Proving Behavioral Refinements of COL-Specifications *

Michel Bidoit [1] Rolf Hennicker [2]

[1] Laboratoire Spécification et Vérification (LSV), CNRS & ENS de Cachan, France
[2] Institut für Informatik, Ludwig-Maximilians-Universität München, Germany

Abstract. The COL institution (constructor-based observational logic) has been introduced as a formal framework to specify both generation- and observation-oriented properties of software systems. In this paper we consider behavioral refinement relations between COL-specifications taking into account implementation constructions. We propose a general strategy for proving the correctness of such refinements by reduction to (standard) first-order theorem proving with induction. Technically our strategy relies on appropriate proof rules and on a lifting construction to encode the reachability and observability notions of the COL institution.

1 Introduction

Within the theory of algebraic specifications, behavioral (or observational) aspects of software systems have been considered since more than twenty years in many approaches in the literature. One of the first studies exposing the importance of a behavioral view for the formalization of implementation notions has been provided by Goguen and Meseguer in [11]. It is motivated by many examples which show that it is essential to abstract from internal implementation details and to rely only on the observable behavior of programs.

As discussed in [7], behavioral refinement concepts can be classified into two principal trends. The first one, pursued e.g. in [18, 19, 17, 3], uses an explicit behavioral abstraction operator to relax the standard model class semantics of the specification to be implemented. The second one uses specifications with built-in features to express behavioral properties. Examples are the hidden algebra institution developed by Goguen and his research group (see e.g. [12]), the CafeOBJ language [9] and the COL institution (constructor-based observational logic [4]). Each of these approaches is equipped with a notion of signature containing a distinguished set of observer operations (to build observable experiments) and with a notion of behavioral satisfaction such that the equality symbol is interpreted by the observational equality of elements (where two elements of an algebra are observationally equal if they cannot be distinguished by observable experiments). In the COL institution signatures contain additionally to the observers a distinguished set of constructor operations which specify those elements

* This work is partially supported by the GLOWA-Danube project (01LW0303A) sponsored by the German Federal Ministry of Education and Research.

which are of interest from the user's point of view thus determining a subpart of an algebra (called the contructor-generated part). Hence a COL-signature $\Sigma_{\mathrm{COL}} = (\Sigma, \mathrm{OP}_{\mathrm{Cons}}, \mathrm{OP}_{\mathrm{Obs}})$ consists of a (standard) many-sorted signature $\Sigma = (S, \mathrm{OP})$ together with distinguished sets $\mathrm{OP}_{\mathrm{Cons}}$ of constructor operations and $\mathrm{OP}_{\mathrm{Obs}}$ of observer operations. The behavioral satisfaction of formulas is then further relaxed to the COL-satisfaction relation $\models_{\Sigma_{\mathrm{COL}}}$ which takes into account only constructor-generated values for the valuation of variables (thus abstracting from junk values).

A simple refinement relation between COL-specifications can be defined if the specification $\mathrm{SP}_{\mathrm{COL}}$ to be implemented and the implementing specification $\mathrm{SPI}_{\mathrm{COL}}$ have the same COL-signature Σ_{COL}. In this case $\mathrm{SPI}_{\mathrm{COL}}$ is a behavioral refinement of $\mathrm{SP}_{\mathrm{COL}}$ if its model class $\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}]$ is included in the model class $\mathcal{M}od[\mathrm{SP}_{\mathrm{COL}}]$ of $\mathrm{SP}_{\mathrm{COL}}$. To prove the correctness of the refinement one has to show, assuming that $\mathrm{SP}_{\mathrm{COL}}$ is a (flat) specification of the form $\langle \Sigma_{\mathrm{COL}}, \mathrm{Ax} \rangle$, that $\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}] \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax}$, i.e that all models of $\mathrm{SPI}_{\mathrm{COL}}$ behaviorally satisfy the axioms of the abstract specification $\mathrm{SP}_{\mathrm{COL}}$. For this purpose one can directly apply the proof techniques for behavioral consequences of COL-specifications developed in [4]. In general, however, the assumption that both specifications have the same signature is much too restrictive because an implementation usually involves some construction steps which led to the concept of a constructor implementation introduced in [19]. In the context of the COL institution this idea is formalized by the notion of a COL-implementation constructor κ_{COL} which can be applied to the models of the implementing specification $\mathrm{SPI}_{\mathrm{COL}}$ to produce models of the specification $\mathrm{SP}_{\mathrm{COL}}$ to be implemented. Hence, to prove the correctness of the refinement one has to show

$$(*) \quad \kappa_{\mathrm{COL}}(\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}]) \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax}$$

with $\mathrm{SP}_{\mathrm{COL}} = \langle \Sigma_{\mathrm{COL}}, \mathrm{Ax} \rangle$ as above. Unfortunatley there is no obvious way for discharging this proof obligation since we cannot expect that κ_{COL} is compatible with COL-satisfaction, i.e. we cannot reduce the proof to $\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}] \models_{\Sigma I_{\mathrm{COL}}} \mathrm{Ax}^*$ (with an appropriate syntactic adjustment Ax^* of Ax and ΣI_{COL} being the signature of $\mathrm{SPI}_{\mathrm{COL}}$). For instance if we consider an implementation of sets by lists, κ_{COL} would be the (standard) reduct functor along a (standard) signature morphism that would not preserve the usual observer operations where sets are observed by the membership test *isin* and lists are observed by *head* and *tail*. Hence, the reduct used for the implementation construction would not be compatible with the COL-satisfaction relations for sets and lists resp. These considerations are in accordance with Goguen's and Malcolm's study on the difference between vertical signature morphisms used for refinements and horizontal signature morphisms used for modular constructions of system specifications; see [15].

In this paper we propose a strategy to discharge the proof obligation (*) which consists of two major steps. In the first step (see Section 4) we show that instead of the COL-specification $\mathrm{SPI}_{\mathrm{COL}}$ used for the implementation it is sufficient to consider the (standard) first-order specification SPI obtained from $\mathrm{SPI}_{\mathrm{COL}}$ by

forgetting the observer and constructor opertions. For the correctness of the corresponding proof rule it is essential that COL-implementation constructors must preserve observational equivalences between algebras (a property which is strongly related to Schoett's notion of stability; see [20]).

As a consequence of the first step it remains to show that the (standard) models of SPI behaviorally satisfy the axioms Ax of the specification $\mathrm{SP}_{\mathrm{COL}}$ to be implemented. Therefore, in the next step (see Section 5), we investigate how the proof of behavioral consequences (w.r.t. $\models_{\Sigma_{\mathrm{COL}}}$) of an arbitrary class of Σ-algebras can be reduced to standard first-order reasoning (plus induction). Technically this is achieved by a "lifting" construction providing an appropriate axiomatization of observational equalities and generated parts. Our proof techniques are illustrated by an example (see Section 6) considering a behavioral refinement of sets by non-redundant lists.

2 Basic Concepts

In this section we summarize the basic concepts that are needed to study behavioral refinements of COL-specifications and corresponding proof techniques.

2.1 Algebraic Preliminaries

We assume that the reader is familiar with the basic notions of algebraic specifications (see, e.g., [22, 14, 1]), like the notions of (many-sorted) *signature* $\Sigma = (S, \mathrm{OP})$ (where S is a set of *sorts* and OP is a set of *operation symbols* $op : s_1, \ldots, s_n \to s$), *signature morphism* $\sigma : \Sigma \to \Sigma'$, *(total)* Σ*-algebra* $A = ((A_s)_{s \in S}, (op^A)_{op \in \mathrm{OP}})$, Σ*-term algebra* $T_\Sigma(X)$ over a family $X = (X_s)_{s \in S}$ of pairwise disjoint sets X_s of variables of sort s and *interpretation* $I_\alpha : T_\Sigma(X) \to A$ w.r.t. a *valuation* $\alpha : X \to A$. The class of all Σ-algebras is denoted by $\mathrm{Alg}(\Sigma)$. Together with Σ-morphisms this class forms a category which, for simplicity, is also denoted by $\mathrm{Alg}(\Sigma)$. For any signature morphism $\sigma : \Sigma \to \Sigma'$, the *reduct functor* $\_|_\sigma : \mathrm{Alg}(\Sigma') \to \mathrm{Alg}(\Sigma)$ is defined as usual and the reduct of a Σ'-algebra A w.r.t. σ is denoted by $A|_\sigma$. In particular, the reduct of A to a subsignature $\Sigma \subseteq \Sigma'$ is denoted by $A|_\Sigma$. In the following we assume that signatures are finite.

The notion of an institution was introduced by Goguen and Burstall [13] to formalize the general concept of a logical system from a model-theoretic point of view; see [21] for an overview. An important example is the institution FOLEq of many-sorted first-order logic with equality as detailed, e.g., in [2]. In FOLEq signatures are many-sorted signatures, models are Σ-algebras and sentences are arbitray first-order Σ-formulas. The satisfaction of a first-order Σ-formula φ by a Σ-algebra A, denoted by $A \models \varphi$, is defined as usual in the first-order predicate calculus with equality. The notation $A \models \varphi$ is extended in a straightforward way to classes of algebras and sets of formulas. The institution CFOLEq is an extension of the FOLEq institution where, in addition to first-order sentences, we consider as extra sentences *sort-generation constraints* of

the form $\mathrm{SGC}(S_{\mathrm{Cons}}, \mathrm{OP}_{\mathrm{Cons}})$. A Σ-algebra A satisfies a sort-generation constraint $\mathrm{SGC}(S_{\mathrm{Cons}}, \mathrm{OP}_{\mathrm{Cons}})$ if it is reachable w.r.t. $\mathrm{OP}_{\mathrm{Cons}}$, i.e. if each element of a carrier set A_s with constrained sort $s \in S_{\mathrm{Cons}}$ can be constructed by the interpretations of the constructors $\mathrm{OP}_{\mathrm{Cons}}$ starting from constants and from arbitrary elements of non-constrained sorts, if any. It is well-known that a *free sort-generation constraint* is just an abbreviation for the corresponding sort-generation constraint plus a finite set of first-order sentences to state that all distinct constructor terms (up to variable renaming) denote distinct values. Therefore, in the following, we will also assume that the CFOLEq institution is equipped with free sort-generation constraints of the form $\mathrm{FSGC}(S_{\mathrm{Cons}}, \mathrm{OP}_{\mathrm{Cons}})$, with the meaning described above (see [16, pp. 152–153]).

Any institution provides a suitable framework to define specifications. The semantics of a specification SP is determined by its signature, denoted by $\mathcal{S}ig[\mathrm{SP}]$, and by its class of models, denoted by $\mathcal{M}od[\mathrm{SP}]$. In this paper we will only consider basic specifications $\langle \Sigma, \mathrm{Ax} \rangle$ consisting of a signature Σ and a set Ax of Σ-sentences, also called the *axioms* of the specification, with semantics:

$$\mathcal{S}ig[\langle \Sigma, \mathrm{Ax} \rangle] \stackrel{\mathrm{def}}{=} \Sigma$$
$$\mathcal{M}od[\langle \Sigma, \mathrm{Ax} \rangle] \stackrel{\mathrm{def}}{=} \{M \in \mathrm{Mod}(\Sigma) \mid M \models_\Sigma \mathrm{Ax}\}.$$

Notations. If SP is a specification and φ is a $\mathcal{S}ig[\mathrm{SP}]$-sentence, we write $\mathrm{SP} \models \varphi$ for $\mathcal{M}od[SP] \models \varphi$ and similarly for sets of $\mathcal{S}ig[\mathrm{SP}]$-sentences. In the context of the CFOLEq institution we will also consider the sum $\mathrm{SP}_1 + \mathrm{SP}_2$ of two specifications SP_1 and SP_2 with semantics:

$$\mathcal{S}ig[\mathrm{SP}_1 + \mathrm{SP}_2] \stackrel{\mathrm{def}}{=} \mathcal{S}ig[\mathrm{SP}_1] \cup \mathcal{S}ig[\mathrm{SP}_2]$$
$$\mathcal{M}od[\mathrm{SP}_1 + \mathrm{SP}_2] \stackrel{\mathrm{def}}{=} \{A \in \mathrm{Alg}(\mathcal{S}ig[\mathrm{SP}_1] \cup \mathcal{S}ig[\mathrm{SP}_2]) \mid A|_{\mathcal{S}ig[\mathrm{SP}_1]} \in \mathcal{M}od[\mathrm{SP}_1] \text{ and } A|_{\mathcal{S}ig[\mathrm{SP}_2]} \in \mathcal{M}od[\mathrm{SP}_2]\}.$$

By analogy, for any class $\boldsymbol{C}$ of Σ-algebras and any specification SP, we denote by $\boldsymbol{C} + \mathrm{SP}$ the class of $\Sigma \cup \mathcal{S}ig[\mathrm{SP}]$-algebras defined by:

$$\boldsymbol{C} + \mathrm{SP} \stackrel{\mathrm{def}}{=} \{A \in \mathrm{Alg}(\Sigma \cup \mathcal{S}ig[\mathrm{SP}]) \mid A|_\Sigma \in \boldsymbol{C} \text{ and } A|_{\mathcal{S}ig[\mathrm{SP}]} \in \mathcal{M}od[\mathrm{SP}]\}.$$

2.2 A Brief Introduction to the Constructor-based Observational Logic COL

The COL institution has been introduced as a formal framework to capture the observational aspects of system specifications; see [4]. The basic idea is to consider distinguished sets of constructor and observer operations. Intuitively, the constructor operations determine those elements which are of interest from the user's point of view while the observer operations determine a set of observable experiments that a user can perform to examine hidden states. Thus we can abstract from junk elements and also from concrete state representations whereby two states are considered to be "observationally equal" if they cannot be distinguished by observable experiments.

Formally, a *constructor operation* is an operation symbol $cons : s_1, \ldots, s_n \to s$ with $n \geq 0$. The result sort s of *cons* is called a *constrained sort*. An *observer*

operation is a pair (obs, i) where obs is an operation symbol $obs : s_1, \ldots, s_n \to s$ with $n \geq 1$ and $1 \leq i \leq n$. The distinguished argument sort s_i of obs is called a *state sort* (or *hidden sort*). If $obs : s_1 \to s$ is a unary observer we simply write obs instead of $(obs, 1)$. A COL-*signature* $\Sigma_{\mathrm{COL}} = (\Sigma, \mathrm{OP}_{\mathrm{Cons}}, \mathrm{OP}_{\mathrm{Obs}})$ consists of a standard many-sorted signature $\Sigma = (S, \mathrm{OP})$ together with a distinguished set $\mathrm{OP}_{\mathrm{Cons}} \subseteq \mathrm{OP}$ of constructor operations and a distinguished set $\mathrm{OP}_{\mathrm{Obs}}$ of observer operations (obs, i) with $obs \in \mathrm{OP}$. We implicitly assume in the following that whenever we consider a COL-signature Σ_{COL} the underlying (standard) signature is Σ and similarly for Σ'_{COL} etc.

The set $S_{\mathrm{Cons}} \subseteq S$ of *constrained sorts* (w.r.t. $\mathrm{OP}_{\mathrm{Cons}}$) consists of all sorts s such that there exists at least one constructor in $\mathrm{OP}_{\mathrm{Cons}}$ with range s. The set $S_{\mathrm{Loose}} \subseteq S$ of *loose sorts* consists of all non-constrained sorts, i.e. $S_{\mathrm{Loose}} = S \backslash S_{\mathrm{Cons}}$. The set $S_{\mathrm{State}} \subseteq S$ of *state sorts* (or *hidden sorts*, w.r.t. $\mathrm{OP}_{\mathrm{Obs}}$) consists of all sorts s_i such that there exists at least one observer (obs, i) in $\mathrm{OP}_{\mathrm{Obs}}$, $obs : s_1, \ldots, s_i, \ldots, s_n \to s$. The set $S_{\mathrm{Obs}} \subseteq S$ of *observable sorts* consists of all sorts which are not a state sort, i.e. $S_{\mathrm{Obs}} = S \setminus S_{\mathrm{State}}$. An observer $(obs, i) \in \mathrm{OP}_{\mathrm{Obs}}$, $obs : s_1, \ldots, s_i, \ldots, s_n \to s$ is called a *direct observer* if $s \in S_{\mathrm{Obs}}$, otherwise it is an *indirect observer*.

The set $\mathrm{OP}_{\mathrm{Cons}}$ of constructor operations (of a COL-signature Σ_{COL}) determines a set of *constructor terms*. A constructor term is a term t of a constrained sort $s \in S_{\mathrm{Cons}}$ which is built only from constructor operations of $\mathrm{OP}_{\mathrm{Cons}}$ and from variables of loose sorts. In particular, if all sorts are constrained, i.e., $S_{\mathrm{Cons}} = S$, the constructor terms are exactly the $(S, \mathrm{OP}_{\mathrm{Cons}})$-ground terms which are built by the constructor symbols. The set of constructor terms determines, for any Σ-algebra A, an S-sorted family of subsets of the carrier sets of A, called the *generated part* and denoted by $\mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)$. For each constrained sort $s \in S_{\mathrm{Cons}}$, the corresponding subset $\mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)_s \subseteq A_s$ consists of those elements that can be constructed by the interpretations of the given constructors (starting from constants and from arbitrary elements of loose sorts, if any). For each loose sort $s \in S_{\mathrm{Loose}}$, $\mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)_s = A_s$. The Σ_{COL}-generated part represents those elements which are of interest from the user's point of view according to the given constructor operations. A Σ-algebra A is *reachable* (w.r.t. Σ_{COL}) if its carrier sets coincide with its Σ_{COL}-generated part.

The set $\mathrm{OP}_{\mathrm{Obs}}$ of observer operations (of a COL-signature Σ_{COL}) determines a set of *observable contexts* which represent the observable experiments that a user can perform. Observable contexts are defined in a coinductive style which will be reflected in the encoding of observable contexts in Section 5.

Definition 1 (Observable context). *Let Σ_{COL} be a* COL-*signature, let $X = (X_s)_{s \in S}$ be a family of pairwise disjoint, countably infinite sets X_s of variables of sort s and let $Z = (\{z_s\})_{s \in S_{\mathrm{State}}}$ be a disjoint family of singleton sets (one for each state sort). The sets $\mathcal{C}(\Sigma_{\mathrm{COL}})_{s \to s'}$ of* observable Σ_{COL}-contexts *with "application sort" s and "observable result sort" s', with $s \in S_{\mathrm{State}}$ and $s' \in S_{\mathrm{Obs}}$, are the least sets such that:*

1. *For each direct observer* (obs, i) *with* $obs : s_1, \ldots, s_i, \ldots, s_n \to s'$ *and pairwise disjoint variables* $x_1{:}s_1, \ldots, x_n{:}s_n$,
   $obs(x_1, \ldots, x_{i-1}, z_{s_i}, x_{i+1}, \ldots, x_n) \in \mathcal{C}(\Sigma_{\mathrm{COL}})_{s_i \to s'}$.
2. *For each observable context* $c \in \mathcal{C}(\Sigma_{\mathrm{COL}})_{s \to s'}$ *, for each indirect observer* (obs, i) *with* $obs : s_1, \ldots, s_i, \ldots, s_n \to s$*, and pairwise disjoint variables* $x_1{:}s_1, \ldots, x_n{:}s_n$ *not occurring in* c,
   $c[obs(x_1, \ldots, x_{i-1}, z_{s_i}, x_{i+1}, \ldots, x_n)/z_s] \in \mathcal{C}(\Sigma_{\mathrm{COL}})_{s_i \to s'}$
   where $c[obs(x_1, \ldots, x_{i-1}, z_{s_i}, x_{i+1}, \ldots, x_n)/z_s]$ *denotes the term obtained from* c *by substituting the term* $obs(x_1, \ldots, x_{i-1}, z_{s_i}, x_{i+1}, \ldots, x_n)$ *for* z_s .

We assume that for any state sort $s \in S_{\mathrm{State}}$ *there exists an observable context with application sort* s.

The set of observable contexts determines, for any Σ-algebra A, an indistinguishability relation, called *observational equality*. The observational equality on A is an S-sorted binary relation $\approx_{\Sigma_{\mathrm{COL}},A}$ such that for any two elements $a, b \in A$, $a \approx_{\Sigma_{\mathrm{COL}},A} b$ holds if either $a = b$ and a, b are observable (i.e. belong to a carrier set of observable sort $s \in S_{\mathrm{Obs}}$) or if a and b are hidden (i.e. belong to a carrier set of a state sort $s \in S_{\mathrm{State}}$) but cannot be distinguished by the application of observable contexts. The application of observable contexts is defined in the usual way apart from the fact that for variables in X (occurring in an observable context c) we consider only valuations in the generated part $\mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)$ (i.e. junk values are disregarded because they should not contribute to distinguish elements). A Σ-algebra A is *fully abstract* if the observational equality coincides (on all carrier sets) with the set-theoretic equality.

The constructor and the observer operations induce certain constraints on Σ-algebras. First, since the constructor operations determine the values of interest, we require that the non-constructor operations should (up to observational equality) respect the constructor-generated part of an algebra, i.e. by the application of non-constructor operations one should at most be able to obtain elements which are observationally equal to some element of the constructor-generated part $\mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)$. Technically this means that for a given Σ-algebra A we first consider the smallest Σ-subalgebra $\langle \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A) \rangle_\Sigma$ of A containing the Σ_{COL}-generated part because this subalgebra represents the only elements a user can compute (over the loose carrier sets) by invoking operations of Σ. Then we require that each element of $\langle \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A) \rangle_\Sigma$ is observationally equal to some element of the Σ_{COL}-generated part $\mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)$ of A. This condition is called *reachability constraint*.

Furthermore, since the declaration of observer operations determines a particular observational equality on any Σ-algebra A, the (interpretations of the) non-observer operations should respect this observational equality, i.e. a non-observer operation should not contribute to distinguish non-observable elements. To ensure this we require that the observational equality is a Σ-congruence on the subalgebra $\langle \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A) \rangle_\Sigma$. (It is sufficient to consider $\langle \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A) \rangle_\Sigma$ instead of A because computations performed by a user can only lead to elements in the Σ-subalgebra $\langle \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A) \rangle_\Sigma$.) This condition is called *observability constraint*.

A Σ-algebra A which satisfies both the reachability and the observability constraints induced by a COL-signature Σ_{COL} is called a Σ_{COL}-*algebra* (or simply a COL-*algebra*). Obviously any Σ-algebra A which is reachable and fully abstract w.r.t. Σ_{COL} is a Σ_{COL}-algebra. The class of all Σ_{COL}-algebras is denoted by $\mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}})$. It can be extended to a category by an appropriate notion of Σ_{COL}-morphism which reflects behavioral relationships between Σ_{COL}-algebras (see [4] for details).

The satisfaction of the reachability and observability constraints allows us to construct for each Σ_{COL}-algebra A its *black box view* which is a reachable and fully abstract algebra representing the behavior of A from the user's point of view. The black box view is constructed in two steps. First, we *restrict* to the Σ_{COL}-generated subalgebra $\langle \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)\rangle_{\Sigma}$ of A thus forgetting junk values. Then, we *identify* all elements of $\langle \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)\rangle_{\Sigma}$ which are observationally equal. Hence the black box view of a Σ_{COL}-algebra A is given by the quotient algebra of $\langle \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)\rangle_{\Sigma}$ w.r.t. $\approx_{\Sigma_{\mathrm{COL}},A}$ which, for simplicity, will be denoted by $A/\!\approx_{\Sigma_{\mathrm{COL}},A}$. Two Σ_{COL}-algebras A and B are *observationally equivalent*, denoted by $A \equiv_{\Sigma_{\mathrm{COL}}} B$, if their black box views $A/\!\approx_{\Sigma_{\mathrm{COL}},A}$ and $B/\!\approx_{\Sigma_{\mathrm{COL}},B}$ are isomorphic Σ-algebras. Observationally equivalent Σ_{COL}-algebras are isomorphic w.r.t. Σ_{COL}-morphisms (see [4]).

A crucial concept to obtain a built-in behavioral semantics for specifications is the COL-*satisfaction relation*, denoted by $\models_{\Sigma_{\mathrm{COL}}}$, which generalizes the standard satisfaction relation of first-order logic by abstracting with respect to reachability and observability. First, from the reachability point of view, the valuations of variables are restricted to the elements of the Σ_{COL}-generated part only. From the observability point of view, the idea is to interpret the equality symbol "=" occurring in a first-order formula φ not by the set-theoretic equality but by the observational equality of elements.

Definition 2 (COL-satisfaction relation). *For any* COL-*signature* Σ_{COL}, *the* COL-satisfaction relation *between* Σ-*algebras and first-order* Σ-*formulas (with variables in* X*) is denoted by* $\models_{\Sigma_{\mathrm{COL}}}$ *and defined as follows. Let* $A \in \mathrm{Alg}(\Sigma)$.

1. *For any two terms* $t, r \in T_{\Sigma}(X)_s$ *of the same sort* s *and for any valuation* $\alpha : X \to \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)$, $A, \alpha \models_{\Sigma_{\mathrm{COL}}} t = r$ *holds if* $I_{\alpha}(t) \approx_{\Sigma_{\mathrm{COL}},A} I_{\alpha}(r)$.
2. *For any* Σ-*formula* φ *and for any valuation* $\alpha : X \to \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)$, $A, \alpha \models_{\Sigma_{\mathrm{COL}}} \varphi$ *is defined by induction over the structure of the formula* φ *in the usual way. In particular,* $A, \alpha \models_{\Sigma_{\mathrm{COL}}} \forall x{:}s.\, \varphi$ *if for all valuations* $\beta : X \to \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)$ *with* $\beta(y) = \alpha(y)$ *for all* $y \neq x$, $A, \beta \models_{\Sigma_{\mathrm{COL}}} \varphi$.
3. *For any* Σ-*formula* φ, $A \models_{\Sigma_{\mathrm{COL}}} \varphi$ *holds if for all valuations* $\alpha : X \to \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)$, $A, \alpha \models_{\Sigma_{\mathrm{COL}}} \varphi$ *holds.*

The notation $A \models_{\Sigma_{\mathrm{COL}}} \varphi$ is extended in the usual way to classes of algebras and sets of formulas. The COL-satisfaction relation is defined not only for Σ_{COL}-algebras but also for arbitrary Σ-algebras which will be important when we consider proof techniques for behavioral refinement relations.

Fact 1 *Let* Σ_{COL} *be a* COL*-signature, let* φ *be a* Σ*-formula and let* A *be a* Σ_{COL}*-algebra. Then:*

$$A \models_{\Sigma_{\mathrm{COL}}} \varphi \text{ if and only if } A/{\approx_{\Sigma_{\mathrm{COL}},A}} \models \varphi.$$

The above definitions provide the basic ingredients that lead to the COL institution. In particular, the COL-satisfaction relation satisfies the satisfaction condition of institutions w.r.t. COL-signature morphisms which are standard signature morphisms fulfilling additional properties related to the preservation of constructor and observer operations (see [4] for details). A basic COL specification $\mathrm{SP}_{\mathrm{COL}} = \langle \Sigma_{\mathrm{COL}}, \mathrm{Ax} \rangle$ consists of a COL-signature Σ_{COL} and a set Ax of Σ-sentences (the axioms of the specification). The semantics of $\mathrm{SP}_{\mathrm{COL}}$ is given by its signature Σ_{COL} and by its class of models:

$$\mathcal{M}od[\mathrm{SP}_{\mathrm{COL}}] = \{A \in \mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}}) \mid A \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax}\}.$$

3 Behavioral Refinements

Generally, specification refinement is a relation between an abstract specification to be implemented and a more concrete specification which satisfies the requirements of the given abstract specification. Taking into account the observable behavior described by COL-specifications, a COL-specification $\mathrm{SPI}_{\mathrm{COL}}$ is considered as a behavioral refinement of a COL-specification $\mathrm{SP}_{\mathrm{COL}}$ if $\mathrm{SPI}_{\mathrm{COL}}$ respects the behavioral properties required by $\mathrm{SP}_{\mathrm{COL}}$. Formally, a simple behavioral refinement relation between two COL-specifications can be defined by requiring that both specifications have the same signature and that the model class of the implementing specification $\mathrm{SPI}_{\mathrm{COL}}$ is included in the model class of $\mathrm{SP}_{\mathrm{COL}}$. Remember that for the sake of simplicity we restrict to basic specifications in the framework of this paper.

Definition 3 (Behavioral refinement: simple case).
Let $\mathrm{SP}_{\mathrm{COL}} = \langle \Sigma_{\mathrm{COL}}, \mathrm{Ax} \rangle$ *and* $\mathrm{SPI}_{\mathrm{COL}} = \langle \Sigma_{\mathrm{COL}}, \mathrm{AxI} \rangle$ *be two* COL*-specifications with the same signature* Σ_{COL}*.* $\mathrm{SPI}_{\mathrm{COL}}$ *is a* behavioral refinement *of* $\mathrm{SP}_{\mathrm{COL}}$*, denoted by* $\mathrm{SP}_{\mathrm{COL}} \rightsquigarrow \mathrm{SPI}_{\mathrm{COL}}$*, if*

$$\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}] \subseteq \mathcal{M}od[\mathrm{SP}_{\mathrm{COL}}].$$

To prove that $\mathrm{SP}_{\mathrm{COL}} \rightsquigarrow \mathrm{SPI}_{\mathrm{COL}}$ holds, one has to show that:

$$\mathrm{SPI}_{\mathrm{COL}} \models_{\Sigma_{\mathrm{COL}}} \varphi \text{ for all axioms } \varphi \in \mathrm{Ax},$$

i.e. that the axioms of $\mathrm{SP}_{\mathrm{COL}}$ are observable consequences of $\mathrm{SPI}_{\mathrm{COL}}$. For this purpose one can directly apply the proof techniques for COL-specifications studied in [4] (since Σ_{COL} is also the signature of $\mathrm{SPI}_{\mathrm{COL}}$).

In general, however, one has to take into account that an implementation involves some construction step, an idea which has been formalized by the notion of constructor implementation introduced in [19] (and similarly in other

implementation concepts; see [17, 10] for an overview). According to [19] an implementation constructor is a function which maps algebras over the signature of the implementing specification to algebras over the signature of the abstract specification. Since it is sufficient if an implementation construction is defined on the models of the implementing specification implementation constructors are, in general, partial functions. We assume that implementation constructions are performed in a uniform way, i.e. preserve isomorphisms. It is obvious that the concept of an implementation constructor can be easily transferred to behavioral refinements of COL-specifications. In particular, the requirement that isomorphisms are preserved means in the context of the COL institution that a COL-implementation constructor preserves COL-isomorphisms, i.e. observational equivalences of COL-algebras (see Section 2.2).

Definition 4 (COL-implementation constructor). *Let Σ_{COL}, ΣI_{COL} be two* COL*-signatures. A* COL-implementation constructor *from ΣI_{COL} to Σ_{COL} is a partial function $\kappa_{\mathrm{COL}} : \mathrm{Alg}_{\mathrm{COL}}(\Sigma I_{\mathrm{COL}}) \to \mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}})$ which is* COL*-isomorphism preserving, i.e. for all $AI, BI \in \mathrm{Alg}_{\mathrm{COL}}(\Sigma I_{\mathrm{COL}})$,*
if $AI \equiv_{\Sigma I_{\mathrm{COL}}} BI$ and $\kappa_{\mathrm{COL}}(AI)$ is defined
then $\kappa_{\mathrm{COL}}(BI)$ is defined and $\kappa_{\mathrm{COL}}(AI) \equiv_{\Sigma_{\mathrm{COL}}} \kappa_{\mathrm{COL}}(BI)$.
The definition domain of κ_{COL} is denoted by $Dom(\kappa_{\mathrm{COL}})$.

Using the notion of a COL-implementation constructor we can generalize Definition 3 to the case where the abstract and implementing specifications have different signatures.

Definition 5 (Behavioral refinement w.r.t. an implementation constructor). *Let* $\mathrm{SP}_{\mathrm{COL}}$, $\mathrm{SPI}_{\mathrm{COL}}$ *be two* COL*-specifications with signatures Σ_{COL}, ΣI_{COL} resp. and let κ_{COL} be a* COL*-implementation constructor from ΣI_{COL} to Σ_{COL}.* $\mathrm{SPI}_{\mathrm{COL}}$ *is a* behavioral refinement *of* $\mathrm{SP}_{\mathrm{COL}}$ w.r.t. κ_{COL}*, denoted by* $\mathrm{SP}_{\mathrm{COL}} \rightsquigarrow^{\kappa_{\mathrm{COL}}} \mathrm{SPI}_{\mathrm{COL}}$*, if*

$$\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}] \subseteq Dom(\kappa_{\mathrm{COL}}) \text{ and } \kappa_{\mathrm{COL}}(\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}]) \subseteq \mathcal{M}od[\mathrm{SP}_{\mathrm{COL}}].$$

As discussed in [7] an important question is, of course, which implementation constructors are appropriate for behavioral refinements. As a first approach one could simply consider COL-signature morphisms $\sigma_{\mathrm{COL}} : \Sigma_{\mathrm{COL}} \to \Sigma I_{\mathrm{COL}}$. Since COL is an institution, the corresponding COL-reduct functor $\_|_{\sigma_{\mathrm{COL}}} : \mathrm{Alg}_{\mathrm{COL}}(\Sigma I_{\mathrm{COL}}) \to \mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}})$ preserves COL-isomorphisms, i.e. is a COL-implementation constructor. Hence it is tempting to consider COL-refinements where the syntactic relationship between the specification $\mathrm{SP}_{\mathrm{COL}}$ to be implemented and the implementing specification $\mathrm{SPI}_{\mathrm{COL}}$ is established by a COL-signature morphism. This approach has, however, a serious drawback because the implementing specification $\mathrm{SPI}_{\mathrm{COL}}$ usually has constructor and observer operations $\mathrm{OPI}_{\mathrm{Cons}}$, $\mathrm{OPI}_{\mathrm{Obs}}$ which are unrelated to the constructor and observer operations $\mathrm{OP}_{\mathrm{Cons}}$, $\mathrm{OP}_{\mathrm{Obs}}$ of the specification $\mathrm{SP}_{\mathrm{COL}}$ to be implemented. As a simple example we consider in Section 6 the implementation of sets by lists where the observer for sets is the membership test *isin* while the observer operations

for lists are, as usual, the *head* and *tail* operations. Hence the COL-specifications of sets and lists cannot be related by a COL-signature morphism which would require the preservation of constructor and observer operations. This is the reason why we want to consider standard signature morphisms and their reduct functors as implementation constructors for COL-specifications.

But before let us still point out that our viewpoint has been inspired by the following remarkable sentences by Goguen and Malcolm [15]: *"Signature morphisms perform two distinct roles. One role is to express the importation of one specification into another... referred to as* horizontal composition*... so that when a specification of a class of objects is imported into a larger specification, the properties of the imported object classes are preserved. The other role performed by signature morphisms is to compare two different specifications. This is referred to as* vertical composition*, and pertains to relationships between layers... In such a case we would not expect that signature morphisms encapsulate object class specifications, but rather expect that signature morphisms preserve the behaviour of object classes... ".*

Interpreting these considerations in the COL framework this means that it is indeed adequate not to stick to COL-signature morphisms when we construct implementations. COL-signature morphisms are the appropriate tool to ensure encapsulation of COL-specifications (formally expressed by the satisfaction condition of an institution) which is indeed important when we construct large design specifications in a modular way (i.e. by *horizontal composition*). But when we discuss refinements by relating abstract and concrete specifications (*vertical composition*) this is a totally different matter where it makes no sense to talk about encapsulation.

Let us now consider two COL-specifications $\mathrm{SP}_{\mathrm{COL}}$, $\mathrm{SPI}_{\mathrm{COL}}$ with signatures Σ_{COL}, ΣI_{COL} resp. together with a (standard) signature morphism $\sigma : \Sigma \to \Sigma I$ (where Σ and ΣI are the underlying standard signatures of Σ_{COL} and ΣI_{COL} resp.). Moreover, let us consider the reduct functor $\_|_\sigma : \mathrm{Alg}(\Sigma I) \to \mathrm{Alg}(\Sigma)$ as a partial function $\_|_\sigma : \mathrm{Alg}_{\mathrm{COL}}(\Sigma I_{\mathrm{COL}}) \to \mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}})$,[1] where:

$\_|_\sigma(AI) \stackrel{\mathrm{def}}{=} AI|_\sigma$ if $AI|_\sigma$ is a Σ_{COL}-algebra,
$\_|_\sigma(AI)$ is undefined otherwise.

Then we have the following fact (see Lemma 1 in [7]).

Fact 2 *$\_|_\sigma : \mathrm{Alg}_{\mathrm{COL}}(\Sigma I_{\mathrm{COL}}) \to \mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}})$ is a* COL*-implementation constructor if $\sigma(S_{\mathrm{Obs}}) \subseteq SI_{\mathrm{Obs}}$ and $\sigma(S_{\mathrm{Loose}}) \subseteq SI_{\mathrm{Loose}}$ where $S_{\mathrm{Obs}}, SI_{\mathrm{Obs}}$ are the observable sorts and $S_{\mathrm{Loose}}, SI_{\mathrm{Loose}}$ are the loose sorts induced by Σ_{COL}, ΣI_{COL} respectively (see Section 2.2).*

Let us stress that vertical signature morphisms used for refinements in [15] satisfy the above conditions due to the fixed universe of visible data. Hence vertical signature morphisms in the sense of [15] are special cases of COL-implementation constructors.

[1] By abuse of notation we use the same symbol $\_|_\sigma$ for the (total) reduct functor on $\mathrm{Alg}(\Sigma I)$ and for its induced partial reduct function on $\mathrm{Alg}_{\mathrm{COL}}(\Sigma I_{\mathrm{COL}})$.

4 Proof Rules for Behavioral Refinements: Part I

In the following we are interested in proof rules for proving behavioural refinement relations $\mathrm{SP}_{\mathrm{COL}} \rightsquigarrow^{\kappa_{\mathrm{COL}}} \mathrm{SPI}_{\mathrm{COL}}$. Obviously, the following basic proof rule follows directly from Definition 5:

For any COL-specifications $\mathrm{SP}_{\mathrm{COL}} = \langle \Sigma_{\mathrm{COL}}, \mathrm{Ax} \rangle$, $\mathrm{SPI}_{\mathrm{COL}} = \langle \Sigma I_{\mathrm{COL}}, \mathrm{AxI} \rangle$, and COL-implementation constructor $\kappa_{\mathrm{COL}} : \mathrm{Alg}_{\mathrm{COL}}(\Sigma I_{\mathrm{COL}}) \to \mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}})$:

$$\text{(Basic)} \quad \frac{\begin{array}{ll} \text{(B1)} & \mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}] \subseteq Dom(\kappa_{\mathrm{COL}}), \\ \text{(B2)} & \kappa_{\mathrm{COL}}(\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}]) \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax} \end{array}}{\mathrm{SP}_{\mathrm{COL}} \rightsquigarrow^{\kappa_{\mathrm{COL}}} \mathrm{SPI}_{\mathrm{COL}}}$$

Note that in (B2) $\kappa_{\mathrm{COL}}(\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}])$ consists of Σ-algebras and that $\models_{\Sigma_{\mathrm{COL}}}$ has been defined not only for COL-algebras but for arbitary Σ-algebras. Of course, the central question is how to prove (B1) and (B2)? For this purpose, we will follow a strategy which consists of two crucial steps. The idea of the first step, elaborated in this section, is to consider instead of the COL-specification $\mathrm{SPI}_{\mathrm{COL}}$ a standard specification SPI (over the FOLEq institution) and instead of κ_{COL} an implementation constructor κ on standard algebras. This idea is related to the (behavioral) refinement notion in [20] and to the concept of an abstractor implementation in [19] where behavioral refinement is, by definition, a relation between the standard interpretation of the implementing specification and the behavioral interpretation of the specification to be implemented. The idea of the second step, elaborated in Section 5, is to reduce the proof of consequences w.r.t. the COL-satisfaction relation $\models_{\Sigma_{\mathrm{COL}}}$ to proofs w.r.t. the standard satisfaction relation of first-order logic with equality.

Let us start by considering COL-implementation constructors which are induced by standard implementation constructors. Given two signatures Σ and ΣI a standard implementation constructor from ΣI to Σ is a function $\kappa : \mathrm{Alg}(\Sigma I) \to \mathrm{Alg}(\Sigma)$ which is iso-preserving. For simplicity, let us assume that κ is total. Since any COL-algebra is also a (standard) algebra it is obvious that any implementation constructor $\kappa : \mathrm{Alg}(\Sigma I) \to \mathrm{Alg}(\Sigma)$ gives rise to a (partial) function $\kappa_{\mathrm{COL}} : \mathrm{Alg}_{\mathrm{COL}}(\Sigma I_{\mathrm{COL}}) \to \mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}})$ where:

$\kappa_{\mathrm{COL}}(AI) \stackrel{\mathrm{def}}{=} \kappa(AI)$ if $\kappa(AI)$ is a Σ_{COL}-algebra,
$\kappa_{\mathrm{COL}}(AI)$ is undefined otherwise.

If this partial function is COL-iso-preserving then κ_{COL} is a COL-implementation constructor *induced* by κ.[2] For instance, Fact 2 provides a simple criterion when reduct functors along standard signature morphisms induce COL-implementation constructors.

[2] In particular this means that κ is compatible with observational equivalences between COL-algebras, a property which is related to the notion of stability introduced by Schoett [20].

To state our second proof rule we consider for any COL-specification $\mathrm{SPI}_{\mathrm{COL}}$ its associated standard specification SPI obtained by forgetting the constructor and observer operations declared in $\mathrm{SPI}_{\mathrm{COL}}$. Then we have for any specifications $\mathrm{SP}_{\mathrm{COL}} = \langle \Sigma_{\mathrm{COL}}, \mathrm{Ax} \rangle$, $\mathrm{SPI}_{\mathrm{COL}} = \langle \Sigma I_{\mathrm{COL}}, \mathrm{AxI} \rangle$, $\mathrm{SPI} = \langle \Sigma I, \mathrm{AxI} \rangle$, where ΣI is the underlying standard signature of ΣI_{COL}, and for any $\kappa : \mathrm{Alg}(\Sigma I) \to \mathrm{Alg}(\Sigma)$ and COL-implementation constructor $\kappa_{\mathrm{COL}} : \mathrm{Alg}_{\mathrm{COL}}(\Sigma I_{\mathrm{COL}}) \to \mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}})$ induced by κ:

$$(\mathrm{Forget}_{\mathrm{COL}}) \quad \frac{\begin{array}{ll} (\mathrm{F1}) & \kappa(\mathcal{M}od[\mathrm{SPI}]) \subseteq \mathrm{Alg}_{\mathrm{COL}}(\Sigma_{\mathrm{COL}}), \\ (\mathrm{F2}) & \kappa(\mathcal{M}od[\mathrm{SPI}]) \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax} \end{array}}{\begin{array}{ll} (\mathrm{B1}) & \mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}] \subseteq Dom(\kappa_{\mathrm{COL}}), \\ (\mathrm{B2}) & \kappa_{\mathrm{COL}}(\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}]) \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax} \end{array}}$$

Lemma 1. *The proof rule* $(\mathrm{Forget}_{\mathrm{COL}})$ *is correct.*

Proof. Assume (F1) and (F2). To prove (B1) and (B2) let $AI \in \mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}]$. Then $AI \models_{\Sigma I_{\mathrm{COL}}} \mathrm{AxI}$ and hence, by Fact 1, $AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}} \models \mathrm{AxI}$. Thus $AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}} \in \mathcal{M}od[\mathrm{SPI}]$. By assumption (F1), $\kappa(AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}})$ is a Σ_{COL}-algebra. Hence, since κ_{COL} is induced by κ, $\kappa_{\mathrm{COL}}(AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}})$ is defined. Since $AI \equiv_{\Sigma I_{\mathrm{COL}}} AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}}$ and κ_{COL} is a COL-implementation constructor, $\kappa_{\mathrm{COL}}(AI)$ is defined as well. Hence, $\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}] \subseteq Dom(\kappa_{\mathrm{COL}})$, i.e. (B1) holds.

Moreover, since $AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}} \in \mathcal{M}od[\mathrm{SPI}]$, the assumption (F2) implies that $\kappa(AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}}) \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax}$. Then, since κ_{COL} is induced by κ, we have $\kappa_{\mathrm{COL}}(AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}}) \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax}$. Since κ_{COL} is a COL-implementation constructor and $AI \equiv_{\Sigma I_{\mathrm{COL}}} AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}}$, we conclude that $\kappa_{\mathrm{COL}}(AI) \equiv_{\Sigma_{\mathrm{COL}}} \kappa_{\mathrm{COL}}(AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}})$. But then $\kappa_{\mathrm{COL}}(AI) \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax}$ holds as well. Hence, $\kappa_{\mathrm{COL}}(\mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}]) \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax}$, i.e. (B2) holds. □

The proof rule $(\mathrm{Forget}_{\mathrm{COL}})$ is also complete if $\mathcal{M}od[\mathrm{SPI}]$ is closed under behavioral quotients, i.e. if any ΣI-algebra $AI \in \mathcal{M}od[\mathrm{SPI}]$ is a ΣI_{COL}-algebra such that $AI/{\approx_{\Sigma I_{\mathrm{COL}}, AI}} \in \mathcal{M}od[\mathrm{SPI}]$. (The proof relies on the fact that, under this assumption, $\mathcal{M}od[\mathrm{SPI}] \subseteq \mathcal{M}od[\mathrm{SPI}_{\mathrm{COL}}]$.)

According to the given proof rules (Basic) and $(\mathrm{Forget}_{\mathrm{COL}})$, the remaining task to prove behavioral refinements is to prove (F1) and (F2). A possible approach to discharge (F1) will be explained with the example in Section 6. A general technique to discharge (F2) is studied in the next section.

5 Proof Rules for Behavioral Refinements: Part II

In this section we focus on how to handle the proof obligation (F2) arising from the proof rule $(\mathrm{Forget}_{\mathrm{COL}})$. Basically we have to show that a set of formulas is behaviorally satisfied by some class of arbitrary algebras. The difficulty here is that these algebras are not COL-algebras w.r.t. the same COL-signature as the one used for the behavioral satisfaction considered, hence we cannot reuse the

ideas and proof techniques detailed in [4]. However, we can rely on another idea, similar to the one introduced in [5], where the proof of behavioral consequences is replaced by the proof of standard consequences using a so-called "*lifting encoding*". The main difference to [5] is, first, that in COL we have distinguished sets of observer and constructor operations which lead to much less observable contexts and constructor terms than in the case of partial observational equalities considered in [5]. Hence, the ideas of [5], which were mainly of theoretical interest, now become practically relevant. Secondly, in contrast to [5], we follow a coinductive style for the encoding of observable contexts which is more appropriate for proving behavioral theorems.

Our lifting encoding relies on a syntactic counterpart of both the constructor terms and the observable contexts. Therefore we need a few preliminary definitions. Remember that given a COL-signature Σ_{COL}, for each state sort s and observable sort s', $\mathcal{C}(\Sigma_{\mathrm{COL}})_{s \to s'}$ denotes the set of the observable Σ_{COL}-contexts with application sort s and result sort s'.

Definition 6 (Lifted signature $\mathcal{AL}(\Sigma_{\mathrm{COL}})$ associated to a COL-signature Σ_{COL}). *Let $\Sigma_{\mathrm{COL}} = (\Sigma, \mathrm{OP}_{\mathrm{Cons}}, \mathrm{OP}_{\mathrm{Obs}})$ be a COL-signature. The induced lifted signature $\mathcal{AL}(\Sigma_{\mathrm{COL}})$ is defined as follows:*

$$\mathcal{AL}(\Sigma_{\mathrm{COL}}) \stackrel{\mathrm{def}}{=} \Sigma \cup \Delta(\mathrm{OP}_{\mathrm{Cons}}) \cup \Lambda(\mathrm{OP}_{\mathrm{Cons}}) \cup \Delta(\mathrm{OP}_{\mathrm{Obs}}) \cup \Lambda(\mathrm{OP}_{\mathrm{Obs}})$$

where $\Delta(\mathrm{OP}_{\mathrm{Cons}})$ *is the signature fragment containing:*
- *for each constrained sort $s \in S_{\mathrm{Cons}}$,* a new sort $c[s]$*;*
- *for each constructor $cons : s_1, \ldots, s_n \to s \in \mathrm{OP}_{\mathrm{Cons}}$,* a new operation $cons^* : \overline{s_1}, \ldots, \overline{s_n} \to c[s]$
  where here and in the following, for any sort $r \in S$, $\overline{r} \stackrel{\mathrm{def}}{=} r$ if $r \in S_{\mathrm{Loose}}$ and $\overline{r} \stackrel{\mathrm{def}}{=} c[r]$ if $r \in S_{\mathrm{Cons}}$;

where $\Lambda(\mathrm{OP}_{\mathrm{Cons}})$ *is the signature fragment containing:*
- *for each constrained sort $s \in S_{\mathrm{Cons}}$,* a new (overloaded) operation $inj : c[s] \to s$*;*
- *for each constrained sort $s \in S_{\mathrm{Cons}}$,* a new (overloaded) unary predicate $G : s$ *on the sort s;*

where $\Delta(\mathrm{OP}_{\mathrm{Obs}})$ *is the signature fragment containing:*
- *for each state sort $s \in S_{\mathrm{State}}$ and observable sort $s' \in S_{\mathrm{Obs}}$, if $\mathcal{C}(\Sigma_{\mathrm{COL}})_{s \to s'}$ is not empty,* a new sort $Cont[s \to s']$*;*[3]
- *for each direct observer $(obs, i) \in \mathrm{OP}_{\mathrm{Obs}}$ with $obs : s_1, \ldots, s_i, \ldots, s_n \to s'$,* a new operation $obs_i^* : \overline{s_1}, \ldots, \overline{s_{i-1}}, \overline{s_{i+1}}, \ldots, \overline{s_n} \to Cont[s_i \to s']$*;*[4]
- *for each indirect observer $(obs, i) \in \mathrm{OP}_{\mathrm{Obs}}$ with $obs : s_1, \ldots, s_i, \ldots, s_n \to s$, and for all observable sorts $s' \in S_{\mathrm{Obs}}$ such that $\mathcal{C}(\Sigma_{\mathrm{COL}})_{s \to s'}$ is not empty,*[5] new (overloaded) operations $obs_i^* : Cont[s \to s'], \overline{s_1}, \ldots, \overline{s_{i-1}}, \overline{s_{i+1}}, \ldots, \overline{s_n} \to Cont[s_i \to s']$*;*

[3] Otherwise, i.e. if $\mathcal{C}(\Sigma_{\mathrm{COL}})_{s \to s'}$ is empty, no new sort is added to reduce the syntactic complexity of the encoding.

[4] The existence of the direct observer (obs, i) entails the non-emptiness of $\mathcal{C}(\Sigma_{\mathrm{COL}})_{s_i \to s'}$, hence the existence of the new sort $Cont[s_i \to s']$.

[5] Hence, the new sort $Cont[s \to s']$ exists, and so does the new sort $Cont[s_i \to s']$.

and where $\Lambda(\mathrm{OP}_{\mathrm{Obs}})$ *is the signature fragment containing:*

- *for each new sort* $Cont[s{\rightarrow}s']$, a new (overloaded) operation $apply : Cont[s{\rightarrow}s'], s \rightarrow s'$;
- *for each state sort* $s \in S_{\mathrm{State}}$, a new (overloaded) binary predicate $\sim : s, s$.

Definition 7 (Lifting axioms $\mathrm{Ax}(\Sigma_{\mathrm{COL}})$ **associated to a** COL**-signature** Σ_{COL}**).** *Let* $\Sigma_{\mathrm{COL}} = (\Sigma, \mathrm{OP}_{\mathrm{Cons}}, \mathrm{OP}_{\mathrm{Obs}})$ *be a* COL*-signature. The lifting axioms* $\mathrm{Ax}(\Sigma_{\mathrm{COL}})$ *associated to the lifted signature* $\mathcal{AL}(\Sigma_{\mathrm{COL}})$ *introduced in Definition 6 are defined as follows:*

$$\begin{aligned}\mathrm{Ax}(\Sigma_{\mathrm{COL}}) \stackrel{\mathrm{def}}{=} {} & \mathrm{SGC}(\Delta(\mathrm{OP}_{\mathrm{Cons}})) \cup \mathrm{Ax}_{\Sigma_{\mathrm{COL}}}(inj) \cup \mathrm{Ax}_{\Sigma_{\mathrm{COL}}}(G) \cup \\ & \mathrm{FSGC}(\Delta(\mathrm{OP}_{\mathrm{Obs}})) \cup \mathrm{Ax}_{\Sigma_{\mathrm{COL}}}(apply) \cup \mathrm{Ax}_{\Sigma_{\mathrm{COL}}}(\sim)\end{aligned}$$

where $\mathrm{SGC}(\Delta(\mathrm{OP}_{\mathrm{Cons}}))$ *is the sort-generation constraint induced by the new sorts* $c[s]$ *and by the new operations* $cons^*$;

where $\mathrm{Ax}_{\Sigma_{\mathrm{COL}}}(inj)$ *states that the operations inj are injective and homomorphic w.r.t.* $\mathrm{OP}_{\mathrm{Cons}}$*, i.e.* $\mathrm{Ax}_{\Sigma_{\mathrm{COL}}}(inj)$ *is the union of:*

- *for each constrained sort* $s \in S_{\mathrm{Cons}}$*, the conditional equation:* $\forall x, y{:}c[s].\ inj(x) = inj(y) \Rightarrow x = y$;
- *for each constrained sort* $s \in S_{\mathrm{Cons}}$ *and constructor* $cons \in \mathrm{OP}_{\mathrm{Cons}}$ *with* $cons : s_1, \ldots, s_n \rightarrow s$*, the implicitly universally quantified equation:* $inj(cons^*(x_1, \ldots, x_n)) = cons(\zeta x_1, \ldots, \zeta x_n)$ *where here and in the following,* $\zeta x_i = x_i$ *if the sort of* x_i *is in* S_{Loose}*, and* $\zeta x_i = inj(x_i)$ *otherwise (i.e., if the sort of* x_i *is of the form* $c[s_i]$ *with* $s_i \in S_{\mathrm{Cons}}$*);*

where $\mathrm{Ax}_{\Sigma_{\mathrm{COL}}}(G)$ *is the set of sentences:*

- *for each constrained sort* $s \in S_{\mathrm{Cons}}$*:* $\forall x{:}s.\ G(s) \Leftrightarrow \exists y{:}c[s].\ x = inj(y)$;

where $\mathrm{FSGC}(\Delta(\mathrm{OP}_{\mathrm{Obs}}))$ *is the free sort-generation constraint induced by the signature fragment* $\Delta(\mathrm{OP}_{\mathrm{Obs}})$*, i.e., by the new sorts* $Cont[s{\rightarrow}s']$ *and the new operations* obs_i^*;

where $\mathrm{Ax}_{\Sigma_{\mathrm{COL}}}(apply)$ *is the set of equations:*

- *for each direct observer* $(obs, i) \in \mathrm{OP}_{\mathrm{Obs}}$ *with* $obs : s_1, \ldots, s_i, \ldots, s_n \rightarrow s'$*, the equation:*
$$\begin{aligned}&\forall x_1{:}\overline{s_1}, \ldots, x_{i-1}{:}\overline{s_{i-1}}, x_{i+1}{:}\overline{s_{i+1}}, \ldots, x_n{:}\overline{s_n}.\ \forall x_i{:}s_i.\\ &apply(obs_i^*(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n), x_i) = \\ &\qquad\qquad obs(\zeta x_1, \ldots, \zeta x_{i-1}, x_i, \zeta x_{i+1}, \ldots, \zeta x_n);\end{aligned}$$
- *for each indirect observer* $(obs, i) \in \mathrm{OP}_{\mathrm{Obs}}$ *with* $obs : s_1, \ldots, s_i, \ldots, s_n \rightarrow s$*, and for all observable sorts* $s' \in S_{\mathrm{Obs}}$ *such that the new sort* $Cont[s{\rightarrow}s']$ *exists, the equations:*
$$\begin{aligned}&\forall c{:}Cont[s{\rightarrow}s'], x_1{:}\overline{s_1}, \ldots, x_{i-1}{:}\overline{s_{i-1}}, x_{i+1}{:}\overline{s_{i+1}}, \ldots, x_n{:}\overline{s_n}.\ \forall x_i{:}s_i.\\ &apply(obs_i^*(c, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n), x_i) = \\ &\qquad\qquad apply(c, obs(\zeta x_1, \ldots, \zeta x_{i-1}, x_i, \zeta x_{i+1}, \ldots, \zeta x_n));\end{aligned}$$

where $\mathrm{Ax}_{\Sigma_{\mathrm{COL}}}(\sim)$ *is the set of sentences:*

– *for each state sort* $s \in S_{\mathrm{State}}$*:*

$$\forall x, y{:}s.\ \left(\bigwedge_{Cont[s \to s']} \forall c{:}Cont[s \to s'].\ apply(c, x) = apply(c, y) \right) \Leftrightarrow x \sim y\ .^{6}$$

The main idea underlying the above definitions is that, to any Σ-algebra A (where Σ is the standard signature underlying the COL-signature Σ_{COL}), corresponds a unique (up to isomorphism) "lifted" $\mathcal{AL}(\Sigma_{\mathrm{COL}})$-algebra $\mathcal{AL}(A)$ which extends A (i.e., $\mathcal{AL}(A)|_{\Sigma} = A$) and satisfies the lifting axioms $\mathrm{Ax}(\Sigma_{\mathrm{COL}})$.[7] Moreover, this lifted algebra $\mathcal{AL}(A)$ is defined in a way which ensures a one to one correspondence between:

– values in the Σ_{COL}-generated part of the Σ-algebra A and values in the carriers of the new sorts $c[s]$ in $\mathcal{AL}(A)$;
– observable contexts, together with appropriate valuations of their variables (in the Σ_{COL}-generated part of A), and values in the (carriers of the) syntactic counterparts $Cont[s \to s']$ in $\mathcal{AL}(A)$. Hence the new sorts $Cont[s \to s']$ reflect the observable contexts in $\mathcal{C}(\Sigma_{\mathrm{COL}})_{s \to s'}$ and they are generated by the constructors obs_i^*. Note that our definition of the constructors of the new sorts $Cont[s \to s']$ follows the coinductive definition of observable contexts given in Definition 1.

We still need a further definition to state the main result of this section.

Definition 8 (Lifted formula $\mathcal{L}(\varphi)$). *Let $\Sigma_{\mathrm{COL}} = (\Sigma, \mathrm{OP}_{\mathrm{Cons}}, \mathrm{OP}_{\mathrm{Obs}})$ be a* COL*-signature and φ be an arbitrary Σ-formula. The lifted formula $\mathcal{L}(\varphi)$ is the $\mathcal{AL}(\Sigma_{\mathrm{COL}})$-formula defined by:*[8]

$$\mathcal{L}(\varphi) \stackrel{\mathrm{def}}{=} \left(\bigwedge_{y:s \in \mathrm{FreeVar}(\varphi) \text{ and } s \in S_{\mathrm{Cons}}} G(y) \right) \Rightarrow \varphi^*$$

where $\mathrm{FreeVar}(\varphi)$ *denotes the free variables of φ, if any, and where φ^* is defined by induction on the structure of φ as follows:*

1. *If φ is an equation $l = r$ between two terms of sort s:*
   if $s \in S_{\mathrm{Obs}}$ then $\varphi^* \stackrel{\mathrm{def}}{=} l = r$, otherwise $s \in S_{\mathrm{State}}$ and $\varphi^* \stackrel{\mathrm{def}}{=} l \sim r$;
2. $(\neg\varphi)^* \stackrel{\mathrm{def}}{=} \neg(\varphi^*)$, $(\varphi_1 \wedge \varphi_2)^* \stackrel{\mathrm{def}}{=} \varphi_1^* \wedge \varphi_2^*$, $(\varphi_1 \vee \varphi_2)^* \stackrel{\mathrm{def}}{=} \varphi_1^* \vee \varphi_2^*$;
3. *If $s \in S_{\mathrm{Loose}}$ then $(\forall x{:}s.\ \varphi)^* \stackrel{\mathrm{def}}{=} \forall x{:}s.\ \varphi^*$,*
   otherwise $s \in S_{\mathrm{Cons}}$ and $(\forall x{:}s.\ \varphi)^* \stackrel{\mathrm{def}}{=} \forall x{:}s.\ [G(x) \Rightarrow \varphi^*]$.

Obviously $\mathcal{L}(\varphi)$ coincides with φ^* if φ is a closed Σ-formula.

---

[6] This sentence is finite, since for any state sort $s \in S_{\mathrm{State}}$, there is only a finite number of sorts $Cont[s \to s']$, where $s' \in S_{\mathrm{Obs}}$ is an observable sort.

[7] In other words, the lifting axioms $\mathrm{Ax}(\Sigma_{\mathrm{COL}})$ induce a strongly persistent free functor from Σ-algebras to $\mathcal{AL}(\Sigma_{\mathrm{COL}})$-algebras.

[8] Note however that the only extra (not in Σ) symbols used in $\mathcal{L}(\varphi)$ are the predicates G and $\sim$. Moreover, note the similarity with [5, Def. 4.1-(iv)].

Theorem 3. *Let* $\Sigma_{\mathrm{COL}} = (\Sigma, \mathrm{OP}_{\mathrm{Cons}}, \mathrm{OP}_{\mathrm{Obs}})$ *be a* COL*-signature. For any class* $\boldsymbol{C} \subseteq \mathrm{Alg}(\Sigma)$ *of* Σ*-algebras and any* Σ*-formula* φ*, we have:*

$$\boldsymbol{C} \models_{\Sigma_{\mathrm{COL}}} \varphi \;\; \textit{if and only if} \;\; \boldsymbol{C} + \langle \mathcal{AL}(\Sigma_{\mathrm{COL}}), \mathrm{Ax}(\Sigma_{\mathrm{COL}}) \rangle \models \mathcal{L}(\varphi)\,.$$

Proof. For lack of space we only detail here the main steps of the proof.

Step 1: In a first step we introduce a semantic lifting of Σ-algebras as follows. Let $\mathcal{L}(\Sigma)$ be the signature Σ enriched by the predicates G and $\sim$ (as they are introduced in Definition 6). Remember that $\mathcal{L}(\varphi)$ is indeed a $\mathcal{L}(\Sigma)$-formula, as pointed out in Definition 8. Now the semantic lifting $\mathcal{L}(A)$ of a Σ-algebra A is defined as being the unique $\mathcal{L}(\Sigma)$-algebra extension of A defined by:

1. $\mathcal{L}(A)|_{\Sigma} \stackrel{\mathrm{def}}{=} A$;
2. For any constrained sort $s \in S_{\mathrm{Cons}}$, and $a \in \mathcal{L}(A)_s = A_s$, $G^{\mathcal{L}(A)}(a)$ if and only if $a \in \mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)_s$;
3. For any state sort $s \in S_{\mathrm{State}}$, and $a, b \in \mathcal{L}(A)_s = A_s$, $a \sim^{\mathcal{L}(A)} b$ if and only if $a \approx_{\Sigma_{\mathrm{COL}}, A, s} b$.

Now we have:

$$A \models_{\Sigma_{\mathrm{COL}}} \varphi \;\; \text{if and only if} \;\; \mathcal{L}(A) \models \mathcal{L}(\varphi).$$

The proof of this fact is similar to the proof of Theorem 4.2 in [5].

Step 2: In a second step we prove that, for any Σ-algebra A:

$$\mathcal{AL}(A)|_{\mathcal{L}(\Sigma)} = \mathcal{L}(A)$$

This indeed results directly from Definitions 6 and 7 (see the comments after the later definition), and from the definitions of $\mathrm{Gen}_{\Sigma_{\mathrm{COL}}}(A)$ and of $\approx_{\Sigma_{\mathrm{COL}}, A}$.

Step 3: From the above we conclude that:
$A \models_{\Sigma_{\mathrm{COL}}} \varphi$ if and only if, according to Step 1,
$\mathcal{L}(A) \models \mathcal{L}(\varphi)$ if and only if, according to Step 2,
$\mathcal{AL}(A)|_{\mathcal{L}(\Sigma)} \models \mathcal{L}(\varphi)$ if and only if, according to the satisfaction condition in CFOLEq, $\mathcal{AL}(A) \models \mathcal{L}(\varphi)$. This is enough to conclude the proof of the theorem, since $\{\, \mathcal{AL}(A) \mid A \in \boldsymbol{C} \,\} = \boldsymbol{C} + \langle \mathcal{AL}(\Sigma_{\mathrm{COL}}), \mathrm{Ax}(\Sigma_{\mathrm{COL}}) \rangle$. □

As a direct consequence of Theorem 3 we obtain the following proof rule. For any COL-specification $\mathrm{SP}_{\mathrm{COL}} = \langle \Sigma_{\mathrm{COL}}, \mathrm{Ax} \rangle$, CFOLEq-specification $\mathrm{SPI} = \langle \Sigma I, \mathrm{AxI} \rangle$ and for any $\kappa : \mathrm{Alg}(\Sigma I) \to \mathrm{Alg}(\Sigma)$:

$$\text{(Lifting)} \quad \frac{\text{(L)} \quad \kappa(\mathcal{M}od[\mathrm{SPI}]) + \langle \mathcal{AL}(\Sigma_{\mathrm{COL}}), \mathrm{Ax}(\Sigma_{\mathrm{COL}}) \rangle \models \mathcal{L}(\mathrm{Ax})}{\text{(F2)} \quad \kappa(\mathcal{M}od[\mathrm{SPI}]) \models_{\Sigma_{\mathrm{COL}}} \mathrm{Ax}}$$

6 Example: Implementation of Sets by Non-redundant Lists

In this section we illustrate the use of our proof rules and proof techniques on a small but non-trivial example.

6.1 The Behavioral Refinement Relation

The following specification SET-COL specifies properties of sets over a loose domain of arbitary elements. As constructors for sets we use the operations *empty* and *add* and as an observer for sets we use the membership test *isin*.[9]

spec SET-COL =
sorts $bool,\ elem,\ set$
ops $true,\ false : bool$;
$empty : set$;
$add : elem \times set \to set$;
$remove : elem \times set \to set$;
$isin : elem \times set \to bool$;
constructors $empty, add$
observer $(isin, 2)$
axioms
$\forall x, y : elem;\ s : set$
%% standard axioms for booleans, plus
- $isin(x, empty) = false$
- $isin(x, add(x, s)) = true$
- $x \neq y \Rightarrow isin(x, add(y, s)) = isin(x, s)$
- $isin(x, remove(x, s)) = false$
- $x \neq y \Rightarrow isin(x, remove(y, s)) = isin(x, s)$
- $add(x, add(x, s)) = add(x, s)$
- $add(x, add(y, s)) = add(y, add(x, s))$

end

As a refinement for sets we consider a classical implementation of sets by non-redundant lists where the set operation *add* is implemented in a such a way that it inserts an element x into a list only if x does not yet occur in the list and the set operation *remove* just removes the first occurrence of an element.

spec LIST-COL =
sorts $bool,\ elem,\ list$
ops $true,\ false : bool$;
$empty : list$;
$cons : elem \times list \to list$;
$head : list \to elem$;
$tail : list \to list$;
$isin : elem \times list \to bool$;
$add : elem \times set \to set$;
$remove : elem \times set \to set$;
constructors $empty, cons$
observers $head, tail$
axioms

[9] All our examples are expressed using a syntactic sugar similar to the one of CASL [8].

$\forall x, y : elem;\ l : list$

%% standard axioms for booleans, plus

- $head(cons(x, l)) = x$
- $tail(cons(x, l)) = l$
- $isin(x, empty) = false$
- $isin(x, cons(x, l)) = true$
- $x \neq y \Rightarrow isin(x, cons(y, l)) = isin(x, l)$
- $isin(x, l) = true \Rightarrow add(x, l) = l$
- $isin(x, l) = false \Rightarrow add(x, l) = cons(x, l)$
- $remove(x, empty) = empty$
- $remove(x, cons(x, l)) = l$
- $x \neq y \Rightarrow remove(x, cons(y, l)) = cons(y, remove(x, l))$

end

To state the refinement relation we still need an appropriate COL-implementation constructor. Since LIST-COL provides already all set operations the simple idea is to forget the list operations *cons*, *head* and *tail* and to perform an appropriate renaming to match the sorts *set* and *list*. For this purpose we consider the (standard) signature morphism $\sigma_{\text{SET}as\text{LIST}} : \Sigma_{\text{SET}} \to \Sigma_{\text{LIST}}$ where Σ_{SET} denotes the underlying (standard) signature of $\mathcal{S}ig[\text{SET-COL}]$,[10] similarly, Σ_{LIST} denotes the underlying (standard) signature of $\mathcal{S}ig[\text{LIST-COL}]$ and $\sigma_{\text{SET}as\text{LIST}}(set) \stackrel{\text{def}}{=} list$, $\sigma_{\text{SET}as\text{LIST}}(x) \stackrel{\text{def}}{=} x$ otherwise.

Since *bool* and *elem* are the observable sorts of both SET-COL and LIST-COL, Fact 2 implies that the reduct functor $\_|_{\sigma_{\text{SET}as\text{LIST}}} : \text{Alg}(\Sigma_{\text{SET}}) \to \text{Alg}(\Sigma_{\text{LIST}})$ on standard algebras induces a COL-implementation constructor:

$$\_|_{\sigma_{\text{SET}as\text{LIST}}} : \text{Alg}_{\text{COL}}(\mathcal{S}ig[\text{LIST-COL}]) \to \text{Alg}_{\text{COL}}(\mathcal{S}ig[\text{SET-COL}]).$$

Then, we claim that LIST-COL is indeed a behavioral refinement of SET-COL w.r.t. $\_|_{\sigma_{\text{SET}as\text{LIST}}}$, i.e. SET-COL $\rightsquigarrow^{\_|_{\sigma_{\text{SET}as\text{LIST}}}}$ LIST-COL.

6.2 Proof of the Refinement

For the proof of the above refinement relation the combination of the rules (Basic) and ($\text{Forget}_{\text{COL}}$) provided in Section 4 shows that it is enough to consider the standard specification LIST obtained from LIST-COL by omitting the declarations of the constructors and observers. Then we have the following two proof obligations:

(F1) $\mathcal{M}od[\text{LIST}]|_{\sigma_{\text{SET}as\text{LIST}}} \subseteq \text{Alg}_{\text{COL}}(\mathcal{S}ig[\text{SET-COL}])$

(F2) $\mathcal{M}od[\text{LIST}]|_{\sigma_{\text{SET}as\text{LIST}}} \models_{\mathcal{S}ig[\text{SET-COL}]} \varphi$ for all axioms φ of SET-COL

To prove (F1) one has to check that the reducts of all models of LIST satisfy the reachability and observability constraints induced by $\mathcal{S}ig[\text{SET-COL}]$; see Section 2. To check the reachability constraint we consider the generated parts of

[10] i.e. Σ_{SET} consists of all sorts and operations of the COL-signature $\mathcal{S}ig[\text{SET-COL}]$ without any constructor or observer declaration.

sort *set* which are constructed by *empty* and *add*. Obviously, due to the implementation of *add*, those parts represent lists without duplicates. Moreover, from the axioms of LIST it follows that the only non-constructor operation *remove* does not introduce duplicates, i.e. the constructor-generated parts are already subalgebras and therefore the reachability constraint is trivially satisfied. For the proof of the observability constraint one has to show that both non-observer operations *add* and *insert* are congruent, i.e. are compatible with the observational equality for sets. For this purpose one can use the lifting encoding considered below and prove the congruence axioms for $\sim$. Another strategy would be first to verify (F2) and then to conclude that both *add* and *remove* are congruent operations since the axioms of SET-COL provide sufficiently complete definitions for *add* and for *remove*.[11]

For the proof of (F2) we will apply the rule (Lifting) of the previous section which says that it is sufficient to prove that for all axioms φ of SET-COL,

$$\mathcal{M}od[\text{LIST}]|_{\sigma_{\text{SET}as\text{LIST}}} + \langle \mathcal{AL}(\mathcal{S}ig[\text{SET-COL}]), \text{Ax}(\mathcal{S}ig[\text{SET-COL}])\rangle \models \mathcal{L}(\varphi)$$

or equivalently, since the (standard) satisfaction relation is compatible with reducts of (standard) algebras,

$$\text{LIST}^* + \langle \mathcal{AL}(\mathcal{S}ig[\text{SET-COL}]), \text{Ax}(\mathcal{S}ig[\text{SET-COL}])\rangle \models \mathcal{L}(\varphi),$$

where LIST^* is the same specification as LIST but with the sort *list* renamed into *set*. For this purpose, we first compute, according to Definition 6, the lifted signature:

$$\mathcal{AL}(\mathcal{S}ig[\text{SET-COL}]) \stackrel{\text{def}}{=} \Sigma_{\text{SET}} \cup \Delta(\text{OP}_{\text{Cons}}) \cup \Lambda(\text{OP}_{\text{Cons}}) \cup \Delta(\text{OP}_{\text{Obs}}) \cup \Lambda(\text{OP}_{\text{Obs}})$$

where $\Delta(\text{OP}_{\text{Cons}})$ consists of
- the new sort $c[set]$
- the new operations $empty^* : c[set]$; $add^* : elem \times c[set] \to c[set]$;

where $\Lambda(\text{OP}_{\text{Cons}})$ consists of
- the new operation $inj : c[set] \to set$;
- the new unary predicate $G : set$;

where $\Delta(\text{OP}_{\text{Obs}})$ consists of
- the new sort $Cont[set \to bool]$;
- the new operation $isin^* : elem \to Cont[set \to bool]$;

and where $\Lambda(\text{OP}_{\text{Obs}})$ consists of
- the new operation $apply : Cont[set \to bool] \times set \to bool$;
- the new binary predicate $\sim : set \times set$.

In the next step, we compute, according to Definition 7, the lifted axioms:

$$\begin{aligned}\text{Ax}(\mathcal{S}ig[\text{SET-COL}]) \stackrel{\text{def}}{=} {} & \text{SGC}(\Delta(\text{OP}_{\text{Cons}})) \cup \text{Ax}_{\mathcal{S}ig[\text{SET-COL}]}(inj) \cup \text{Ax}_{\mathcal{S}ig[\text{SET-COL}]}(G) \cup \\ & \text{FSGC}(\Delta(\text{OP}_{\text{Obs}})) \cup \text{Ax}_{\mathcal{S}ig[\text{SET-COL}]}(apply) \cup \text{Ax}_{\mathcal{S}ig[\text{SET-COL}]}(\sim)\end{aligned}$$

[11] This idea follows a general result presented in [6] for observational logic and equational specifications which still has to be extended to COL and conditional equations.

where SGC($\Delta(\mathrm{OP}_{\mathrm{Cons}})$) is the sort-generation constraint
generated type $c[set] ::= empty^* \mid add^*(elem;\ c[set])$;
where $\mathrm{Ax}_{\mathcal{S}ig[\text{SET-COL}]}(inj)$ consists of
– the conditional equation:
$\forall s, s'{:}c[set].\ inj(s) = inj(s') \Rightarrow s = s'$;
– the implicitly universally quantified equations:
$inj(empty^*) = empty$
$inj(add^*(x, s)) = add(x, inj(s))$
where $\mathrm{Ax}_{\mathcal{S}ig[\text{SET-COL}]}(G)$ consists of
– $\forall s{:}set.\ G(s) \Leftrightarrow \exists s'{:}c[set].\ s = inj(s')$;
where FSGC($\Delta(\mathrm{OP}_{\mathrm{Obs}})$) is the free sort-generation constraint
free type $Cont[set{\rightarrow}bool] ::= isin^*(elem)$;
where $\mathrm{Ax}_{\mathcal{S}ig[\text{SET-COL}]}(apply)$ is the equation:
– $\forall x{:}elem, s{:}set.\ apply(isin^*(x), s) = isin(x, s)$
where $\mathrm{Ax}_{\mathcal{S}ig[\text{SET-COL}]}(\sim)$ is the sentence:
– $\forall s, s'{:}set.$
$(\forall c{:}Cont[set{\rightarrow}bool].\ apply(c, s) = apply(c, s')\ ) \Leftrightarrow s \sim s'$

According to the above axioms, the unary predicate G characterizes those lists (of type *set* because of the performed renaming) which are built with *empty* and *add*. These lists are exactly the lists with no duplicates used for the representation of sets. On the other hand, the above axioms provide also a specification of the binary predicate $\sim$ which relates any two lists containing the same elements (independently of the order and the number of occurrences of these elements).

Let us now compute the lifting $\mathcal{L}(\varphi)$ of all axioms φ of SET-COL which leads, according to Definition 8, to the following set of sentences:

$\forall x, y : elem;\ s : set$
- $isin(x, empty) = false$
- $G(s) \Rightarrow isin(x, add(x, s)) = true$
- $G(s) \Rightarrow (x \neq y \Rightarrow isin(x, add(y, s)) = isin(x, s))$
- $G(s) \Rightarrow isin(x, remove(x, s)) = false$ (1)
- $G(s) \Rightarrow (x \neq y \Rightarrow isin(x, remove(y, s)) = isin(x, s))$
- $G(s) \Rightarrow add(x, add(x, s)) \sim add(x, s)$
- $G(s) \Rightarrow add(x, add(y, s)) \sim add(y, add(x, s))$ (2)

Of course, the remaining task is to show that the lifted axioms given above are consequences of LIST* + $\langle \mathcal{AL}(\mathcal{S}ig[\text{SET-COL}]), \mathrm{Ax}(\mathcal{S}ig[\text{SET-COL}])\rangle$. In most cases the proof is already a direct consequence of the axioms of the LIST specification without the need of the predicates G and $\sim$. The situation is, however, different for the sentences (1) and (2). In the case of (1) the relativization w.r.t. $G(s)$ is indeed crucial, because $isin(x, remove(x, s)) = false$ holds only for those list interpretations of s which have no duplicates, but these are exactly the lists characterized by the predicate symbol G. In the case of (2), the use of $\sim$ instead of "=" is crucial as well, since two lists (also two non-redundant lists) are different

if they contain the same elements but in a different order. In this case they are, however, observationally equal which is axiomatized by $\sim$.[12]

7 Conclusion

We have provided proof techniques to verify behavioral refinements of COL-specifications based on a reduction to first-order specifications and (standard) inductive reasoning. Hence, any inductive theorem prover can be used to prove behavioral refinements. Let us stress that we do not use coinduction to prove the behavioral validity of equations but we use an encoding of observational equalities and generated parts which works for arbitrary first-order formulas. Typical proofs of consequences of the encoding are then performed by induction on the (coinductive) structure of observable contexts. Next steps are the extension of our approach to take into account structured specifications and the study of further examples of implementation constructors like, e.g., specification extension.

Acknowledgement. We are grateful to the anonymous referee of a previous version of this paper for valuable remarks.

References

1. E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors. *Algebraic Foundations of Systems Specification*. Springer, 1999.
2. M. Bidoit, M.-V. Cengarle, and R. Hennicker. Proof systems for structured specifications and their refinements. In *[1]*, chapter 11, pages 385–433. Springer, 1999.
3. M. Bidoit and R. Hennicker. Modular correctness proofs of behavioural implementations. *Acta Informatica*, 35:951–1005, 1998.
4. M. Bidoit and R. Hennicker. Constructor-based observational logic. *Journal of Logic and Algebraic Programming*, 67 (1-2):3–51, 2006. Preliminary version available at `www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/BID-HEN-JLAP.pdf`.
5. Michel Bidoit and Rolf Hennicker. Behavioural theories and the proof of behavioural properties. *Theoretical Computer Science*, 165(1):3–55, 1996.
6. Michel Bidoit and Rolf Hennicker. Observer complete definitions are behaviourally coherent. In *Proc. OBJ/CafeOBJ/Maude Workshop at Formal Methods'99, Toulouse, France, Sep. 1999*, pages 83–94. THETA, 1999.
7. Michel Bidoit and Rolf Hennicker. Externalized and internalized notions of behavioral refinement. In Dang Van Hung and Martin Wirsing, editors, *Proceedings of the 2nd International Colloquium on Theoretical Aspects of Computing (ICTAC'05)*, volume 3722 of *Lecture Notes in Computer Science*, pages 334–350, Hanoi, Vietnam, October 2005. Springer.
8. Michel Bidoit and Peter D. Mosses. CASL *User Manual – Introduction to Using the Common Algebraic Specification Language*, volume 2900 of *Lecture Notes in Computer Science*. Springer, 2004.

[12] As a side remark the reader may note that in the lifted sentence requiring the idempotency of *add* the predicate symbol $\sim$ could indeed be replaced by "=" due to the relativization w.r.t. $G(s)$ and the implementation of *add*.

9. R. Diaconescu and K. Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.
10. H. Ehrig and H.-J. Kreowski. Refinement and implementation. In *[1]*, chapter 7, pages 201–242. Springer, 1999.
11. J. Goguen and J.A. Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In *Proc. ICALP'82*, volume 140 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 1982.
12. J. Goguen and G. Roşu. Hiding more of hidden algebra. In J.M. Wing, J. Woodcock, and J. Davies, editors, *Proc. Formal Methods (FM'99)*, volume 1709 of *Lecture Notes in Computer Science*, pages 1704–1719. Springer, 1999.
13. Joseph Goguen and Rod Burstall. Institutions: abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
14. J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of Abstract Data Types*. Wiley and Teubner, 1996.
15. G. Malcolm and J. Goguen. Proving correctness of refinement and implementation. Technical Report PRG-114, Oxford University Computing Laboratory, 1994.
16. Peter D. Mosses, editor. CASL *Reference Manual*, volume 2960 of *Lecture Notes in Computer Science*. Springer, 2004.
17. F. Orejas, M. Navarro, and A. Sanchez. Implementation and behavioural equivalence. In *Recent Trends in Data Type Specification*, volume 655 of *Lecture Notes in Computer Science*, pages 93–125. Springer, 1993.
18. D. Sannella and A. Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Sciences*, 34:150–178, 1987.
19. D.T. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: implementation revisited. *Acta Informatica*, 25:233–281, 1988.
20. O. Schoett. Data abstraction and correctness of modular programming. Technical Report CST-42-87, University of Edinburgh, 1987.
21. Andrzej Tarlecki. Institutions: An Abstract Framework for Formal Specification. In *[1]*, chapter 4, pages 105–130. Springer, 1999.
22. Martin Wirsing. Algebraic Specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 13, pages 676–788. Elsevier Science Publishers B.V., 1990.