11 Proof Systems for Structured Specifications and Their Refinements

Michel Bidoit[1], María Victoria Cengarle[2], and Rolf Hennicker[2]

[1] LSV, CNRS and ENS de Cachan, 61 avenue du Président Wilson, F-94235 Cachan Cedex, France
[2] Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstr. 67, D-80538 München, Germany

11.1 Introduction

Reasoning about specifications is one of the fundamental activities in the process of formal program development. This ranges from proving the consequences of a specification, during the prototyping or testing phase for a requirements specification, to proving the correctness of refinements (or implementations) of specifications. The main proof techniques for algebraic specifications have their origin in equational Horn logic and term rewriting. These proof methods have been well studied in the case of nonstructured specifications (see Chapters 9 and 10). For large systems of specifications built using the structuring operators of specification languages, relatively few proof techniques have been developed yet; for such proof systems, see [SB83,HST94, Wir91,Far92,Cen94,HWB97].

In this chapter we focus on proof systems designed particularly for modular specifications. The aim is to concentrate on the structuring concepts, while abstracting as much as possible from the particular logic and proof methods used for specification "in the small". Technically this is achieved by considering a kernel language of structured specifications defined over an arbitrary institution (see Chapter 4). Hence the proof methods presented can be instantiated by any kind of proof system chosen for a particular logic. The structuring operators of the kernel language are representative of a wide spectrum of specification languages (see Chapter 8). They include operators for the combination of specifications, and for hiding (export) and renaming specifications. The semantics of a specification expression is determined by the loose model-class semantics described in Chapter 5.

On this basis we study proof systems for verifying two kinds of statements. On the one hand, we are interested in proving that a sentence φ is a consequence (i.e., theorem) of a specification SP, denoted by $\mathrm{SP} \models \varphi$, which means that φ is valid in all models of SP. On the other hand, we are interested in proving refinement relations, written $\mathrm{SP}_1 \rightsquigarrow \mathrm{SP}_2$, between a more "abstract" specification SP_1 and a more "concrete" specification SP_2. The refinement relation is based on model class inclusion, i.e., it is a simplified version of the implementation definition given in Chapter 7.

After having introduced the syntax and semantics of the kernel specification language in Section 11.2, we first consider different kinds of proof systems for proving theorems of specifications (over the kernel language). In Section 11.3, noncompositional proof systems are studied, where the underlying idea is to compute, for a given specification, a flat, unstructured set of axioms and rules which (combined with some standard proof system for the chosen logic) may be used for deriving theorems of the specification. Here the most prominent representative is the normal-form approach based on the laws of the module algebra [BHK90]. The drawback of this method is that it does not respect the modular structure of specifications. Therefore we present in Section 11.4 a structured proof system whose derivations are performed in accordance with the modular structure of a specification. In this case the idea is to provide for each specification-building operator, Op, proof rules of the form

$$\frac{\mathrm{SP}_1 \vdash \varphi_1, \ldots, \mathrm{SP}_n \vdash \varphi_n}{\mathrm{Op}(\mathrm{SP}_1, \ldots, \mathrm{SP}_n) \vdash \varphi}$$

where the intended meaning is: if $\varphi_1, \ldots, \varphi_n$ are respectively theorems of $\mathrm{SP}_1, \ldots, \mathrm{SP}_n$, then φ is a theorem of $\mathrm{Op}(\mathrm{SP}_1, \ldots, \mathrm{SP}_n)$.

An important issue in program development is the proof of refinements. In Section 11.5 we present a proof system for deriving refinement relations $\mathrm{SP}_1 \rightsquigarrow \mathrm{SP}_2$, following the modular structure of the specification SP_1 to be implemented. The proof system for refinements can be built on top of any proof system for theorems of a specification.

In many specification languages, some parameterization concept is available in order to define specifications in a generic way. Based on the λ-calculus style of parameterization (see Chapter 8), we present in Section 11.6 an extension of the kernel language to include parameterization. We also extend the structured proof systems of the kernel language for proving theorems and refinements in the context of parameterized specifications.

For all proof systems considered so far, we provide soundness and (relative) completeness results. Up to Section 11.6 the proof systems are given for specifications over an arbitrary institution. In the remaining sections we consider the particular institution of many-sorted first-order logic with equality, in which we can define two further kinds of specification-building operators, namely the reachability and observability operators. The reachability operator increases the expressive power of the specification language by imposing reachability constraints (i.e., generation principles) on the models of a specification. Two observability operators are considered (see also Chapter 5) which are particularly useful in the context of refinements (see below). In Section 11.7 we first present noncompositional proof systems for specifications with reachability and observability operators. In both cases the idea is to introduce particular semiformal rules (the "infinitary induction" and "infinitary context induction" rules) which allow one to characterize the reachability and the observational equality of elements. In this way proof systems consisting of a set of (nonlogical) axioms and rules, adapted to the particular form of a specification expression, can be constructed. Again soundness and (relative) completeness results for the proof systems of the extended language are provided. We then show that a sound and complete structured proof system can also be defined for the extended specification language, provided we allow infinitary sentences (hence working in the institution of many-sorted infinitary first-order logic with equality).

Finally we show how the proof system for refinement studied in Section 11.5 can be extended to take into account the new specification-building operators, and we also consider observational refinements of specifications.

11.2 The kernel specification language

11.2.1 Basic concepts and notations

The basis of the kernel specification language to be introduced in the next subsection is the concept of an *institution* as defined in Chapter 4. Recall that an institution is a tuple of the form[1]

$$\mathrm{INS} = \langle \mathrm{Sign}, \mathrm{Sen}, \mathrm{Str}, (\models_\Sigma)_{\Sigma\in|\mathrm{Sign}|}\rangle$$

where Sign is a category of signatures, $\mathrm{Sen}\colon \mathrm{Sign} \to \mathrm{Set}$ and $\mathrm{Str}\colon \mathrm{Sign}^{op} \to \mathrm{Cat}$ are functors, and, for each signature Σ, $\models_\Sigma$ is a satisfaction relation such that the satisfaction condition holds. The elements of $\mathrm{Sen}(\Sigma)$ are called Σ-sentences and the objects of $\mathrm{Str}(\Sigma)$ are called Σ-structures. As in Chapter 4 we use the following notation.

Notation. For any signature morphism $\sigma\colon \Sigma \to \Sigma'$, the function $\mathrm{Sen}(\sigma)\colon \mathrm{Sen}(\Sigma) \to \mathrm{Sen}(\Sigma')$ (between sets of sentences) is, for simplicity, also denoted by σ, and the functor $\mathrm{Str}(\sigma)\colon \mathrm{Str}(\Sigma') \to \mathrm{Str}(\Sigma)$ is denoted by $\_|_\sigma$. Hence the satisfaction condition can be rephrased as $A' \models_{\Sigma'} \sigma(\varphi)$ if and only if $A'|_\sigma \models_\Sigma \varphi$ for any $\sigma\colon \Sigma \to \Sigma'$, $\varphi \in \mathrm{Sen}(\Sigma)$, and $A' \in |\mathrm{Str}(\Sigma')|$. The satisfaction relation is extended as usual to sets of Σ-sentences and classes of Σ-structures. For instance, if C is a class of Σ-structures and φ is a Σ-sentence, then $C \models \varphi$ if and only if $A \models \varphi$ for each $A \in C$.

Here and in the following sections, we assume a fixed, arbitrary institution INS is given.

Institutions provide a model-theoretic view of logical systems. However we still require the notion of a proof system which allows us to extract (soundly) the theorems of a theory presented over an institution. More precisely, given a presentation $\langle \Sigma, E\rangle$ (called a *basic specification* later on) with $\Sigma \in |\mathrm{Sign}|$ and $E \subseteq \mathrm{Sen}(\Sigma)$, we want to compute its associated theory, consisting of the

[1] In Chapter 4, the functor Str is called Mod. We reserve the use of this name for the collection of models associated with a specification as defined in Section 11.2.2.

sentences $\varphi \in \mathrm{Sen}(\Sigma)$ which are implied by the presentation, i.e., those φ for which, for all $A \in |\mathrm{Str}(\Sigma)|$, if $A \models_\Sigma E$ then $A \models_\Sigma \varphi$. By abuse of notation, we then also write $E \models_\Sigma \varphi$ where $\models_\Sigma$ denotes the consequence relation (see Chapter 4).

A proof-theoretic view of institutions is captured by the notions of *entailment relation* and *entailment system* as defined in Chapter 4. An entailment relation on $\mathrm{Sen}(\Sigma)$ is a relation $\vdash\ \subseteq \wp(\mathrm{Sen}(\Sigma)) \times \mathrm{Sen}(\Sigma)$ satisfying the reflexivity, transitivity, and weakening conditions. Moreover, an entailment system (w.r.t. the given institution INS) is a tuple of the form

$$\mathrm{ES} = \langle \mathrm{Sign}, \mathrm{Sen}, (\vdash_\Sigma)_{\Sigma \in |\mathrm{Sign}|} \rangle$$

such that, for each $\Sigma \in |\mathrm{Sign}|$, $\vdash_\Sigma$ is an entailment relation on $\mathrm{Sen}(\Sigma)$, preserved under translation of sentences by signature morphisms. (Intuitively this means that the entailment relations are constructed in a uniform way.) An entailment system is *sound (complete)* if $\vdash_\Sigma$ is sound (complete) w.r.t. the semantic consequence relation $\models_\Sigma$ for each $\Sigma \in |\mathrm{Sign}|$.

For dealing with proofs, we consider entailment systems which are generated by the axioms and rules of a proof system. A proof system for INS consists of a (uniformly defined) family of sets $\Pi(\Sigma)$ (for $\Sigma \in |\mathrm{Sign}|$), where the elements of $\Pi(\Sigma)$ are pairs (Φ, φ), consisting of a set $\Phi \subseteq \mathrm{Sen}(\Sigma)$ and a Σ-sentence φ. The pair (Φ, φ) can be considered as an (instantiated) proof rule or as an axiom if $\Phi = \emptyset$.

Definition 11.1 (Proof system). A *proof system* for INS is a family $\Pi = (\Pi(\Sigma))_{\Sigma \in |\mathrm{Sign}|}$ such that

for each $\Sigma \in |\mathrm{Sign}|$, $\Pi(\Sigma) \subseteq \wp(\mathrm{Sen}(\Sigma)) \times \mathrm{Sen}(\Sigma)$, and
for each signature morphism $\sigma\colon \Sigma \to \Sigma'$, if $(\Phi, \varphi) \in \Pi(\Sigma)$ then $(\sigma(\Phi), \sigma(\varphi)) \in \Pi(\Sigma')$.

Π is *sound* if $\Pi(\Sigma) \subseteq\ \models_\Sigma$ for each $\Sigma \in |\mathrm{Sign}|$. Π is called *formal* (or *finitary*) if $\Pi(\Sigma) \subseteq \wp_{\mathrm{f}}(\mathrm{Sen}(\Sigma)) \times \mathrm{Sen}(\Sigma)$ for each $\Sigma \in |\mathrm{Sign}|$, where $\wp_{\mathrm{f}}(\mathrm{Sen}(\Sigma))$ is the set of the *finite* subsets of $\mathrm{Sen}(\Sigma)$; otherwise Π is called *semiformal* (or *infinitary*).

Usually a proof system is presented by a finite set of axiom schemes and inference rules containing metavariables for the sentences [Men87]. Then $\Pi(\Sigma)$ just consists of all instantiations (of the axiom schemes and inference rules) where the metavariables are replaced by sentences; see also the proof system presented for the first-order logic with equality **FOLEq** in the example below. Formally, any relation R between sets of Σ-sentences and Σ-sentences generates an entailment relation in the following way.

Definition 11.2. Given $R \subseteq \wp(\mathrm{Sen}(\Sigma)) \times \mathrm{Sen}(\Sigma)$, where $\Sigma \in |\mathrm{Sign}|$, $\vdash_R$ is the least entailment relation on $\mathrm{Sen}(\Sigma)$ containing R.

Notation. $\Phi \vdash_R \varphi$ stands for $(\Phi, \varphi) \in\ \vdash_R$, $\ \vdash_R \varphi$ stands for $\emptyset \vdash_R \varphi$.

Given $R \subseteq \wp(\mathrm{Sen}(\Sigma)) \times \mathrm{Sen}(\Sigma)$, we call φ an *axiom* of R if $(\emptyset, \varphi) \in R$. Obviously, for each set Φ of Σ-sentences, the set $\{\varphi \in \mathrm{Sen}(\Sigma) \mid \Phi \vdash_R \varphi\}$ is the least set of Σ-sentences which contains all sentences of Φ and all axioms of R, and is closed under the "rules of inference" of R [Kei71].

If R is finitary, i.e., $R \subseteq \wp_{\mathrm{f}}(\mathrm{Sen}(\Sigma)) \times \mathrm{Sen}(\Sigma)$, then a necessary and sufficient condition for $\Phi \vdash \varphi$ is that there exists a finite sequence $\varphi_1, \ldots, \varphi_k$ of Σ-sentences, called *deduction* [Men87], such that $\varphi_k = \varphi$, and for each $i \in \{1, \ldots, k\}$

1. φ_i is an axiom (i.e., $(\emptyset, \varphi) \in R$), or
2. φ_i is a hypothesis (i.e., $\varphi_i \in \Phi$), or
3. there is a set of indices $J \subseteq \{1, \ldots, i-1\}$ such that $(\{\varphi_j : j \in J\}, \varphi_i) \in R$.[2]

The advantage of the inductive definition of $\vdash_R$ is that it provides a procedure for testing theoremhood, which is semidecidable if the proof system is formal.

It is straightforward to show that any proof system for INS generates an entailment system, i.e., we have the following proposition.

Proposition 11.3. *Let Π be a proof system for* INS*. Then*

$$\mathrm{ES}_\Pi \stackrel{\mathrm{def}}{=} \langle \mathrm{Sign}, \mathrm{Sen}, (\vdash_{\Pi(\Sigma)})_{\Sigma \in |\mathrm{Sign}|} \rangle$$

is an entailment system. If Π is sound then ES_Π *is sound.*

Remark 11.4. If the proof system Π is sound, then INS together with ES_Π constitute a general logic (see Chapter 4).

Definition 11.5. A proof system Π for INS is called *complete* if ES_Π is complete, i.e., for each $\Sigma \in |\mathrm{Sign}|$, $\Phi \subseteq \mathrm{Sen}(\Sigma)$, and Σ-sentence φ, if $\Phi \models_\Sigma \varphi$ then $\Phi \vdash_{\Pi(\Sigma)} \varphi$.

Example 11.6 (First-order logic with equality). An important example is the institution **FOEq** of first-order logic with equality as defined in Chapter 4, Section 4.2. We use here a slightly different institution, called **FOLEq**, where for each first-order signature Σ the sentences over Σ are arbitrary first-order Σ-formulas (possibly containing free variables)[3] and the Σ-structures have nonempty carrier sets.[4] We will use the following notations and definitions:

- $\mathbf{FOLEq} = \langle \mathbf{FOLSig}, \mathbf{FOLWff}, \mathbf{FOLStr}, \models \rangle$

[2] In fact, the first case is subsumed by the third with $J = \emptyset$; traditionally, however, these two cases are distinguished as one distinguishes axioms from inference rules.

[3] This is necessary since formulas occurring in the inference rules of proof systems for **FOLEq** may contain free variables.

[4] Hence no pathological situation concerning the satisfaction relation can arise [KK67].

- A (first-order) signature $\Sigma \in |\mathbf{FOLSig}|$ is a triple $\langle S, F, P\rangle$ of *sorts* S, *function symbols* F, and *predicate symbols* P. With each $f \in F$, a *profile* $s_1, \ldots, s_n \to s$ is associated and with each $p \in P$, a *profile* $s_1, \ldots, s_n$ is associated, where $s_1, \ldots, s_n, s \in S$. We often write $\mathrm{Sorts}(\Sigma)$ for S, $\mathrm{Opns}(\Sigma)$ for F, and $\mathrm{Preds}(\Sigma)$ for P.
- A signature morphism $\sigma\colon \Sigma \to \Sigma'$ from a signature $\Sigma = \langle S, F, P\rangle$ to a signature $\Sigma' = \langle S', F', P'\rangle$ is a triple $\langle \sigma_{\mathrm{Sorts}}, \sigma_{\mathrm{Opns}}, \sigma_{\mathrm{Preds}}\rangle$ of mappings $\sigma_{\mathrm{Sorts}}\colon S \to S'$, $\sigma_{\mathrm{Opns}}\colon F \to F'$, $\sigma_{\mathrm{Preds}}\colon P \to P'$, such that the profiles are respected.
- For each signature $\Sigma \in |\mathbf{FOLSig}|$, we assume that we have a $\mathrm{Sorts}(\Sigma)$-sorted family $X = (X_s)_{s\in\mathrm{Sorts}(\Sigma)}$ of denumerable sets X_s of variables (of sort s). The set $T_\Sigma(X)$ of Σ-terms with variables in X and the set $\mathbf{FOLWff}(\Sigma)$ of first-order Σ-formulas with equality and with (possibly free) variables in X are defined as usual. Substitution of a term t of sort s for a variable $x \in X_s$ in a formula φ is denoted by $\varphi[t/x]$.
- For each signature $\Sigma \in |\mathbf{FOLSig}|$, the category $\mathbf{FOLStr}(\Sigma)$ of (total) Σ-structures (whose carrier sets are not empty) and Σ-homomorphisms is defined as usual. If Σ contains no predicate symbols, i.e., is of the form $\langle S, F, \emptyset\rangle$ (abbreviated to $\langle S, F\rangle$), then Σ-structures are called Σ-*algebras*. For each signature morphism $\sigma\colon \Sigma \to \Sigma'$, $\mathbf{FOLStr}(\sigma)$ is the usual forgetful functor and is denoted by $\_|_\sigma$.
- The satisfaction relation $\models_\Sigma$ between Σ-structures and Σ-formulas is defined as usual. Sometimes we drop the index Σ if it is obvious from the context.

Similarly, the institution **IFOLEq** of infinitary first-order logic with denumerable conjunctions and disjunctions is defined.

An example of a (formal) proof system for the institution **FOLEq** is the following Hilbert-like system $\Pi(\mathbf{FOLEq})$, which is sound and complete [Men87]. For each signature Σ,

(A1) $\varphi_1 \Rightarrow (\varphi_2 \Rightarrow \varphi_1)$ is an axiom of $\Pi(\mathbf{FOLEq})$, i.e.,
if $\varphi_1, \varphi_2 \in \mathbf{FOLWff}(\Sigma)$ then $(\emptyset, \varphi_1 \Rightarrow (\varphi_2 \Rightarrow \varphi_1)) \in \Pi(\mathbf{FOLEq})(\Sigma)$;
(A2) $(\varphi_1 \Rightarrow (\varphi_2 \Rightarrow \varphi_3)) \Rightarrow ((\varphi_1 \Rightarrow \varphi_2) \Rightarrow (\varphi_1 \Rightarrow \varphi_3))$ is an axiom of $\Pi(\mathbf{FOLEq})$;
(A3) $(\neg\varphi_2 \Rightarrow \neg\varphi_1) \Rightarrow ((\neg\varphi_2 \Rightarrow \varphi_1) \Rightarrow \varphi_2)$ is an axiom of $\Pi(\mathbf{FOLEq})$;
(A4) if $s \in \mathrm{Sorts}(\Sigma)$ and φ_1 contains no free occurrences of $x \in X_s$,[5]
then $(\forall x : s.\ \varphi_1 \Rightarrow \varphi_2) \Rightarrow (\varphi_1 \Rightarrow (\forall x : s.\ \varphi_2))$ is an axiom of $\Pi(\mathbf{FOLEq})$;
(A5) if t is a Σ-term of sort s free for $x \in X_s$ in φ,[6]
then $(\forall x{:}s.\ \varphi) \Rightarrow \varphi[t/x]$ is an axiom of $\Pi(\mathbf{FOLEq})$;

[5] The notion of free occurrence of a variable in a formula is defined as usual.

[6] A term t is free for a variable x in a formula φ if no occurrence of x in the formula φ lies within the scope of any quantifier $(\forall x')$, where x' is a variable in the term t. Thus, the substitution of the term for the variable in the formula causes no variable trapping in the term.

(A6) if $s \in \mathrm{Sorts}(\Sigma)$ and $x \in X_s$,
then $\forall x\!:\!s.\ x = x$ is an axiom of $\Pi(\mathbf{FOLEq})$;

(A7) if $s \in \mathrm{Sorts}(\Sigma)$ and $x, x' \in X_s$,
then $x = x' \Rightarrow (\varphi(x,x) \Rightarrow \varphi(x,x'))$[7] is an axiom of $\Pi(\mathbf{FOLEq})$;

(MP) $\dfrac{\varphi_1 \Rightarrow \varphi_2,\ \varphi_1}{\varphi_2}$ is an inference rule of $\Pi(\mathbf{FOLEq})$, i.e.,
if $\varphi_1, \varphi_2 \in \mathbf{FOLWff}(\Sigma)$ then $(\{\varphi_1 \Rightarrow \varphi_2, \varphi_1\}, \varphi_2) \in \Pi(\mathbf{FOLEq})(\Sigma)$;

(Gen) $\dfrac{\varphi}{\forall x\!:\!s.\ \varphi}$ is an inference rule of $\Pi(\mathbf{FOLEq})$.

Note that φ, φ_1, φ_2, and φ_3 may be considered as metavariables which can be instantiated by well-formed formulas. Hence $\Pi(\mathbf{FOLEq})$ can be presented just by giving seven axiom schemes and two inference rules.

A sound and complete proof system $\Pi(\mathbf{IFOLEq})$ for infinitary first-order logic can be found in [Kei71]. It contains, for any denumerable set Φ of formulas, infinitary axioms of the form $\bigwedge\Phi \Rightarrow \varphi$ with $\varphi \in \Phi$, and infinitary inference rules of the form $\dfrac{\psi \Rightarrow \varphi \text{ for all } \varphi \in \Phi}{\psi \Rightarrow \bigwedge\Phi}$. ∎

The next proposition is important for proving the equivalence of the two noncompositional proof systems for specification expressions considered in Section 11.3.

Proposition 11.7. *Let Π be a proof system for* INS. *For each signature $\Sigma \in |\mathrm{Sign}|$ and for all $\Phi, \Phi' \subseteq \mathrm{Sen}(\Sigma)$ and $\varphi \in \mathrm{Sen}(\Sigma)$:*

$$\Phi' \vdash_{\Pi(\Sigma)\cup\Phi} \varphi \text{ if and only if } \Phi' \cup \Phi \vdash_{\Pi(\Sigma)} \varphi$$

where $\Pi(\Sigma) \cup \Phi$ stands for $\Pi(\Sigma) \cup \{(\emptyset, \phi) \mid \phi \in \Phi\}$.

11.2.2 Structured specifications: the kernel language

In this section we define a kernel language for specifications in the institution INS, consisting of presentations over INS (which we call basic specifications) and three specification-building operators. The semantics of the language is defined by two functions Sig and Mod, returning respectively the signature and the model class associated with each specification expression, see Chapter 4, Section 4.6.

Assumptions. We require that the given institution INS has semicomposable signatures, see Chapter 4.[8] Moreover, we assume that Sign is an inclusive

[7] $\varphi(x,x')$ arises from $\varphi(x,x)$ by replacing some, but not necessarily all, free occurrences of x by x', with the proviso that x' is free for x in $\varphi(x,x)$. Thus, $\varphi(x,x')$ may or may not contain free occurrences of x.

[8] An institution with semicomposable signatures is called an *exact* institution in [Mes89] and *semiexact* in [DGS93]. This prerequisite is equivalent to the amalgamation lemma, see Chapter 4.

$$\begin{tikzcd}
 & \Sigma_0 \arrow[dl, "\sigma_1"'] \arrow[dr, "\sigma_2"] & \\
\Sigma_1 \arrow[dr, dashed, "\mathrm{po}_1(\sigma_1,\sigma_2)"'] & & \Sigma_2 \arrow[dl, dashed, "\mathrm{po}_2(\sigma_1,\sigma_2)"] \\
 & \mathrm{PO}(\sigma_1,\sigma_2) &
\end{tikzcd}$$

Fig. 11.1. Pushout diagram

$$\begin{tikzcd}
 & \Sigma_0 \arrow[dl, hook'] \arrow[dr, hook] & \\
\Sigma_1 \arrow[dr, dashed, "inl"'] & & \Sigma_2 \arrow[dl, dashed, "inr"] \\
 & \Sigma_1 +_{\Sigma_0} \Sigma_2 &
\end{tikzcd}$$

Fig. 11.2. Amalgamated sum

category with initial object $\emptyset$ and inclusion-preserving pushouts [DGS93].[9] If $\sigma\colon \Sigma \to \Sigma'$ is an inclusion (also called embedding) morphism, then we write $\sigma\colon \Sigma \hookrightarrow \Sigma'$, or simply $\Sigma \subseteq \Sigma'$.[10] In this case, for any Σ'-structure A we write $A|_\Sigma$ for $A|_\sigma$. Similarly, if C is a class of Σ-structures, then $C|_\Sigma$ denotes the class of all Σ-structures $A|_\Sigma$ with $A \in C$. Pushouts are depicted as in Figure 11.1. If σ_1 and σ_2 are inclusions, the respective pushout morphisms are denoted by *inl* and *inr*, and the pushout signature is denoted by $\Sigma_1 +_{\Sigma_0} \Sigma_2$ and is called the *amalgamated sum* of Σ_1 and Σ_2 over Σ_0, see Figure 11.2.[11] An isomorphism between signatures is also called a *renaming*.

*Example 11.8 (***FOLEq***).* If Σ_1, Σ_2 are first-order signatures such that Σ_1 is included componentwise in Σ_2 (i.e., $\mathrm{Sorts}(\Sigma_1) \subseteq \mathrm{Sorts}(\Sigma_2)$, $\mathrm{Opns}(\Sigma_1) \subseteq \mathrm{Opns}(\Sigma_2)$, and $\mathrm{Preds}(\Sigma_1) \subseteq \mathrm{Preds}(\Sigma_2)$), then Σ_1 is called a *subsignature* of Σ_2, denoted by $\Sigma_1 \subseteq \Sigma_2$. Inclusion morphisms are defined as subsignature relations $\Sigma_1 \subseteq \Sigma_2$. ■

Definition 11.9 (Structured specifications). The collection Spec of *specification expressions* (or specifications for short) in INS, together with the functions $\mathrm{Sig}\colon \mathrm{Spec} \to |\mathrm{Sign}|$ and $\mathrm{Mod}\colon \mathrm{Spec} \to \{C \subseteq \mathrm{Str}(\Sigma) \mid \Sigma \in \mathrm{Sign}\}$ (returning the signature and the class of models of a given specification, respectively), are inductively defined as follows.

1. A *basic specification* is a pair $\langle \Sigma, E \rangle$ where Σ is a signature and E is a set of Σ-sentences.
   $\mathrm{Sig}(\langle \Sigma, E \rangle) \overset{\mathrm{def}}{=} \Sigma$
   $\mathrm{Mod}(\langle \Sigma, E \rangle) \overset{\mathrm{def}}{=} \{A \in \mathrm{Str}(\Sigma) \mid A \models_\Sigma E\}$

[9] Inclusive categories capture the notion of image subobjects. In other words, if $\sigma\colon \Sigma \to \Sigma'$ is a signature morphism, then it makes sense to talk about $\sigma(\Sigma) \in |\mathrm{Sign}|$ (which is intuitively the image of Σ under σ).

[10] Note that, by definition of inclusive categories, there is at most one inclusion morphism between two signatures, see [DGS93].

[11] The pushout can be chosen in such a way that either *inl* or *inr* is an inclusion, since Sign is assumed to be a category with inclusion-preserving pushouts, see [DGS93].

2. If SP_1 and SP_2 are specification expressions, and Σ is a signature such that $\Sigma \subseteq \mathrm{Sig}(\mathrm{SP}_1)$ and $\Sigma \subseteq \mathrm{Sig}(\mathrm{SP}_2)$, then the *amalgamated sum (or union)* $\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2$ of SP_1 and SP_2 w.r.t. Σ is a specification expression.
   $\mathrm{Sig}(\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2) \stackrel{\mathrm{def}}{=} \mathrm{Sig}(\mathrm{SP}_1) +_\Sigma \mathrm{Sig}(\mathrm{SP}_2)$ (see Figure 11.2)
   $\mathrm{Mod}(\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2) \stackrel{\mathrm{def}}{=} \{A \in \mathrm{Str}(\mathrm{Sig}(\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2)) \mid A|_{inl} \in \mathrm{Mod}(\mathrm{SP}_1)$ and $A|_{inr} \in \mathrm{Mod}(\mathrm{SP}_2)\}$
3. If SP is a specification expression and Σ is a signature such that $\Sigma \subseteq \mathrm{Sig}(\mathrm{SP})$, then $\mathrm{SP}|_\Sigma$ is a specification expression, called an *export* specification.
   $\mathrm{Sig}(\mathrm{SP}|_\Sigma) \stackrel{\mathrm{def}}{=} \Sigma$
   $\mathrm{Mod}(\mathrm{SP}|_\Sigma) \stackrel{\mathrm{def}}{=} \mathrm{Mod}(\mathrm{SP})|_\Sigma$
   The symbols of SP that do not occur in Σ are called the *hidden symbols* of $\mathrm{SP}|_\Sigma$, whereas the symbols of Σ are its *exported symbols*.
4. If SP is a specification expression and $r\colon \mathrm{Sig}(\mathrm{SP}) \to \Sigma$ is a renaming, then `rename` SP `by` r is a specification expression called the *renaming* of SP by r.
   $\mathrm{Sig}(\texttt{rename}\ \mathrm{SP}\ \texttt{by}\ r) \stackrel{\mathrm{def}}{=} \Sigma$
   $\mathrm{Mod}(\texttt{rename}\ \mathrm{SP}\ \texttt{by}\ r) \stackrel{\mathrm{def}}{=} \{A \in \mathrm{Str}(\Sigma) \mid A|_r \in \mathrm{Mod}(\mathrm{SP})\}$

Amalgamated sum, export, and renaming are called *specification-building operators*. Further operators can be derived, for instance, the *plain sum (or union)* defined as follows:
$\mathrm{SP}_1 + \mathrm{SP}_2 \stackrel{\mathrm{def}}{=} \mathrm{SP}_1 +_\Sigma \mathrm{SP}_2$ where $\Sigma = \mathrm{Sig}(\mathrm{SP}_1) \cap \mathrm{Sig}(\mathrm{SP}_2)$.[12]
The morphisms *inl* and *inr* associated with a plain sum are the usual embeddings since no undesired name conflicts can arise.

If Σ is the signature of a specification expression SP, then we write Sorts(SP) to denote Sorts(Σ), Opns(SP) to denote Opns(Σ), and Preds(SP) to denote Preds(Σ). If $\mathrm{SP} = \langle \Sigma, E \rangle$ is a basic specification, then the axioms E of SP are also denoted by Axs(SP).

Example 11.10 (`CONT`*).* The following basic specification `CONT` (in the institution **FOLEq**) describes properties of containers which can store arbitrary elements. It has a sort `Cont` for the containers and a sort `Elem` for the elements of containers. Elements can be stored in a container with the operation `add` and the membership test is denoted by the function symbol `member`.

```
spec CONT
    sorts Bool, Elem, Cont
    functions true, false :  → Bool
              empty       :  → Cont
              add         : Elem, Cont → Cont
              member      : Elem, Cont → Bool
```

[12] For this definition to make sense, the category of signatures must have pullbacks, which would mean that INS is a *reasonable* institution in the terminology of [DGS93].

axioms $\forall x, y\colon \texttt{Elem}, s\colon \texttt{Cont}.$
$\texttt{true} \neq \texttt{false} \wedge$
$\texttt{member}(x, \texttt{empty}) = \texttt{false} \wedge$
$\texttt{member}(x, \texttt{add}(x, s)) = \texttt{true} \wedge$
$(x \neq y \Rightarrow \texttt{member}(x, \texttt{add}(y, s)) = \texttt{member}(x, s))$
endspec

Note that `CONT` has only one axiom, which is a universally quantified conjunction of formulas. Of course, the axioms of `CONT` could equally be presented by a *set* of universally quantified formulas. For technical convenience, in the following, we often disregard which kind of presentation of the axioms of a (basic) specification is chosen. ∎

Example 11.11 (`CONT-I`*).* The specification `CONT-I` constructed below can be considered as an implementation of containers by stacks. It is based on a common specification `STACK` of stacks which is combined with a specification `MEMBER`, that implements the membership test over stacks. In order to adapt the sum of these specifications (which is called `STACK-EXT`) to the signature of the container specification, an appropriate renaming is performed and the signature of `CONT` is exported from the renamed specification (i.e., the stack operations `top` and `pop` used for the implementation are hidden).

spec `STACK`
sorts `Elem, Stack`
functions `emptyst` : $\to$ `Stack`
`push` : `Elem, Stack` $\to$ `Stack`
`pop` : `Stack` $\to$ `Stack`
`top` : `Stack` $\to$ `Elem`
axioms $\forall x\colon \texttt{Elem}, s\colon \texttt{Stack}.$
$\texttt{push}(x, s) \neq \texttt{emptyst} \wedge$
$\texttt{pop}(\texttt{push}(x, s)) = s \wedge$
$\texttt{top}(\texttt{push}(x, s)) = x$
endspec

spec `MEMBER`
sorts `Bool, Elem, Stack`
functions `true, false` : $\to$ `Bool`
`emptyst` : $\to$ `Stack`
`push` : `Elem, Stack` $\to$ `Stack`
`pop` : `Stack` $\to$ `Stack`
`top` : `Stack` $\to$ `Elem`
`member` : `Elem, Stack` $\to$ `Bool`
axioms $\forall x\colon \texttt{Elem}, s\colon \texttt{Stack}.$
$\texttt{true} \neq \texttt{false} \wedge$
$\texttt{member}(x, \texttt{emptyst}) = \texttt{false} \wedge$
$[s \neq \texttt{emptyst} \Rightarrow$
$((\texttt{top}(s) = x \Rightarrow \texttt{member}(x, s) = \texttt{true}) \wedge$
$(\texttt{top}(s) \neq x \Rightarrow \texttt{member}(x, s) = \texttt{member}(x, \texttt{pop}(s))))]$
endspec

spec STACK-EXT = STACK + MEMBER

spec CONT-I = (rename STACK-EXT by r)$|_{\text{Sig(CONT)}}$

where r renames `Stack` into `Cont`, `emptyst` into `empty`, and `push` into `add`.

■

For any specification expression SP, a Sig(SP)-sentence is also called an SP-sentence. Given an SP-sentence φ, we write SP $\models_\Sigma \varphi$ if Mod(SP) $\models_\Sigma \varphi$, i.e., if φ is satisfied by all models of SP. In this case φ is called a *theorem* of SP. It is a central aim of this chapter to study proof systems that allow us to prove that a given sentence is a theorem of a specification. In the following, we consider two different possibilities: noncompositional proof systems and structured proof systems which allow us to perform proofs according to the structure of a given specification.

11.3 Noncompositional proof systems

Let INS $= \langle \text{Sign}, \text{Sen}, \text{Str}, (\models_\Sigma)_{\Sigma \in |\text{Sign}|} \rangle$ be an arbitrary institution which satisfies the assumptions of Section 11.2.2 and, moreover, let Π be a sound proof system for INS. There are two approaches to noncompositional proof systems: either a normal form nf(SP) equivalent to a given specification SP is computed whose theorems are exactly those of SP, or a proof system is designed for each particular specification.

In both cases, hidden symbols play an important role in the derivation of theorems. Consider, for example, a first-order signature with just one sort and three constants a, b, and c of that sort. Define the basic specification consisting of that signature and two axioms $a = c$ and $b = c$. Now hide the symbol c. In the resulting specification, there are just two (visible) constants, and no "visible axiom". Nevertheless, the equality $a = b$ is satisfied by each of its models.

11.3.1 Proof system associated with a specification

In this subsection we construct an individual proof system for each specification SP that allows us to prove theorems of SP. Technically this is done by providing, for each structured specification SP, a set Π_{SP} of particular "non-logical" axioms and rules according to the form of SP. We show that when combining Π_{SP} with the given proof system Π for the underlying institution INS, the result is a sound proof system for SP (which is also complete if Π is complete).

The proof systems Π_{SP} are inductively defined according to the structure of the specification SP. In the simplest case SP is a basic specification and Π_{SP} just consists of the axioms of SP. In all other cases the proof system for SP is constructed by using the (appropriately modified) proof systems for the specifications from which SP was built by the specification-building operators.

As already sketched above, the hidden symbols cannot be simply forgotten when trying to derive the theorems that are valid in a specification. However special care is then needed because of possible name conflicts with hidden symbols, as shown in the following examples.

Example 11.12. There may be name conflicts caused by the combination of "export" and "rename". Indeed, assume that SP is a basic specification in **FOLEq** with one sort s, three constants a, b, c of sort s, and the axiom $a = c$. Now hide the symbol c, i.e., construct the specification $\mathrm{SP}' = \mathrm{SP}|_\Sigma$ with $\Sigma = \langle \{s\}, \{a, b\}, \emptyset \rangle$. Since we cannot forget axioms concerning hidden symbols, the equation $a = c$ is also an axiom of (the proof system for) SP′. In a next step we "re-introduce" the constant c by renaming b to c, i.e., we consider the specification `rename` SP′ `by` r where $r_{\mathrm{Sorts}}(s) = s$, $r_{\mathrm{Opns}}(a) = a$, and $r_{\mathrm{Opns}}(b) = c$. The renamed specification now has two constants a and c. If we construct the proof system for `rename` SP′ `by` r by renaming only the symbols of Σ according to r, then $a = c$ is also an axiom of (the proof system for) `rename` SP′ `by` r. Hence we can derive $a = c$ which, nevertheless, is not valid in all models of `rename` SP′ `by` r. ∎

The problem in this example is that a previously hidden symbol is re-introduced by the renaming. This difficulty is solved by using a pushout construction which renames not only the (visible) symbols of SP′, but also all its hidden symbols in a consistent way when constructing the proof system for `rename` SP′ `by` r.

Example 11.13. There may also be name conflicts caused by the combination of "export" and "amalgamated sum". Indeed, let SP be as in the previous example and let SP_2 be a basic specification with the same signature as SP, but with the axiom $b = c$. Now hide c in SP, i.e., construct the specification $\mathrm{SP}_1 = \mathrm{SP}|_\Sigma$ where $\Sigma = \langle \{s\}, \{a, b\}, \emptyset \rangle$. Again, since we cannot forget axioms concerning hidden symbols, the equation $a = c$ is also an axiom of (the proof system for) SP_1. Now construct the amalgamated sum $\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2$ (which is just the plain sum $\mathrm{SP}_1 + \mathrm{SP}_2$). Clearly, if we assume that the proof system for $\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2$ is simply the union of the proof systems for the single specifications, then both equations $a = c$ and $b = c$ belong (as axioms) to the proof system for $\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2$. This means that we can derive $a = b$, which is not valid in all models of $\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2$. ∎

The problem in the last example is that the constant c which is hidden in SP_1 is identified with the "visible" constant c of SP_2. This confusion can again be avoided using a pushout construction which consistently renames the hidden symbols of SP_1 (and, symmetrically, of SP_2) when combining the proof systems for SP_1 and SP_2. In summary, these considerations lead to the following definition of the symbols of a specification.

$$\begin{tikzcd}
\Sigma \arrow[r, hook] \arrow[d, hook] & \mathrm{Sig}(\mathrm{SP}_2) \arrow[r, hook] \arrow[d, dashed, "inr"] & \mathrm{Symbols}(\mathrm{SP}_2) \arrow[dd, dashed, "inr'"] \\
\mathrm{Sig}(\mathrm{SP}_1) \arrow[r, dashed, "inl"] \arrow[d, hook] & \begin{array}{c} \mathrm{Sig}(\mathrm{SP}_1) +_\Sigma \mathrm{Sig}(\mathrm{SP}_2) \\ = \mathrm{Sig}(\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2) \end{array} \arrow[dr, dashed, hook] & \\
\mathrm{Symbols}(\mathrm{SP}_1) \arrow[rr, dashed, "inl'"] & & \begin{array}{c} \mathrm{Symbols}(\mathrm{SP}_1) \\ +_\Sigma \\ \mathrm{Symbols}(\mathrm{SP}_2) \end{array}
\end{tikzcd}$$

Fig. 11.3. Symbols of an amalgamated union

$$\begin{tikzcd}
\mathrm{Sig}(\mathrm{SP}') \arrow[r, "r"] \arrow[d, hook] & \Sigma \arrow[d, dashed, hook] \\
\mathrm{Symbols}(\mathrm{SP}') \arrow[r, dashed, "\sigma"] & \mathrm{PO}(\mathrm{Sig}(\mathrm{SP}') \hookrightarrow \mathrm{Symbols}(\mathrm{SP}'), r)
\end{tikzcd}$$

Fig. 11.4. Symbols of a renamed specification

Definition 11.14 (Symbols). For any specification expression SP, the signature Symbols(SP) of the symbols of SP is inductively defined as follows.

1. If SP is a specification expression of the form $\langle \Sigma, E\rangle$, then $\mathrm{Symbols}(\mathrm{SP}) \stackrel{\mathrm{def}}{=} \Sigma$.
2. If SP is a specification expression of the form $\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2$, then $\mathrm{Symbols}(\mathrm{SP}) \stackrel{\mathrm{def}}{=} \mathrm{Symbols}(\mathrm{SP}_1) +_\Sigma \mathrm{Symbols}(\mathrm{SP}_2)$ (see Figure 11.3).
3. If SP is a specification expression of the form $\mathrm{SP}'|_\Sigma$, then $\mathrm{Symbols}(\mathrm{SP}) \stackrel{\mathrm{def}}{=} \mathrm{Symbols}(\mathrm{SP}')$.
4. If SP is a specification expression of the form `rename` SP' `by` r, then $\mathrm{Symbols}(\mathrm{SP}) \stackrel{\mathrm{def}}{=} \mathrm{PO}(\mathrm{Sig}(\mathrm{SP}') \hookrightarrow \mathrm{Symbols}(\mathrm{SP}'), r)$ (see Figure 11.4).

Note that Symbols(SP) is always well defined since pushout signatures are defined up to isomorphism and, by assumption, preserve inclusions. Hence we can always choose Symbols(SP) such that $\mathrm{Sig}(\mathrm{SP}) \subseteq \mathrm{Symbols}(\mathrm{SP})$.

Intuitively, the symbols in Symbols(SP) that are not included in Sig(SP) are the hidden symbols of SP. The pushout technique ensures that hidden symbols are not confused with other symbols when computing the proof system for an amalgamated union or for a specification obtained by renaming.

Example 11.15 *(*Symbols(`CONT-I`)*).* Let `CONT-I` be the specification of Example 11.11. Obviously, the signature of the symbols of the (sub)specifications `STACK`, `MEMBER`, `STACK-EXT`, and `rename STACK-EXT by` r is just the signature of each of the given specifications, since no export operator is applied in these cases. Then, by definition of the symbols of an export specification, Symbols(`CONT-I`) = Sig(`rename STACK-EXT by` r) which is the signature

$$\begin{array}{l}\langle\{\mathtt{Bool},\mathtt{Elem},\mathtt{Cont}\},\\ \ \{\mathtt{true},\mathtt{false}:\ \to \mathtt{Bool},\\ \ \ \mathtt{empty}:\ \to \mathtt{Cont},\\ \ \ \mathtt{add}:\mathtt{Elem},\mathtt{Cont}\to \mathtt{Cont},\\ \ \ \mathtt{pop}:\mathtt{Cont}\to \mathtt{Cont},\\ \ \ \mathtt{top}:\mathtt{Cont}\to \mathtt{Elem},\\ \ \ \mathtt{member}:\mathtt{Elem},\mathtt{Cont}\to \mathtt{Bool}\}\rangle\ .\end{array}$$

■

Definition 11.16 (Π_{SP}). For any specification expression SP, the corresponding proof system Π_{SP} is inductively defined as follows.

1. If SP is a basic specification $\langle\Sigma, E\rangle$,
   then $\Pi_{\mathrm{SP}} \stackrel{\mathrm{def}}{=} E$.
2. If SP is a specification expression of the form $\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2$,
   then $\Pi_{\mathrm{SP}} \stackrel{\mathrm{def}}{=} \mathit{inl}'(\Pi_{\mathrm{SP}_1}) \cup \mathit{inr}'(\Pi_{\mathrm{SP}_2})$
   where $\mathit{inl}' = \mathrm{po}_1(\Sigma \hookrightarrow \mathrm{Symbols}(\mathrm{SP}_1), \Sigma \hookrightarrow \mathrm{Symbols}(\mathrm{SP}_2))$,
   $\mathit{inr}' = \mathrm{po}_2(\Sigma \hookrightarrow \mathrm{Symbols}(\mathrm{SP}_1), \Sigma \hookrightarrow \mathrm{Symbols}(\mathrm{SP}_2))$
   (see Figure 11.3), and $\mathit{inl}'(\Pi_{\mathrm{SP}_1})$ and $\mathit{inr}'(\Pi_{\mathrm{SP}_2})$ are respectively the straightforward extensions of *inl*' and *inr*' to proof systems.
3. If SP is a specification expression of the form $\mathrm{SP}'|_\Sigma$,
   then $\Pi_{\mathrm{SP}} \stackrel{\mathrm{def}}{=} \Pi_{\mathrm{SP}'}$.
4. If SP is a specification expression of the form `rename` SP′ `by` r,
   then $\Pi_{\mathrm{SP}} \stackrel{\mathrm{def}}{=} \sigma(\Pi_{\mathrm{SP}'})$,
   where $\sigma = \mathrm{po}_1(\mathrm{Sig}(\mathrm{SP}') \hookrightarrow \mathrm{Symbols}(\mathrm{SP}'), r)$ (see Figures 11.1 and 11.4).

For any specification SP, the proof system Π_{SP} can be combined with the given proof system Π for deriving SP-sentences. We write $\vdash_{\Pi_{\mathrm{SP}}} \varphi$ if φ is an SP-sentence such that $\vdash_{\Pi(\Sigma)\cup\Pi_{\mathrm{SP}}} \varphi$, where $\Sigma = \mathrm{Symbols}(\mathrm{SP})$.[13] The next theorem states that in this way we obtain a sound proof system for SP. Moreover, if Π is complete, we even obtain a complete proof system for SP.

Theorem 11.17 (Soundness and completeness of $\Pi_{\mathrm{SP}} \cup \Pi$). *Let* SP *be a specification expression and let φ be an* SP*-sentence. If $\vdash_{\Pi_{\mathrm{SP}}} \varphi$, then* SP $\models_{\mathrm{Sig(SP)}} \varphi$*. The converse holds if Π is complete.*

[13] By $\Pi(\Sigma) \cup \Pi_{\mathrm{SP}}$ we simply mean, in the same way as in Proposition 11.7, the proof system defined by $\Pi(\Sigma) \cup \{(\emptyset, \phi) \mid \phi \in \Pi_{\mathrm{SP}}\}$. In Section 11.7 inference rules, as well as axioms, will be included in Π_{SP}.

Proof. The theorem is a consequence of the equivalence of the given proof system and the normal form approach studied in the following subsections (see Corollary 11.21 and Proposition 11.24). □

Example 11.18 ($\boldsymbol{\Pi}_{\texttt{CONT-I}}$). The proof system of the specification CONT-I of Example 11.11 is constructed by first, combining the axioms of the (sub)specifications STACK and MEMBER, and then, renaming the resulting set of sentences according to r. Hence $\Pi_{\texttt{CONT-I}}$ consists of the following sentences over the signature Symbols(CONT-I):

$$
\begin{array}{l}
\Pi_{\texttt{CONT-I}} = \\
\{\ \forall x\colon \texttt{Elem}, s\colon \texttt{Cont}. \\
\qquad \texttt{add}(x,s) \neq \texttt{empty} \wedge \texttt{pop}(\texttt{add}(x,s)) = s \wedge \texttt{top}(\texttt{add}(x,s)) = x, \\
\quad \forall x\colon \texttt{Elem}, s\colon \texttt{Cont}. \\
\qquad \texttt{true} \neq \texttt{false} \wedge \texttt{member}(x,\texttt{empty}) = \texttt{false} \wedge \\
\qquad [s \neq \texttt{empty} \Rightarrow ((\texttt{top}(s) = x \Rightarrow \texttt{member}(x,s) = \texttt{true}) \wedge \\
\qquad\qquad\qquad (\texttt{top}(s) \neq x \Rightarrow \texttt{member}(x,s) = \texttt{member}(x,\texttt{pop}(s))))]\ \}
\end{array}
$$

Proofs of theorems of CONT-I can now be performed by combining the "non-logical" axioms of the proof system $\Pi_{\texttt{CONT-I}}$ with the axioms and rules of $\Pi(\mathbf{FOLEq})(\text{Symbols}(\texttt{CONT-I}))$ (which is just the first-order proof system $\Pi(\mathbf{FOLEq})$ used for formulas over the signature Symbols(CONT-I)).
For instance, one can derive $\vdash_{\Pi_{\texttt{CONT-I}}} \varphi$ where φ is the Sig(CONT-I)-sentence $x \neq y \Rightarrow \texttt{member}(x,\texttt{add}(y,s)) = \texttt{member}(x,s)$. We will not give a proof here as an explicit proof of this theorem, using the structured proof system to be introduced in Section 11.4, will be presented in Example 11.27. ■

11.3.2 Normal form of a given specification

The idea of the normal form approach is to compute, for a given specification SP, an equivalent but "simpler" specification nf(SP), called the *normal form* of SP, such that one can use just the axioms of this simpler specification (together with the proof system Π associated with the underlying institution) in order to derive the theorems of the given specification SP.

Two specification expressions are said to be *equivalent* if they have the same signature and the same model class.

Since we have to take into account the hidden symbols of a specification expression SP, a normal form equivalent to SP cannot generally be just a basic specification, it has to be a basic specification restricted to an export signature (at least, if we restrict ourselves to finite presentations; for examples and proofs of this fact, see [BM84,BT87,Maj77,Maj79,TWW82,Wir90]).

More precisely, given a specification expression SP, there exists a *normal form* specification expression nf(SP) of the form $\langle \text{Symbols(SP)}, E\rangle|_{\text{Sig(SP)}}$ equivalent to SP. For computing the normal form of SP it is enough to compute the symbols of SP (as described in the previous subsection) and an appropriate set E of axioms. This can be done inductively on the structure of the given specification.

Definition 11.19 (Normal form). The normal form nf(SP) associated with a specification expression SP is defined by induction on the structure of SP as follows.

1. If SP is a specification expression of the form $\langle \Sigma, E \rangle$, then $\mathrm{nf}(\mathrm{SP}) \stackrel{\text{def}}{=} \langle \Sigma, E \rangle|_{\Sigma}$.
2. If SP is a specification expression of the form $\mathrm{SP}_1 +_{\Sigma} \mathrm{SP}_2$ with $\mathrm{nf}(\mathrm{SP}_i) = \langle \mathrm{Symbols}(\mathrm{SP}_i), E_i \rangle|_{\mathsf{Sig}(\mathrm{SP}_i)}$ for $i = 1, 2$, then $\mathrm{nf}(\mathrm{SP}) \stackrel{\text{def}}{=} \langle \mathrm{Symbols}(\mathrm{SP}), \mathit{inl}'(E_1) \cup \mathit{inr}'(E_2) \rangle|_{\mathsf{Sig}(\mathrm{SP})}$, where inl' and inr' are the signature morphisms shown in Figure 11.3.
3. If SP is a specification expression of the form $\mathrm{SP}'|_{\Sigma}$ with $\mathrm{nf}(\mathrm{SP}') = \langle \mathrm{Symbols}(\mathrm{SP}'), E \rangle|_{\mathsf{Sig}(\mathrm{SP}')}$, then $\mathrm{nf}(\mathrm{SP}) \stackrel{\text{def}}{=} \langle \mathrm{Symbols}(\mathrm{SP}'), E \rangle|_{\Sigma}$.
4. If SP is a specification expression of the form `rename` SP' `by` r with $\mathrm{nf}(\mathrm{SP}') = \langle \mathrm{Symbols}(\mathrm{SP}'), E \rangle|_{\mathsf{Sig}(\mathrm{SP}')}$, then $\mathrm{nf}(\mathrm{SP}) \stackrel{\text{def}}{=} \langle \mathrm{Symbols}(\mathrm{SP}), \sigma(E) \rangle|_{\mathsf{Sig}(\mathrm{SP})}$, where σ is the signature morphism shown in Figure 11.4.

Theorem 11.20. *For any specification expression* SP, nf(SP) *is well defined and, moreover, equivalent to* SP.

Proof. The proof is straightforward by induction on the structure of SP, using the amalgamation lemma (which holds since, by assumption, INS has semicomposable signatures) for the induction step when SP is an amalgamated union or a renaming. □

As a direct consequence of the last theorem, we obtain that the normal form of a specification SP provides a means for proving the theorems of SP. This proof method is sound, and also complete provided Π is complete.

Notation. We write $\mathrm{SP} \vdash_{\mathrm{nf}} \varphi$ if φ is an SP-sentence such that $E \vdash_{\Pi(\Sigma)} \varphi$, where $\mathrm{nf}(\mathrm{SP}) = \langle \Sigma, E \rangle|_{\Sigma'}$.

Corollary 11.21 (Soundness and completeness of the normal form approach). *Let* SP *be a specification expression and let* φ *be an* SP*-sentence. If* $\mathrm{SP} \vdash_{\mathrm{nf}} \varphi$*, then* $\mathrm{SP} \models_{\mathsf{Sig}(\mathrm{SP})} \varphi$*. The converse holds if* Π *is complete.*

Proof. Let $\mathrm{nf}(\mathrm{SP}) = \langle \Sigma, E \rangle|_{\Sigma'}$.

Then	$\mathrm{SP} \vdash_{\mathrm{nf}} \varphi$	
iff	$E \vdash_{\Pi(\Sigma)} \varphi$	(by definition)
implies	$\langle \Sigma, E \rangle \models \varphi$	(since Π is sound) $(*)$
iff	$\mathrm{nf}(\mathrm{SP}) \models_{\mathsf{Sig}(\mathrm{SP})} \varphi$	(by satisfaction condition)
iff	$\mathrm{SP} \models_{\mathsf{Sig}(\mathrm{SP})} \varphi$	(by Theorem 11.20).

The converse of $(*)$ holds if Π is complete. □

Example 11.22 ($\mathbf{nf}(\texttt{CONT-I})$*).* The normal form of the specification CONT-I (see Example 11.11) is given by the specification

$$\mathrm{nf}(\texttt{CONT-I}) = \langle \mathrm{Symbols}(\texttt{CONT-I}), \Pi_{\texttt{CONT-I}} \rangle|_{\mathrm{Sig}(\texttt{CONT})}$$

where Symbols(CONT-I) is the signature of the symbols of CONT-I shown in Example 11.15, and $\Pi_{\texttt{CONT-I}}$ is the set of sentences presented in Example 11.18. For proving theorems using the normal form approach, i.e., for proving $\texttt{CONT-I} \vdash_{\mathrm{nf}} \varphi$, we can use the first-order proof system $\Pi(\mathbf{FOLEq})$ and prove $\Pi_{\texttt{CONT-I}} \vdash_{\Pi(\mathbf{FOLEq})(\mathrm{Symbols}(\texttt{CONT-I}))} \varphi$ from the hypotheses $\Pi_{\texttt{CONT-I}}$. We will see in the next section that such proofs are equivalent to proofs with the proof system $\Pi_{\texttt{CONT-I}}$ considered in Section 11.3.1. ■

11.3.3 Equivalence of $\vdash_{\Pi_{\mathrm{SP}}}$ and $\vdash_{\mathrm{nf}}$

Let us now compare the two proposals for noncompositional proof systems. As a first step we see that if $\mathrm{nf}(\mathrm{SP}) = \langle \Sigma, E \rangle|_{\Sigma'}$, then $\Pi_{\mathrm{SP}} = E$, and vice versa (in particular, $\Sigma = \mathrm{Symbols}(\mathrm{SP})$ and $\Sigma' = \mathrm{Sig}(\mathrm{SP})$).

Lemma 11.23. *For any specification expression* SP*,* $\mathrm{nf}(\mathrm{SP}) = \langle \Sigma, E \rangle|_{\Sigma'}$ *if and only if* $\Pi_{\mathrm{SP}} = E$*.*

Proof. The thesis can be shown by induction on the structure of SP. □

This lemma shows that computing the proof system Π_{SP} for a specification SP yields the same set of axioms as when computing the axioms of the normal form of SP. Hence for proving theorems of SP, there should be no difference between using the proof system Π_{SP} and using the normal form approach. That our intuition is right can be formally stated using the result of Proposition 11.7.

Proposition 11.24. *Let* SP *be a specification expression and let* φ *be an* SP*-sentence.* $\vdash_{\Pi_{\mathrm{SP}}} \varphi$ *if and only if* $\mathrm{SP} \vdash_{\mathrm{nf}} \varphi$*.*

Proof. Let $\mathrm{nf}(\mathrm{SP}) = \langle \Sigma, E \rangle|_{\Sigma'}$.

$\vdash_{\Pi_{\mathrm{SP}}} \varphi$
iff $\vdash_{\Pi(\Sigma) \cup \Pi_{\mathrm{SP}}} \varphi$ (by definition)
iff $\vdash_{\Pi(\Sigma) \cup E} \varphi$ (by Lemma 11.23)
iff $E \vdash_{\Pi(\Sigma)} \varphi$ (by Proposition 11.7)
iff $\mathrm{SP} \vdash_{\mathrm{nf}} \varphi$ (by definition). □

In summary, the two noncompositional proof systems are equivalent for specifications of the kernel language. However, they differ in the philosophy by which they were inspired. This becomes evident when the kernel language is extended by further operators (for reachability and observability) in the framework of the **FOLEq** institution in Section 11.7. In this case, the normal form approach needs infinitary axioms while proof systems for specifications use semiformal rules (for instance, infinitary induction). That is, a proof system Π_{SP} consists not only of axioms (as above) but may also contain rules designed according to the form of SP.

11.4 Structured proof systems

The disadvantage of the approaches detailed in Section 11.3 is not only the technical complexity of computing a normal form or a proof system, which can be of importance for larger specifications, but also the loss of structure. The structured nature of the specification is ignored, and too many axioms may cause the proof to become hard to handle. In this section we give a modular alternative, in the spirit of the proof system of Chapter 4, which takes the structure of specifications into account.

The structured proof system, called Π_s, is presented, as usual, by a set of axioms and rules which now define a relation $\vdash_{\Pi_s}$ between specification expressions and sentences. This proof system was inspired by the structured proof system of [Wir91]. It is based on the given proof system Π for the institution INS.

Definition 11.25 (Π_s). The relation $\vdash_{\Pi_s}$ between specifications SP and SP-sentences φ is inductively defined as follows:

(basic) $\quad \langle \Sigma, E \rangle \vdash_{\Pi_s} \varphi \quad$ for all $\varphi \in E$

(pi) $$\frac{\mathrm{SP} \vdash_{\Pi_s} \varphi_i \ (i \in I)}{\mathrm{SP} \vdash_{\Pi_s} \varphi} \quad \text{if } \{\varphi_i : i \in I\} \vdash_{\Pi(\mathrm{Sig}(\mathrm{SP}))} \varphi$$

(sum-l) $$\frac{\mathrm{SP}_1 \vdash_{\Pi_s} \varphi}{\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2 \vdash_{\Pi_s} \mathit{inl}(\varphi)}$$

where $\mathit{inl} = \mathrm{po}_1(\Sigma \hookrightarrow \mathrm{Sig}(\mathrm{SP}_1), \Sigma \hookrightarrow \mathrm{Sig}(\mathrm{SP}_2))$

(sum-r) $$\frac{\mathrm{SP}_2 \vdash_{\Pi_s} \varphi}{\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2 \vdash_{\Pi_s} \mathit{inr}(\varphi)}$$

where $\mathit{inr} = \mathrm{po}_2(\Sigma \hookrightarrow \mathrm{Sig}(\mathrm{SP}_1), \Sigma \hookrightarrow \mathrm{Sig}(\mathrm{SP}_2))$

(exp) $$\frac{\mathrm{SP} \vdash_{\Pi_s} \varphi}{\mathrm{SP}|_\Sigma \vdash_{\Pi_s} \varphi} \quad \text{if } \varphi \in \mathrm{Sen}(\Sigma)$$

(ren) $$\frac{\mathrm{SP} \vdash_{\Pi_s} r^{-1}(\varphi)}{\texttt{rename}\ \mathrm{SP}\ \texttt{by}\ r \vdash_{\Pi_s} \varphi}$$

The proof system Π_s is sound as shown in the next theorem.

Theorem 11.26 (Soundness of Π_s). *Let* SP *be a specification expression and let* φ *be an* SP*-sentence. If* $\mathrm{SP} \vdash_{\Pi_s} \varphi$*, then* $\mathrm{SP} \models_{\mathrm{Sig}(\mathrm{SP})} \varphi$.

Proof. The theorem is proved by induction on the length of the derivation for $\mathrm{SP} \vdash_{\Pi_s} \varphi$. □

Example 11.27. Consider the specification `CONT-I` of Example 11.11. Let us prove `CONT-I` $\vdash_{\Pi_s} \varphi$, where φ is the sentence $x \neq y \Rightarrow \texttt{member}(x, \texttt{add}(y, s)) = \texttt{member}(x, s)$ (used as an axiom in the specification `CONT` of Example 11.10):

1. $\texttt{MEMBER} \vdash_{\Pi_s} s \neq \texttt{emptyst} \Rightarrow$
$((\texttt{top}(s) = x \Rightarrow \texttt{member}(x, s) = \texttt{true}) \wedge$
$(\texttt{top}(s) \neq x \Rightarrow \texttt{member}(x, s) = \texttt{member}(x, \texttt{pop}(s))))$
(basic), (pi)
2. $\texttt{STACK} \vdash_{\Pi_s} \texttt{push}(y, s) \neq \texttt{emptyst}$ (basic), (pi)
3. $\texttt{STACK-EXT} \vdash_{\Pi_s} s \neq \texttt{emptyst} \Rightarrow$
$((\texttt{top}(s) = x \Rightarrow \texttt{member}(x, s) = \texttt{true}) \wedge$
$(\texttt{top}(s) \neq x \Rightarrow \texttt{member}(x, s) = \texttt{member}(x, \texttt{pop}(s))))$
(sum-r), (1)
4. $\texttt{STACK-EXT} \vdash_{\Pi_s} \texttt{push}(y, s) \neq \texttt{emptyst}$ (sum-l), (2)
5. $\texttt{STACK-EXT} \vdash_{\Pi_s} \texttt{push}(y, s) \neq \texttt{emptyst} \Rightarrow$
$((\texttt{top}(\texttt{push}(y, s)) = x \Rightarrow \texttt{member}(x, \texttt{push}(y, s)) = \texttt{true}) \wedge$
$(\texttt{top}(\texttt{push}(y, s)) \neq x \Rightarrow \texttt{member}(x, \texttt{push}(y, s)) =$
$\texttt{member}(x, \texttt{pop}(\texttt{push}(y, s)))))$
(pi), (3)
6. $\texttt{STACK-EXT} \vdash_{\Pi_s} (\texttt{top}(\texttt{push}(y, s)) = x \Rightarrow \texttt{member}(x, \texttt{push}(y, s)) = \texttt{true}) \wedge$
$(\texttt{top}(\texttt{push}(y, s)) \neq x \Rightarrow \texttt{member}(x, \texttt{push}(y, s)) =$
$\texttt{member}(x, \texttt{pop}(\texttt{push}(y, s))))$
(pi), (4,5)
7. $\texttt{STACK} \vdash_{\Pi_s} \texttt{top}(\texttt{push}(y, s)) = y$ (basic), (pi)
8. $\texttt{STACK-EXT} \vdash_{\Pi_s} \texttt{top}(\texttt{push}(y, s)) = y$ (sum-l), (7)
9. $\texttt{STACK-EXT} \vdash_{\Pi_s} (y = x \Rightarrow \texttt{member}(x, \texttt{push}(y, s)) = \texttt{true}) \wedge$
$(y \neq x \Rightarrow \texttt{member}(x, \texttt{push}(y, s)) =$
$\texttt{member}(x, \texttt{pop}(\texttt{push}(y, s))))$
(pi), (6,8)
10. $\texttt{STACK} \vdash_{\Pi_s} \texttt{pop}(\texttt{push}(y, s)) = s$ (basic), (pi)
11. $\texttt{STACK-EXT} \vdash_{\Pi_s} \texttt{pop}(\texttt{push}(y, s)) = s$ (sum-l), (10)
12. $\texttt{STACK-EXT} \vdash_{\Pi_s} (y = x \Rightarrow \texttt{member}(x, \texttt{push}(y, s)) = \texttt{true}) \wedge$
$(y \neq x \Rightarrow \texttt{member}(x, \texttt{push}(y, s)) = \texttt{member}(x, s))$
(pi), (9,11)
13. $\texttt{STACK-EXT} \vdash_{\Pi_s} y \neq x \Rightarrow \texttt{member}(x, \texttt{push}(y, s)) = \texttt{member}(x, s)$ (pi), (12)
14. $\texttt{rename STACK-EXT by } r \vdash_{\Pi_s}$
$y \neq x \Rightarrow \texttt{member}(x, \texttt{add}(y, s)) = \texttt{member}(x, s)$ (ren), (13)
15. $\texttt{CONT-I} \vdash_{\Pi_s} y \neq x \Rightarrow \texttt{member}(x, \texttt{add}(y, s)) = \texttt{member}(x, s)$ (exp), (14)
16. $\texttt{CONT-I} \vdash_{\Pi_s} x \neq y \Rightarrow \texttt{member}(x, \texttt{add}(y, s)) = \texttt{member}(x, s)$ (pi), (15)

■

In the rest of this section we address the completeness of Π_s. The issue is converse to that of soundness, that is, whether $\mathrm{SP} \models_{\mathrm{Sig(SP)}} \varphi$ implies $\mathrm{SP} \vdash_{\Pi_s} \varphi$. To begin, we consider the institution **FOLEq** and the (sound and complete) proof system $\Pi(\mathbf{FOLEq})$ of Example 11.6. We show that in this case the structured proof system Π_s (based on $\Pi(\mathbf{FOLEq})$) is complete if the axioms of the basic specifications are closed sentences.

Theorem 11.28. *Let* SP *be a specification expression in* **FOLEq** *such that the axioms of the basic specifications are closed. Let* φ *be an* SP*-sentence. If* $\mathrm{SP} \models_{\mathrm{Sig(SP)}} \varphi$*, then* $\mathrm{SP} \vdash_{\Pi_s} \varphi$*.*

Proof. The proof is by induction on the structure of SP. The critical case is when SP is of the form $\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2$, for which Theorem 11.20, as well as the interpolation property, are needed [Cen94]. □

In the general case, the completeness of Π_s is ensured if, in addition to the assumptions of Section 11.2.2, we have:[14]

- INS has conjunctions and negations, that is, for each $\Sigma \in |\mathrm{Sign}|$, $\mathrm{Sen}(\Sigma)$ is closed under finite conjunctions and negations (with the obvious meaning);
- INS is compact, that is, whenever $\Phi \models_\Sigma \varphi$, then there exists a finite set $\Psi \subseteq \Phi$ such that $\Psi \models_\Sigma \varphi$.
- Π is finitary, complete, and satisfies both the deduction theorem (see, e.g., [Men87]) and the interpolation property.

Thereby we say that Π satisfies the interpolation property if the following holds: if $\varphi_1 \vdash_{\Pi(\Sigma)} \varphi_2$ where φ_i is a Σ_i-sentence with $\Sigma_i \subseteq \Sigma$ ($i = 1, 2$), then there exists a $(\Sigma_1 \cap \Sigma_2)$-sentence[15] φ such that $\varphi_1 \vdash_{\Pi(\Sigma)} \varphi$ and $\varphi \vdash_{\Pi(\Sigma)} \varphi_2$ [Bar77,BM86].[16] The sentence φ is called interpolant. In equational logic, for example, interpolation does not hold [BHK90].[17] Classical and intuitionistic first-order predicate calculi [BM86], as well as the logic of partial terms [Bee85], satisfy interpolation.

If INS has denumerable conjunctions and negations and Π satisfies the conditions stated above, apart from being finitary, i.e., Π has a semiformal rule of the form

$$\frac{\varphi_i : i \in I}{\varphi}$$

with denumerable I, then Π_s is complete. In particular, Theorem 11.28 can be extended to the institution **IFOLEq** of infinitary first-order logic (see Example 11.6).

The structured proof system Π_s has the advantage, mentioned above, that it handles the modular structure of the specification of interest. Nevertheless, in practice, one does not work purely in this way: sometimes we *do* compute a halfway normal form, in particular, when dealing with the combination of two specifications by means of the sum. That is, the techniques presented in Section 11.3 and in the present subsection deal with the extremes, but a system designer normally works somewhere in between. An interesting discussion about this subject can be found in [HST94].

[14] See also [Bor98].

[15] Here we need to assume, moreover, that the category of signatures has pullbacks, i.e., that INS is reasonable [DGS93].

[16] A different notion of interpolation (defined on the semantic level) is given in [DGS93], where sets of sentences rather than single sentences are allowed as hypotheses.

[17] Note, however, that equational logic satisfies the interpolation property as defined in [DGS93].

11.5 Proof systems for refinement

A system specified in an abstract manner is implemented by any structure in its class of models. This structure is usually specified by adding design decisions to the original specification in a stepwise way. Refinement is a partial order relation that holds between a more "abstract" and a more "concrete" specification of that chain. That is, two specifications are in this relation if they have the same signature and if the model class of the "concrete" specification is included in the model class of the "abstract" specification, which is expressed by the following definition; see Chapter 7 and also [ST88, SW83,Wir91].

Definition 11.29. A specification SPI *refines* (or *implements*) a specification SP, written $\mathrm{SP} \rightsquigarrow \mathrm{SPI}$, if $\mathrm{Sig}(\mathrm{SPI}) = \mathrm{Sig}(\mathrm{SP})$ and $\mathrm{Mod}(\mathrm{SPI}) \subseteq \mathrm{Mod}(\mathrm{SP})$.

The refinement relation has some obvious properties: given a signature Σ and sets of Σ-sentences E and E', if $E' \vdash_{\Pi(\Sigma)} \varphi$ for every $\varphi \in E$, then $\langle \Sigma, E \rangle \rightsquigarrow \langle \Sigma, E' \rangle$. Furthermore, all specification-building operators considered here are monotonic with respect to $\rightsquigarrow$, e.g., if $\mathrm{SP} \rightsquigarrow \mathrm{SPI}$, then $\mathrm{SP} +_\Sigma \mathrm{SP}' \rightsquigarrow \mathrm{SPI} +_\Sigma \mathrm{SP}'$ (provided the specification expressions are well formed).

In the following we present a proof system, denoted by $\Pi_{\rightsquigarrow_s}$, which allows us to derive a binary relation $\mathrm{SP} \rightsquigarrow_s \mathrm{SPI}$ inductively according to the modular structure of the specification SP, such that whenever $\mathrm{SP} \rightsquigarrow_s \mathrm{SPI}$, then SPI refines SP. Thereby we assume given an arbitrary sound proof system Π_{spec} for the derivation of the theorems φ of a specification expression SP (see Sections 11.3 and 11.4). We write $\mathrm{SP} \vdash \varphi$ if φ can be proved by the axioms and rules of Π_{spec}. In order to introduce $\Pi_{\rightsquigarrow_s}$ we need the notion of extension.

Definition 11.30. A specification SP_2 is a *(syntactic) extension* of a specification SP_1 if $\mathrm{Sig}(\mathrm{SP}_1) \subseteq \mathrm{Sig}(\mathrm{SP}_2)$. SP_2 is a *conservative* extension of SP_1 if it is a syntactic extension of SP_1 such that $\mathrm{Mod}(\mathrm{SP}_1) = \mathrm{Mod}(\mathrm{SP}_2)|_{\mathrm{Sig}(\mathrm{SP}_1)}$.

Definition 11.31 (Proof system $\Pi_{\rightsquigarrow_s}$ for refinement). The proof system $\Pi_{\rightsquigarrow_s}$ for the refinement relation consists of the following axiom scheme and inference rules, that inductively define a relation $\rightsquigarrow_s$ between specifications:

(ref-basic) $\quad \langle \Sigma, E \rangle \rightsquigarrow_s \mathrm{SPI} \quad$ if $\mathrm{Sig}(\mathrm{SPI}) = \Sigma$ and $\mathrm{SPI} \vdash \varphi$ for every $\varphi \in E$

(ref-sum) $\quad \dfrac{\mathrm{SP}_1 \rightsquigarrow_s \mathrm{SPI}|_{inl},\ \mathrm{SP}_2 \rightsquigarrow_s \mathrm{SPI}|_{inr}}{\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2 \rightsquigarrow_s \mathrm{SPI}} \quad$ if $\mathrm{Sig}(\mathrm{SPI}) = \mathrm{Sig}(\mathrm{SP}_1 +_\Sigma \mathrm{SP}_2)$

where $inl = \mathrm{po}_1(\Sigma \hookrightarrow \mathrm{Sig}(\mathrm{SP}_1), \Sigma \hookrightarrow \mathrm{Sig}(\mathrm{SP}_2))$
and $inr = \mathrm{po}_2(\Sigma \hookrightarrow \mathrm{Sig}(\mathrm{SP}_1), \Sigma \hookrightarrow \mathrm{Sig}(\mathrm{SP}_2))$

$$\text{(ref-exp)} \quad \frac{\mathrm{SP} \rightsquigarrow_s \mathrm{SPI}'}{\mathrm{SP}|_\Sigma \rightsquigarrow_s \mathrm{SPI}} \quad \begin{array}{l}\text{if } \Sigma = \mathrm{Sig}(\mathrm{SPI}) \subseteq \mathrm{Sig}(\mathrm{SP}), \text{ and} \\ \text{if } \mathrm{SPI}' \text{ is a conservative extension of SPI}\end{array}$$

$$\text{(ref-ren)} \quad \frac{\mathrm{SP} \rightsquigarrow_s \texttt{rename}\ \mathrm{SPI}\ \texttt{by}\ r^{-1}}{\texttt{rename}\ \mathrm{SP}\ \texttt{by}\ r \rightsquigarrow_s \mathrm{SPI}}$$

Theorem 11.32 (Soundness and completeness of $\Pi_{\rightsquigarrow_s}$). *Given specifications* SP *and* SPI*, if* SP $\rightsquigarrow_s$ SPI*, then* SP $\rightsquigarrow$ SPI*. The converse holds if* Π_{spec} *is complete.*

Proof. Soundness can be proved by induction on the length of the derivation, completeness by induction on the structure of SP. □

Remark 11.33. The proof system $\Pi_{\rightsquigarrow_s}$ has the following difficulty. Rule (ref-exp) requires an appropriate conservative extension of SPI to be found. That means, completeness is achieved at the cost of a proof obligation which is of semantic nature. Assume, for the moment, that we are working in the institution **FOLEq**. Then, to prove conservativeness, we have to show that each model of SPI can be extended by interpretations of the symbols $\mathrm{Sig}(\mathrm{SPI}') \setminus \mathrm{Sig}(\mathrm{SPI})$, in such a way that the extension is a model of SPI$'$. In general, this has to be done by a semantical construction. However, if the axioms for the function symbols in $\mathrm{Sig}(\mathrm{SPI}') \setminus \mathrm{Sig}(\mathrm{SPI})$ are in a constructive form, such as explicit or implicit definition, then it is well known that conservativeness can be checked by syntactic conditions (under mild assumptions).

Example 11.34. Let `CONT` and `CONT-I` be the specifications of Examples 11.10 and 11.11, respectively. In order to prove `CONT` $\rightsquigarrow_s$ `CONT-I`, we only have to show that `CONT-I` $\vdash \varphi$ for every $\varphi \in \mathrm{Axs}($`CONT`$)$ (since `CONT` is a basic specification). This can be done using the structured proof system of Section 11.4, as illustrated in Example 11.27. More elaborate structured proofs of implementation relations will be shown in Example 11.48 in the context of parameterized specifications, and in Example 11.70 in the context of the reachability and observability operators. ■

11.6 Parameterization

In this section we define parameterized specifications, together with associated proof systems. There are many practical examples that motivate parameterization, for instance, storing structures like stacks, lists, and queues, which are parameterized with respect to the elements to be stored.

In the literature one can find two major approaches to parameterization, the so-called pushout approach (e.g., [BG80,EM85]), and that based on the λ-calculus (e.g., [SST92,Wir90]; see also Chapter 8 for a discussion of both approaches). The drawback of the former approach is that it does not easily generalize to abstraction over arbitrary higher-order variables. The need for higher-order parameterization arises naturally: an interpreter for a language with block structure may need a stack as an actual parameter (in order to save the return address each time a new block is entered), whereby the stack is also a parameterized specification. In this chapter, we discuss only the λ-calculus parameterization mechanism. For proof systems in the framework of pushout parameterization, the reader may consult [Pad84,Pad87,Pad88].

We define a language of simply typed λ-terms whose words are the specification expressions of Section 11.2.2, together with variables, λ-abstraction, and application. This language allows the definition of parameterized specification expressions but not the specification of parameterized program modules (see Chapter 8 and [SST92]). There may be a parameter restriction (also called *semantic type*) associated with the formal parameter variable of a λ-abstraction. An application is well defined if the actual parameter refines (or implements) the parameter restriction. That is, we can require an argument to "behave" like a specification expression intended, in particular, to satisfy a number of axioms. This implies that a parameterized specification applied to an argument may yield an erroneous expression in two ways, namely, either by violating the parameter restriction, or by disregarding the formation laws (also called *syntactic typing*) of specification expressions (see Definition 11.9). In order to keep the presentation as simple as possible, we do not take syntactic typing of specification expressions into account here, and assume always that specifications are well formed. A proof system that allows the derivation of syntactic well-definedness is given in [Cen94,CW95].

In the following we define a proof system that allows us to derive the theorems of a (parameterized and instantiated) specification λ-term without considering parameter restrictions. In the same way as in Section 11.5, this proof system is used for the definition of an inference system for refinement relations between specification λ-terms. In particular, this mechanism can then be applied to the verification of parameter restrictions. By adding parameter restrictions and combining the two inference systems, we finally obtain a system for deriving theorems and refinements in the context of parameterized specifications.

11.6.1 A λ-calculus for parameterized specifications

We enhance the kernel language defined over the institution INS (see Section 11.2.2) with parameterization based on the simply typed λ-calculus. The set of types of the λ-terms is the closure under arrow application of the type *sp*, which is the base type of specification terms. As a notational convention, we choose X, Y, Z to denote variables of the λ-calculus of parameterized specifications.

Pure λ-terms are defined just as a set of typed variables, closed under λ-abstraction and application. Additionally in our case, we have as constants the *basic* specifications over the chosen institution and, moreover, the specification-building operators as further term constructors. In particular, any specification of the kernel language is a λ-term of type *sp* without variables. Types and terms are formally defined as follows.

Definition 11.35 (Types). The set T of *types* is inductively defined by

$$T ::= sp \mid (T \rightarrow T)$$

Definition 11.36 (λ-Terms). Let $Var = (Var_\tau)_{\tau \in T}$ be a family of (denumerable) sets Var_τ of variables of type τ. The family $\Lambda = (\Lambda_\tau)_{\tau \in T}$ of sets Λ_τ of λ-terms of type τ over Var and INS is inductively defined by

$$\begin{array}{rll}
\Lambda_{sp} & ::= \langle \Sigma, E \rangle & \text{for each } \Sigma \in |\mathrm{Sign}| \text{ and } E \subseteq \mathrm{Sen}(\Sigma) \\
& \mid \ \Lambda_{sp} +_\Sigma \Lambda_{sp} & \text{for each } \Sigma \in |\mathrm{Sign}| \\
& \mid \ \Lambda_{sp}|_\Sigma & \text{for each } \Sigma \in |\mathrm{Sign}| \\
& \mid \ \texttt{rename } \Lambda_{sp} \texttt{ by } r & \text{for each renaming } r \\
\Lambda_{(\tau_1 \rightarrow \tau_2)} & ::= (\lambda\, Var_{\tau_1} . \Lambda_{\tau_2}) & \\
\Lambda_{\tau_2} & ::= (\Lambda_{(\tau_1 \rightarrow \tau_2)}\ \Lambda_{\tau_1}) & \\
& \mid \ Var_{\tau_2} &
\end{array}$$

The semantics of the λ-terms is straightforward. The semantics $[\![M]\!]$ of a term M is a function on environments where an environment assigns values to variables. If e is an environment and M is of type *sp*, then $[\![M]\!]_e$ is a specification in the collection Spec of specification expressions (see Section 11.2.2), whereby we assume that the formation rules of specification expressions are fulfilled.[18] If M is of arrow type, then $[\![M]\!]_e$ is a function. More precisely, the semantics of types and λ-terms is defined as follows.

Definition 11.37 (Semantics of types and terms). The semantics $[\![\tau]\!]$ of a type τ is defined by induction on the structure of τ:

$[\![sp]\!] \stackrel{\text{def}}{=} \mathrm{Spec}$

$[\![(\tau_1 \rightarrow \tau_2)]\!] \stackrel{\text{def}}{=} ([\![\tau_1]\!] \rightarrow [\![\tau_2]\!])$, the collection of all mappings from $[\![\tau_1]\!]$ to $[\![\tau_2]\!]$.

An *environment* is a family $e = (e_\tau)_{\tau \in T}$ of functions $e_\tau \colon Var_\tau \rightarrow [\![\tau]\!]$ for each $\tau \in T$. The index τ is usually omitted and we simply write e. We let $e_{[X:=S]}$ denote the environment defined in terms of an environment e, a variable $X \in Var_\tau$, and a value $S \in [\![\tau]\!]$, by

$$(e_{[X:=S]})(Y) \stackrel{\text{def}}{=} \begin{cases} S & \text{if } Y = X, \\ e(Y) & \text{if } Y \neq X. \end{cases}$$

The semantics $[\![M]\!]$ of a λ-term M is a function on environments defined by induction on the structure of M:

$[\![\langle \Sigma, E \rangle]\!]_e \stackrel{\text{def}}{=} \langle \Sigma, E \rangle$;

$[\![M +_\Sigma N]\!]_e \stackrel{\text{def}}{=} [\![M]\!]_e +_\Sigma [\![N]\!]_e$;

$[\![M|_\Sigma]\!]_e \stackrel{\text{def}}{=} [\![M]\!]_e|_\Sigma$;

[18] Otherwise, $[\![M]\!]_e$ should be a distinct error value [Cen94].

$[\![\texttt{rename}\ M\ \texttt{by}\ r]\!]_e \stackrel{\text{def}}{=} \texttt{rename}\ [\![M]\!]_e\ \texttt{by}\ r\,;$
$[\![(\lambda X.M)]\!]_e \colon [\![\tau_1]\!] \to [\![\tau_2]\!]$ (where τ_1 is the type of X and τ_2 that of M) is the mapping defined by $[\![(\lambda X.M)]\!]_e(S) \stackrel{\text{def}}{=} [\![M]\!]_{e_{[X:=S]}}$ for all $S \in [\![\tau_1]\!]\,;$
$[\![(M\ N)]\!]_e \stackrel{\text{def}}{=} [\![M]\!]_e([\![N]\!]_e)\,;$
$[\![X]\!]_e \stackrel{\text{def}}{=} e(X)\,.$

The notions of free variables of a term (denoted by $\mathrm{FV}(M)$, if M is a λ-term) and of substitution of terms for (free) variables (denoted by $M[X := N]$, if X is a variable and M, N are λ-terms with X and N of the same type) are defined as usual. In particular, M is a ground λ-term if M contains no free variable.

Definition 11.38 (β-equality). The binary relation of β-reduction is the least set of pairs of λ-terms containing $((\lambda X.M)\ N) \to_\beta M[X := N]$[19] for every M, N, X with X and N of the same type, that commutes with the specification-building operators, λ-abstraction, and application. The binary relation of β-equality is the equivalence relation generated by the reflexive, symmetric, and transitive closure of β-reduction.

Lemma 11.39. *For any λ-terms M, N of the same type, $M =_\beta N$ implies $[\![M]\!] = [\![N]\!]$.*

This lemma can be proved by induction on the definition of β-equality. As a consequence of being a simply typed λ-calculus, for any term M there is a unique β-normal form N (i.e., N is such that $N \not\to_\beta$) with $M =_\beta N$.

The notion of β-reduction only uses substitution, that is, no name adjustment, typically through a parameter-passing morphism, takes place. A proposal in this direction in the framework of the λ-calculus approach to parameterization can be found in [Bau93].

Example 11.40 ($\texttt{CONT}_{par}$). A parameterized specification of containers in the institution **FOLEq** can be defined by

$$\texttt{CONT}_{par} \stackrel{\text{def}}{=} (\lambda X.X +_{\texttt{ELEM}} \texttt{CONT}),$$

where `ELEM` is the signature containing only the sort `Elem`, and `CONT` is the specification of containers as defined in Example 11.10. The amalgamated union ensures that no name conflict can arise by applying $\texttt{CONT}_{par}$ to an argument that contains, besides `Elem`, further symbols of `CONT`. ∎

Example 11.41 ($\texttt{CONT-I}_{par}$). In the same way we can use the specification `CONT-I`, that implements `CONT`, in order to define (at a more concrete level)

[19] We adopt the *variable convention* of [Bar81] and require that free variables and bound variables have different names. Because of the variable convention, for $M[X := N]$ we do not have to be concerned about variable capturing in the case where M is an abstraction term.

a parameterized specification of containers. That is, we define

$$\mathtt{CONT\text{-}I}_{par} \stackrel{\text{def}}{=} (\lambda X.X +_{\mathtt{ELEM}} \mathtt{CONT\text{-}I}),$$

where ELEM is the signature as in the example above. ■

11.6.2 Proving theorems of specification λ-terms

In this subsection we present a proof system Π_{ps} which allows us to derive theorems of specification λ-terms of type *sp*. This proof system extends the structured proof system Π_s of Section 11.4, and contains additional rules according to the λ-term constructors. Its judgements are of the form

$M \vdash_{\Pi_{ps}} \varphi$ where M is a ground term of type *sp* and φ is a $[\![M]\!]$-sentence.

In the same way as Π_s, Π_{ps} is also based on a given proof system Π for the underlying institution (see rule (prf-pi) below) and, moreover, on the axioms and rules of β-equality[20] (see rule (prf-refl) below).

Definition 11.42 (Π_{ps}). The relation $\vdash_{\Pi_{ps}}$ between ground λ-terms $M \in \Lambda_{sp}$ and $[\![M]\!]$-sentences φ is inductively defined as follows:

(prf-ax) $\langle \Sigma, E\rangle \vdash_{\Pi_{ps}} \varphi$ for all $\varphi \in E$

(prf-pi) $\dfrac{M \vdash_{\Pi_{ps}} \varphi_i \ (i \in I)}{M \vdash_{\Pi_{ps}} \varphi}$ if $\{\varphi_i : i \in I\} \vdash_{\Pi(\mathrm{Sig}([\![M]\!]))} \varphi$

(prf-sum-l) $\dfrac{M_1 \vdash_{\Pi_{ps}} \varphi}{M_1 +_\Sigma M_2 \vdash_{\Pi_{ps}} inl(\varphi)}$

(prf-sum-r) $\dfrac{M_2 \vdash_{\Pi_{ps}} \varphi}{M_1 +_\Sigma M_2 \vdash_{\Pi_{ps}} inr(\varphi)}$

(prf-exp) $\dfrac{M \vdash_{\Pi_{ps}} \varphi}{M|_\Sigma \vdash_{\Pi_{ps}} \varphi}$ if φ is a Σ-sentence

(prf-ren) $\dfrac{M \vdash_{\Pi_{ps}} r^{-1}(\varphi)}{\mathtt{rename}\ M\ \mathtt{by}\ r \vdash_{\Pi_{ps}} \varphi}$

(prf-refl) $\dfrac{M_1 \vdash_{\Pi_{ps}} \varphi, M_1 =_\beta M_2}{M_2 \vdash_{\Pi_{ps}} \varphi}$

Note that if $M \vdash_{\Pi_{ps}} \varphi$, then M is necessarily of basic type, that is, it cannot be a λ-abstraction. Nevertheless it may contain occurrences of the "λ" symbol, e.g., if M is of the form $((\lambda X.N_1)\ N_2)$, where N_1 is of type *sp*. In this case, theorems of M can be proved using the rule (prf-refl) in the following way. Suppose that we have derived $N_1[X := N_2] \vdash_{\Pi_{ps}} \varphi$. Then, by rule (prf-refl), we can infer $M \vdash_{\Pi_{ps}} \varphi$.

[20] Defined according to Definition 11.38.

Remark 11.43. If M is a λ-abstraction, then any sentence φ which is valid in all applications $(M\ N)$ of M to an arbitrary argument N may be considered as a theorem of M. Hence, according to the following soundness theorem, for proving a theorem φ of M, it is sufficient to prove $(M\ N) \vdash_{\Pi_{ps}} \varphi$ for all arguments N of M. For instance, let M be a parameterized specification of the form $(\lambda X.X +_{\Sigma} \mathrm{SP})$, where SP is a specification of the kernel language. If we can prove $\mathrm{SP} \vdash_{\Pi_s} \varphi$, then we can also prove $\mathrm{SP} \vdash_{\Pi_{ps}} \varphi$ (since $\Pi_s \subseteq \Pi_{ps}$). Therefore by (prf-sum-r), $N +_{\Sigma} \mathrm{SP} \vdash_{\Pi_{ps}} inr(\varphi)$ for any N of type *sp* and hence, by rule (prf-refl), $(M\ N) \vdash_{\Pi_{ps}} inr(\varphi)$ for all arguments N of M. This means that we can deduce $inr(\varphi)$ for any application of M and, therefore, $inr(\varphi)$ may be considered as a theorem of M.

Theorem 11.44 (Soundness). *Let M be a ground term of type sp and let φ be a $[\![M]\!]$-sentence. If $M \vdash_{\Pi_{ps}} \varphi$, then $[\![M]\!] \models_{\mathrm{Sig}([\![M]\!])} \varphi$.*

The theorem can be proved by induction on the length of the derivation of $M \vdash_{\Pi_{ps}} \varphi$ [Cen94]. Thereby one shows that $M \vdash_{\Pi_{ps}} \varphi$ implies $[\![M]\!] \vdash_{\Pi_s} \varphi$. This is fairly obvious if M is a β-normal form; if M is not a β-normal form, one computes the unique β-normal form N with $M =_{\beta} N$, thus the result applies to N and, by (prf-refl), also to M. The theorem is then a consequence of the correctness of Π_s (see Theorem 11.26).

Furthermore, given a ground term M of type *sp*, one can show that $[\![M]\!] \vdash_{\Pi_s} \varphi$ implies $M \vdash_{\Pi_{ps}} \varphi$. Indeed, this implication is evident since $\Pi_s \subseteq \Pi_{ps}$, and because of (prf-refl). Completeness therefore depends on the same conditions that determine the completeness of Π_s. That is, under certain assumptions on INS and Π (see Theorem 11.28 and the discussion of completeness of Π_s in Section 11.4), the proof system Π_{ps} is also complete.

Example 11.45. Let us specify containers of natural numbers by instantiating the parameterized specification $\texttt{CONT-I}_{par}$ of Example 11.41 with a specification `NAT` of the natural numbers as follows:
$\texttt{NATCONT-I} = (\texttt{CONT-I}_{par}\ \texttt{rename NAT by}\ r')$, where r' renames `Nat` into `Elem`. Let φ be the sentence $x \neq y \Rightarrow \texttt{member}(x, \texttt{add}(y, s)) = \texttt{member}(x, s)$. Now we show that $\texttt{NATCONT-I} \vdash_{\Pi_{ps}} \varphi$:

1. $\texttt{CONT-I} \vdash_{\Pi_{ps}} \varphi$ — analogously to Example 11.27
2. $(\texttt{rename NAT by}\ r') +_{\texttt{ELEM}} \texttt{CONT-I} \vdash_{\Pi_{ps}} \varphi$ — (prf-sum-r), (1)
3. $\texttt{NATCONT-I} \vdash_{\Pi_{ps}} \varphi$ — (prf-refl), (2)

Note that in the last step, rule (prf-refl) can be applied since

$$\begin{aligned}\texttt{NATCONT-I} &= (\texttt{CONT-I}_{par}\ \texttt{rename NAT by}\ r') \\ &=_{\beta} (X +_{\texttt{ELEM}} \texttt{CONT-I})[X := \texttt{rename NAT by}\ r'] \\ &= (\texttt{rename NAT by}\ r') +_{\texttt{ELEM}} \texttt{CONT-I}\ .\end{aligned}$$

Note also that φ is a `CONT`-axiom (see Example 11.10) and, moreover, that $\texttt{NATCONT-I}|_{inr} \vdash_{\Pi_{ps}} \psi$ for every $\psi \in \mathrm{Axs}(\texttt{CONT})$. ∎

11.6.3 Refinement of parameterized specifications

The refinement relation between specifications can be extended to values in $[\![\tau]\!]$, for arbitrary τ, by a standard pointwise definition. If $\tau = sp$, and if S_1 and S_2 are values in $[\![sp]\!]$, then $S_1 \rightsquigarrow S_2$ if S_2 refines S_1 according to Definition 11.29. If $\tau = (\tau_1 \to \tau_2)$ and F_1 and F_2 are values in $[\![(\tau_1 \to \tau_2)]\!]$, then $F_1 \rightsquigarrow F_2$ if $F_1(S) \rightsquigarrow F_2(S)$ for all $S \in [\![\tau_1]\!]$.

Since all the specification-building operators considered here are monotonic with respect to $\rightsquigarrow$, it can be proved that monotonicity extends to denotable values, i.e., if N_1 and N_2 are ground terms such that $[\![N_1]\!] \rightsquigarrow [\![N_2]\!]$, then $[\![(M\ N_1)]\!] \rightsquigarrow [\![(M\ N_2)]\!]$ for any ground term M of an appropriate type. As a consequence, given denotable values F_1, F_2, S_1, and S_2, if $F_1 \rightsquigarrow F_2$ and $S_1 \rightsquigarrow S_2$, then $F_1(S_1) \rightsquigarrow F_2(S_2)$ [Cen94,Cen95].

The refinement relation between specification terms of type *sp* can be derived by the following proof system, which extends the proof system for refinements presented in Section 11.5 by additional rules that handle λ-abstraction and application. Its judgements are of the form $M \rightsquigarrow_{ps} N$, where M and N are ground terms of type *sp*.

Considering the axiom scheme (ref-basic) below, one can see that this system, analogously to that introduced in Section 11.5, is based on a proof system for proving theorems of a specification. In our case, we have chosen the proof system Π_{ps} of the previous subsection. In principle, however, any other proof system could be used.

Definition 11.46 (Proof system $\Pi_{\rightsquigarrow_{ps}}$ for refinement). The proof system $\Pi_{\rightsquigarrow_{ps}}$ for the refinement relation consists of the following axiom scheme and inference rules that inductively define a relation $\rightsquigarrow_{ps}$ between ground λ-terms of type *sp*:

(ref-basic) $\langle \Sigma, E\rangle \rightsquigarrow_{ps} M \quad$ if $M \vdash_{\Pi_{ps}} \varphi$ for all $\varphi \in E$

(ref-sum) $\dfrac{M_1 \rightsquigarrow_{ps} N|_{inl}, M_2 \rightsquigarrow_{ps} N|_{inr}}{M_1 +_\Sigma M_2 \rightsquigarrow_{ps} N}$

(ref-exp) $\dfrac{M \rightsquigarrow_{ps} M_2}{M|_\Sigma \rightsquigarrow_{ps} M_1}$ $\quad$ if $\Sigma = \mathrm{Sig}([\![M_1]\!]) \subseteq \mathrm{Sig}([\![M]\!])$, and if $[\![M_2]\!]$ is a conservative extension of $[\![M_1]\!]$

(ref-ren) $\dfrac{M \rightsquigarrow_{ps} \texttt{rename}\ N\ \texttt{by}\ r^{-1}}{\texttt{rename}\ M\ \texttt{by}\ r \rightsquigarrow_{ps} N}$

(ref-refl) $\dfrac{M_1 \rightsquigarrow_{ps} N, M_1 =_\beta M_2}{M_2 \rightsquigarrow_{ps} N}$

Theorem 11.47 (Soundness and completeness of $\Pi_{\rightsquigarrow_{ps}}$). *Let M and N be ground terms of type sp. If $M \rightsquigarrow_{ps} N$, then $[\![M]\!] \rightsquigarrow [\![N]\!]$. The converse holds if Π_{ps} is complete.*

Example 11.48. Consider the following (abstract) specification of containers of natural numbers obtained by instantiating the parameterized specification $\mathtt{CONT}_{par}$ of Example 11.40 with the specification NAT of the natural numbers: $\mathtt{NATCONT} = (\mathtt{CONT}_{par}$ **rename** NAT **by** $r')$, where r' renames Nat into Elem. Let us prove that NATCONT-I of Example 11.45 is an implementation of NATCONT, i.e., $\mathtt{NATCONT} \rightsquigarrow_{ps} \mathtt{NATCONT\text{-}I}$:

1. $\mathtt{CONT} \rightsquigarrow_{ps} \mathtt{NATCONT\text{-}I}|_{inr}$
   (ref-basic), since $\mathtt{NATCONT\text{-}I}|_{inr} \vdash_{\Pi_{ps}} \varphi$ for every $\varphi \in \mathrm{Axs}(\mathtt{CONT})$; see Example 11.45
2. $\mathtt{NAT} \rightsquigarrow_{ps} \mathtt{rename}\ (\mathtt{NATCONT\text{-}I}|_{inl})\ \mathtt{by}\ (r')^{-1}$
   (ref-basic), since one can prove $\mathtt{rename}\ (\mathtt{NATCONT\text{-}I}|_{inl})\ \mathtt{by}\ (r')^{-1} \vdash_{\Pi_{ps}} \varphi$ for every $\varphi \in \mathrm{Axs}(\mathtt{NAT})$
3. $\mathtt{rename\ NAT\ by}\ r' \rightsquigarrow_{ps} \mathtt{NATCONT\text{-}I}|_{inl}$ (ref-ren), (2)
4. $(\mathtt{rename\ NAT\ by}\ r') +_{\mathtt{ELEM}} \mathtt{CONT} \rightsquigarrow_{ps} \mathtt{NATCONT\text{-}I}$ (ref-sum), (1,3)
5. $\mathtt{NATCONT} \rightsquigarrow_{ps} \mathtt{NATCONT\text{-}I}$ (ref-refl), (4)

■

11.6.4 Semantic typing

We can demand that an actual parameter exhibits a particular behavior in order to be a valid argument of a parameterized specification. Then the derivation of the refinement relation between specification λ-terms provides a means for verifying that an argument satisfies the semantic type imposed on the parameters of a parameterized specification.

First, we enhance the calculus of parameterized specifications by allowing an additional parameter restriction, accompanying the formal parameter variable of a λ-abstraction. Formally, we define

$$\Lambda_{(\tau_1 \to \tau_2)} ::= (\lambda\, Var_{\tau_1} : \Lambda_{\tau_1} . \Lambda_{\tau_2})$$

With respect to the semantics of this generalized λ-calculus, we have to take into account that the parameter restriction may be violated by an actual parameter of a λ-abstraction. Therefore, it is necessary to introduce a distinguished semantic value $\perp$ to denote erroneous λ-applications. Hence, the semantics of the base type sp is defined by $[\![sp]\!] = \mathrm{Spec} \cup \{\perp\}$, and the semantics of arbitrary arrow types τ is inductively defined in the same way as in Section 11.6.1 over this new base case. The semantics of the term constructors that correspond to the specification-building operators (see Section 11.6.1) are extended in the obvious (strict) way. The semantics of a term $(\lambda X : R.M)$ of type $(\tau_1 \to \tau_2)$ in an environment e is a function from $[\![\tau_1]\!]$ to $[\![\tau_2]\!]$, which is now defined by

$$[\![(\lambda X : R.M)]\!]_e(S) \stackrel{\text{def}}{=} \begin{cases} [\![M]\!]_{e[X:=S]} & \text{if } [\![R]\!]_e \rightsquigarrow S \\ \perp_{\tau_2} & \text{otherwise,} \end{cases}$$

where $\perp_\tau$ is a "totally undefined" constant for each interpretation of types in T, inductively defined on the structure of τ by

$\perp_{sp} \in [[sp]] \stackrel{\text{def}}{=} \perp$
$\perp_{(\tau_1 \to \tau_2)} \in [[(\tau_1 \to \tau_2)]]$ is the constant function defined by $\perp_{(\tau_1 \to \tau_2)}(S) \stackrel{\text{def}}{=} \perp_{\tau_2}$ for each $S \in [[\tau_1]]$.

The semantics of basic specification terms, variables, and application have the same (inductive) definition as in Section 11.6.1.

Example 11.49. (a) A parameterized specification of sorted lists in **FOLEq** can be defined by $\texttt{SORT}_{par} \stackrel{\text{def}}{=} (\lambda X : \texttt{ORD}.X +_{\texttt{RELEM}} \texttt{SORT})$, where `RELEM` is the signature containing the sort `Elem` and a binary relation `R` over `Elem`, `SORT` is the usual specification of lists over elements of sort `Elem` supplied with a sorting function, and `ORD` is the specification with signature `RELEM` and the axioms of partial order for `R`. The parameter restriction ensures that any actual parameter will be equipped with a partial order, so that it makes sense to compute the (topological) sort of a list of elements of sort `Elem`.

(b) It is possible to require that one parameter refines another, by writing, for instance, $(\lambda X.(\lambda Y : X.M))$. ■

The introduction of parameter restrictions requires the redefinition of the notions of β-reduction and β-equality of Definition 11.38; we have to state as base case the following rule of *conditional* β-relation:

$$\text{(cond-beta)} \qquad \frac{R \rightsquigarrow_{ps} N}{((\lambda X : R.M)\ N) \to_\beta M[X := N]}$$

This means that, in the same way as in [Fei89], a step of β-reduction is only possible if the parameter restriction associated with the λ-abstraction can be derived for the actual parameter. Note that Lemma 11.39 remains valid with this new definition of β-equality.

In order to take into account parameter restrictions (i.e., semantic typing) in the proof systems, we replace the axiom of β-reduction (see Definition 11.38) with the rule (cond-beta), and generalize the systems Π_{ps} and $\Pi_{\rightsquigarrow_{ps}}$ to terms of the extended language (whereby the axioms and rules remain unchanged). Note that the systems are now mutually dependent, since as before $\Pi_{\rightsquigarrow_{ps}}$ uses proofs of Π_{ps} (see rule (ref-basic)) and Π_{ps} uses proofs of β-equality (see rule (prf-refl)), for which proofs of refinements are now necessary (see rule (cond-beta)). These derivations of refinement relations may be considered as parameter-passing verification.

The soundness results of Theorems 11.44 and 11.47 carry over to the new system. Moreover, if the conditions for completeness of Π_s (see the discussions in Sections 11.4 and 11.6.2) are satisfied, then the new system is also complete (for ground terms of base type).

Remark 11.50. Similarly to Remark 11.43, if M is a λ-abstraction of the form $(\lambda X : R.M')$, then for proving a theorem φ over M it is now sufficient to prove

$(M\ N) \vdash_{\Pi_{ps}} \varphi$ for all correct arguments N of M. For this it is enough to prove $M'[X := R] \vdash_{\Pi_{ps}} \varphi$, because $R \rightsquigarrow_{ps} N$ holds for any correct parameter N of M and hence, by monotonicity, $M'[X := N] \vdash_{\Pi_{ps}} \varphi$. Thus, by (prf-refl), $((\lambda X : R.M')\ N) \vdash_{\Pi_{ps}} \varphi$. This means φ is a theorem of any (valid) application of $(\lambda X : R.M)$.

Further Extensions. The drawback of the parameter restrictions in their present form is that they require an actual parameter to match *exactly* the signature of the parameter restriction. This can be relaxed by requiring an argument to implement the parameter restriction R when restricted to the signature of R, i.e., if $[\![R]\!]_e \rightsquigarrow S|_{\mathrm{Sig}([\![R]\!]_e)}$ then $[\![(\lambda X : R.M)]\!]_e(S) \stackrel{\mathrm{def}}{=} [\![M]\!]_{e[X:=S]}$. Therefore, in order to be able to derive that an actual parameter conforms to a parameter restriction, we have to compute the signature of parameter restrictions and thus further terms denoting signatures are necessary.

If we have a signature base type, say *sg*, and terms denoting signatures, then we could use them in those places where a signature is necessary, for instance, $M|_N$ if M is of type *sp* and N of type *sg*. Soon we discover that specification-building operators also need to become signature-building operators, and the function Sig has to play the role of a term constructor as well. For a detailed study, see [Cen94].

In summary, the defined λ-calculus, together with the extensions discussed, supplies a powerful typing mechanism whose syntactic angle is relaxed with respect to previous approaches and its semantic angle is given more expressiveness. By permitting the derivation of syntactic and semantic typing, parameter verification is supported.

An objective of further research is an extension of Π_{ps} and $\Pi_{\rightsquigarrow_{ps}}$ such that it will be possible to derive $M \vdash_{\Pi_{ps}} \varphi$, and $M \rightsquigarrow_{ps} N$, for M and N not necessarily of type *sp* but of the same arbitrary type.

11.7 Reachability and observability operators

The kernel specification language considered in Sections 11.2–11.5 will now be extended in two directions. On the one hand, we will consider a *reachability* specification-building operator, able to express *reachability constraints* (see Chapter 5, Section 4). On the other hand, we will consider two new operators, *observational behavior* and *observational quotient*, that are necessary to express properties related to the observable behavior of software systems (see Chapter 5, Section 4). The motivations for considering observability operators are strongly related to the refinement relation between specifications as discussed in Section 11.5. Indeed, in contrast with the idea that requirements should describe those properties which have to be satisfied by implementations, many classical refinements of, for instance, set-like data structures do not preserve all requirements axioms. A careful analysis shows that only the observable consequences of the axioms are preserved. This means that instead of attempting to prove that a specification SPI is a refinement of a given requirements specification SP (an attempt that will often fail in practice), it is much more satisfactory to attempt to prove that the specification SPI is a refinement of the specification "behavior of SP", where the latter is assumed to capture the observable consequences of the requirements provided by SP. For a more detailed discussion of the refinement relation and the related problems, see Chapter 7.

11.7.1 Extension of the kernel specification language

In order to define the new specification-building operators, we will assume that specifications are built in the framework of the **FOLEq** institution (see Example 11.6, Section 11.2.1); unfortunately it is not convenient to define these operators in the framework of an arbitrary institution, since we will rely heavily on the (internal) structure of the signatures considered. However, note that even if for technical reasons we work within the framework of the **FOLEq** institution, nevertheless we assume, in addition, that all specification signatures are countable and contain no predicate symbols, i.e., they are of the form (S, F), with S and F countable.[21] To define the new specification-building operators, we first need some preliminary definitions.

Definition 11.51 (Preliminary technical definitions). Let $\Sigma = (S, F)$ be a signature.

1. The *reachability constraint* over Σ induced by a subset $\Omega \subseteq F$ of function symbols is the pair $\mathcal{R}_\Omega = (S_\Omega, \Omega)$, with $S_\Omega = \{s \in S \mid \text{there exist } f \in \Omega \text{ with } \mathrm{range}(f) = s\}$. A sort $s \in S_\Omega$ is called a *constrained sort* and a function symbol $f \in \Omega$ is called a *constructor symbol* (or briefly constructor).
2. A *constructor term* of sort s is a term $t \in T_{\Sigma'}(X')_s$, where $\Sigma' = (S, \Omega)$, $X' = (X'_s)_{s \in S}$ with $X'_s = X_s$, if $s \in S \backslash S_\Omega$, and $X'_s = \emptyset$, if $s \in S_\Omega$ (thereby $X = (X_s)_{s \in S}$ is the generally assumed family of countably infinite sets X_s of variables of sort $s \in S$). The set of constructor terms is denoted by T^Ω_Σ.
3. A Σ-algebra A satisfies the reachability constraint induced by Ω, denoted by $A \models \mathcal{R}_\Omega$, if, for all $s \in S$ and for all $a \in A_s$, there exists a constructor term $t \in T_{\Sigma'}(X')_s$ of sort s and a valuation $\alpha \colon X' \to A$, such that $I_\alpha(t) = a$. (Thereby $I_\alpha \colon T_{\Sigma'}(X') \to A$ denotes the unique Σ'-morphism

[21] Indeed the reachability specification-building operator could be easily defined for specifications containing predicate symbols. The assumption that specification signatures do not contain predicate symbols is necessary for the observability operators, since otherwise the definition of the observational equality (see Chapter 5, Section 4) would have to be extended to take into account predicate symbols. However, note that predicate symbols, though not used in specification signatures, will be used later on when defining the hidden symbols of a specification.

that extends the valuation α. Note that the definition is independent of X since X_s is countably infinite for all $s \in S$.)

4. Let $\mathrm{In} \subseteq S$ be a set of sorts. The signature Σ is *sensible* w.r.t. In if for all $s \in S \setminus \mathrm{In}$ (and hence for all $s \in S$), there exists a term t of sort s built with function symbols in Σ and variables of sort $s \in \mathrm{In}$.
5. Let Obs and In be two sets of sorts such that $\mathrm{Obs} \subseteq S$ and $\mathrm{In} \subseteq S$, and such that Σ is sensible w.r.t. In. Then for any Σ-algebra A, $\approx_{\mathrm{Obs},\mathrm{In},A}$ denotes the *(partial) observational equality* on A induced by Obs and In (see Chapter 5, Section 4).
   $\mathrm{Beh}_{\approx_{\mathrm{Obs},\mathrm{In}}}$ is the corresponding *observational behavior operator* defined by, for any class C of Σ-algebras:
   $\mathrm{Beh}_{\approx_{\mathrm{Obs},\mathrm{In}}}(C) \stackrel{\mathrm{def}}{=} \{A \in \mathrm{Alg}(\Sigma) \mid A/{\approx_{\mathrm{Obs},\mathrm{In},A}} \in C\}$.
   $./{\approx_{\mathrm{Obs},\mathrm{In}}}$ is the corresponding *observational quotient operator* defined by, for any class C of Σ-algebras: $C/{\approx_{\mathrm{Obs},\mathrm{In}}} \stackrel{\mathrm{def}}{=} \{A/{\approx_{\mathrm{Obs},\mathrm{In},A}} \mid A \in C\}$.

We now extend the kernel specification language defined in Section 11.2.2 by new specification-building primitives:

Definition 11.52 (Structured specifications). Definition 11.9 (instantiated by the specific institution **FOLEq**) is extended by the following specification-building primitives:

5. If SP is a specification expression, and if Ω is a set of function symbols such that $\Omega \subseteq \mathrm{Opns}(\mathrm{SP})$, then `reach` SP `w.r.t.` Ω is a specification expression.
   $\mathrm{Sig}(\texttt{reach}\ \mathrm{SP}\ \texttt{w.r.t.}\ \Omega) \stackrel{\mathrm{def}}{=} \mathrm{Sig}(\mathrm{SP})$
   $\mathrm{Mod}(\texttt{reach}\ \mathrm{SP}\ \texttt{w.r.t.}\ \Omega) \stackrel{\mathrm{def}}{=} \{A \in \mathrm{Mod}(\mathrm{SP}) \mid A \models \mathcal{R}_\Omega\}$
   where $\mathcal{R}_\Omega$ is the reachability constraint over Sig(SP) induced by Ω.
6. If SP is a specification expression, if Obs and In are two sets of sorts such that $\mathrm{Obs} \subseteq \mathrm{Sorts}(\mathrm{SP})$ and $\mathrm{In} \subseteq \mathrm{Sorts}(\mathrm{SP})$, and if, moreover, Sig(SP) is sensible w.r.t. In, then `behavior` SP `w.r.t.` Obs, In is a specification expression.
   $\mathrm{Sig}(\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs},\mathrm{In}) \stackrel{\mathrm{def}}{=} \mathrm{Sig}(\mathrm{SP})$
   $\mathrm{Mod}(\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs},\mathrm{In}) \stackrel{\mathrm{def}}{=} \mathrm{Beh}_{\approx_{\mathrm{Obs},\mathrm{In}}}(\mathrm{Mod}(\mathrm{SP}))$
   where $\mathrm{Beh}_{\approx_{\mathrm{Obs},\mathrm{In}}}$ is the observational behavior operator defined above.
7. Under the same assumptions as in 6 above, $\mathrm{SP}/{\approx_{\mathrm{Obs},\mathrm{In}}}$ is a specification expression.
   $\mathrm{Sig}(\mathrm{SP}/{\approx_{\mathrm{Obs},\mathrm{In}}}) \stackrel{\mathrm{def}}{=} \mathrm{Sig}(\mathrm{SP})$
   $\mathrm{Mod}(\mathrm{SP}/{\approx_{\mathrm{Obs},\mathrm{In}}}) \stackrel{\mathrm{def}}{=} \mathrm{Iso}(\mathrm{Mod}(\mathrm{SP})/{\approx_{\mathrm{Obs},\mathrm{In}}})$
   where Iso closes the given class under isomorphism, and $./{\approx_{\mathrm{Obs},\mathrm{In}}}$ is the observational quotient operator defined above.

An observational abstraction specification-building primitive [SW83,ST88] can be defined as a derived operator, using the relationships between observational behavior and observational abstraction (see Chapter 5):

$$\texttt{abstract}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs},\mathrm{In} \stackrel{\mathrm{def}}{=} \texttt{behavior}\ (\mathrm{SP}/{\approx_{\mathrm{Obs},\mathrm{In}}})\ \texttt{w.r.t.}\ \mathrm{Obs},\mathrm{In}\,.$$

Unfortunately it is impossible to define sound and complete *finitary* proof systems for specifications built with reachability and observability operators in the framework of the **FOLEq** institution. We will therefore study below two alternatives: the first is to use infinitary proof rules, hence defining semi-formal proof systems. The second is to use infinitary sentences as nonlogical axioms, hence defining the proof systems in the framework of the **IFOLEq** institution (see Example 11.6). In both cases we will use the fact that there is a natural "embedding" of the **FOLEq** institution into the **IFOLEq** institution. As a consequence, any specification SP of our specification language (defined in the framework of the **FOLEq** institution) can also be considered as a specification of the corresponding specification language defined in the framework of the **IFOLEq** institution. The important point is that, while the reachability and observability operators are proper new operators when working in the framework of the **FOLEq** institution (since they increase the expressive power of the kernel specification language), this is no longer the case when we work in the framework of the **IFOLEq** institution: then the three new operators can indeed be considered as derived operators of the kernel specification language, as explained in the next subsection.

11.7.2 Reachability and observability operators as derived operators

In this subsection we show that, in the framework of the **IFOLEq** institution, the reachability, observational behavior, and observational quotient operators can be considered as derived operators of the specification language.

The reachability operator.

Proposition 11.53. *Let* SPR *be a specification expression of the form* `reach` SP `w.r.t.` Ω *, and let* $\Sigma = (S, F) = \mathrm{Sig}(\mathrm{SP})$ *(hence* $\Omega \subseteq F$*). Then* SPR *is* equivalent *to the specification* $\mathrm{SP} + \langle \Sigma, \{\mathrm{GEN}_s^\Omega \mid s \in S_\Omega\}\rangle$ *, where* S_Ω *is defined as in Definition 11.51(1) and where* GEN_s^Ω *is the infinitary sentence defined by* $\mathrm{GEN}_s^\Omega \stackrel{\mathrm{def}}{=} \forall x{:}s.\, \bigvee_{t \in (T_\Sigma^\Omega)_s} \exists \mathrm{Var}(t).\, x = t$ *. Thereby* $\exists \mathrm{Var}(t)$ *is an abbreviation for* $\exists x_1 : s_1. \dots \exists x_n : s_n$ *where* $x_1, \dots, x_n$ *are the variables (of sort* $s_1, \dots, s_n$*) of the term* t*.*

Proof. A Σ-algebra A is a model of SPR if and only if A is a model of SP and, moreover, $A \models \mathcal{R}_\Omega$. But $A \models \mathcal{R}_\Omega$ if and only if $A \models \mathrm{GEN}_s^\Omega$, for each $s \in S_\Omega$. □

For each sort $s \in S_\Omega$, the infinitary disjunction GEN_s^Ω expresses exactly that all elements of sort s are denotable by some constructor term $t \in (T_\Sigma^\Omega)_s$.[22]

[22] Infinitary formulas of this kind were used in [MVS85].

The observational behavior operator. The observational behavior operator can also be considered as a derived operator of the kernel specification language, provided we allow predicate symbols in the signatures of specifications and, again, infinitary axioms. The main idea is to introduce on the one hand unary predicate symbols T_s , together with an appropriate axiomatization $\mathrm{Ax}[T_s]$, such that, for each algebra A satisfying $\mathrm{Ax}[T_s]$, the interpretation of T_s in A is just the domain of $\approx_{\mathrm{Obs,In},A}$ (w.r.t. the sort s). On the other hand, we introduce binary predicate symbols $\sim_s$ and an axiomatization $\mathrm{Ax}[\sim_s]$, such that $\sim_s$ denotes the observational equality $\approx_{\mathrm{Obs,In},A}$ for each algebra A.[23] Since in the definition of the observational behavior operator, Σ-algebras A and their observational quotient $A/{\approx_{\mathrm{Obs,In},A}}$ are related, the next idea is to encode both algebras within a single structure. For describing the connection between any Σ-algebra A and its quotient $A/{\approx_{\mathrm{Obs,In},A}}$, we first consider them as algebras over different (isomorphic) signatures Σ and $\mathrm{Copy}(\Sigma)$, and we specify an appropriate "pseudo" epimorphism c which connects sorts of Σ with their copy in $\mathrm{Copy}(\Sigma)$.

Proposition 11.54. *Let* SPB *be a specification expression of the form* `behavior` SP `w.r.t.` Obs, In *, and let* $\Sigma = (S, F)$ *be the signature of* SP *(hence also of* SPB*). We have* $\mathrm{Obs} \subseteq S$, $\mathrm{In} \subseteq S$, *and the signature* Σ *is sensible w.r.t.* In. SPB *is* equivalent *to the specification* $\mathrm{SP}'|_\Sigma$ *, where* SP' *is defined by:*

$$\mathrm{SP}' = (\texttt{rename}\ \mathrm{SP}\ \texttt{by}\ \mathrm{Copy}) + \\ \langle\ (S \cup \mathrm{Copy}(S),\ F \cup \mathrm{Copy}(F) \cup \{T_s \mid s \in S \setminus \mathrm{In}\} \cup \{\sim_s, c_s \mid s \in S\}), \\ \{\mathrm{Ax}[T_s] \mid s \in S \setminus \mathrm{In}\} \cup \{\mathrm{Ax}[\sim_s] \mid s \in S\} \cup \mathrm{Ax}[\{c_s \mid s \in S\}]\ \rangle$$

where:

1. $\mathrm{Copy}(\Sigma)$ *is a disjoint copy of the signature* Σ *(i.e., we have* $\Sigma \cap \mathrm{Copy}(\Sigma) = \emptyset$*), and* $\mathrm{Copy}\colon \Sigma \to \mathrm{Copy}(\Sigma)$ *is the corresponding bijective signature morphism.*
2. *For each sort* $s \in S \setminus \mathrm{In}$*,* T_s *is a unary predicate symbol of domain* s*, intended to denote the values of sort* s *for which the partial observational equality is defined.*
3. *For each sort* $s \in S$*,* $\sim_s$ *is an infix binary predicate symbol of domain* s, s*, intended to denote the values of sort* s *which are observationally equal.*
4. *For each sort* $s \in S$*,* $c_s\colon s \to \mathrm{Copy}(s)$ *is a new function symbol (i.e.,* $c_s \notin \Sigma \cup \mathrm{Copy}(\Sigma)$*) connecting the sort* s *and its copy* $\mathrm{Copy}(s)$*.*
5. *For each sort* $s \in S \setminus \mathrm{In}$*,* $\mathrm{Ax}[T_s]$ *is the infinitary sentence defined by:*
   $\mathrm{Ax}[T_s] \stackrel{\mathrm{def}}{=} \forall x{:}s.\ \Big[T_s(x) \Leftrightarrow \bigvee_{t \in T_\Sigma(X_{\mathrm{In}})_s} \exists \mathrm{Var}(t).\, x = t\Big]$,
   where $(X_{\mathrm{In}})_s \stackrel{\mathrm{def}}{=} X_s$*, if* $s \in \mathrm{In}$*, and* $(X_{\mathrm{In}})_s \stackrel{\mathrm{def}}{=} \emptyset$*, if* $s \in S \setminus \mathrm{In}$*.*
6. *For each sort* $s \in S$*,* $\mathrm{Ax}[\sim_s]$ *is the infinitary sentence defined by:*
   $\mathrm{Ax}[\sim_s] \stackrel{\mathrm{def}}{=} \forall x, y{:}s.\ \Big[x \sim_s y \Leftrightarrow \bigwedge_{c \in T_\Sigma(X_{\mathrm{In}} \cup \{z_s\})} \forall \mathrm{Var}(c).\, c[x] = c[y]\Big]$
   if $s \in \mathrm{In}$*,*

[23] Another approach which axiomatizes the observational equality in higher-order logic is presented in [HS96].

$\mathrm{Ax}[\sim_s] \stackrel{\mathrm{def}}{=} \forall x, y : s.$
$$\Big[x \sim_s y \Leftrightarrow T_s(x) \wedge T_s(y) \wedge \textstyle\bigwedge_{c \in T_\Sigma(X_{\mathrm{In}} \cup \{z_s\})} \forall \mathrm{Var}(c).\, c[x] = c[y]\Big]$$
if $s \in S \setminus \mathrm{In}$*,*
where $T_\Sigma(X_{\mathrm{In}} \cup \{z_s\})$ *is the set of the observable contexts, with context variable* z_s *of sort* s *(see Chapter 5, Section 4),* $\mathrm{Var}(c)$ *denotes the set of the variables of the context* c*, apart from the context variable* z_s *, and* $c[x]$ *and* $c[y]$ *denote respectively the terms obtained by substituting the variables* x *and* y *for the context variable* z_s *in* c*.*

7. $\mathrm{Ax}[\{c_s \mid s \in S\}] \stackrel{\mathrm{def}}{=} [\mathrm{Hom}] \cup [\mathrm{Epi}] \cup [\sim=]$ *, where:*
[Hom] *is the union, for each function symbol* f
with profile $s_1, \ldots, s_n \to s$ *in* F*, of the sentence:*
$$\forall x_1 : s_1 . \ldots . \forall x_n : s_n .\, T_{s_{r_1}}(x_{r_1}) \wedge \ldots \wedge T_{s_{r_k}}(x_{r_k}) \Rightarrow$$
$$c_s(f(x_1, \ldots, x_n)) = \mathrm{Copy}(f)(c_{s_1}(x_1), \ldots, c_{s_n}(x_n))\,,$$
where $s_{r_1}, \ldots, s_{r_k}$ *are the non input sorts among the argument sorts of* f*,*
[Epi] *is the union of:*
- *for each sort* $s \in \mathrm{In}$*, the sentence:*
  $\forall x' : \mathrm{Copy}(s) . \exists x : s .\, x' = c_s(x)\,,$
- *for each sort* $s \in S \setminus \mathrm{In}$*, the sentence:*
  $\forall x' : \mathrm{Copy}(s) . \exists x : s .\, T_s(x) \wedge x' = c_s(x)\,,$

[$\sim=$] *is the union of:*
- *for each sort* $s \in \mathrm{In}$*, the sentence:*
  $\forall x, y : s .\, x \sim_s y \Leftrightarrow c_s(x) = c_s(y)\,,$
- *for each sort* $s \in S \setminus \mathrm{In}$*, the sentence:*
  $\forall x, y : s .\ [T_s(x) \wedge T_s(y) \Rightarrow (x \sim_s y \Leftrightarrow c_s(x) = c_s(y))]\,.$

Proof. Let B be a model of SP', and let $A \stackrel{\mathrm{def}}{=} B|_\Sigma$ (hence A is a model of $\mathrm{SP}'|_\Sigma$). The axioms $\mathrm{Ax}[T_s]$ ensure that the interpretations of the predicate symbols T_s in B coincide with $\mathrm{Dom}(\approx_{\mathrm{Obs,In},A})$. The axioms $\mathrm{Ax}[\sim_s]$ ensure that the interpretations of the predicate symbols $\sim_s$ in B coincide with $\approx_{\mathrm{Obs,In},A}$. The axioms $\mathrm{Ax}[\{c_s \mid s \in S\}]$ ensure that the interpretations of the function symbols c_s in B can be considered as defining a surjective Σ-morphism from $\mathrm{Dom}(\approx_{\mathrm{Obs,In},A})$ to $(B|_{\mathrm{Copy}(\Sigma)})|_{\mathrm{Copy}}$, the kernel of which coincides with the kernel of the quotient projection from $\mathrm{Dom}(\approx_{\mathrm{Obs,In},A})$ to $A/\!\approx_{\mathrm{Obs,In},A}$. Hence $(B|_{\mathrm{Copy}(\Sigma)})|_{\mathrm{Copy}}$ (which, by definition of SP', is a model of SP) is isomorphic to $A/\!\approx_{\mathrm{Obs,In},A}$. Hence models of SP' are exactly those algebras comprising a Σ-part A and a $\mathrm{Copy}(\Sigma)$-part which, when restricted along Copy to Σ, is both a model of SP and isomorphic to $A/\!\approx_{\mathrm{Obs,In},A}$. This is enough to show that SPB and $\mathrm{SP}'|_\Sigma$ are equivalent specifications (for more details, see [Hen97]). □

The observational quotient operator. To show that the observational quotient operator is a derived operator of the kernel specification language, we use exactly the same ideas as for the observational behavior operator, but now in the "reverse" direction.

Proposition 11.55. *Let* SPQ *be a specification expression of the form* $\mathrm{SP}/{\approx_{\mathrm{Obs},\mathrm{In}}}$ *, and let* $\Sigma = (S, F)$ *be the signature of* SP *(hence also of* SPQ*). We have* $\mathrm{Obs} \subseteq S$*,* $\mathrm{In} \subseteq S$*, and the signature* Σ *is sensible w.r.t.* In. SPQ *is* equivalent *to the specification* $\mathrm{SP}'|_\Sigma$ *, where* SP' *is defined by:*

$$\begin{aligned}
\mathrm{SP}' = {} & (\texttt{rename}\ \mathrm{SP}\ \texttt{by}\ \mathrm{Copy}) + \\
& \langle\ (\ S \cup \mathrm{Copy}(S),\ F \cup \mathrm{Copy}(F) \cup \{T_{\mathrm{Copy}(s)} \mid s \in S \setminus \mathrm{In}\} \cup \\
& \qquad \{\sim_{\mathrm{Copy}(s)}, c_{\mathrm{Copy}(s)} \mid s \in S\}\ ), \\
& \quad \{\ \mathrm{Ax}[T_{\mathrm{Copy}(s)}] \mid s \in S \setminus \mathrm{In}\} \cup \{\mathrm{Ax}[\sim_{\mathrm{Copy}(s)}] \mid s \in S\} \cup \\
& \qquad \mathrm{Ax}[\{c_{\mathrm{Copy}(s)} \mid s \in S\}]\ \rangle
\end{aligned}$$

where:

1. *As for the observational behavior operator,* $\mathrm{Copy}(\Sigma)$ *is a disjoint copy of the signature* Σ*, and* $\mathrm{Copy}\colon \Sigma \to \mathrm{Copy}(\Sigma)$ *is the corresponding bijective signature morphism.*
2. *For each sort* $s \in S \setminus \mathrm{In}$*,* $T_{\mathrm{Copy}(s)}$ *is a unary predicate symbol of domain* $\mathrm{Copy}(s)$ *intended to denote the values of sort* $\mathrm{Copy}(s)$ *for which the partial observational equality (w.r.t.* $\mathrm{Copy}(\mathrm{Obs})$ *and* $\mathrm{Copy}(\mathrm{In})$*) is defined.*
3. *For each sort* $s \in S$*,* $\sim_{\mathrm{Copy}(s)}$ *is an infix binary predicate symbol of domain* $\mathrm{Copy}(s), \mathrm{Copy}(s)$ *intended to denote the values of sort* $\mathrm{Copy}(s)$ *which are observationally equal (w.r.t.* $\mathrm{Copy}(\mathrm{Obs})$ *and* $\mathrm{Copy}(\mathrm{In})$*).*
4. *For each sort* $s \in S$*,* $c_{\mathrm{Copy}(s)}\colon \mathrm{Copy}(s) \to s$ *is a new function symbol (i.e.,* $c_{\mathrm{Copy}(s)} \notin \Sigma \cup \mathrm{Copy}(\Sigma)$*) connecting the sort* $\mathrm{Copy}(s)$ *to* s*.*
5. *For each sort* $s \in S \setminus \mathrm{In}$*,* $\mathrm{Ax}[T_{\mathrm{Copy}(s)}]$ *is the same infinitary sentence as* $\mathrm{Ax}[T_s]$*, but now with* $\mathrm{Copy}(\Sigma)$*,* $\mathrm{Copy}(\mathrm{In})$*, and* $\mathrm{Copy}(s)$ *instead of* Σ*,* In*, and* s *respectively.*
6. *For each sort* $s \in S$*,* $\mathrm{Ax}[\sim_{\mathrm{Copy}(s)}]$ *is the same infinitary sentence as* $\mathrm{Ax}[\sim_s]$*, but now with* $\mathrm{Copy}(\Sigma)$*,* $\mathrm{Copy}(\mathrm{In})$*, and* $\mathrm{Copy}(s)$ *instead of* Σ*,* In*, and* s *respectively.*
7. $\mathrm{Ax}[\{c_{\mathrm{Copy}(s)} \mid s \in S\}]$ *is defined similarly to* $\mathrm{Ax}[\{c_s \mid s \in S\}]$*, but w.r.t. the function symbols* $c_{\mathrm{Copy}(s)}$*, instead of the function symbols* c_s*.*

Proof. Similar to that of Proposition 11.54. It is easy to show that the models of SP' comprise a $\mathrm{Copy}(\Sigma)$-part A^{Copy}, which is simply a renamed version of a model A of SP, and a Σ-part isomorphic to $A/{\approx_{\mathrm{Obs},\mathrm{In},A}}$. This is sufficient to show that SPQ and $\mathrm{SP}'|_\Sigma$ are equivalent specifications. □

11.7.3 Noncompositional proof systems

Propositions 11.53–11.55 show that the reachability and observability operators can be considered as derived operators of the kernel specification language, in the framework of the **IFOLEq** institution. Using these results, it is easy to define, for each specification SP of our specification language (defined in the framework of the **FOLEq** institution), an equivalent normal form nf(SP) which, however, is then a specification with infinitary axioms, and hence a specification in the framework of the **IFOLEq** institution. Then, using a sound and complete proof system $\Pi(\mathbf{IFOLEq})$ for the institution **IFOLEq** (for instance, that presented in [Kei71]), we obtain a sound and complete proof method for proving theorems of SP (see Section 11.3.2).

We will now show how the noncompositional approach of Section 11.3.1, where an individual proof system is constructed for each specification expression, can be extended to take into account the reachability and observability operators, by introducing adequate additional (infinitary) proof rules. Let us stress that, for this noncompositional approach, we will stay strictly within the framework of the **FOLEq** institution (and hence we will just rely on the standard proof system $\Pi(\mathbf{FOLEq})$ for first-order logic). In that case, the results of Propositions 11.53–11.55 cannot be used directly. However, they provide us with the necessary intuition about which additional rules are necessary: the idea we shall use is to "reflect" the infinitary axioms of the **IFOLEq** equivalent specifications by adequate infinitary proof rules. In the following, we study each new specification-building operator independently, and then we define the noncompositional proof system associated with each specification expression.

Rules for the reachability operator. We will define the proof system associated with `reach` SP `w.r.t.` Ω as being the proof system for SP suitably enriched by some (infinitary) rules which, intuitively speaking, reflect the axioms GEN_s^{Ω}. To achieve this, we define the infinitary induction rules associated with a signature Σ and a set of constructors Ω:

Definition 11.56 (Infinitary induction rules). Let $\Sigma = (S, F)$ be a signature and $\Omega \subseteq F$ be a set of constructors. The infinitary induction rules associated with Σ and Ω, denoted by $(\mathrm{iI}_{\Sigma,\Omega})$, are given by the countable set $(\mathrm{iI}_{\Sigma,\Omega}) \stackrel{\mathrm{def}}{=} \{(\mathrm{iI}_{\Sigma,\Omega,s}) \mid s \in S_\Omega\}$ of infinitary proof rules:

$$(\mathrm{iI}_{\Sigma,\Omega,s}) \qquad \frac{\varphi[t/x] \text{ for all } t \in (T_\Sigma^\Omega)_s}{\forall x\!:\!s.\,\varphi}$$

where φ is an arbitrary first-order formula built with arbitrary symbols and with at least a free variable x of sort s.

The infinitary induction rules $(\mathrm{iI}_{\Sigma,\Omega})$ express that a formula $\forall x\!:\!s.\,\varphi$ is derivable if all instantiations $\varphi[t/x]$, where the variable x is replaced by the constructor terms t, are derivable. The soundness of the infinitary induction rules is obvious (compare with the infinitary sentences GEN_s^{Ω}).

Rules for the observability operators. We can use Proposition 11.54 to understand how to derive a proof system for `behavior` SP `w.r.t.` Obs, In from a proof system for SP: we will add to the proof system for `rename` SP `by` Copy (obtained from the proof system for SP, as explained in Section 11.3.1) new hidden predicate symbols T_s, $\sim_s$, new hidden function symbols c_s, the finitary axioms $\mathrm{Ax}[\{c_s \mid s \in S\}]$, and new finitary axioms and infinitary rules that will reflect the infinitary axioms $\mathrm{Ax}[T_s]$ and $\mathrm{Ax}[\sim_s]$.

Definition 11.57 (Infinitary rules for the observational equality). Let $\Sigma = (S, F)$ be a signature and $\mathrm{Obs}, \mathrm{In} \subseteq S$ (such that Σ is sensible w.r.t. In). Let $\{\sim_s : s\ s \mid s \in S\}$ be a set of binary predicate symbols and let $\{T_s : s \mid s \in S \setminus \mathrm{In}\}$ be a set of unary predicate symbols.

1. $(\mathrm{T}_{\Sigma,\mathrm{In}})$ is the following countable set of axioms:
   $\{T_s(t) \mid t \in T_\Sigma(X_{\mathrm{In}})_s,\ s \in S \setminus \mathrm{In}\}$
2. $(\mathrm{iT}_{\Sigma,\mathrm{In}})$ is the countable set $(\mathrm{iT}_{\Sigma,\mathrm{In}}) \stackrel{\mathrm{def}}{=} \{(\mathrm{iT}_{\Sigma,\mathrm{In},s}) \mid s \in S \setminus \mathrm{In}\}$ of infinitary proof rules for relativized induction:
   $$(\mathrm{iT}_{\Sigma,\mathrm{In},s}) \quad \frac{\varphi[t/x] \text{ for all } t \in T_\Sigma(X_{\mathrm{In}})_s}{\forall x{:}s.\, T_s(x) \Rightarrow \varphi}$$
   where φ is an arbitrary first-order formula built with arbitrary symbols and with at least a free variable x of sort s.
3. $(\mathrm{iCI}_{\sim,\Sigma,\mathrm{Obs},\mathrm{In}})$ is the countable set $(\mathrm{iCI}_{\sim,\Sigma,\mathrm{Obs},\mathrm{In}}) \stackrel{\mathrm{def}}{=} \{(\mathrm{iCI}_{\sim,\Sigma,\mathrm{Obs},\mathrm{In},s}) \mid s \in S\}$ of infinitary context induction proof rules, defined as follows.
   If $s \in \mathrm{In}$ then:
   $$(\mathrm{iCI}_{\sim,\Sigma,\mathrm{Obs},\mathrm{In},s}) \quad \frac{\varphi \vee \forall \mathrm{Var}(c).\, c[x] = c[y] \text{ for all } c \in T_\Sigma(X_{\mathrm{In}} \cup \{z_s\})}{\varphi \vee x \sim_s y}$$
   If $s \in S \setminus \mathrm{In}$ then $(\mathrm{iCI}_{\sim,\Sigma,\mathrm{Obs},\mathrm{In},s})$ is the rule:
   $$\frac{\varphi \vee (T_s(x) \wedge T_s(y) \wedge \forall \mathrm{Var}(c).\, c[x] = c[y]) \text{ for all } c \in T_\Sigma(X_{\mathrm{In}} \cup \{z_s\})}{\varphi \vee x \sim_s y}$$
   In the above rules, φ denotes an arbitrary first-order formula.
4. $(\mathrm{Comp}_{\sim,\Sigma,\mathrm{Obs},\mathrm{In}})$ is the following countable set of compatibility axioms:
   $\{\forall x, y{:}s.\, x \sim_s y \Rightarrow c[x] = c[y] \mid s \in S,\ c \in T_\Sigma(X_{\mathrm{In}} \cup \{z_s\})\} \cup$
   $\{\forall x, y{:}s.\, x \sim_s y \Rightarrow T_s(x) \wedge T_s(y) \mid s \in S \setminus \mathrm{In}\}$

The axioms $(\mathrm{T}_{\Sigma,\mathrm{In}})$, together with the rules $(\mathrm{iT}_{\Sigma,\mathrm{In}})$, reflect the infinitary axioms $\mathrm{Ax}[T_s]$, and the rules $(\mathrm{iCI}_{\sim,\Sigma,\mathrm{Obs},\mathrm{In}})$, together with the axioms $(\mathrm{Comp}_{\sim,\Sigma,\mathrm{Obs},\mathrm{In}})$, reflect the infinitary axioms $\mathrm{Ax}[\sim_s]$.

Similarly, the proof system for $\mathrm{SP}/{\approx_{\mathrm{Obs},\mathrm{In}}}$ will be derived from the proof system for `rename` SP `by` Copy by adding new hidden predicate symbols $T_{\mathrm{Copy}(s)}$, $\sim_{\mathrm{Copy}(s)}$, new hidden function symbols $c_{\mathrm{Copy}(s)}$, the finitary axioms $\mathrm{Ax}[\{c_{\mathrm{Copy}(s)} \mid s \in S\}]$, and new finitary axioms and infinitary rules that will reflect the infinitary axioms $\mathrm{Ax}[T_{\mathrm{Copy}(s)}]$ and $\mathrm{Ax}[\sim_{\mathrm{Copy}(s)}]$. For this purpose, we again use the infinitary rules for the observational equality defined in Definition 11.57 (but now w.r.t. $\mathrm{Copy}(\Sigma)$, Copy(Obs), and Copy(In)).

Definition of the proof systems. We now have all the necessary ingredients to define the proof system associated with a structured specification that may contain reachability or observability operators. First, we define, as in Section 11.3.1, the set of all visible and hidden symbols of such specifications.

Definition 11.58 (Symbols of a structured specification). The inductive definition of the symbols of a specification (see Definition 11.14) is extended as follows:

5. If SP is a specification expression of the form `reach` SP' `w.r.t.` Ω, then $\mathrm{Symbols}(\mathrm{SP}) \stackrel{\mathrm{def}}{=} \mathrm{Symbols}(\mathrm{SP}')$.
6. If SP is a specification expression of the form `behavior` SP' `w.r.t.` $\mathrm{Obs}, \mathrm{In}$, then $\mathrm{Symbols}(\mathrm{SP}) \stackrel{\mathrm{def}}{=} \mathrm{PO}(\mathrm{Copy}, \mathrm{Sig}(\mathrm{SP}') \hookrightarrow \mathrm{Symbols}(\mathrm{SP}')) \ \cup\ \mathrm{Sig}(\mathrm{SP}') \cup \{T_s \mid s \in S \setminus \mathrm{In}\} \cup \{\sim_s, c_s \mid s \in S\}$.
7. If SP is a specification expression of the form $\mathrm{SP}'/\!\approx_{\mathrm{Obs,In}}$, then $\mathrm{Symbols}(\mathrm{SP}) \stackrel{\mathrm{def}}{=} \mathrm{PO}(\mathrm{Copy}, \mathrm{Sig}(\mathrm{SP}') \hookrightarrow \mathrm{Symbols}(\mathrm{SP}')) \cup\ \mathrm{Sig}(\mathrm{SP}') \cup \{T_{\mathrm{Copy}(s)} \mid s \in S \setminus \mathrm{In}\} \cup \{\sim_{\mathrm{Copy}(s)}, c_{\mathrm{Copy}(s)} \mid s \in S\}$.

Since pushout signatures are defined up to isomorphism, we can always choose the pushout signature used above in (6) and (7) to be a disjoint copy of $\mathrm{Symbols}(\mathrm{SP}')$ which contains $\mathrm{Copy}(\mathrm{Sig}(\mathrm{SP}'))$. Note also that, although specification signatures do not contain predicate symbols, symbol signatures may now contain (hidden) predicate symbols. Without loss of generality, we assume that all newly introduced symbols are fresh.

The proof system associated with a specification expression can now be defined as follows:

Definition 11.59 (Extension of Π_{SP}). Definition 11.16 of the proof system Π_{SP} associated with a specification expression SP is extended as follows:

5. If SP is a specification expression of the form `reach` SP' `w.r.t.` Ω, then $\Pi_{\mathrm{SP}} \stackrel{\mathrm{def}}{=} \Pi_{\mathrm{SP}'} \cup (\mathrm{iI}_{\mathrm{Sig}(\mathrm{SP}),\Omega})$.
6. If SP is a specification expression of the form `behavior` SP' `w.r.t.` $\mathrm{Obs}, \mathrm{In}$, then $\Pi_{\mathrm{SP}} \stackrel{\mathrm{def}}{=} \mathrm{Copy}^\star(\Pi_{\mathrm{SP}'}) \cup (\mathrm{T}_{\mathrm{Sig}(\mathrm{SP}),\mathrm{In}}) \cup (\mathrm{iT}_{\mathrm{Sig}(\mathrm{SP}),\mathrm{In}}) \cup (\mathrm{iCI}_{\sim,\mathrm{Sig}(\mathrm{SP}),\mathrm{Obs},\mathrm{In}}) \cup (\mathrm{Comp}_{\sim,\mathrm{Sig}(\mathrm{SP}),\mathrm{Obs},\mathrm{In}}) \cup \mathrm{Ax}[\{c_s \mid s \in S\}]$.
7. If SP is a specification expression of the form $\mathrm{SP}'/\!\approx_{\mathrm{Obs,In}}$, then
$$\begin{aligned}\Pi_{\mathrm{SP}} \stackrel{\mathrm{def}}{=}\ & \mathrm{Copy}^\star(\Pi_{\mathrm{SP}'}) \cup (\mathrm{T}_{\mathrm{Copy}(\mathrm{Sig}(\mathrm{SP})),\mathrm{Copy}(\mathrm{In})}) \cup \\ & (\mathrm{iT}_{\mathrm{Copy}(\mathrm{Sig}(\mathrm{SP})),\mathrm{Copy}(\mathrm{In})}) \cup \\ & (\mathrm{iCI}_{\sim,\mathrm{Copy}(\mathrm{Sig}(\mathrm{SP})),\mathrm{Copy}(\mathrm{Obs}),\mathrm{Copy}(\mathrm{In})}) \cup \\ & (\mathrm{Comp}_{\sim,\mathrm{Copy}(\mathrm{Sig}(\mathrm{SP})),\mathrm{Copy}(\mathrm{Obs}),\mathrm{Copy}(\mathrm{In})}) \cup \\ & \mathrm{Ax}[\{c_{\mathrm{Copy}(s)} \mid s \in S\}]\,.\end{aligned}$$

where in (6) and (7), $\mathrm{Copy}^\star \stackrel{\mathrm{def}}{=} \mathrm{po}_2(\mathrm{Copy}, \mathrm{Sig}(\mathrm{SP}') \hookrightarrow \mathrm{Symbols}(\mathrm{SP}'))$, i.e., $\mathrm{Copy}^\star$ produces a disjoint copy of $\mathrm{Symbols}(SP')$.

As in Section 11.3.1, for any specification SP, the proof system Π_{SP} is combined with the proof system $\Pi(\mathbf{FOLEq})$ for deriving SP-sentences. We write $\vdash_{\Pi_{\mathrm{SP}}} \varphi$ if φ is an SP-sentence such that $\vdash_{\Pi(\mathbf{FOLEq})(\Sigma)\cup\Pi_{\mathrm{SP}}} \varphi$, where $\Sigma = \mathrm{Symbols}(\mathrm{SP})$. The next theorem states that by this method we obtain a sound and complete proof system for SP (since $\Pi(\mathbf{FOLEq})$ is complete).

Theorem 11.60 (Soundness and completeness of the proof systems). *Let* SP *be a specification expression and let* φ *be an* SP*-sentence.*
$\mathrm{SP} \models_{\mathrm{Sig(SP)}} \varphi$ *if and only if* $\vdash_{\Pi_{\mathrm{SP}}} \varphi$.

For a proof (which relies on the extended omitting-types theorem for showing the completeness of the infinitary rules), see [HWB97].

11.7.4 Structured proof systems

Let us now discuss how to obtain a structured proof system, in the sense of Section 11.4, for the reachability and observability operators. Unfortunately, it is not yet known how to define a sound and complete structured proof system (even with infinitary rules) for our extended specification language in the framework of the **FOLEq** institution. Therefore we will again use the results established in Section 11.7.2 and work within the **IFOLEq** institution. In that case, we could derive directly from Propositions 11.53–11.55, proof rules for the reachability, observational behavior, and observational quotient operators. It seems therefore rather easy to obtain a sound and complete structured proof system for the extended specification language in the framework of the **IFOLEq** institution. However, let us stress that the structured proof system obtained in that way is perhaps not so useful from a practical point of view, since the proof rules for the reachability and observability operators are quite complex.[24] The aim of this subsection is to show that we can define "practically better" proof rules. For this we again study each specification-building operator separately.

The reachability operator. Let us consider a specification of the form `reach` SP `w.r.t.` Ω, and let $\Sigma = (S, F)$ be the signature of SP (hence $\Omega \subseteq F$). A direct application of Proposition 11.53 leads to the following proof rule:

$$(\mathrm{R}) \qquad \frac{\mathrm{SP} + \langle \Sigma, \{\mathrm{GEN}_s^{\Omega} \mid s \in S_\Omega\}\rangle \vdash_{\Pi_s} \varphi}{\texttt{reach}\ \mathrm{SP}\ \texttt{w.r.t.}\ \Omega \vdash_{\Pi_s} \varphi}$$

Using the rule (sum-l), we derive from (R) the following rule:

$$(\mathrm{reach}) \qquad \frac{\mathrm{SP} \vdash_{\Pi_s} \varphi}{\texttt{reach}\ \mathrm{SP}\ \texttt{w.r.t.}\ \Omega \vdash_{\Pi_s} \varphi}$$

Using the rules (sum-r) and (basic), we derive from (R) the following rules:

$$(\text{reach-GEN}_s) \qquad \texttt{reach}\ \mathrm{SP}\ \texttt{w.r.t.}\ \Omega \vdash_{\Pi_s} \mathrm{GEN}_s^{\Omega}\ \text{(for each sort } s \in S_\Omega)$$

[24] This criticism also applies to the noncompositional proof systems defined in the previous subsection.

It is intuitively clear that the set of rules $\{(\text{reach})\} \cup \{(\text{reach-GEN}_s) \mid s \in S_\Omega\}$ is equivalent to rule (R) above, and it will provide a set of sound and complete structured proof rules for the reachability operator. Now, to understand how this set of rules will allow us to prove theorems of specifications with reachability constraints, let us stress that the following rule is a derived rule of $\Pi(\mathbf{IFOLEq})$, for each sort $s \in S_\Omega$:

$$(\text{iI}_s) \qquad \frac{\text{GEN}_s^\Omega,\ \varphi[t/x] \text{ for all } t \in (T_\Sigma^\Omega)_s}{\forall x\!:\!s.\,\varphi}$$

where φ is an arbitrary formula composed of arbitrary symbols and at least a free variable x of sort s.

Hence, whenever we consider a specification expression SP' containing a subspecification of the form `reach` SP `w.r.t.` Ω, the sentences GEN_s^Ω can first be propagated to the larger specification SP' using the rules of the structured proof system of Definition 11.25, and then we can apply the infinitary induction rules (iI_s) to SP'. In practice, for proving the infinitely many hypotheses $\varphi[t/x]$ for all $t \in (T_\Sigma^\Omega)_s$, one would use an induction scheme like structural induction with respect to $(T_\Sigma^\Omega)_s$.

Example 11.61. As an example let us consider the usual specification `NAT` of the natural numbers with constructors $\Omega = \{\mathtt{0}, \mathtt{succ}\}$ and the following extension `NAT1` of `NAT`:

spec `NAT1` $=$ `NAT` $+$ `NATf`

where

spec `NATf`
  sorts `Nat`
  functions `0` : $\to$ `Nat`
    `succ, f` : `Nat` $\to$ `Nat`
  axioms $\forall n : \mathtt{Nat}.\ \mathtt{f}(\mathtt{0}) = \mathtt{0} \wedge \mathtt{f}(\mathtt{succ}(n)) = \mathtt{f}(n)$
endspec

Let us prove that $\mathtt{NAT1} \vdash_{\Pi_s} \forall n : \mathtt{Nat}.\ \mathtt{f}(n) = \mathtt{0}$:

1. $\mathtt{NATf} \vdash_{\Pi_s} \forall n : \mathtt{Nat}.\ \mathtt{f}(\mathtt{0}) = \mathtt{0} \wedge \mathtt{f}(\mathtt{succ}(n)) = \mathtt{f}(n)$ (basic)
2. $\mathtt{NAT1} \vdash_{\Pi_s} \forall n : \mathtt{Nat}.\ \mathtt{f}(\mathtt{0}) = \mathtt{0} \wedge \mathtt{f}(\mathtt{succ}(n)) = \mathtt{f}(n)$ (sum-r), (1)
3. $\mathtt{NAT1} \vdash_{\Pi_s} \mathtt{f}(t) = \mathtt{0}$ for all $t \in (T^\Omega_{\mathrm{Sig}(\mathtt{NAT})})_{\mathtt{Nat}}$
   by induction on the structure of the terms t, using (2)
4. $\mathtt{NAT} \vdash_{\Pi_s} \text{GEN}^\Omega_{\mathtt{Nat}}$ $(\text{reach-GEN}_{\mathtt{Nat}})$
5. $\mathtt{NAT1} \vdash_{\Pi_s} \text{GEN}^\Omega_{\mathtt{Nat}}$ (sum-l), (4)
6. $\mathtt{NAT1} \vdash_{\Pi_s} \forall n : \mathtt{Nat}.\ \mathtt{f}(n) = \mathtt{0}$ (pi) with $(\text{iI}_{\mathtt{Nat}})$, using (5) and (3)

∎

Remark 11.62. Let us stress that a more naive approach would lead to an incomplete set of structured proof rules. Indeed, instead of the rules (reach-GEN_s) above (which require the use of infinitary sentences), one could be tempted to try to avoid infinitary sentences by introducing just the following infinitary proof rules (for each $s \in S_\Omega$):

$$(\text{reach-iI}_s) \quad \frac{\texttt{reach}\ \text{SP}\ \texttt{w.r.t.}\ \Omega \vdash_{\Pi_s} \varphi[t/x] \text{ for all } t \in (T_\Sigma^\Omega)_s}{\texttt{reach}\ \text{SP}\ \texttt{w.r.t.}\ \Omega \vdash_{\Pi_s} \forall x\!:\!s.\,\varphi}$$

These rules are obviously sound, but unfortunately not complete. For instance, in the example above, it would not be possible to derive the desired theorem using just the rules (reach), (reach-iI$_s$), and the structured rules of Definition 11.25. The problem is that the rules (reach-iI$_{\texttt{Nat}}$) can only be applied to NAT, but not to the larger specification NAT1. This clearly demonstrates why it is unfortunately not possible to define the structured proof system for specifications with reachability constraints in the framework of the institution **FOLEq** (at least not in this way).

The observability operators. In Section 11.7.2 we defined an infinitary axiomatization of the observational equality $\approx_{\mathrm{Obs,In}}$, using auxiliary predicate symbols T_s and $\sim_s$ and infinitary axioms $\mathrm{Ax}[T_s]$ and $\mathrm{Ax}[\sim_s]$ (i.e., again formulas of the institution **IFOLEq**), which characterize the observational equality $\approx_{\mathrm{Obs,In},A}$ for each Σ-algebra A. We will again use this infinitary axiomatization, but in a different way compared to Proposition 11.55:

Proposition 11.63. *Let us consider a specification expression of the form* $\mathrm{SP}/\!\approx_{\mathrm{Obs,In}}$ *, and let* $\Sigma = (S, F)$ *be the signature of* SP*. We have* $\mathrm{Obs} \subseteq S$*,* $\mathrm{In} \subseteq S$*, and the signature* Σ *is sensible w.r.t.* In*. Let* φ *be a closed* Σ*-formula. Let* $\mathcal{L}(\Sigma) \stackrel{\text{def}}{=} \Sigma \cup \{T_s \mid s \in S \setminus \mathrm{In}\} \cup \{\sim_s \mid s \in S\}$ *and let* $\mathrm{AX}[\mathcal{L}] \stackrel{\text{def}}{=} \{\mathrm{Ax}[T_s] \mid s \in S \setminus \mathrm{In}\} \cup \{\mathrm{Ax}[\sim_s] \mid s \in S\}$ *(see Proposition 11.54).*
$\mathrm{SP}/\!\approx_{\mathrm{Obs,In}} \models \varphi$ *if and only if* $\mathrm{SP} + \langle \mathcal{L}(\Sigma), \mathrm{AX}[\mathcal{L}]\rangle \models \mathcal{L}(\varphi)$ *, where* $\mathcal{L}(\varphi)$ *is a syntactic translation of* φ *inductively defined as follows:*[25]

1. *If* φ *is an equation* $l = r$ *between two terms* l, r *of sort* s*,*
   then $\mathcal{L}(\varphi) \stackrel{\text{def}}{=} l \sim_s r$ *.*
2. $\mathcal{L}(\neg\varphi) \stackrel{\text{def}}{=} \neg(\mathcal{L}(\varphi))$ *,* $\mathcal{L}(\varphi \wedge \varphi') \stackrel{\text{def}}{=} \mathcal{L}(\varphi) \wedge \mathcal{L}(\varphi')$ *.*
3. *If* $s \in \mathrm{In}$*, then* $\mathcal{L}(\forall x\!:\!s.\varphi) \stackrel{\text{def}}{=} \forall x\!:\!s.\mathcal{L}(\varphi)$ *,*
   and if $s \in S \setminus \mathrm{In}$*, then* $\mathcal{L}(\forall x\!:\!s.\varphi) \stackrel{\text{def}}{=} \forall x\!:\!s.(T_s(x) \Rightarrow \mathcal{L}(\varphi))$ *.*

For a proof see [BH96].

According to the above proposition, the proof of a theorem φ of a specification $\mathrm{SP}/\!\approx_{\mathrm{Obs,In}}$ can be reduced to the proof of a corresponding theorem, called "lifting" of φ and denoted by $\mathcal{L}(\varphi)$, with respect to an axiomatic extension of SP obtained by adding the predicate symbols T_s and $\sim_s$ and the axioms $\mathrm{AX}[\mathcal{L}]$ to SP. (Hence this extension of SP is defined in the institution **IFOLEq**.) This leads to the following proof rule:

$$(\text{quot}) \quad \frac{\mathrm{SP} + \langle \mathcal{L}(\Sigma), \mathrm{AX}[\mathcal{L}]\rangle \vdash_{\Pi_s} \mathcal{L}(\varphi)}{\mathrm{SP}/\!\approx_{\mathrm{Obs,In}} \vdash_{\Pi_s} \varphi}$$

[25] Similar constructions, called relativizations, are used in [Wir91,HS96].

where $\Sigma = \mathrm{Sig}(\mathrm{SP})$ and $\mathcal{L}(\Sigma), \mathrm{AX}[\mathcal{L}]$, $\mathcal{L}(\varphi)$ are defined in Proposition 11.63.

For the observational behavior operator we use the fact that $\mathrm{Beh}_{\approx_{\mathrm{Obs,In}}}$ and $./{\approx_{\mathrm{Obs,In}}}$ form a Galois connection. Hence, for any specification SP, we know that $\mathrm{Mod}((\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In})/\approx_{\mathrm{Obs,In}}) \subseteq \mathrm{Mod}(\mathrm{SP})$. Therefore $\mathrm{SP} \models \varphi$ implies $(\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In})/\approx_{\mathrm{Obs,In}} \models \varphi$. Using Proposition 11.63, this leads to the rule:

$$\text{(beh-1)} \qquad \frac{\mathrm{SP} \vdash_{\Pi_s} \varphi}{\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In} + \langle \mathcal{L}(\Sigma), \mathrm{AX}[\mathcal{L}] \rangle \vdash_{\Pi_s} \mathcal{L}(\varphi)}$$

where $\Sigma = \mathrm{Sig}(\mathrm{SP})$ and $\mathcal{L}(\Sigma), \mathrm{AX}[\mathcal{L}]$, $\mathcal{L}(\varphi)$ are defined in Proposition 11.63.

Now we remark that $(\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In} + \langle \mathcal{L}(\Sigma), \mathrm{AX}[\mathcal{L}] \rangle)|_{\Sigma}$ and $\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In}$ are equivalent specifications. Hence, for Σ-formulas φ, we obtain the rule:

$$\text{(beh-2)} \qquad \frac{\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In} + \langle \mathcal{L}(\Sigma), \mathrm{AX}[\mathcal{L}] \rangle \vdash_{\Pi_s} \varphi}{\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In} \vdash_{\Pi_s} \varphi}$$

where $\Sigma = \mathrm{Sig}(\mathrm{SP})$, $\mathcal{L}(\Sigma), \mathrm{AX}[\mathcal{L}]$ are defined in Proposition 11.63, and φ is a Σ-formula.

Intuitively, the idea behind the two proof rules (beh-1) and (beh-2) above is to derive theorems of $\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In}$ by using lifted versions of the theorems of SP, and then the axioms $\mathrm{AX}[\mathcal{L}]$ and the proof rules of the underlying proof system to eliminate the hidden symbols.

Definition of the structured proof system. The rules introduced in the previous subsection lead to a sound and complete proof system for specifications with reachability and observability operators. In the following, we assume that Π_s denotes the structured proof system of Definition 11.25 with the underlying proof system $\Pi(\mathbf{IFOLEq})$ of infinitary first-order logic (see Example 11.6), extended by the proof rules (reach), (reach-GEN_s), (quot), (beh-1), and (beh-2).

Theorem 11.64 (Soundness and completeness of Π_s). *Let* SP *be a specification expression (with reachability and/or observability operators) such that the axioms of the basic specifications are closed formulas. Let φ be an* SP*-sentence.* $\mathrm{SP} \models_{\mathrm{Sig(SP)}} \varphi$ *if and only if* $\mathrm{SP} \vdash_{\Pi_s} \varphi$.

For a proof (which relies on a normal form construction in the institution **IFOLEq** and on the interpolation property), see [Hen97].

The above proof system Π_s is based on infinitary first-order logic because not only the formulas GEN_s^{Ω} of (reach-GEN_s) but also the axioms $\mathrm{AX}[\mathcal{L}]$, which describe the observational equality, are infinitary sentences. In order to simplify the rules (quot), (beh-1), and (beh-2), one can try to replace the infinitary axiomatization of the observational equality by a finitary one. Unfortunately, it is generally not possible to find a finitary axiomatization of the observational equality in first-order logic (see, e.g., [Wol87,Sch92]). However, it is shown in [BH96] that the observational equality can always be axiomatized by finitary axioms if we use auxiliary (hidden) sorts and functions symbols. As a consequence, the basic specification $\langle \mathcal{L}(\Sigma), \mathrm{AX}[\mathcal{L}] \rangle$ occurring in the rules (quot), (beh-1), and (beh-2) can always be replaced by a specification of the form $\mathrm{HID} + \langle \mathcal{L}(\Sigma) \cup \mathrm{Sig}(\mathrm{HID}), \mathrm{FINAX}[\mathcal{L}] \rangle$, where HID is a specification of the **FOLEq** institution (possibly containing a reachability operator) and $\mathrm{FINAX}[\mathcal{L}]$ is a set of finitary first-order axioms which specify the observational equality with the help of the hidden sorts and function symbols of HID. Hence the use of the infinitary logic can be restricted to the formulas GEN_s^{Ω} which are themselves only necessary if reachability constraints occur in the specification of interest.

A further improvement can be obtained by noticing that we do not need a finitary axiomatization of the observational equality which is valid for arbitrary algebras, but only for the models of the underlying specification (see again [BH96]). This last remark allows us in particular to obtain finitary axiomatizations without a hidden part in many cases, so that we can replace the rule (quot) above by the rule

$$\text{(quot')} \qquad \frac{\mathrm{SP} + \langle \mathcal{L}(\Sigma), \mathrm{FINAX}[\mathcal{L}] \rangle \vdash_{\Pi_s} \mathcal{L}(\varphi)}{\mathrm{SP}/{\approx_{\mathrm{Obs,In}}} \vdash_{\Pi_s} \varphi}$$

provided $\mathrm{FINAX}[\mathcal{L}]$ is a correct finitary axiomatization of the observational equality with respect to the models of SP.[26]

Example 11.65. The following specification `STATE-I` can be considered as an observational implementation of environments (also called states) which assign natural numbers to identifiers (see Example 11.70). In the specification `STATE-I` a state is represented by a sequence of pairs consisting of an identifier and its associated value. The reachability constraint requires that states are constructed by the constant `init`, the operation $\langle .,. \rangle$ (forming the pair of an identifier and a natural number) and the operation $.\&.$ (concatenating two sequences of pairs). The specification `STATE-I` is built on top of the usual specification `NAT` of the natural numbers and a specification `ID` of identifiers. It exports the signature Σ-`STATE` which contains only the typical state operations `init`, `update`, and `lookup`.

[26] To achieve the same simplifications for the rules (beh-1) and (beh-2) would require a finitary axiomatization without a hidden part of the observational equality with respect to the models of `behavior` SP `w.r.t.` Obs, In which, in practice, is not easy to find.

spec STATE-SEQ0
  sorts Nat, Id, State
  functions $\texttt{init} \;:\; \to \texttt{State}$
    $\langle .,. \rangle \;:\; \texttt{Id, Nat} \to \texttt{State}$
    $.\&. \;:\; \texttt{State, State} \to \texttt{State}$
    $\texttt{update} \;:\; \texttt{Id, Nat, State} \to \texttt{State}$
    $\texttt{lookup} \;:\; \texttt{Id, State} \to \texttt{Nat}$
  axioms $\forall x, y : \texttt{Id},\ \ n : \texttt{Nat},\ \ s, t, u : \texttt{State}.$
    $s\&\texttt{init} = s \wedge \texttt{init}\&s = s \wedge$
    $(s\&t)\&u = s\&(t\&u) \wedge$
    $\texttt{update}(x, n, s) = \langle x, n\rangle\&s \wedge$
    $\texttt{lookup}(x, \langle x, n\rangle\&s) = n \wedge$
    $(x \neq y \Rightarrow \texttt{lookup}(x, \langle y, n\rangle\&s) = \texttt{lookup}(x, s))$
endspec

spec STATE-SEQ $=$ reach NAT $+$ ID $+$ STATE-SEQ0 w.r.t. $\{\texttt{init}, \langle .,. \rangle, .\&.\}$

spec STATE-I $=$ STATE-SEQ$|_{\Sigma\texttt{-STATE}}$
where Σ-STATE $= \mathrm{Sig}(\texttt{STATE-SEQ}) \setminus \{\langle .,. \rangle, .\&.\}$

Consider the sorts Obs $= \{\texttt{Nat}, \texttt{Id}\}$ to be observable and let all sorts be input sorts, i.e., In $=$ Sorts(STATE-I). One can show, using the method described in [BH96], that it is "enough" to consider the "crucial" context $\texttt{lookup}(x, z_{\texttt{State}})$ to obtain a finitary axiomatization of the observational equality, valid for all models of STATE-I, with $\texttt{FINAX}[\mathcal{L}]$ defined as follows:[27]

$$\texttt{FINAX}[\mathcal{L}] \stackrel{\text{def}}{=} \forall S_1, S_2 : \texttt{State}.\\ [S_1 \sim_{\texttt{State}} S_2 \Leftrightarrow (\forall x : \texttt{Id}.\ \texttt{lookup}(x, S_1) = \texttt{lookup}(x, S_2))]\,.$$

Now consider the observational quotient specification $\texttt{STATE-I}/\!\approx_{\mathrm{Obs,In}}$. We want to prove that the (implicitly universally quantified) equation $\texttt{update}(x, n, \texttt{update}(x, m, s)) = \texttt{update}(x, n, s)$ is valid in the observational quotient of STATE-I (although this equation is obviously not valid in STATE-I). For this we can first prove:

$$\texttt{STATE-I} + \langle \mathcal{L}(\mathrm{Sig}(\texttt{STATE-I})), \texttt{FINAX}[\mathcal{L}] \rangle \vdash \texttt{update}(x, n, \texttt{update}(x, m, s))\\ \sim_{\texttt{State}} \texttt{update}(x, n, s)$$

using the proof rules of the structured proof system (which is not difficult). Then, we can apply the simplified rule (quot') and we obtain:

$$\texttt{STATE-I}/\!\approx_{\mathrm{Obs,In}} \vdash \texttt{update}(x, n, \texttt{update}(x, m, s)) = \texttt{update}(x, n, s)$$

Similarly, we can prove that the update operation is commutative in the observational quotient of STATE-I, i.e., that:

$$\texttt{STATE-I}/\!\approx_{\mathrm{Obs,In}} \vdash x \neq y \Rightarrow \texttt{update}(x, n, \texttt{update}(y, m, s)) =\\ \texttt{update}(y, m, \texttt{update}(x, n, s))\,. \qquad \blacksquare$$

[27] Since the observational equality is total, the predicates T_s are not needed in the lifting here. Moreover, since the predicates $\sim_s$ coincide with the equality on observable sorts, we only need one predicate $\sim_{\texttt{State}}$.

11.7.5 Proof systems for refinement

The refinement relation between specifications introduced in Section 11.5 (Definition 11.29) can be extended in a straightforward way to specification expressions built with the reachability and observability operators. As explained in the introduction of this section, the main motivation for having extended the kernel specification language with observability operators is to make the refinement relation more meaningful with respect to practical applications. Indeed using the observational behavior operator, we can define the following notion of observational refinement which captures the idea that, from the observational point of view, it is sufficient if an implementation satisfies the intended behavior of the given specification (and not necessarily the specification itself).

Definition 11.66 (Observational refinement). A specification SPI is an *observational refinement* of a specification SP w.r.t. a set $\mathrm{Obs} \subseteq \mathrm{Sorts(SP)}$ of observable sorts and a set $\mathrm{In} \subseteq \mathrm{Sorts(SP)}$ of input sorts, written $\mathrm{SP} \overset{\mathrm{Obs,In}}{\rightsquigarrow} \mathrm{SPI}$, if the observational behavior specification $\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In}$ is refined by the specification SPI (in the sense of Definition 11.29), i.e., if $\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In} \rightsquigarrow \mathrm{SPI}$.

Proving an observational refinement relation $\mathrm{SP} \overset{\mathrm{Obs,In}}{\rightsquigarrow} \mathrm{SPI}$ is thus equivalent to proving the refinement relation $\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In} \rightsquigarrow \mathrm{SPI}$. For this purpose we extend the proof system for refinements defined in Section 11.5 as follows.

Definition 11.67 (Extension of the proof system $\Pi_{\rightsquigarrow_s}$ for refinement). Definition 11.31 of the proof system $\Pi_{\rightsquigarrow_s}$ for the refinement relation between structured specifications is extended as follows (again assuming given a sound proof system Π_{spec} for the derivation of the theorems of a specification expression):

$$\text{(ref-reach)} \quad \frac{\mathrm{SP} \rightsquigarrow_s \mathrm{SPI}}{\texttt{reach}\ \mathrm{SP}\ \texttt{w.r.t.}\ \Omega \rightsquigarrow_s \mathrm{SPI}} \quad \begin{array}{l}\text{if } \mathrm{Mod(SPI)} \models \mathcal{R}_\Omega \\ \text{(cf. Definition 11.51)}\end{array}$$

$$\text{(ref-beh)} \quad \frac{\mathrm{SP} \rightsquigarrow_s \mathrm{SPI}/\approx_{\mathrm{Obs,In}}}{\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs,In} \rightsquigarrow_s \mathrm{SPI}}$$

Remark 11.68. The proof system $\Pi_{\rightsquigarrow_s}$ as defined above contains two difficulties:

1. To apply rule *(ref-reach)*, one has to prove the validity of a reachability constraint. In practice, this can often be achieved by proving a "sufficient completeness" condition (see, e.g., [Wir90,Hen97]) and/or by applying the techniques for proving reachability constraints studied in [Far92].
2. There is no rule for the observational quotient, i.e., if the implementation relation to be proved is of the form $\mathrm{SP}/\approx_{\mathrm{Obs,In}} \rightsquigarrow \mathrm{SPI}$, one has to

perform a direct proof. As a consequence, the proof system for the refinement relation is complete (see below) only if the specification to be implemented does not contain observational quotient operators. This is not a severe restriction, since we will usually not apply an observational quotient operator to a specification to be implemented, but, on the contrary, in this case we will apply an observational behavior operator. Then the observational quotient operator will only appear inside an implementation proof on the right-hand side of the implementation relation when applying rule (ref-beh). Hence, for finally applying (ref-basic) in an implementation proof, it is enough to have sound and complete proof rules for the satisfaction of formulas with respect to specifications containing observational quotient operators (which were considered in the previous section). However, a consequence of the absence of proof rules for the refinement of observational quotient specifications is that we cannot directly apply our proof rules for proving an implementation relation of the form $\texttt{abstract}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs}, \mathrm{In} \rightsquigarrow \mathrm{SPI}$ as considered, e.g., in [ST88].[28] In this case we have to add the following rule, which is sound since $\mathrm{Mod}(\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs}, \mathrm{In}) \subseteq \mathrm{Mod}(\texttt{abstract}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs}, \mathrm{In})$:

$$\text{(ref-abstract)} \qquad \frac{\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs}, \mathrm{In} \rightsquigarrow_s \mathrm{SPI}}{\texttt{abstract}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs}, \mathrm{In} \rightsquigarrow_s \mathrm{SPI}}$$

If the specification SP is behaviorally closed, i.e., if we have $\mathrm{Mod}(\mathrm{SP}) \subseteq \mathrm{Mod}(\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs}, \mathrm{In})$, we know from Theorem 5.19 that $\mathrm{Mod}(\texttt{abstract}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs}, \mathrm{In}) = \mathrm{Mod}(\texttt{behavior}\ \mathrm{SP}\ \texttt{w.r.t.}\ \mathrm{Obs}, \mathrm{In})$ and the rule above is also complete.

Theorem 11.69 (Soundness and completeness of $\boldsymbol{\Pi}_{\rightsquigarrow_s}$). *For any specification* SP *containing no observational quotient operator and for any specification* SPI*,* $\mathrm{SP} \rightsquigarrow \mathrm{SPI}$ *if and only if* $\mathrm{SP} \rightsquigarrow_s \mathrm{SPI}$*.*

For a proof see [HWB97].

Example 11.70. The following specification `STATE` provides an abstract description of states (see Example 11.65). The last two axioms of the underlying specification `STATE0` express that only the current value of an identifier is stored and that the `update` operation is commutative (when different identifiers x and y are used). The operations `init` and `update` are declared as constructors for states.

```
spec STATE0
    sorts Nat, Id, State
    functions init   :  → State
              update : Id, Nat, State → State
              lookup : Id, State → Nat
```

[28] Remember that the observational abstraction operator was defined as a derived operator of the language using the observational behavior and the observational quotient operators (see Section 11.7.1).

$$
\begin{array}{ll}
\textbf{axioms} & \forall x, y : \texttt{Id},\ \ n, m : \texttt{Nat},\ \ s : \texttt{State}. \\
& \texttt{lookup}(x, \texttt{update}(x, n, s)) = n \wedge \\
& (x \neq y \Rightarrow \texttt{lookup}(x, \texttt{update}(y, n, s)) = \texttt{lookup}(x, s)) \wedge \\
& \texttt{update}(x, n, \texttt{update}(x, m, s)) = \texttt{update}(x, n, s) \wedge \\
& (x \neq y \Rightarrow\ \texttt{update}(x, n, \texttt{update}(y, m, s)) = \\
& \qquad\qquad\quad \texttt{update}(y, m, \texttt{update}(x, n, s))\ \ )
\end{array}
$$

endspec

spec $\texttt{STATE} = \texttt{reach NAT} + \texttt{ID} + \texttt{STATE0 w.r.t. } \{\texttt{init}, \texttt{update}\}$

States can be implemented by sequences of pairs (consisting of an identifier and its associated value), as described in the specification `STATE-I` of Example 11.65. In this implementation, each sequence stores not only the current value of an identifier x, but also all previous values of x. Hence the last two axioms of `STATE0` are not satisfied by the implementation. Nevertheless `STATE-I` is a correct observational refinement of `STATE` if we consider only the sorts `Nat` and `Id` as observable (and all sorts as input sorts), i.e., with Obs and In as in Example 11.65, we have:

$$\texttt{STATE} \ {}^{\mathrm{Obs,In}}\!\!\rightsquigarrow \texttt{STATE-I}$$

To prove this, we have to prove that $\texttt{behavior STATE w.r.t. } \mathrm{Obs}, \mathrm{In} \rightsquigarrow$ `STATE-I`, which can be done by applying the rules of the proof system for refinement as illustrated below. Thereby `STATE0-Ax` denotes the axioms of the basic specification `STATE0`.

1.	$\texttt{STATE-I}/{\approx_{\mathrm{Obs,In}}} \vdash \texttt{STATE0-Ax}$	(see Example 11.65)
2.	$\texttt{STATE0} \rightsquigarrow_s \texttt{STATE-I}/{\approx_{\mathrm{Obs,In}}}$	(ref-basic), (1)
3.	$\texttt{NAT} + \texttt{ID} \rightsquigarrow_s (\texttt{STATE-I}/{\approx_{\mathrm{Obs,In}}})\vert_{\mathrm{Sig}(\texttt{NAT+ID})}$	(ref-sum)
4.	$\texttt{NAT} + \texttt{ID} + \texttt{STATE0} \rightsquigarrow_s \texttt{STATE-I}/{\approx_{\mathrm{Obs,In}}}$	(ref-sum) with (3) and (2)
5.	$\mathrm{Mod}(\texttt{STATE-I}/{\approx_{\mathrm{Obs,In}}}) \models \mathcal{R}_{\{\texttt{init},\texttt{update}\}}$	(see below)
6.	$\texttt{STATE} \rightsquigarrow_s \texttt{STATE-I}/{\approx_{\mathrm{Obs,In}}}$	(ref-reach) with (4) and (5)
7.	$\texttt{behavior STATE w.r.t. } \mathrm{Obs}, \mathrm{In} \rightsquigarrow_s \texttt{STATE-I}$	(ref-beh) with (6)

For the application of the rule (ref-reach), one has to show that the models of $\texttt{STATE-I}/{\approx_{\mathrm{Obs,In}}}$ satisfy the reachability constraint $\mathcal{R}_{\{\texttt{init},\texttt{update}\}}$. This, however, is a simple task since the underlying specification `STATE-SEQ` already satisfies this reachability constraint, which follows from the reachability constraint of `STATE-SEQ` and the validity of the equations $\langle x, n\rangle = \langle x, n\rangle \& \texttt{init}$ and $\langle x, n\rangle \& s = \texttt{update}(x, n, s)$ in `STATE-SEQ`. ∎

References

[Bar77] Jon Barwise, editor. *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Company, 1977.

[Bar81] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, revised edition, 1981.

[Bau93] Hubert Baumeister. Parameter passing in the typed λ-calculus approach to parameterized specifications. Available at URL `http://www.mpi-sb.mpg.de/~hubert/index/index6.html`, 1993.

[Bee85] Michael J. Beeson. *Foundations of Constructive Mathematics*, volume 6 of *Ergebnisse der Mathematik und ihrer Grenzgebiete (3. Folge).* Springer, revised edition, 1985.

[BG80] R.M. Burstall and J.A. Goguen. The semantics of CLEAR, a specification language. In D. Björner, editor, *Proc. Copenhagen Winter School on Abstract Software Specification*, volume 86 of *Lecture Notes in Computer Science*, pages 292–332. Springer, 1980.

[BH96] Michel Bidoit and Rolf Hennicker. Behavioural theories and the proof of behavioural properties. *Theoretical Computer Science*, 165(1):3–55, 1996.

[BHK90] J.A. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the Association for Computing Machinery*, 37(2):335–372, 1990.

[BM84] J.A. Bergstra and J.J. Meyer. On specifying sets of integers. *Elektronische Informationsverarbeitung und Kybernetik*, 20:531–541, 1984.

[BM86] John L. Bell and Moshé Machover. *A course in mathematical logic.* North-Holland, 1986.

[Bor98] Tomasz Borzyszkowski. Completeness of a logical system for structured specifications. In F. Parisi Presicce, editor, *Recent Trends in Algebraic Development Techniques. Selected Papers, 12th Intl. Workshop, WADT'97*, volume 1376 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 1998.

[BT87] J.A. Bergstra and J.V. Tucker. Algebraic specifications of computable and semicomputable data types. *Theoretical Computer Science*, 50:137–181, 1987.

[Cen94] María Victoria Cengarle. *Formal Specifications with Higher-Order Parameterization.* PhD thesis, Institut für Informatik, Ludwig-Maximilians-Universität München, 1994.

[Cen95] María Victoria Cengarle. Semantic typing for parametric algebraic specifications. In V.S. Alagar and M. Nivat, editors, *Proc. Algebraic Methodology and Software Technology (AMAST'95)*, volume 936 of *Lecture Notes in Computer Science*, pages 261–276. Springer, 1995.

[CW95] María Victoria Cengarle and Martin Wirsing. A calculus of higher-order parameterization for algebraic specifications. *Bulletin of the Interest Group in Pure and Applied Logics (IGPL)*, 3(4):615–641, 1995. Special Issue 'Workshop on Logic, Language, Information and Computation 1994'.

[DGS93] Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gérard Huet and Gordon Plotkin, editors, *Logical Environments, Proc. Workshop on Logical Frameworks '91*, pages 83–130. Cambridge University Press, 1993.

[EM85] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[Far92] Jordi Farrés-Casals. *Verification in ASL and Related Specification Languages*. PhD thesis, University of Edinburgh, 1992. Report CST-92-92.

[Fei89] L.M.G. Feijs. The calculus $\lambda\pi$. In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, volume 394 of *Lecture Notes in Computer Science*, pages 307–328. Springer, 1989.

[Hen97] Rolf Hennicker. Structured specifications with behavioural operators: Semantics, proof methods and applications. Habilitation Thesis, Ludwig-Maximilians-Universität München, 1997.

[HS96] M. Hofmann and D. Sannella. On behavioural abstraction and behavioural satisfaction in higher-order logic. *Theoretical Computer Science*, 167, 1996.

[HST94] Robert Harper, Donald Sannella, and Andrzej Tarlecki. Structured theory presentations and logic representations. *Annals of Pure and Applied Logic*, 67(1):113–160, 1994. An earlier version: Structure and representation in LF. *Proc. 4th IEEE Symposium on Logics in Computer Science* LiCS'89, Asilomar, Cal. (USA) 1989, 226–237.

[HWB97] Rolf Hennicker, Martin Wirsing, and Michel Bidoit. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173:393–443, 1997.

[Kei71] H. Jerome Keisler. *Model theory for infinitary logic*, volume 62 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, 1971.

[KK67] G. Kreisel and J.L. Krivine. *Eléments de Logique Mathématique*. Dunod (Paris), 1967.

[Maj77] M.E. Majster. Limits of the "algebraic" specification of abstract data types. *ACM SIGPLAN Notices*, 12:37–42, 1977.

[Maj79] M.E. Majster. Data types, abstract data types and their specification problem. *Theoretical Computer Science*, 8:89–127, 1979.

[Men87] Elliot Mendelson. *Introduction to Mathematical Logic*. Mathematics Series. Wadsworth & Brooks/Cole, third edition, 1987.

[Mes89] José Meseguer. General Logics. In H.-D. Ebbinghaus, J. Fernández-Prida, M. Garrido, D. Lascar, and M. Rodríguez Artalejo, editors, *Logic Colloquium '87*, pages 275–329. North-Holland, 1989.

[MVS85] T.S.E. Maibaum, P.A.S. Veloso, and M.R. Sadler. A theory of abstract data types for program development: bridging the gap? In H. Ehrig, C. Floyd, M. Nivat, and J. Thatcher, editors, *Proc. Joint Conference on Theory and Practice of Software Development (TAPSOFT'85)*, volume 186 of *Lecture Notes in Computer Science*, pages 214–230. Springer, 1985.

[Pad84] Peter Padawitz. Towards a proof theory of parameterized specifications. In G. Kahn, D.B. MacQueen, and G. Plotkin, editors, *Intl. Symposium on Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 375–391. Springer, 1984.

[Pad87] Peter Padawitz. Parameter preserving data type specifications. *Journal of Computer and System Sciences*, 34:179–209, 1987.

[Pad88] Peter Padawitz. The equational theory of parameterized specifications. *Information and Computation*, 76:121–137, 1988.

[SB83] Donald Sannella and R.M. Burstall. Structured theories in LCF. In G. Ausiello and M. Protasi, editors, *Proc. 8th Colloquium on Trees in Algebra and Programming (CAAP)*, volume 159 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 1983.

[Sch92] O. Schoett. Two impossibility theorems on behavioural specifications of abstract data types. *Acta Informatica*, 29:595–621, 1992.

[SST92] Donald Sannella, Stefan Sokołowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Parameterisation revisited. *Acta Informatica*, 29(8):689–736, 1992.

[ST88] Donald Sannella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica*, 25(3):233–281, 1988.

[SW83] Donald Sannella and Martin Wirsing. A kernel language for algebraic specification and implementation. In M. Karpinski, editor, *Proc. 11th Colloquium on Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 1983.

[TWW82] J.W. Thatcher, E.G. Wagner, and J.B. Wright. Data type specification: parameterization and the power of specification techniques. *ACM TOPLAS*, 4:711–732, 1982.

[Wir90] Martin Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, chapter 13, pages 675–788. Elsevier Science Publishers B.V. (North Holland), 1990.

[Wir91] Martin Wirsing. Structured specifications: Syntax, semantics and proof calculus. In F.L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, NATO ASI Series F94: Computer and Systems Sciences, pages 411–442. Springer, 1991.

[Wol87] U. Wolter. The power of behavioural validity. Preprint 1-87, TU Magdeburg, Section Mathematik, 1987.