf{P_1}[\mathbf{X_1}]_\star) & \\
\mathsf{Mod}(\mathbf{X_1}_\star) \arrow[ur, "\mathsf{F}_1"] \arrow[rr, "\mathsf{F}_{21}"'] & & \mathsf{Mod}(\mathbf{P_2}[\mathbf{X_1}]_\star) \arrow[ul, hook, dotted, "\mathsf{i}"'] \\
 & \mathsf{Mod}(\mathbf{P_1}[\mathbf{X_2}]_\star) \arrow[uu, hook, "|_{\nu_1}"'] & \\
\mathsf{Mod}(\mathbf{X_2}_\star) \arrow[uu, hook, "|_{\nu}"'] \arrow[ur, "\mathsf{F}'_1"] \arrow[rr, "\mathsf{F}'_{21}"] & & \mathsf{Mod}(\mathbf{P_2}[\mathbf{X_2}]_\star) \arrow[uu, hook, "|_{\nu_2}"']
\end{tikzcd}$$

The diagram illustrates this situation where, given an app $\nu : \mathbf{X_1}_\star \rightarrow \mathbf{X_2}_\star$ such that $|_\nu : \mathsf{Mod}(\mathbf{X_2}_\star) \rightarrow \mathsf{Mod}(\mathbf{X_1}_\star)$ is an inclusion, the $|_{\nu_2}$, obtained by pullback, is an inclusion, too.

4.4 Refinement

We now formalize the concept of refinement of PDT. As we have emphasized, it amounts not only to the simple model class inclusion but, primarily, to introduction of additional *structure* on the PDTs. The following definition captures the general concept

Definition 4.4.1 *A PDT:*

$$\mathbf{P}' = (\mu', \mathbf{X}'_\star, \mathbf{P}[\mathbf{X}]'_\star, \delta')$$

refines a PDT:

$$\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$$

$\mathbf{P} \rightsquigarrow \mathbf{P}'$*, if there exist two functors, the left and the right refinement functor:*

$$\mathsf{R^L}_{\mathbf{X}} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}'_\star) \text{ and } \mathsf{R^R}_{\mathbf{P}[\mathbf{X}]} : \mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$$

such that for any semantic functor F' *for* $\mathbf{P}'$*, the functor* $\mathsf{R^L}_{\mathbf{X}}; \mathsf{F}'; \mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}$ *is a semantic functor for* $\mathbf{P}$*.*

The following diagram illustrates the requirement:

$$\begin{tikzcd}
 & & \mathsf{Mod}(\mathbf{X}_\star) \arrow[dll, "\mathsf{R^L}_{\mathbf{X}}"'] \arrow[d, "\iota"'] \arrow[drr, dotted, "\mathsf{R^L}_{\mathbf{X}};\mathsf{F}';\mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}"] \arrow[dr, phantom, "1."] & & \\
\mathsf{Mod}(\mathbf{X}'_\star) \arrow[d, "\iota'"'] \arrow[drrr, "\mathsf{F}'"] \arrow[dr, phantom, "2."] & & \mathsf{Mod}(\mathbf{X}_-) & & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[ll, "|_\mu"'] \\
\mathsf{Mod}(\mathbf{X}'_-) & & & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \arrow[lll, "|_{\mu'}"] \arrow[ur, "\mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}"'] &
\end{tikzcd}$$

The relation is trivially transitive, i.e., $\mathbf{P} \rightsquigarrow \mathbf{P}' \rightsquigarrow \mathbf{P}'' \to \mathbf{P} \rightsquigarrow \mathbf{P}''$.

The contravariance of $\mathsf{R^L}_{\mathbf{X}}$ and $\mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}$ on the parameter side in the refinement definition allows the refinement $\mathbf{P}'$ to have less semantical functors than the refined PDT $\mathbf{P}$, i.e. refinement of PDT's corresponds to a functorial subspace of semantic functors. It means that our refinement notion fits nicely with the traditional view of refinement as a subclass relation. Note however the difference between refinement of PDT's and refinement of flat specifications. Suppose that:
$\mathbf{X}_\star \rightsquigarrow \mathbf{X}'_\star$, i.e., $\mathsf{Mod}(\mathbf{X}_\star) \supseteq \mathsf{Mod}(\mathbf{X}'_\star)$. If this refinement is strict, i.e.:
$\mathsf{Mod}(\mathbf{X}_\star) \supset \mathsf{Mod}(\mathbf{X}'_\star)$, a semantic functor F' with source $\mathsf{Mod}(\mathbf{X}'_\star)$ could not, in general, be used in places were one assumes a functor with source $\mathsf{Mod}(\mathbf{X}_\star)$.

A simple example of refinement is when $\mathbf{P}[\mathbf{X}]_\star \rightsquigarrow \mathbf{P}[\mathbf{X}]'_\star$, i.e., when $\mathsf{R^L}_{\mathbf{X}}$ is identity and $\mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}$ is a model class inclusion $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \subseteq \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ as was the case of vertical composition in subsection 4.3.2.

Other examples were 4.3.10 and 4.3.11 in section 4.3.4, where both $\mathsf{R^L}_{\mathbf{X}}$ and $\mathsf{R^R}_{\mathbf{P[X]}}$ were identities but where intermediary parameter forced additional requirements which did not (necessarily) follow from the original, refined PDT.

Finally, we give an example showing yet another case of refinement by adding structure, where the formal parameter of a PDT is itself refined to a PDT.

Example 4.4.2 *Let* $\mathbf{P} = (\mu, \mathbf{Set}_\star, \sqcup[\mathbf{Set}]_\star, \delta)$ *be the following PDT which extends a (deterministic) specification of sets with a nondeterministic choice operation.*

$$\boxed{\begin{array}{rrrcl}
\multicolumn{5}{l}{\mathbf{spec}\ \mathbf{Set}_\star =} \\
\mathbf{S}: & \multicolumn{4}{l}{Set, El} \\
\Omega: & \emptyset: & & \to & Set \\
 & \dashv: & El \times Set & \to & Set \\
S^\star: & \star_{El}: & & \to & El \\
 & \star_{Set}: & & \to & Set \\
\Phi: & 1. & x \dashv (y \dashv S) & \doteq & y \dashv (x \dashv S) \\
 & 2. & x \dashv (x \dashv S) & \doteq & x \dashv S \\
\Gamma: & 3. & x & \prec & \star_{EL} \\
 & 4. & S & \prec & \star_{Set}
\end{array}}$$

$\mu(\star_{El}) = \delta(\star_{El}) = ok_{El}$

$\mu(\star_{Set}) = \delta(\star_{Set}) = \star_{Set}$

$$\boxed{\begin{array}{rrrcl}
\multicolumn{5}{l}{\sqcup[\mathbf{Set}]_\star =} \\
\mathbf{S}: & \multicolumn{4}{l}{Set, El} \\
\Omega: & \emptyset: & & \to & Set \\
 & \dashv: & El \times Set & \to & Set \\
 & \sqcup: & Set & \to & El \\
S^\star: & ok, \star_{El}: & & \to & El \\
 & \star_{Set}: & & \to & Set \\
\Phi: & 1. & x \prec ok, y \prec ok & \to & x \dashv (y \dashv S) \doteq y \dashv (x \dashv S) \\
 & 2. & x \prec ok & \to & x \dashv (x \dashv S) \doteq x \dashv S \\
 & 3. & x \prec ok & \to & \sqcup(x \dashv S) \prec ok \\
 & 4. & x \prec ok, z \prec \sqcup(x \dashv S) & \to & z \doteq x, z \prec \sqcup(S) \\
\Gamma: & 5. & & & x \prec \star_{EL} \\
 & 6. & & & S \prec \star_{Set}
\end{array}}$$

We admit here **extending carrier** El, $\mu(\star_{El}) = ok$, *which is motivated by the possible need of a new, "error" element to be returned by* $\sqcup(\emptyset)$.

A possible semantic functor F *may send a* $\mathbf{Set}_\star$ *algebra* A *on the algebra* $\mathsf{F}(A)$ *where, for any nonempty set* S, *the operation* $\sqcup^{\mathsf{F}(A)}(S)$ *returns all the elements of the set* S. $\sqcup^{\mathsf{F}(A)}(\emptyset)$ *may return a new, "error" element* $\perp$, *added to the carrier of* A. *Adding this element to a set,* $\perp \dashv^{\mathsf{F}(A)} S$ *may then result in the empty set* $\emptyset^{\mathsf{F}(A)}$.

Obviously, the specification $\mathbf{Set}_\star$ *can be naturally parameterized by elements, i.e., we "extract" from it a parameter (sub)specification. We obtain then:* $\mathbf{P}' = (\mu', \mathbf{El}_\star, \mathbf{Set}[\mathbf{El}]_\star, \delta')$*, where* $\mathbf{El}_\star$ *contains merely the sort El and the global guard* $x \prec \star_{El}$*, while* $\mathbf{Set}[\mathbf{El}]_\star$ *is exactly the same as* $\mathbf{Set}_\star$*.* μ' *and* δ' *are identities on* $\star_{El}$*.*

The point is now that the composed PDT $\mathbf{P}';\mathbf{P}$ *is a refinement of* $\mathbf{P}$ *according to definition 4.4.1.*

$$\begin{tikzcd}
\mathsf{Mod}(\mathbf{El}_\star) \arrow[d, dotted] \arrow[dr, "\mathsf{F}'"] & & & \mathsf{Mod}(\mathbf{Set}_\star) \arrow[lll, "|_{\mu'} \;=\; \mathsf{R^L}_{\mathbf{Set}}"'] \arrow[dll, dotted, dash, "id"'] \arrow[d, dotted] \arrow[dr, dotted, "\mathsf{G}"] & \\
\mathsf{Mod}(\mathbf{El}_-) & \mathsf{Mod}(\mathbf{Set}[\mathbf{El}]_\star) \arrow[l, "|_{\mu'}"'] \arrow[d, dotted] \arrow[dr, "\mathsf{F}"] & & \mathsf{Mod}(\mathbf{Set}_-) & \mathsf{Mod}(\sqcup[\mathbf{Set}]_\star) \arrow[l, "|_{\mu}"'] \arrow[dll, dotted, dash, "id \;=\; \mathsf{R^R}_{\sqcup[\mathbf{Set}]}"] \\
& \mathsf{Mod}(\mathbf{Set}[\mathbf{El}]_-) & \mathsf{Mod}(\sqcup[\mathbf{Set}[\mathbf{El}]]_\star) \arrow[l, "|_{\mu}"'] & &
\end{tikzcd}$$

Since $\mu'(\star_{El}) = \star_{El}$ *the* $|_{\mu'}$ *reduct will, actually, return an* $\mathbf{El}_\star$*-algebra (and not only an* $\mathbf{El}_-$*-algebra).*

Now, given F' *and* F*, the functor* G *can be trivially chosen to be* $|'_{\mu};\mathsf{F}';\mathsf{F};id$*. I.e.* $\mathbf{P} \rightsquigarrow \mathbf{P}';\mathbf{P}$*, where* $\mathsf{R^L}_{\mathbf{Set}} = |_{\mu'}$ *and* $\mathsf{R^R}_{\sqcup[\mathbf{Set}]} = id$*, the identity.*

However the refinement consists in requiring a more structured data type, which consists of building first an algebra of sets over a given algebra of elements, and then an algebra with choice (i.e., composing two functors $\mathsf{F}';\mathsf{F}$*). In this sense, it is reasonable to call* $\mathbf{P}';\mathbf{P}$ *a refinement of* $\mathbf{P}$*.*

Moreover, the functor F' *will not, in general, be surjective on the objects, i.e., it may choose only a subclass of all* $\mathbf{Set}[\mathbf{El}]_\star$ *algebras. In this case, the application of the composition to all models of* $\mathbf{El}_\star$*,* $\mathsf{F}';\mathsf{F}(\mathsf{Mod}(\mathbf{El}_\star))$ *may result in fewer* $\sqcup[\mathbf{Set}[\mathbf{El}]]_\star$ *algebras than* $\mathsf{F}(\mathsf{Mod}(\mathbf{Set}[\mathbf{El}]_\star))$*, which is another reason for viewing this composition as a refinement of the original PDT.*

There is yet another possibility of viewing the above as an example of refinement. Suppose that we have implemented the PDT $(\mu, \mathbf{Set}_\star, \sqcup[\mathbf{Set}]_\star, \delta)$, i.e., we have a functor G. Then, having implemented also a functor F', we can compose it with G. Since F' will not, typically, be surjective on the objects, this composition will, in general, yield a smaller subclass of $\mathsf{Mod}(\sqcup[\mathbf{Set}]_\star)$ than the image of G.

In any case, we can view the above process as a gradual development of a design for the flat specification $\sqcup[\mathbf{Set}]_\star$. In the first step, we extract from it the parameter $\mathbf{Set}_\star$ which prescribes a more specific, structured implementation. In the second step, we again extract the parameter $\mathbf{El}_\star$, requiring even more structure. Viewed in this way, our setting gives a concrete specialization of the general concept of "constructor implementations" from [47].

4.4.1 Parameter introduction

We now take a closer look at the introduction of the parameter $\mathbf{El}_\star$ in example 4.4.2:

$$\begin{tikzcd}
\mathsf{Mod}(\mathbf{El}_\star) \arrow[d, dotted] \arrow[dr, "\mathsf{F}'"] & & \mathsf{Mod}(\mathbf{Set}_\star) \arrow[ll, "|_{\mu'} = \mathsf{R^L}_{\mathbf{Set}}"'] \arrow[dl, dash, dotted, "\mathsf{id}{=}\mathsf{R^R}_{\mathbf{Set}}"] \\
\mathsf{Mod}(\mathbf{El}_-) & \mathsf{Mod}(\mathbf{Set}[\mathbf{El}]_\star) \arrow[l, "|_\mu"'] &
\end{tikzcd}$$

In this case μ was a specification morphism, hence we can view the parameter introduction as a refinement of the "PDT" $\mathsf{Mod}(\mathbf{Set}_\star)$ where $\mathsf{R^L}_{\mathbf{Set}} = |_{\mu'}$ and $\mathsf{R^R}_{\mathbf{Set}} = id$. But as noted in fact 4.1.18 in general the parameterization morphism may not be a specification morphism hence $|_\mu$ could not be chosen as the corresponding R. We now illustrate that parameter introduction could, nevertheless, always be seen as a refinement.

Fact 4.4.3 *Given a flat specification* $\mathbf{P}$ *we can view it as a PDT, parameterized by the empty model category, since the empty category,* $\emptyset$ *is initial in* $\mathbf{Cat}$.

$$\emptyset \xrightarrow{\ !\ } \mathsf{Mod}(\mathbf{P}[\emptyset]_\star)$$

Hence we can view any PDT as a refinement of a flat specification in the following way:

Fact 4.4.4 *Any PDT,* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ *is a refinement of the flat specification* $\mathbf{P}[\mathbf{X}]_\star$, *where* $\mathsf{R^L} = !$ *and* $\mathsf{R^R} = \mathsf{id}$ *i.e. we have the following diagram:*

$$\begin{tikzcd}
\emptyset \arrow[r, "!"] \arrow[d, "! = \mathsf{R^L}_\emptyset"] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \\
\mathsf{Mod}(\mathbf{X}_\star) \arrow[r, "\mathsf{F}"] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[u, "\mathsf{id} = \mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}"]
\end{tikzcd}$$

Hence we can view the PDT: $\mu : \mathbf{Set}_\star \to \sqcup[\mathbf{Set}]_\star$ as a refinement of the flat specification $\sqcup[\mathbf{Set}]_\star$ as illustrated in the following diagram:

$$\begin{tikzcd}
\emptyset \arrow[r, "!"] \arrow[d, "! = \mathsf{R^L}_\emptyset"] & \mathsf{Mod}(\sqcup[\mathbf{Set}]_\star) \\
\mathsf{Mod}(\mathbf{Set}_\star) \arrow[r, "\mathsf{F}"] & \mathsf{Mod}(\sqcup[\mathbf{Set}]_\star) \arrow[u, "\mathsf{id} = \mathsf{R^R}_{\sqcup[\mathbf{Set}]}"]
\end{tikzcd}$$

4.4.2 Composition of refinements

We will now illustrate that refinements can be both vertical and horizontal composed.

Vertical composition of refinements

Fact 4.4.5 *Given a PDT* $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$*, and it's refinement* $\mathbf{P}' = (\mu', \mathbf{X}'_\star, \mathbf{P}[\mathbf{X}]'_\star, \delta')$*, whit corresponding refinement* $\mathbf{P}'' = (\mu'', \mathbf{X}''_\star, \mathbf{P}[\mathbf{X}]''_\star, \delta'')$*, i.e. given the refinements in the following diagram:*

$$\begin{tikzcd}
\mathsf{Mod}(\mathbf{X}_\star) \arrow[r, "\mathsf{F}"] \arrow[d, "\mathsf{R^L}_{\mathbf{X}}"'] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \\
\mathsf{Mod}(\mathbf{X}'_\star) \arrow[r, "\mathsf{F}'"] \arrow[d, "\mathsf{R^L}_{\mathbf{X}'}"'] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \arrow[u, "\mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}"'] \\
\mathsf{Mod}(\mathbf{X}''_\star) \arrow[r, "\mathsf{F}''"] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]''_\star) \arrow[u, "\mathsf{R^R}_{\mathbf{P}[\mathbf{X}]'}"']
\end{tikzcd}$$

we have that: $\mathsf{R^L}_{\mathbf{X}}; \mathsf{R^L}_{\mathbf{X}'}$ *and* $\mathsf{R^R}_{\mathbf{P}[\mathbf{X}]'}; \mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}$ *is a refinement of* $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$*, i.e. vertical composition ensures that refinements are transitive.*

Horizontal composition of refinements

Example 4.4.6 *On the other hand suppose that we have the PDT* $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$*, and it's refinement* $\mathbf{P}' = (\mu', \mathbf{X}'_\star, \mathbf{P}[\mathbf{X}]'_\star, \delta')$*, together with the PDT* $\mathbf{Q} = (\nu, \mathbf{P}[\mathbf{X}]_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, \overline{\delta})$ *with corresponding refinement* $\mathbf{Q}' = (\nu', \mathbf{Y}_\star, \mathbf{W}[\mathbf{Y}]'_\star, \overline{\delta'})$*. I.e. given the following diagrams:*

$$\begin{tikzcd}
\mathsf{Mod}(\mathbf{X}_\star) \arrow[r, "\mathsf{F}"] \arrow[d, "\mathsf{R^L}_{\mathbf{X}}"'] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \\
\mathsf{Mod}(\mathbf{X}'_\star) \arrow[r, "\mathsf{F}'"] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \arrow[u, "\mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[r, "\mathsf{G}"] \arrow[d, "\mathsf{R^L}_{\mathbf{P}[\mathbf{X}]}"'] & \mathsf{Mod}(\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star) \\
\mathsf{Mod}(\mathbf{Y}_\star) \arrow[r, "\mathsf{G}'"] & \mathsf{Mod}(\mathbf{W}[\mathbf{Y}]'_\star) \arrow[u, "\mathsf{R^R}_{\mathbf{W}[\mathbf{P}[\mathbf{X}]]}"']
\end{tikzcd}$$

The two refinements of $\mathbf{P}$ *and* $\mathbf{Q}$ *can be composed horizontally in the following manner:*

$$
\begin{tikzcd}
\mathsf{Mod}(\mathbf{X}_\star) \arrow[rr, "\mathsf{F}"] \arrow[d, "\mathsf{R^L}_{\mathbf{X}}"'] & & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[rr, "\mathsf{G}"] \arrow[dr, "\mathsf{R^L}_{\mathbf{P}[\mathbf{X}]}"] & & \mathsf{Mod}(\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star) \\
\mathsf{Mod}(\mathbf{X}'_\star) \arrow[r, "\mathsf{F}'"] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \arrow[ur, "\mathsf{R^R}_{\mathbf{P}[\mathbf{X}]}"] & & \mathsf{Mod}(\mathbf{Y}_\star) \arrow[r, "\mathsf{G}'"] & \mathsf{Mod}(\mathbf{W}[\mathbf{Y}]'_\star) \arrow[u, "\mathsf{R^R}_{\mathbf{W}[\mathbf{P}[\mathbf{X}]]}"']
\end{tikzcd}
$$

The above example illustrates the following general fact:

Fact 4.4.7 *Suppose that two PDTs* $\mathbf{P}$ *and* $\mathbf{Q}$ *can be composed to a PDT* $\mathbf{P};\mathbf{Q}$ *and we are given PDTs* $\mathbf{P}'$ *and* $\mathbf{Q}'$*, such that:* $\mathbf{P} \rightsquigarrow \mathbf{P}'$ *and* $\mathbf{Q} \rightsquigarrow \mathbf{Q}'$*, then* $\mathbf{P};\mathbf{Q} \rightsquigarrow \mathbf{P}';\mathsf{R};\mathbf{Q}'$*, where* $\mathsf{R} = \mathsf{R^L};\mathsf{R^R}$ *of the specification that combines* $\mathbf{P}$ *and* $\mathbf{Q}$.

One immediate consequence of the facts; 4.4.4, 4.4.5 and 4.4.7 is that if we want to implement a specification we can first view it as a PDT and then implement the PDT by introducing new parameters, moreover the sub PDTs can be independently refined. Note that the final result of this process offers a model of the original specification. We illustrate this by taking a closer look at a possible implementation of the specification of sets with nondeterministic choice, $\sqcup\mathbf{Set}_\star$, from example 4.4.2.

Example 4.4.8 *Suppose that we start with the flat specification* $\sqcup\mathbf{Set}_\star$*, we first consider it as a specification parameterized by the empty category, we then refine the parameterized specification by introducing the parameter* $\mathbf{Set}_\star$ *and we make a further refinement step by parameterizing* $\mathbf{Set}_\star$ *by* $\mathbf{El}_\star$*. The refinements corresponds to the following diagram:*

$$
\begin{tikzcd}
 & \emptyset \arrow[r] \arrow[d, "\mathsf{R^L}_\emptyset"', "!"] \arrow[dr, phantom, "\rightsquigarrow"] & \mathsf{Mod}(\sqcup\mathbf{Set}[\emptyset]_\star) \\
\emptyset \arrow[r, "!"] \arrow[d, "\mathsf{R^L}_\emptyset"', "!"] \arrow[dr, phantom, "\rightsquigarrow"] & \mathsf{Mod}(\mathbf{Set}[\emptyset]_\star) \arrow[r, "\mathsf{F}"'] & \mathsf{Mod}(\sqcup[\mathbf{Set}]_\star) \arrow[u, "\mathsf{Id}", "\mathsf{R^R}_{\sqcup\mathbf{Set}[\emptyset]}"'] \\
\mathsf{Mod}(\mathbf{El}_\star) \arrow[r, "\mathsf{F}'"'] & \mathsf{Mod}(\mathbf{Set}[\mathbf{El}]_\star) \arrow[r, dotted, "\mathsf{F}"'] \arrow[u, "\mathsf{Id}", "\mathsf{R^R}_{\mathbf{Set}[\emptyset]}"'] & \mathsf{Mod}(\sqcup[\mathbf{Set}[\mathbf{El}]]_\star) \arrow[u, dotted, "\mathsf{Id}"]
\end{tikzcd}
$$

Of course we could have refined the PDT: $\mathbf{Set}[\mathbf{El}]_\star \to \sqcup[\mathbf{Set}[\mathbf{El}]]_\star$ *further, anyway, this example indicates how refinements of PDTs can be used to implement specifications by adding structure to the original specification.*

In this section we have illustrated some immediate consequences of the refinement definition. We have seen that one gradually can introduce new parameters, which correspond to identification and implementation of subprograms.

Moreover in a complex PDT could each sub PDT be refined independently, it means that one can develop each module of a program in isolation as long as it fulfills the refinement definition. We believe that a more detailed study of the refinement notion may result in a concrete methodology for implementations of PDTs, but this is left for a future work.

4.5 Concluding remarks

We have presented a framework for specifying parameterized data types. The syntax of PDTs is defined by a series of restrictions on the syntax of parameterized specifications, with the additional means for indicating the possibility of extending the carrier of the parameter algebras as well as the axioms of the parameter specification (to apply also to the "new" elements).

Semantics of PDTs is defined by a class of functors which satisfy a generalization of the classical persistency requirement – the parameter has to be a (tight) subalgebra of its image under the semantic functor. This generalization leads to a great flexibility in specifying PDTs which was illustrated by a series of examples.

We have re-stated and proved the counterparts of the theorems of vertical and horizontal composition of PDTs in our setting. An important concept emerging from these theorems concerns refinement of PDTs. We have given a general definition of such a refinement which is reflected in concrete examples primarily as introduction of additional structure into the specified PDT.

We view PDTs as design specifications which put requirements not only on the abstract (input-output) properties of the implementations but also on the actual structure of the implementation. In this way, our PDTs give a concrete realization of a more general concept of 'constructor specifications' from [47] which has recently been included into CASL [23] as 'architectural specifications'. Indeed, the suggested refinement of PDTs can be naturally seen as an example of program development based on constructor specifications where successive stages amount to splitting the original, loose specification into smaller pieces, according to the desired structure of the intended implementation.

Although we have presented the whole setting using the institution of multialgebras, it should be easy to recognize the generic aspect of our definitions and constructions. Entirely analogous extensions can be made on the top of many common specification frameworks, as long as they satisfies a few requirements: they are semi-exact institutions (admit amalgamation lemma); signatures contain symbols which in the model class are interpreted as unary predicates; signature morphisms respect this distinction, i.e. send such predicate symbols only on such symbols; the model classes are concrete categories where monomorphisms are injective. Although the list may seem rather restrictive, most commonly used institutions do satisfy the requirements. The results, in particular 4.2.14 hold not only for $\mathcal{MA}$, but also for institutions of total algebras (with predicates), partial algebras (with predicates) and membership algebras. In this way, we have quite a general way of applying our setting which vastly extends the specific context of order sorted algebras used in [42]. We would also maintain that our syntactic requirements are simpler than those introduced in the above paper.

A point which certainly requires a further study concerns reasoning about PDTs. We expect that addition of generic axiom schemata expressing closure of the parameter algebras (i.e., that operations applied to the "old" elements return "old" elements) will lead to a complete axiomatization but this issue needs to be investigated.

On the other hand, we would like to use PDTs for study and, perhaps, design of more specific structuring mechanisms at the level of implementations. We believe that the current work can provide a useful starting point for designing more detailed constructs, for instance, for architectural specifications in languages like CASL.

Conclusion

We have studied the concept of multialgebras as an algebraic specification formalism. In chapter 1 we gave the mathematical foundation for the thesis. The main result from this chapter is that multialgebras form an exact institutions, $\mathcal{MA}$, especially this means that the amalgamation lemma holds for multialgebras. This result is used to define the induced actualization functor semantics for parameterized datatypes in chapter 4. In chapter 1 we also constructed the actual institution embedding from the institutions of membership algebras and the institution of partial algebras to multialgebras. These results legitimate the title of the thesis since any thing that can be specified in one of the mentioned frameworks has an natural encoding within multialgebras. Embedding from many other algebraic specification formalisms can be easily envisaged, following the work of Mossakowski in [38]. In chapter 2 we gave two sound and complete reasoning systems for multialgebras, one Raisowa-Sikorski type of logic that should be well suited for implementation of a theorem prover, we also gave a Gentzen style logic that is more convenient for doing proofs by hand. In chapter 3 we illustrate how one can use nondeterminism and multialgebras as a powerful tool for partiality handling; by weakening the institution embedding from chapter 1 to institution transformation we also illustrate how one can reuse partial algebra specifications within the framework of multialgebras and develop the specifications to particular error recovery. In chapter 4 we defined the concept of parameterized datatype specification, PDT. We have given several results with respect to composition of PDT's and we have also introduced the notion of refinements of PDT's.

We hope that the work in the thesis illustrates the specification power of multialgebras and we propose that multialgebras migtht be used as a unifying framework for algebraic specifications. As a consequence of the institution embeddings mentioned above we are able to reuse a desired concept from a particular specification formalism and refine it within the institution of multialgebras, in the similar way we may use multialgebras to combine concepts from different specification formalisms. This means that we actually can reuse specifications from other algebraic specification formalisms, either to further refine them or else combine them with other specifications. One should observe that in many cases (e.g. $\mathcal{MEMB}$ and $\mathcal{PA}$) such reuse of specifications in $\mathcal{MA}$ is based on a simple translation without any elaborate coding.

Of course one has to pay the price for the generality offered by multialgebras. One major problem that is left for further work is to find syntactical restrictions that ensure the existence of canonical models for multialgebra specifications. A step in this direction is to describe the class of multialgebras that has initial models, we have summarized the present results concerning initial models in chapter 1, but the problem is still unsolved. It should be interesting to implement a theorem prover for the Raisowa-Sikorski logic given in chapter 2. Moreover we will try to develop a system for reasoning about PDT's. We also think that it will be fruitful to study implementation aspect of PDT's, and we hope to obtain a desirable software developing methodology for PDT's.

Bibliography

[1] Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories.* Cambridge University Press, 1994.

[2] Arnon Avron and Beata Konikowska. Decomposition proof systems for gödel-dummett logics. *Studia Logica*, 69:197–219, 2001.

[3] Michael Barr and Charles Wells. *Category Theory for Computing Science.* Prentice Hall, 1990.

[4] Gilles Bernot and Pascale Le Gall. Label algebras: a systematic use of terms. In Bidot and Choppy [7], pages 144–163.

[5] Marcin Białasik and Beata Konikowska. Reasoning with first-order nondeterministic specifications. Technical Report 830, Depeartment of Theoretical Informatics, Polish Academy of Sciences, 1997.

[6] Marcin Białasik and Beata Konikowska. Reasoning with first-order nondeterministic specifications. *Acta Informatica*, 36:357–403, 1999.

[7] Michel Bidot and Christine Choppy, editors. *Recent Trends in Data Type Spesification*, volume 655 of *Lecture Notes in Computer Science*. Springer, 1993.

[8] Peter Burmeister. Partial algebra - an introductory survey. *Algebra Universalis*, 15:306–358, 1982.

[9] Maura Cerioli. A lazy approach to partial algebras. In E. Astesiano, G. Reggio, and A Tarlecki, editors, *Recent Trends in Data Type Specification: 10th Workshop on Specification of Abstract Data Types-Selected Papers*, volume 906 of *Lecture Notes in Computer Science*. Springer, 1995.

[10] Maura Cerioli, Till Mossakowski, and Horst Reichel. From total equational to partial first order. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *on Algebraic Foundations of Systems Specification*, chapter 3. Springer, 1999.

[11] H. Ehrig, H.-J. Kreowski, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Parameter passing in algebraic specification languages. In *Proceedings of Workshop on Program Specification*, volume 134 of *LNCS*, 1982.

[12] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1*. Springer, 1985.

[13] Harald Ganzinger. Parametric specifications, parameter passing and optimizing implementations. Technical Report TUM-18110, Technical University of Munich, 1981.

[14] Joseph Goguen and Răzvan Diaconescu. An oxford survey of order-sorted algebra. *Journal of Mathematical Structures in Computer Science*, 4:363–392, 1994.

[15] Joseph A. Goguen. Abstract errors for abstract data types. In *IFIP Working Conference on the Formal Description of Programming Concepts*, volume 116, pages 89–103. North-Holland, 1978.

[16] Joseph A. Goguen. Order-sorted algebra i. Semantics and Theory of Computation Series 14, UCLA Computer Science Department, 1978.

[17] Joseph A. Goguen and R. M. Burstal. Some fundamental algebraic tools for the semantic of computation. part 1: Comma categories, colimits, signatures, and theories. *Theoretical Computer Science*, 31:175–209, 1984.

[18] Joseph A. Goguen and R. M. Burstal. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39:95–146, 1992.

[19] Joseph A. Goguen and Rod M. Burstall. Cat: a system for the structural elaboration of correct programs from structured specifications. Technical Report CSL-118, SRI International, 1980.

[20] Joseph A. Goguen and José Meseguer. Order-sorted algebra i: Equational deduction for multiple inheritance, overloading, exceptions, and partial operations. Technical Report SRI-CSL-89-10, Computer Science Laboratory SRI International, 1989.

[21] Magne Haveraaen and Eric G. Wagner. Guarded algebras and data type specification. Technical Report 108, Depeartment of Informatics, University of Bergen, 1995.

[22] Heinrich Hussmann. *Nondeterminism in Algebraic Specifications and Algebraic Programs*. Birkhäuser, 1993.

[23] CoFI. The Common Framework Initiative. *CASL – The Common Algebraic Specification Language*. http://www.brics.dk/Projects/CoFI/Documents/CASL.

[24] Rosa Jiménez, Fernando Orejas, and Hartmut Ehrig. Compositionality and compatibility of parameterization and parameter passing in specification languages. *Mathematical Structures in Computer Science*, 5(2):283–314, 1995.

[25] Beata Konikowska. Rasiowa-sikorski deduction systems in computer science applications. [to appear in a special issue of *Theoretical Computer Science*].

[26] Beata Konikowska. Rasiowa-sikorski deduction systems: a handy tool for computer science logic. In J. Fiadeiro, editor, *Recent Trends in Algebraic Specificagtion Techniques*, volume 1589 of *LNCS*. Springer, 1999.

[27] Yngve Lamo and Michał Walicki. Modeling partiality by nondeterminism - from abstract specifications to flexible error handling. Technical Report 178, Depeartment of Informatics, University of Bergen, 1999.

[28] Yngve Lamo and Michał Walicki. The instution of multialgebras. Technical Report 209, Department of Informatics, University of Bergen, 2000.

[29] Yngve Lamo and Michał Walicki. Specification of parameterized data types. Technical Report 210, Department of Informatics, University of Bergen, 2000.

[30] Yngve Lamo and Michał Walicki. Modeling partiality by nondeterminism. In N. Callaos, J. M. Pineda, and M. Sanchez, editors, *Proceedings of SCI/ISAS 2001*, volume I. Orlando, FL, 2001.

[31] Yngve Lamo and Michał Walicki. Specification of parameterized data types: Persistency revisited. *Nordic Journal of Computing*, 8:278–303, 2001.

[32] Yngve Lamo and Michał Walicki. Quantifier-free logic for multialgebraic theories. Technical Report 228, Depeartment of Informatics, University of Bergen, 2002.

[33] Yngve Lamo and Michał Walicki. Composition and refinement of specifications of parameterized data types. *Electronical Notes in Theoretical Computer Science*, 70.4, to appear.

[34] J. A. Makowsky. Why horn formulas matter in computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34:266–292, 1987.

[35] Alfio Martini and Uwe Wolter. A systematic study of mappings between institutions. In Francesco Parisi Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*, pages 300–315. Springer, 1998.

[36] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Logic colloquium'87*, pages 275–329. Elsevier Science Publisher B.V.(North-Holland), 1989.

[37] José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.

[38] Till Mossakowski. Equivalences among various logical frameworks of partial algebras. In H. K. Büning, editor, *Computer Science Logic*, volume 1092 of *Lecture Notes in Computer Science*, pages 403–433. Springer, 1996.

[39] Peter D. Mosses. The use of sorts in algebraic spesifications. In Bidot and Choppy [7], pages 66–91.

[40] Fernando Orejas. Structuring and modularity. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *on Algebraic Foundations of Systems Specification*, chapter 6. Springer, 1999.

[41] Aida Pliuškievičienė. Specialization of the use of axioms for deduction search in axiomatic theories with equality. *J. Soviet Math.*, 1, 1973.

[42] Axel Poigné. Another look at parameterization using algebras with subsorts. In *Proceedings of MFPC*, volume 176 of *LNCS*. Springer, 1984.

[43] Axel Poigné. Error handling for parameterized data types. In *Proceedings of 3rd WADT*, volume 116 of *LNCS*. Springer, 1984.

[44] H. Rasiowa and R. Sikorski. *The Mathematics of Metamatemathics*. PWN [Polish Scientific Publishers], 1963.

[45] Horst Reichel. *Initial Computability Algebraic Specifications and Partial Algebras*. Oxford Science Publications, 1987.

[46] Donald Sanella, Stefan Sokołowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: parameterisation revisited. *Acta Informatica*, 29:689–736, 1992.

[47] Donald Sanella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: implementations revisited. *Acta Informatica*, 25:233–281, 1988.

[48] Andrzej Tarlecki. Institutions: An abstract framework for formal specifications. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *on Algebraic Foundations of Systems Specification*, chapter 4. Springer, 1999.

[49] J.W. Thatcher, E.G. Wagner, and J.B. Wright. Data type specification: parameterization and the power of specification techniques. In *Proceedings of POPL*. ACM, 1979.

[50] Michał Walicki. *Algebraic Specification of Nondeterminism*. PhD thesis, Department of Informatics, University of Bergen, Norway, 1993.

[51] Michał Walicki and Marcin Białasik. Categories of relational structures. In Francesco Parisi Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*. Springer, 1998.

[52] Michał Walicki, Adis Hodzic, and Sigurd Meldal. Compositional homomorphisms of relational structures (modeled as multialgebras). In *Proceedings of 13e-th International Symposium on Fundamentals of Computing Theory*, Lecture Notes in Computer Science. Springer, 2001.

[53] Michał Walicki and Sigurd Meldal. A complete calculus for the multialgebraic and functional semantics of nondeterminism. *ACM TOPLAS*, 17(2), 1995.

[54] Michał Walicki and Sigurd Meldal. Multialgebras, power algebras and complete calculi of identities and inclusions. volume 906 of *Lecture Notes in Computer Science*. Springer, 1995.

[55] Michał Walicki and Sigurd Meldal. Algebraic approaches to nondeterminism - an overview. *ACM Computing Surveys*, 29, 1997.