LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

Proving correctness of constructor implementations

Proving correctness of constructor implementations

by

Jordi Farres-Casals

ECS-LFCS-89-72
(also published as CSR-286-89)

LFCS Report Series

January 1989

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Proving correctness of constructor implementations

Jordi Farrés-Casals

January 11, 1989

Abstract

In [ST 86] the notion of constructor implementation was introduced generalizing previous well-known implementation definitions such as in [EKMP 82]. In this paper we explore a proof strategy for this kind of implementation in a specification language close to ASL. The results show that these proofs are feasible in some cases, but since a general result is not attainable we are satisfied by coping with the most common cases.

1 Introduction

Any formal program development method, and in particular the algebraic ones, consists of a specification language, an implementation notion, and some techniques for proving correctness of an implementation with respect to a specification. In recent years a new framework for specification has been defined [SW 83, ST 85, ST 86] where stress has been put on defining an elegant implementation notion and using as specification language a small collection of powerful specification building operations. Some important features of this work are the generalization to an arbitrary institution [GB 84] of as much of the work done on algebraic specification as possible, and the study of behaviourally abstract specifications in the context of a model-oriented language. Our purpose in this paper is to explore some techniques for proving correctness of implementations in this framework.

The study of implementation proofs for such a language is particularly interesting for two reasons.

Firstly the degree of generality of the specification language and the implementation definition considered will bring together different results and problems which have already been studied in more particular frameworks.

In second place the abstraction provided by the language with respect to the underlying logic used and the stress on matters of structure will focus our proofs also towards structural considerations contrary to many other approaches which stick to theorem proving problems.

The paper is divided into six sections.

In the second section we describe the framework on which the paper is based, first recalling some background on algebra and then defining the specification language and the implementation notion as they were defined in [ST 85, ST 86].

In section three we show the problems which arise when we try to handle proofs of implementations simply by using theories to describe the class of models of a structured specification. An analysis of the possible relationships between a theory and a class of models is given, and the requirements for such a method to work are shown.

In section four we argue informally about the feasibility of a proof for an example. The language is slightly simplified and a graphical representation is defined.

In section five a proof strategy is presented. Firstly the general procedure relying on some transformation rules is described and proven to be a generalization of the approach in section three, and then the transformation rules are listed and proven correct. These rules constitute an *incomplete* calculus of specification transformation built around a collection of *basic* judgements on structured specifications.

Finally some remarks are made in section six on possible extensions and related work.

2 Preliminaries

In order to make this paper self-contained we recall in this section some preliminary concepts from [ST 85] and [ST 86].

2.1 Algebraic background

In this section some of the usual concepts of algebra and algebraic specification are reviewed. Some definitions from category theory are assumed known.

A **signature** is a pair $\langle S, \Omega \rangle$ where S is a set of names (sort names) and Ω is a family of sets of names (operation names) $\{\Omega_{w,s}\}_{w \in S^*, s \in S}$. An element f of $\Omega_{w,s}$ is denoted by $f : w \rightarrow s$.

A **signature morphism** $\sigma : \langle S, \Omega \rangle \rightarrow \langle S', \Omega' \rangle$ is a pair $\langle \sigma_S, \sigma_\Omega \rangle$ where $\sigma_S : S \rightarrow S'$ and σ_Ω is a family of sets of functions $\{\sigma_{w,s} : \Omega_{w,s} \rightarrow \Omega'_{\sigma_S^*(w),\sigma_S(s)}\}_{w \in S^*, s \in S}$ where σ_S^* means application of σ_S to every component of w. For the sake of readability we will omit the subscripts and superscripts of σ.

Signatures together with signature morphisms form a category.

Given a signature $\Sigma = \langle S, \Omega \rangle$, a **(total) Σ-algebra** consists of a S-indexed family of sets (carriers) $|A| =_{def} \{|A|_s\}_{s \in S}$ and a function $f_A : |A|_{s_1} \times ... \times |A|_{s_n} \rightarrow |A|_s$ for every $f : s_1, ..., s_n \rightarrow s$ in Ω.

Let A and B be two Σ-algebras. A **Σ-homomorphism** from A to B, $h : A \rightarrow B$, is a S-indexed family of functions $\{h_s\}_{s \in S}$ between the corresponding carriers $h_s : |A|_s \rightarrow |B|_s$ consistent with the operations in Ω; i.e. for all $f : s_1, ..., s_n \rightarrow s$ in Ω and $a_1 \in |A|_{s_1}, ..., a_n \in |A|_{s_n}$, $h_s(f_A(a_1, ..., a_n)) = f_B(h_{s_1}(a_1), ..., h_{s_n}(a_n))$.

The algebras over a signature Σ together with their Σ-homomorphisms form a category, denoted by $Alg(\Sigma)$. In general the collection of objects of a category C are denoted by $|C|$.

A **first-order Σ-sentence** is a closed term built from the logical connectives $\neg, \wedge, \vee$ and $\Rightarrow$, the quantifiers $\forall$ and $\exists$, and equations $t = t'$ as atomic formulae, where t and t' are Σ-terms.

A Σ-algebra A **satisfies** $t = t'$ for a valuation $\nu : X \rightarrow |A|$, if the values of t and t' in A under valuation ν, denoted by $\nu^{\#}(t)$ and $\nu^{\#}(t')$ respectively, are the same. Satisfaction of a first-order Σ-sentence φ by an algebra A is defined as usual from the satisfaction of its atomic formulae (equations), and denoted $A \models \varphi$. This notation can be extended to satisfation of a set of Σ-sentences, Δ, by a class of Σ-algebras, Γ, when it is the case that each algebra in Γ satisfies each sentence in Δ; $\Gamma \models \Delta$.

Given a signature morphism $\sigma : \Sigma \rightarrow \Sigma'$, the **$\sigma$-reduct** of a Σ'-algebra A' is defined as the Σ-algebra $A'|_\sigma$ such that for all sorts s in Σ, $|(A'|_\sigma)|_s =_{def} |A'|_{\sigma(s)}$ and for all $f : s_1, ..., s_n \rightarrow s$ in Σ, $f_{A'|_\sigma} = \sigma(f)_{A'}$.

We can also define the σ-reduct of a Σ'-homomorphism $h' : A' \rightarrow B'$, where A' and B' are Σ'-algebras, as the Σ-homomorphism $h'|_\sigma : A'|_\sigma \rightarrow B'|_\sigma$ with $(h'|_\sigma)_s =_{def} h'_{\sigma(s)}$ for all sorts s in Σ.

$-|_\sigma$ turns out to be a functor from $Alg(\Sigma')$ to $Alg(\Sigma)$.

Given a set of Σ'-equations eq we can define $Mod[\langle \Sigma', eq \rangle]$ to be the full subcategory of $Alg(\Sigma')$ composed by those algebras satisfying the equations in eq. It is a known result of universal algebra that:

Fact 2.1 *For every morphism $\sigma : \Sigma \rightarrow \Sigma'$ and set of Σ'-equations eq, the left adjoint $Free_{-|_\sigma}$ of the reduct functor $-|_\sigma : Mod[\langle \Sigma', eq \rangle] \rightarrow Alg(\Sigma)$ exists.*

A **subalgebra** B of a Σ-algebra A is a Σ-algebra such that for every sort s in Σ, $|B|_s \subseteq |A|_s$ and for all $f : s_1, ..., s_n \rightarrow s$ in Σ and $b_1 \in |B|_{s_1}, ..., b_{s_n} \in |B|_n$, $f_A(b_1, ..., b_n) = f_B(b_1, ..., b_n)$.

We say that a subalgebra of a Σ-algebra A is **reachable on sort** s if it differs from A only on $|A|_s$ and it contains no proper subalgebra which differs also only on $|A|_s$. Similarly reachability can be defined with respect to a set of sorts.

Fact 2.2 *For all Σ-algebras A and set of Σ-sorts S there is a unique reachable subalgebra $Reach_S(A)$ of A on sorts S.*

Let $V/\equiv$ denote the quotient of the set V modulo the equivalence relation $\equiv \subseteq V \times V$.

A **congruence $\sim$ on an algebra** A is an equivalence relation $\sim \subseteq |A| \times |A|$ which is consistent with the operations of A; i.e. for all $f : s_1, ..., s_n \rightarrow s$ in the signature of A, $a_1, b_1 \in |A|_1, ..., a_n, b_n \in |A|_n$ and $a_1 \sim b_1, ..., a_n \sim b_n$ then $f_A(a_1, ..., a_n) \sim f_A(b_1, ..., b_n)$.

Fact 2.3 *Let $\sim$ be a congruence on A. Then $A/\sim$ is a well-defined Σ-algebra, where $|A/\sim|_s = |A|_s/\sim_s$ and for all $f : s_1, ...s_n \rightarrow s$ in Σ and $a_1 \in |A|_1, ..., a_n \in |A|_n$, $f_{A/\sim}([a_1], ..., [a_n]) = [f_A(a_1, ..., a_n)]$.*

For any set of Σ-equations eq, we denote by $\sim_A^{eq}$ the least congruence on an algebra A such that for all equations $\forall X.t = t'$ in eq and valuations $\nu : X \rightarrow |A|$, the values of the two terms are congruent: $\nu^\#(t) \sim_A^{eq} \nu^\#(t')$.

2.2 Semantics of specification building operations

A specification SP denotes a pair $\langle Sig[SP], Mod[SP] \rangle$ where $Sig[SP]$ is the signature of SP and $Mod[SP] \subseteq |Alg(Sig[SP])|$ are the models of SP.

The specification language considered is composed of the following specification building operations (SBO's for short).

ϵ_Σ Unrestricted specification.

$$Sig[\epsilon_\Sigma] = \Sigma \qquad Mod[\epsilon_\Sigma] = |Alg(\Sigma)|$$

A_Φ Imposition of some axioms (Σ-sentences) Φ on a specification SP[1] over Σ.

$$Sig[A_\Phi SP] = Sig[SP] \qquad Mod[A_\Phi SP] = \{M \in Mod[SP] |\ M \models \Phi\}$$

U_I Union of a collection of specifications SP_i for $i \in I$ over the same signature Σ.

$$Sig[U_{i \in I} SP_i] = \Sigma \qquad Mod[U_{i \in I} SP_i] = \bigcap_{i \in I} Mod[SP_i]$$

T_σ Translate a specification over Σ according to a morphism $\sigma : \Sigma \rightarrow \Sigma'$.

$$Sig[T_\sigma SP] = \Sigma' \qquad Mod[T_\sigma SP] = \{M' \in |Alg(\Sigma')| \mid M'|_\sigma \in Mod[SP]\}$$

[1] In [SW 83] a basic specification is defined which fixes the signature and imposes some axioms at the same time. Both presentations are trivially equivalent but ours is closer to the graphical notation to be introduced in section 4. Throughout the paper A_Φ will be used for $A_\Phi \epsilon_\Sigma$ when Σ consists of the symbols used in Φ.

D_σ Derive a specification over Σ' according to a morphism $\sigma : \Sigma \to \Sigma'$.

$$Sig[D_\sigma SP] = \Sigma \qquad Mod[D_\sigma SP] = \{M|_\sigma \mid M \in Mod[SP]\}$$

M_S Select those models of a specification over Σ which are reachable on a set of sorts S of Σ.

$$Sig[M_S SP] = Sig[SP] \qquad Mod[M_S SP] = \{M \in Mod[SP] \mid Reach_S(M) = M\}$$

R_S Restrict to the reachable submodels of a specification over Σ on a set of sorts S of Σ.

$$Sig[R_S SP] = Sig[SP] \qquad Mod[R_S SP] = \{Reach_S(M) \mid M \in Mod[SP]\}$$

Q_{eq} Quotient the models of a specification over Σ w.r.t. a set of Σ-equations eq.

$$Sig[Q_{eq} SP] = Sig[SP] \qquad Mod[Q_{eq} SP] = \{M/\sim_{eq} \mid M \in Mod[SP]\}$$

F_σ^{eq} Free extension of the models of a specification over Σ according to a morphism $\sigma : \Sigma \to \Sigma'$ and Σ'-equations eq.

$$Sig[F_\sigma^{eq} SP] = \Sigma' \qquad Mod[F_\sigma^{eq} SP] = \{Free_{-|\sigma}(M) \mid M \in Mod[SP]\}$$

These SBO's correspond to the ones directly definable in any institution [ST 85] plus the most common ones involved in implementations [ST 86]. They can be classified into three groups: **selectors** such as A, U, and M which select some of the models of a specification; **constructors** such as D, R, Q and F which return the *image class* formed by applying a function to all the models of a specification; and **inverse-constructors** such as T which apply the inverse of a function to the models of a specification.

2.3 Constructor implementation

A primitive implementation relation can be defined as the simple inclusion of model classes, and is called **refinement.**

$$For\ SP_1, SP_2\ such\ that\ Sig[SP_1] = Sig[SP_2],\ SP_1 \rightsquigarrow SP_2 \stackrel{\text{def}}{\Longleftrightarrow} Mod[SP_1] \supseteq Mod[SP_2]$$

On top of this, more elaborate concepts can be defined such as constructor and abstractor implementations [ST 86]. In this paper we consider constructor implementations only.

A **constructor** κ is defined as a SBO characterized by its *concreteness.* It can be viewed as a program $Func_\kappa$ built on top of a specification SP_2, in the sense

that as soon as we replace that specification SP_2 by a program p implementing it, we get a program $Func_\kappa(p)$ for the whole specification SP_1. Since our models are algebras, a constructor κ is a SBO whose semantics is determined by a function $Func_\kappa : Alg(\Sigma_2) \to Alg(\Sigma_1)$ in the following way:

$$For\ SP_2\ such\ that\ Sig[SP_2] = \Sigma_2,\ \ Sig[\kappa SP_2] = \Sigma_1\ \ Mod[\kappa SP_2] = \{Func_\kappa(M) | M \in Mod[SP$$

Constructor implementations are defined as follows:

$$SP_1 \overset{\kappa}{\rightsquigarrow} SP_2 \overset{\text{def}}{\Longleftrightarrow} SP_1 \rightsquigarrow \kappa(SP_2)$$

This implementation notion is transitive and compatible with the SBO's; i.e. implementations compose vertically and horizontally. Thus it is suitable for a framework of structured program development (as in the CAT system [GB 80]).

Among our SBO's there are four constructors: D, Q, R and F. Also, the composition of two constructors is a constructor.

Constructor implementations can be compared with other common implementation notions such as the one in [EKMP 82], concluding that these constructors correspond to other approaches' semantical constructions which have been shifted from the implementation's semantics to the specification language. An important difference is that in our framework constructors can be composed freely instead of following a fixed pattern as in [EKMP 82].

Example: $SetNat \overset{\kappa}{\rightsquigarrow} BagNat$.

In the following Nat stands for the specification of the natural numbers including booleans and '$\langle\langle$`sorts` S `opns` $\Omega\rangle$, `axioms` $\Phi\ \rangle$' is our notation for the basic specification $A_\Phi \epsilon_{\langle S,\Omega\rangle}$. We begin by giving specifications of sets and bags of natural numbers, but where the sort *nat* and operations on natural numbers need not be interpreted as usual:

$$\begin{array}{lll} Set =_{def} \langle Sig[Nat] \cup \langle \texttt{sorts} & set & \\ \qquad\qquad\qquad\qquad \texttt{opns} & \emptyset :\rightarrow set & \\ & add : nat, set \to set & \\ & isempty : set \to bool & \\ & \in : nat, set \to bool\ \rangle, & \\ \qquad\qquad \texttt{axioms} & add(a, add(b, S)) = add(b, add(a, S)) & \\ & add(a, add(a, S)) = add(a, S) & \\ & isempty(\emptyset) = true & \\ & isempty(add(a, S)) = false & \\ & a \in \emptyset = false & \\ & a \in add(a, S) = true & \\ & a \neq b \Rightarrow a \in add(b, S) = a \in S\rangle & \end{array}$$

$$
\begin{array}{lll}
Bag =_{def} \langle Sig[Nat] \cup \langle \texttt{sorts} & bag & \\
\qquad\qquad \texttt{opns} & \emptyset :\rightarrow bag & \\
& add : nat, bag \rightarrow bag & \\
& isempty : bag \rightarrow bool & \\
& count : nat, bag \rightarrow nat \rangle, & \\
\texttt{axioms} & add(a, add(b, S)) = add(b, add(a, S)) & \\
& isempty(\emptyset) = true & \\
& isempty(add(a, S)) = false & \\
& count(a, \emptyset) = 0 & \\
& count(a, add(a, S)) = succ(count(a, S)) & \\
& a \neq b \Rightarrow count(a, add(b, S)) = count(a, S) \rangle &
\end{array}
$$

The specifications of standard sets and bags of natural numbers can be obtained by combining *Set* and *Bag* with the given specification *Nat* of natural numbers, and taking the reachable models.

Let ι_1 and ι_2 be the inclusion morphisms of $Sig(Nat)$ into $Sig[Set]$ and $Sig[Bag]$ respectively. Then define:

$SetNat = M_{set}((T_{\iota_1} Nat) U Set)$
$BagNat = M_{bag}((T_{\iota_2} Nat) U Bag)$

One way to construct sets from bags is to add the membership operation $\in$, rename *bag* to *set*, forget the function *count* and identify those bags which have different numbers of occurrences of the same elements. This can expressed using three constructors as follows:

$$\kappa =_{def} Q_{eq_1} D_{\sigma_1} F^{eq_2}_{\sigma_2}$$

In $F^{eq_2}_{\sigma_2}$, $\sigma_2 : Sig[BagNat] \hookrightarrow (Sig[BagNat] \cup \langle \texttt{opns} \in : nat, bag \rightarrow bool \rangle)$ is the inclusion morphism which adds the membership operation and $eq_2 =_{def} \{\forall\ a : Nat,\ B : bag.\ a \in B = count(a, B) > 0\}$ defines it.

In D_{σ_1}, $\sigma_1 : Sig[SetNat] \rightarrow (Sig[BagNat] \cup \langle \texttt{opns} \in : nat, bag \rightarrow bool \rangle)$ is the signature morphism mapping *set* to *bag* which is the identity on the other sorts and operations. Note that *count* is not in the image of σ_1 and therefore it will be forgotten.

In Q_{eq_1}, $eq_1 =_{def} \{\forall a : Nat, S : set.\ add(a, add(a, S)) = add(a, S)\}$ identifies those bags with the same elements.

Now we can conjecture that $SetNat \overset{\kappa}{\rightsquigarrow} BagNat$, but it has still to be proven correct.

3 Simple strategy

An intuitively appealing way of proving a refinement correct is to prove that those properties required by the specification are satisfied by the implementation. In

order to do this we can use theories to represent these collections of properties. We can easily see how a basic specification, A_Φ, is represented by the collection of sentences (properties) Φ and a structured specification $A_{\Phi_1} U A_{\Phi_2}$ is similarly represented by $\Phi_1 \cup \Phi_2$.

Our first approach, called the *simple strategy*, will explore the possibility of representing specifications by theories and reducing implementation proofs to syntactic proofs using those theories.

3.1 Theories versus models

Since not every class of algebras can be represented by a theory and, in particular, this turns out to be the case for the model classes of some specifications in our language, we study the possible relations between a class of Σ-algebras Γ and a theory Δ over Σ which tries to *capture* the properties of Γ.

Definition 3.1 *The* **closure** *of a set of Σ-sentences Φ under semantic entailment is the set of those sentences satisfied by all those algebras satisfying the sentences in Φ, and is denoted by $Cl(\Phi)$.*

A **theory** *Δ over Σ is a set of Σ-sentences closed under semantic entailment; i.e. $Cl(\Delta) = \Delta$.*

Corollary 3.2 *For every two theories Δ_1 and Δ_2, semantic entailment between theories, $Mod[\Delta_1] \models \Delta_2$ ($\Delta_1 \models \Delta_2$ for short), is equivalent to theory inclusion, $\Delta_1 \supseteq \Delta_2$.*

Defining $Th[\Gamma]$ as the theory consisting of the Σ-sentences satisfied by all the algebras in Γ and $Mod[\Delta]$ as the class of Σ-algebras satisfying all the sentences in Δ, we can compare by inclusion Γ with $Mod[\Delta]$ and Δ with $Th[\Gamma]$.

Definition 3.3 *Given a class of Σ-algebras Γ and a theory Δ over Σ,*

Δ is **sound** *w.r.t. Γ if $\Delta \subseteq Th[\Gamma]$.*

Δ is **complete** *w.r.t. Γ if $\Delta \supseteq Th[\Gamma]$.*

Δ is **M-sound** *w.r.t. Γ if $Mod[\Delta] \supseteq \Gamma$.*

Δ is **M-complete** *w.r.t. Γ if $Mod[\Delta] \subseteq \Gamma$.*

When the inclusions are proper we use the terms strictly sound, strictly complete, and so on.

Δ is **exact** *w.r.t. Γ if $Mod[\Delta] = \Gamma$.*

These definitions are related by the following lemmas.

Lemma 3.4 *Δ is sound w.r.t. Γ iff Δ is M-sound w.r.t. Γ.*

Lemma 3.5 *If Δ is M-complete w.r.t. Γ then Δ is complete w.r.t. Γ.*

Corollary 3.6 *If Δ is exact w.r.t. Γ then Δ is sound and complete w.r.t. Γ.*

The proof of each of these lemmas is an immediate application of the Galois connection between theories and classes of algebras. For example:

Proof of Lemma 3.4 If Δ is sound then $\Delta \subseteq Th[\Gamma]$. Applying Mod on both sides gives $Mod[\Delta] \supseteq Mod[Th[\Gamma]]$, and recalling that $Mod[Th[\Gamma]] \supseteq \Gamma$ we get $Mod[\Delta] \supseteq \Gamma$ which is the definition of M-sound.

If Δ is M-sound then $Mod[\Delta] \supseteq \Gamma$. Applying Th on both sides, and recalling that by definition of a theory $\Delta = Th[Mod[\Delta]]$ we get $\Delta \subseteq Th[\Gamma]$ which is the definition of sound.

Summing up, we end with six different possible cases: Δ can be exact; sound and complete but not exact; sound but not complete; M-complete but not sound; complete but neither M-complete nor sound; or not related w.r.t. Γ. But not all these cases can occur for the same class of algebras.

Definition 3.7 *A class of algebras Γ is axiomatizable if it there exists a theory Δ_Γ such that $\Gamma = Mod[\Delta_\Gamma]$.*

Proposition 3.8 *A class of algebras Γ is axiomatizable iff every complete theory w.r.t. Γ is also M-complete w.r.t. Γ.*

Now we can classify theories again. For an axiomatizable class of algebras a theory can be exact, sound (M-sound) and not complete, complete (M-complete) and not sound, or not related. For a non-axiomatizable class of algebras all the cases are possible except the exact one.

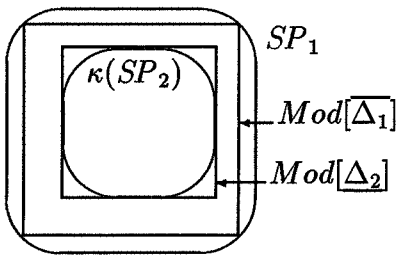
3.2 Basic proof rule

The previous definitions are connected to our implementation notion as follows:

Theorem 3.9 (Basic rule) *If there exists an M-complete theory $\overline{\Delta_1}$ for SP_1, a sound theory $\underline{\Delta_2}$ for $\kappa(SP_2)$ and $\underline{\Delta_2} \vdash \overline{\Delta_1}$, then $SP_1 \overset{\kappa}{\rightsquigarrow} SP_2$. Here $\vdash$ stands for the syntactic consequence relation associated to $\models$ in FOLEQ.*

Proof Since $\vdash$ is sound, $\underline{\Delta_2} \vdash \overline{\Delta_1}$ implies $\underline{\Delta_2} \models \overline{\Delta_1}$, which by corollary 3.2 yields $\underline{\Delta_2} \supseteq \overline{\Delta_1}$. Applying Mod we get $Mod[\underline{\Delta_2}] \subseteq Mod[\overline{\Delta_1}]$, and by definition of M-complete and M-sound (equivalent to sound) $Mod[\kappa(SP_2)] \subseteq Mod[\underline{\Delta_2}] \subseteq Mod[\overline{\Delta_1}] \subseteq Mod[SP_1]$.

The situation where the theorem applies can be represented by the diagram on the right. Every figure represents a class of models; in particular, squares represent those classes of models which have an exact theory.
The theorem states that the inclusion of the two ovals can be proved by showing the inclusion of the two squares.



Looking at the drawing the basic rule seems obvious but also the lack of anything better unless every class of models described by the specification language can be characterized by an exact theory. The situation can only be improved by increasing the expressibility of theories and/or weakening the expressibility of the specification language.

3.3 Inference rules for SBO's

The inference rules are those rules which allow us to infer a theory from another theory.

Definition 3.10 *An* **inference rule** Λ *from* Σ_1 *to* Σ_2 *is a binary relation between sets of* Σ_1*-sentences and* Σ_2*-sentences.*

The **theory** Δ_{inf} **inferred** *from a theory* Δ *using* Λ *is* $\Delta_{inf} = Cl(\{\varphi | \exists \Phi_1 \subseteq \Delta.\ (\Phi_1, \varphi) \in \Lambda\})$.

Inference rules are related to SBO's analogously as theories are related to specifications, therefore we can extend the definitions given for theories w.r.t. classes of models to definitions for inference rules w.r.t. SBO's:

Definition 3.11 *For a unary SBO* ξ *we say that:*

An inference rule Λ *is* **sound** *w.r.t.* ξ*, if whenever is applied to a theory* Δ *which is sound w.r.t. a specification SP (w.r.t.* $Mod[SP]$*) yields a sound theory w.r.t.* $\xi(SP)$.

An inference rule Λ *is* **M-complete** *w.r.t.* ξ*, if whenever applied to a theory* Δ *which is M-complete w.r.t. a specification SP yields an M-complete theory w.r.t.* $\xi(SP)$.

An inference rule Λ *is* **complete** *w.r.t.* ξ*, if whenever is applied to a theory* Δ *which is complete w.r.t. a specification SP yields a complete theory w.r.t.* $\xi(SP)$.

For binary or n-ary SBO's we can easily extend the definitions of inference rule and inferred theory in order to infer a theory from several theories. Then, soundness (M-completeness or completeness) of the inference requires sound (M-complete or complete) theories for each of the arguments.

Sound inference rules are usually denoted by expressions such as $SP \vdash f(\varphi) \Rightarrow \xi SP \vdash g(\varphi)$ where f and g are functions from sentences to sentences and ξ is a explicit reference to the SBO with which the rule is related. The inference rule denoted is $\Lambda = \{(\{f(\varphi)\}, g(\varphi)) \mid for\ all\ sentences\ \varphi\}$.

Sometimes the inferred sentences are independent of the specification to which ξ is applied, then expressions like $\xi SP \vdash \varphi$ constrained by a side condition relating ξ to φ are used. The inference rule denoted is $\Lambda = \{(true, \varphi)\}$ for all φ satisfying the side condition.

Moreover, the union of inference rules between two signatures is also an inference rule. This allows us to charactarize as complete or M-complete collection of rules for a given SBO.

Some inference rules given in [ST 85], after slightly adapting them to our SBO's, can be characterized as follows (see section 5.4 for proofs):

$SP \vdash \varphi \Rightarrow A_\Phi SP \vdash \varphi$ *and* $A_\Phi SP \vdash \varphi$ (if $\Phi \vdash \varphi$)	Sound and M-complete (exact).
$SP_j \vdash \varphi, j \in I \Rightarrow U_{j \in I} SP_j \vdash \varphi$	Sound and M-complete (exact).
$SP \vdash \varphi \Rightarrow T_\sigma SP \vdash \sigma(\varphi)$	Sound and M-complete (exact).
$SP \vdash \sigma(\varphi) \Rightarrow D_\sigma SP \vdash \varphi$	Sound and complete.
$SP \vdash \varphi \Rightarrow M_S SP \vdash \varphi$	Sound.

The fact of having a sound and complete rule for D entails by proposition 3.8 that there is no exact rule for D at all.

Moreover there are no similar proof rules for R, Q and F. In particular, none of these is necessarily consistent w.r.t. its parameter; i.e. what is true in SP does not need to be true in $R_S SP$, $Q_{eq} SP$ or $F_\sigma SP$. For example, in a sentence like $\exists x : s. \forall y : s. x > y$, x can refer to a non-reachable element, therefore this sentence may be true in a specification SP and false in $R_S SP$.

The simple strategy for proving correctness of constructor implementations $SP_1 \overset{\kappa}{\rightsquigarrow} SP_2$ is the one which relies on the basic proof rule, and which tries to infer $\overline{\Delta_1}$ using M-complete inference rules on the structure of SP_1 (specification branch) and $\underline{\Delta_2}$ using sound inference rules on the structure of $\kappa(SP_2)$ (implementation branch). This is too restrictive for use in practice since it only allows SBO's with M-complete inference rules to be used in specifications and SBO's with sound rules as constructors (provided they are constructors). This leaves us with only A, U and T in the specification language, and this is far too litle.

4 Problems, Structure and Strategy

The problem mentioned above in the use of the simple rule compels us to search for other strategies. We know that it is not difficult to give an ad hoc proof of correctness of implementations such as the example in section 2.3, therefore there must be a way of taking advantage of the structure of the specifications involved and represent specifications not by plain first order theories but by use of a more powerful formalism.

In this section a few simplifications on the use of constructors, some graphic notation and an example of a refinement proof are given, concluding with the informal presentation of a two-step strategy.

4.1 Constructors versus SBO's

Although every SBO can be viewed as a primitive operation of a specification language, and therefore constructor implementations are refinements where the implementation branch is of the form $\kappa(SP_2)$, we think that stronger implementation notions must be balanced with a simplification in the specification language primitives.

In this sense, SBO's such as R, Q and F can be relegated to be used as constructors in implementations and avoided in specifications.

The consequences are important in the proof techniques to be considered; namely, there are two simplifications:

1. D and M have no M-complete and complete inference rules respectively. The way to handle these in our strategy, particularly M, is to give commutativity rules w.r.t. the other SBO's so that we are able to transform the specification branch to a more convenient form. The more we restrict the set of SBO's used in the specification the easier it should be to get commutativity rules, especially considering the difficulties in finding them for Q, R and F.

2. The constructors Q, R and F are meant to appear only at the 'top' of the implementation branch, thus only sound proof rules are required for them.

4.2 Graphic representation

In order to show the structure of specifications we shall define a graphic notation. The graphs considered are DAG's (directed acyclic graphs) with two different kinds of oriented edges, optional labels on nodes and edges, and a key mark $\rightsquigarrow$ in each graph. Nodes denote specifications and the sign $\rightsquigarrow$ a relation of refinement between two nodes; therefore, a graph denotes an implementation. The function $Spec$ defined in the following yields the denoted value of node and $Spec_A$ is an interminade function which yields the value of a node regardless of its annotations.

- A node denotes a specification. Its signature is given next to it by an underlined name, and its class of models is defined by the arrows and edges leading to it from other nodes and the annotations (see below) attached to it. If a node is not the target of any arrow or edge then it denotes, apart from possible annotations, the class of all the algebras over its signature i.e. $Spec_A[node] = \epsilon_\Sigma$.

- An edge denotes a T operation whose argument is the specification denoted by the lower (source) node and whose result is the specification denoted by the upper (target) node. The confluence of several edges denotes a U. A node reached by one or more edges, apart from possible annotations, is defined as $Spec_A[target] = U_{i \in I}(T_{\sigma_i} Spec[source_i])$. Morphisms σ_i are denoted by labels next to the respective edges unless they are inclusions which are omitted.

- An arrow denotes a D operation whose argument is the specification denoted by the source node and whose result is the specification denoted by the target node. A node cannot be the target of more than one arrow nor of one arrow and one or more edges. The target node is defined, apart from annotations, as $Spec_A[target] = D_\sigma Spec[source]$. Morphisms are denoted as in the edges.

- Annotations are labels attached to the nodes. These are of two kinds: sets of axioms and reachability requirements. Therefore, a node with annotations such as $Ax: \Phi_1, ..., \Phi_n$; $Gen: s_1, ...s_m$ denotes
$Spec[node] = A_{\Phi_1}...A_{\Phi_n}M_{s_1}...M_{s_m}Spec_A[node]$ where $Spec_A$ is the denoted value of the node apart from annotations defined above.

- $\rightsquigarrow$ is a mark placed from the node denoting the original specification to the node denoting the proposed implementation.

The recursive definition of $Spec_A$ and $Spec$ is well-defined provided that there are no cycles in the graph.

The choice of an edge for T and an arrow for D is based on the *reversibility* of T which allows the original class of models to be reconstructed from the resulting one unless it uses a non-injective morphism, whereas once we have applied a D we cannot guess what the original class of models looked like. The treatment of A and M as annotations is suggested by their 'selector' behaviour which does not contribute to the structure of the specification.

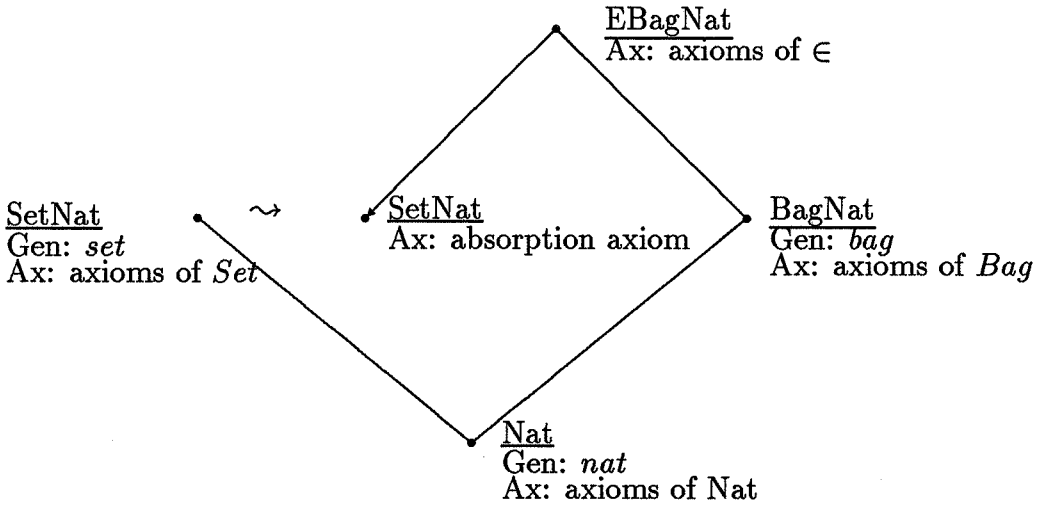
The graph is usually drawn with the edges going upwards and the arrows downwards, so that in the case (very likely) of injective signature morphisms, lower specifications correspond to smaller signatures.

With this graphic notation it is easy to represent any specification term involving A, T, U, D and M. On the other hand the constructors R, Q and F need some extra notation; we pick a dashed arrow labeled with R, Q or F respectively.

In the case of equational logic R is M, $Q_{eq}SP$ when there are no free extensions in SP then amounts to annotating SP with eq, and F consists of a T plus M for the new sorts plus a general principle of 'no-confusion' for the new terms. Hence, in the example we do not need any of these dashed arrows.

Example :

The example presented in section 2.3 is represented by the following graph.



Note that the basic specifications Set and Bag have been simplified from $(T_\iota Nat)UA_\Phi$ to $A_\Phi T_\iota Nat$, and the new annotation for 'no confusion' and reachability corresponding to the free extension ignored since F, in this particular case[2], does not introduce new values for any old sort or any new sort at all.

4.3 Aim: Sharing, Moving and Matching

If we look at the example, it seems clear that the proof should work as follows: forget about the annotations on Nat because it is shared, show that M_{set} is satisfied in the implementation branch by 'moving' M_{bag} to the end of the implementation branch and 'matching' M_{set}, and show that the axioms of Set are satisfied by 'moving' axioms and theorems to the end of the implementation branch and 'matching' them with the axioms of Set.

Our aim is to formalize this sort of proof and give some rules for 'moving' M, 'moving' theorems and some criterion for recognizing when shared subspecifications imply shared submodels. Compared with the simple strategy this is a two-step strategy, in the sense that 'moving' theorems is the new name for inference rules as we had before, but now we have a previous 'step' in which we resolve questions of shared submodels and reachability constraints without relying on theories.

5 Two-step strategy

In this section a new strategy is proposed. First the two-step strategy is presented, then the inclusion of the simple strategy in the new one is shown and finally some auxiliary properties are listed and proven.

[2]It is undecidable to check this in general.

5.1 Strategy

In general, when we try to prove the correctness of a constructor implementation syntactically or, equivalently, a refinement between structured specifications, we proceed by transforming the specifications until they are in a form where refinement is provable (basic case). Then provided the applied transformation rules preserve refinement the proof is finished. Hence, we propose backward proofs of refinement.

First we present the basic cases of our strategy and then some transformation rules. The particular rules chosen among all the possible specification transformations are those embedding the informal ideas exposed in section 4.3 but we try, at the same time, to give a broader motivation.

Basic case

From the definition of refinement there are some SBO's which can be interpreted as immediate refinements, namely, the selectors; i.e. for any selector L and specification SP, $SP \rightsquigarrow LSP$. This can be generalized in the following way:

Definition 5.1 *A* **correct partial ordering between selectors** $\leq$, *is a partial ordering between selectors such that:*

$$\forall L_1, L_2 : selectors;\ SP : specification.\ \ L_1 \leq L_2 \Rightarrow L_1 SP \rightsquigarrow L_2 SP$$

And now we may define an ordering for the three different kinds of selectors used in our specification language.

Fact 5.2 *The ordering* $\leq$ *between selectors defined by*

- *for any two collection of sentences* Φ_1 *and* Φ_2, *if* $\Phi_2 \models \Phi_1$ *then* $A_{\Phi_1} \leq A_{\Phi_2}$;
- *for any two sets of sorts, if* $s_2 \supseteq s_1$ *then* $M_{S_1} \leq M_{S_2}$;
- *for any specification SP,* $U\epsilon_\Sigma \leq USP$ *(currying one of the arguments of U);*
- *and no other two selectors are related*

is correct.

The proof is immediate by referring to the definitions of these selectors in section 2.2. We shall call basic rules the ones derived from the ordering according to the previous definition.

Rule 1

$$\forall SP : spec.\ \ A_{\Phi_1} SP \rightsquigarrow A_{\Phi_2} SP \quad (if\ \Phi_2 \models \Phi_1)$$

Rule 2

$$\forall SP : spec.\ \ M_{S_1} SP \rightsquigarrow M_{S_2} SP \quad (if\ s_2 \supseteq s_1)$$

Rule 3

$$\forall SP_1, SP_2 : spec.\ \ SP_1 \rightsquigarrow SP_1\ U\ SP_2$$

We shall see on the example in section 5.5 how these three basic cases correspond to the three distinct steps sketched in section 4.3, namely: proving that a shared specification involves a sharing of submodels, proving that the implementation satisfies the axioms required by the specification, and proving that the reachability requirements are fulfilled in the implementation.

Transformation rules

In general, refinement transformation rules are given in the form:

$$\frac{SP_1 \rightsquigarrow SP_2}{SP'_1 \rightsquigarrow SP'_2}$$

where the transformation is correct if $SP_1 \rightsquigarrow SP_2$ entails $SP'_1 \rightsquigarrow SP'_2$, or equivalently if $SP'_1 \rightsquigarrow SP'_2$ can be proven correct by proving $SP_1 \rightsquigarrow SP_2$ correct. We can say this is the presentation of an inference system $\vdash_{\rightsquigarrow}$ for deducing refinement relations where the axioms are called basic rules (rules 1-3) and the inference rules are called transformation rules (4-13); however, we do not use the standard terminology in order to avoid confusions with the inference rules for $\vdash$ given in section 3.

All the rules to be presented but one transform only one of the specifications, either the specification branch SP'_1 or the implementation branch SP'_2.

Definition 5.3 *A* **strong transformation**, *written* $SP'_1 \Rightarrow SP_1$, *is a specification transformation such that:*

$$\forall SP_2 : spec.\ \ \frac{SP_1 \rightsquigarrow SP_2}{SP'_1 \rightsquigarrow SP_2}$$

is a correct refinement transformation.

A **weak transformation**, *written* $SP'_2 \Rightarrow SP_2$, *is a specification transformation such that:*

$$\forall SP_1 : spec.\ \ \frac{SP_1 \rightsquigarrow SP_2}{SP_1 \rightsquigarrow SP'_2}$$

is a correct refinement transformation.

It must be noted that these definitions are mere notation since we can define equivalently $SP' \Rightarrow SP$ to be a strong transformation iff $SP' \rightsquigarrow SP$, and a weak transformation iff $SP \rightsquigarrow SP'$. A particular case are those rules preserving the class of models. These are strong and weak at the same time and are called equivalent transformations.

In order to provide a criterion for chosing some schematic refinements to be transformation rules, we must have in mind which are the basic cases to which we want to reduce a proposed problem. In this section we present the skeleton of our proof strategy by showing first a simplification rule which takes into account the possible common top structure of specification and implementation, followed by a collection of strong and weak transformations which precede and prepare the application of the basic rules in the strategy.

Rule 4 (Simplification) *For any SBO* ξ,

$$\frac{SP_1 \rightsquigarrow SP_2}{\xi SP_1 \rightsquigarrow \xi SP_2}$$

Proof Immediate since all the SBO's defined are monotonic.

Strong transformations aim to transform the specification branch in such a way that the basic rules are applicable, therefore they can be classified according to the basic rule they work for into three kinds:

Rule 5 (Pulling axioms)

$$\forall SP : spec.\ \ \xi A_{\Phi_1} SP \Rightarrow A_{\Phi_2} \xi SP$$

Rule 6 (Pulling reachability constraints)

$$\forall SP : spec.\ \ \xi M_{S_1} SP \Rightarrow M_{S_2} \xi SP$$

Rule 7 (Pulling subspecifications)

$$\forall SP, C : spec.\ \ \xi(SP\ U\ T_\iota C) \Rightarrow \xi SP\ U\ T_{\iota'} C$$

Similarly the weak transformations aim to transform the implementation branch in such a way that the basic rules are applicable. As the strong transformations they can be classified according to the basic rules they work for:

Rule 8 (Inheriting axioms)

$$\forall SP : spec.\ \ \xi A_{\Phi_1} SP \Rightarrow A_{\Phi_2} \xi A_{\Phi_1} SP$$

Rule 9 (Inheriting reachability constraints)

$$\forall SP : spec.\ \ \xi M_{S_1} SP \Rightarrow M_{S_2} \xi M_{S_1} SP$$

Rule 10 (Inheriting subspecifications)

$$\forall SP : spec.\ \ \xi(SP\ U\ T_\iota C) \Rightarrow \xi(SP\ U\ T_\iota C)\ U\ T_{\iota'} C$$

It must be noted that rules 5-10 are schematic and that they will be completed for the differet values of ξ in section 5.2 and 5.4.

The previous rules already sketch the strategy, but there are three equivalent transformations which although auxiliary are necessary for the complete definition of the strategy.

Rule 11 (Commutativity of selectors)

$$\forall L_1, L_2 : selectors;\ S : spec.\ \ L_1 L_2 S \Leftrightarrow L_2 L_1 S$$

Rule 12 (Merging axioms)

$$\forall \Phi_1, \Phi_2 : sets\ of\ axioms;\ SP : spec.\ \ A_{\Phi_1} A_{\Phi_2} SP \Leftrightarrow A_{\Phi_1 \cup \Phi_2} SP$$

Rule 13 (Collapse) *For all specifications E built up from T, U, ϵ and D (injective case), then*

$$E \Leftrightarrow \epsilon_{Sig[E]}$$

The first one can be shown correct by the pointwise behaviour of the selectors, the second one by the definition of A_Φ, and the third one inductively by the effect of each operation on ϵ.

Strategy

The so-called two-step strategy works as follows:

1. Discard the shared superstructure with the simplification rule.
2. Apply the strong transformations to the specification branch and eventually rule 13 until it is reduced to a sequence of selectors.
3. Apply the weak transformations to the implementation branch until we have inherited enough selectors.
4. Apply some auxiliary rules such as commutativity and merging of selector, and apply the basic rules.

The strategy contains some degree of freedom in chosing how much the simplification rule is applied (step 1), which subspecifications must be considered such and not try to transform them (step 2), which properties must try to inherite and from where (step 3), and finally in using the underlying theorem prover needed for rule 1.

The strategy is clearly incomplete for two reasons: firstly, the conditions for the applicability of the transformation rules can lead us to a cul-de-sac at steps 2 and 3; and in the second place, since transformations are not in general equivalent, it can happen that the transformed implementation is not a refinement of the transformed specification for an originally correct refinement. Nevertheless, the undecidability of specification equivalence disallows beforehand a complete effective strategy.

5.2 Strong transformations

At this point, we could start giving a rule for each SBO and each kind of proposed weak and strong transformation (rules from 5 to 10), but we rather prefer to give the strong transformations (and the weak transformations later) as consequences of some *elementary* properties of the specifications or the SBO's themselves. This approach raises the need for an inference system for these properties (see section 5.4) which in turn will generate new properties needed in side conditions.

- strong transformations which pull up axioms.

  These transformation rules can be derived from M-complete inference rules (see section 3.3) in the following way:

  Theorem 5.4 *For any M-complete inference rule* $SP \vdash f(\varphi) \Rightarrow \xi SP \vdash g(\varphi)$, *provided* ξ *distributes w.r.t.* U[3], *then*

  $$\xi A_{f(\varphi)} SP_1 \Rightarrow A_{g(\varphi)} \xi SP_1$$

  is a strong transformation.

  Proof By the definition of A and U, and the distributivity of ξ we know that:

  $$\xi A_{f(\varphi)} SP_1 = \xi(SP_1 \; U \; A_{f(\varphi)}\epsilon) = (\xi SP_1) U (\xi A_{f(\varphi)}\epsilon)$$

  The theory $Cl(f(\varphi))$ is M-complete (exact) w.r.t. $A_{f(\varphi)}\epsilon$, then by definition of an M-complete inference rule, $Cl(g(\varphi))$ is an M-complete theory for $\xi A_{f(\varphi)}\epsilon$; i.e. $Mod[A_{g(\varphi)}\epsilon] \subseteq Mod[\xi A_{f(\varphi)}\epsilon]$. Hence, by monotonicity of U,

  $$\xi A_{f(\varphi)} SP_1 = (\xi SP_1) U (\xi A_{f(\varphi)}\epsilon) \rightsquigarrow (\xi SP_1) U (A_{g(\varphi)}\epsilon) = A_{g(\varphi)} \xi SP_1$$

  Fact 5.5 T, *selectors and injective constructors distribute w.r.t.* U.

- strong transformations which pull up reachability constraints.

  These transformation rules can be derived from commutativity rules for M (section 5.4) in the following way:

  Theorem 5.6 *For any commutativity rule of reachability requirements w.r.t. a SBO* ξ *and a specification* SP, $\xi M_{f(s)} SP = M_{g(s)} \xi SP$, *then*

  $$\xi M_{f(s)} SP \Rightarrow M_{g(s)} \xi SP$$

  is a strong transformation.

[3]This condition is not necessary when SP_1 is axiomatizable.

Proof Trivial.

- strong transformations which pull up shared subspecifications.

  These transformations rules can be derived from preserving-submodels rules (see section 5.4).

Fact 5.7 *All the SBO's considered and, in general, any selector, constructor or anticonstructor distribute w.r.t. the union of classes of models.*

Theorem 5.8 *For any preserving-submodels rule for a SBO ξ w.r.t. a signature morphism ι,*

$$\forall SP : spec.\ D_{\iota'}(\xi SP) \subseteq D_{\iota} SP$$

where ι' is the suitable morphism by which both terms have the same signature, then

$$\forall SP_1, C : spec.\ \ \xi(SP_1\ U\ T_{\iota}C) \ \Rightarrow\ \xi SP_1\ U\ T_{\iota'}C$$

is a strong transformation.

Proof We factorize SP_1 into two parts, one agreeing with C and the rest

$$Mod[SP_1] = Mod[SP_1\ U\ T_{\iota}C] \cup Mod[SP_1\ U\ T_{\iota}\overline{C}]$$

where $\overline{C}$ does not need to be a real specification (made of SBO's) but it has $Sig[\overline{C}] = Sig[C]$ and $Mod[\overline{C}] = Alg(Sig[C]) \setminus Mod[C]$. Applying ξ to both sides and by distributivity we obtain

$$Mod[\xi SP_1] = Mod[\xi(SP_1\ U\ T_{\iota}C)] \cup Mod[\xi(SP_1\ U\ T_{\iota}\overline{C})]$$

selecting in both sides those models with a submodel in C and by distributivity we obtain

$$Mod[\xi SP_1\ U\ T_{\iota'}C] = Mod[\xi(SP_1\ U\ T_{\iota}C)\ U\ T_{\iota'}C] \cup Mod[\xi(SP_1\ U\ T_{\iota}\overline{C})\ U\ T_{\iota'}C]$$

Now, we apply the preserving-submodels rule to say that: $D_{\iota'}(\xi(SP_1\ U\ T_{\iota}C)) \subseteq D_{\iota}(SP_1\ U\ T_{\iota}C) \subseteq C$ and that $D_{\iota'}(\xi(SP_1\ U\ T_{\iota}\overline{C})) \subseteq \overline{C}$, from which it follows that in the first term the union with $T_{\iota'}C$ is redundant and that the second term is empty.

$$Mod[\xi SP_1\ U\ T_{\iota'}C] = Mod[\xi(SP_1\ U\ T_{\iota}C)]$$

hence the transformation proposed is a refinement.

Weak transformations

We repeat what has just been done for the strong transformations in the last section for the weak transformations.

- weak transformations which inherit axioms.

  These transformation rules are derived from the sound inference rules (see section 3.3) in the following way:

  Theorem 5.9 *For any sound inference rule* $SP \vdash f(\varphi) \Rightarrow \xi SP \vdash g(\varphi)$, *then*

  $$\forall SP_1 : spec.\ \ \xi A_{f(\varphi)} SP_1 \ \Rightarrow\ A_{g(\varphi)} \xi A_{f(\varphi)} SP_1$$

  is a weak transformation.

  Proof By definition of A we know that $A_{f(\varphi)} SP_1 \models f(\varphi)$ and by definition of soundness we conclude that $\xi A_{f(\varphi)} SP_1 \models g(\varphi)$. And by definition of A again,

  $$A_{g(\varphi)} \xi A_{f(\varphi)} SP_1 = \xi A_{f(\varphi)} SP_1.$$

- weak transformations which inherit reachability constraints.

  These transformation rules can be derived from inheritance rules for M (see section 5.4) in the following way:

  Theorem 5.10 *For any inheritance rule of reachability requirements w.r.t. a SBO* ξ; $\xi M_{f(s)} SP = M_{g(s)} \xi M_{f(s)} SP$, *then*

  $$\xi M_{f(s)} SP \ \Rightarrow\ M_{g(s)} \xi M_{f(s)} SP$$

  is a weak transformation.

  Proof Trivial.

- weak transformations which inherit shared subspecifications.

  These transformations rules can be derived from conserving-submodels rules.

  Theorem 5.11 *For any conserving-submodels rule for SBO* ξ *w.r.t. a signature morphism* ι *and subspecification* C,

  $$D_{\iota'} \xi T_\iota C \subseteq C$$

  where ι' *is the suitable morphism by which both terms have the same signature, then*

  $$\forall SP : spec.\ \ \xi(SP\ U\ T_\iota C) \ \Rightarrow\ \xi(SP\ U\ T_\iota C)\ U\ T_{\iota'} C$$

  is a weak transformation.

Proof Since ξ is monotonic,

$$\xi(SP\ U\ T_\iota C) \subseteq \xi T_\iota C$$

and applying $T_{\iota'}$ to the rule and simplifying we obtain that $\xi T_\iota C \subseteq T_{\iota'} C$, then

$$\xi(SP\ U\ T_\iota C) = \xi(SP\ U\ T_\iota C)\ U\ \xi T_\iota C \subseteq \xi(SP\ U\ T_\iota C)\ U\ T_{\iota'} C$$

5.3 Simple strategy revisited

In this section we relate the two-step strategy to the simple strategy (section 3), the informal two-step motivation (section 4.3) and the graphic notation (section 4.2).

Theorem 5.12 *The simple strategy is the two-step strategy when considering only a subset of the rules; namely, the basic rule concerning A_Φ (rule 1), the auxiliary rules (rules 12-13) and those strong and weak transformations handling the axioms (rules 5 and 8).*

Proof sketch For any sequence of strong transformations $SP_1 \Rightarrow ... \Rightarrow A_{\varphi_1}...A_{\varphi_n}E$ where E stands for a specification as defined in rule 13, another sequence of theories going in the opposite direction can be built. Starting from $\overline{\Delta_0} = Cl(\{\varphi_1, ..., \varphi_n\})$ which is an M-complete theory for $A_{\varphi_1}...A_{\varphi_n}E$, and applying successively those M-complete inference rules from which the strong transformations used were derived, we obtain $\overline{\Delta_0}, ..., \overline{\Delta_{SP_1}}$ where each theory is an M-complete theory w.r.t. the corresponding specification in the first sequence. We conclude that $\overline{\Delta_{SP_1}}$ is an M-complete theory w.r.t. SP_1. Working similarly in the implementation branch with sound inference rules, weak transformations and sound theories, we can construct a sound theory $\underline{\Delta_{SP_2}}$ w.r.t. SP_2. The inference $\underline{\Delta_{SP_2}} \vdash \overline{\Delta_{SP_1}}$ is performed by the basic rule (rule 1) after merging the axioms (rule 12) and collapsing E (rule 13). The construction can be built either from specification transformations to theories or from theories to specification transformations, concluding that the considered subset of the two-step strategy is exactly the simple strategy.

After the informal reasoning in section 4.3 we called the new strategy a *two-step* strategy. Now, this can be justified by using, in a proof of refinement, first all the rules excluded in the simple strategy, and then exclusively those in the simple strategy. Because of the commutativity of the selectors and the independence of the rules handling axioms from the rules handling reachability restrictions or shared submodels, we can perform proofs in two steps without any loss of generality. In fact, a stronger result can be expected:

Conjecture 5.13 *Consider a sequence of refinements* $SP_1 \rightsquigarrow SP_2, ..., \epsilon_{Sig[SP_3]} \rightsquigarrow SP_3$, *where each refinement is obtained from the previous by one of the transformation rules; i.e. a proof of refinement. Another sequence can be built where the transformations handling shared submodels are applied first, followed by the transformations handling reachability constraints and finally applying those handling axioms (the simple strategy).*

Finally, the transformation rules can be seen as modifications to the graph representing a presumed refinement.

The basic case corresponds to a graph where the specification branch has been reduced to the root node with some annotations and a subgraph hanging from it which happens to be hanging also from the root node of the implementation branch (sharing of submodels). The implementation branch can be of any shape. In this case correctness can be immediately proven if the annotations at the root of the specification are included in the ones at the root of the implementation.

Strong transformations affect the specification branch, each of them pulling up the constraints of those subspecifications not shared. In the end, rule 13 prunes the meaningless structure and leaves the specification branch ready for the application of the basic rules.

Weak transformations affect the implementation branch by adding some redundant annotations and copies of subspecifications. These rules provide the way to show up on the implementation branch those properties required by the already transformed specification branch.

The simplification rule clears the way to those subspecifications where the two branches differ.

5.4 Properties of the SBO's

In the presentation of the strategy, strong and weak transformations are synthesized from M-complete inference rules, sound inference rules, commutativity rules for M, inheritance rules for M, preserving-submodels rules and conserving-submodels rules. All these rules are considered to state elementary properties of the SBO's and not mere intermediate requirements in refinement proofs. In fact, we could have given strong transformations immediately instead of proving some rules on commutativity of M and that the synthesis of strong transformations from commutativity rules for M is correct, but we consider that the commutativity of M, or any of the other properties, is interesting in its own right and might be useful in other problems apart from proofs of refinement.

First we define some more properties of specifications and morphisms which are used later in side conditions of the rules in this section.

1. A sort s is **new** w.r.t. a signature morphism σ iff $\sigma^{-1}(s) = \emptyset$. And a term is new w.r.t. a signature morphism σ iff it does not include function symbols

which are not in the image of σ.

2. A specification SP is **closed under subalgebras** w.r.t. a set of sorts S
iff $D_\iota R_S T_\iota SP \subseteq SP$
where $\iota : Sig[SP] \rightarrow (Sig[SP] \cup (\{c_s : s, f_s : s \rightarrow s \mid s \in S\}))$

   The extension of the signature provides an arbitrary function f_s for each sort in S which can make reachable any subalgebra of an algebra in $Mod[SP]$ where the sorts not in S are preserved. Therefore, if we take the reachable part for every possible interpretation of f_s we obtain all the subalgebras[4] of all the algebras in $Mod[SP]$. The construction does not work for algebras with empty sorts.

   In case we use equational logic the models of basic specifications (varieties) are closed under subalgebras but there are SBO's such as D_σ that do not preserve this property.

3. A signature morphism σ **forgets a set of equations** eq
iff $\forall\ SP : spec.\ D_\sigma Q_{eq} SP = D_\sigma SP$.

   An alternative definition which refers to the congruence induced by eq is:
$\forall \dot{s} \in sorts(\Sigma).\ (\exists e1, e2 \in |S|_s.\ (e1, e2) \in (\sim_{eq} \backslash ident)) \Rightarrow (s\ is\ a\ new\ sort\ w.r.t.\ \sigma)$,
where $(e1, e2)$ denote a pair of different values of the same sort which are identified if we impose the equations eq; i.e. the pair belongs to the minimal congruence generated by eq but not to the identity $((e1, e2) \in (\sim_{eq} \backslash ident))$.

   In order to prove this property we can check that all the sorts mentioned in the equations eq are forgotten in σ plus those sorts which although not directly mentioned depend on the ones mentioned (see [BKM 87] for a definition of dependency).

4. A specification is **closed under quotients**
iff $\forall eq : set\ of\ equations.\ Q_{eq} SP \subseteq SP$.

   In case we use equational logic the models of basic specifications (varieties) are closed under quotients but there are SBO's such as F_σ that do not preserve this property.

5. A signature morphism σ_2 **forgets a free extension** F_{σ_1}
iff $\forall SP : spec.\ D_{\sigma_1 \cdot \sigma_2} F_{\sigma_1} SP = D_{\sigma_2} SP$.

   As in the case of forgetting equations, σ_2 must forgets all those sorts altered by F_{σ_1} which, in particular, are those mentioned directly in σ_1 plus the ones depending on them.

[4] In fact only the countable subalgebras are obtained, so we should properly talk about closure under countable subalgebras and indeed this is all that is required in the rules.

6. A free extension $F^{eq}_{\sigma}SP$ is **sufficiently complete** iff $D_{\sigma}F^{eq}_{\sigma}SP = D_{\sigma}Q_{eq}T_{\sigma}SP$.

   This can be redefined referring to the models as $\forall A \in Mod[SP].\ (Free^{eq}_{\sigma}A)|_{\sigma} \cong A/(\sim_{eq}|_{\sigma})$ where the reduct of a congruence is the congruence associated to the reduct of the (surjective) homomorphism representing the original congruence. Intuitively, a free extension is sufficiently complete if it does not add new values to the old sorts, although it can confuse (quotient) some of the old values.

   The standard definition refers exclusively to the reachable models of SP or, equivalently, to its initial model (see [Ber 87] for discussion).

7. A free extension $F^{eq}_{\sigma}SP$ is **consistent**
   iff $\forall A \in Mod[SP].\ A$ *is a subalgebra of* $(Free^{eq}_{\sigma}A)|_{\sigma}$.

   Intuitively, a free extension is consistent (or hierarchically consistent) if it preserves the old values of the old sorts into the extension, although it may add new ones.

   Similarly to sufficient completeness, the standard definition refers exclusively to the reachable models of SP or, equivalently, to its initial model (see [Ber 87] for discussion).

8. A signature morphism σ does not add **new operations** for a sort s
   iff $\sigma(Lang(s, SP)) = Lang(\sigma(s), T_{\sigma}SP)$
   where $Lang(s, SP)$ is defined recursively as the collection of operations in $Sig[SP]$ which produce elements of sort s or of any other sort on which s depends (see [KM 87] for definition).

9. Two sets of sorts S_1 and S_2 are **junk-independent** in a specification SP
   iff $R_{S_1}R_{S_2}SP = R_{(S_2 \cup S_1)}SP = R_{S_2}R_{S_1}SP$.

   It is sufficient to show that neither S_1 nor S_2 depend on the other.

10. A sentence φ **has no negation** w.r.t. a specification SP, if either φ (assumed in prenex form) does not contain any inequality or each inequality $A \neq B$ it contains is trivial (its truth-value does not affect the truth-value of φ) or SP makes it false, $SP \models A = B$.

11. A sentence φ **has no existentials** w.r.t. a specification SP, if either φ (assumed in prenex form) does not contain any $\exists$ or each subformula $\exists x : s.F(x)$ it contains is trivial (its truth-value does not affect the truth-value of φ) or SP makes it false, $SP \models \nexists x : s.F(x)$.

M-complete & sound inference rules

As mentioned in section 3.3, there are some inference rules associated to SBO's. Most of them were presented in [ST 85] but we go through all of them since they

need still to be proven sound and sometimes M-complete according to the new definitions given in 3.3.

- $SP_i \vdash \varphi \Rightarrow U_i SP_i \vdash \varphi$

  Fact 5.14 *This is a sound and M-complete inference rule.*

  Proof Let $\{\Delta_i\}_{i \in I}$ be a family of M-complete theories w.r.t. $\{SP_i\}_{i \in I}$, i.e. $Mod[\Delta_i] \subseteq Mod[SP_i]$ for every $i \in I$. The inferred theory is $\Delta_{inf} = Cl(\cup_i \Delta_i)$. Then, by monotonicity of the intersection, $\bigcap_i Mod[\Delta_i] \subseteq \bigcap_i Mod[SP_i]$, and by the definition of the models of a theory $\bigcap_i Mod[\Delta_i] = Mod[\cup_i \Delta_i]$. We conclude that Δ_{inf} is an M-complete theory for $U_i SP_i$.

  Analogously, and considering the inclusion in the opposite way the inference rule can be proved M-sound and therefore sound.

- $SP \vdash \varphi \Rightarrow T_\sigma SP \vdash \sigma(\varphi)$

  Fact 5.15 *This is a sound and M-complete inference rule.*

  Proof Given an M-complete theory Δ for SP and considering $\sigma : \Sigma \to \Sigma'$, the inferred theory has models $Mod[\Delta_{inf}] = \{A \in |Alg(\Sigma')| \mid A \models \sigma(\Delta)\}$ which by the satisfaction condition are $\{A \in |Alg(\Sigma')| \mid A|_\sigma \models \Delta\}$. Finally, by definition of the models of a theory, the same class of models can be expressed as $\{A \in |Alg(\Sigma')| \mid A|_\sigma \in Mod[\Delta]\}$ which, by M-completeness of Δ, is included in $\{A \in |Alg(\Sigma')| \; |A|_\sigma \in Mod[SP]\} = Mod[T_\sigma SP]$. We conclude that Δ_{inf} is an M-complete theory for $T_\sigma SP$.

  Analogously, and considering the inclusion in the opposite way the inference rule can be proved M-sound and therefore sound.

- $SP \vdash \sigma(\varphi) \Rightarrow D_\sigma SP \vdash \varphi$

  Fact 5.16 *This is a sound and complete inference rule.*

  Proof Let be Δ a sound theory w.r.t. SP. The inferred theory is $\Delta_{inf} = Cl(\sigma^{-1}\Delta)$ where $\sigma^{-1}\Delta = \{\varphi_1 \; is \; a \; \Sigma - sentence \mid \exists \varphi_2 \in \Delta.\; \varphi_2 = \sigma(\varphi_1)\}$.
  Then,
  $Mod[SP] \subseteq Mod[\Delta] \subseteq Mod[\sigma(\sigma^{-1}\Delta)] = \{A \in |Alg(\Sigma)| \; |A \models \sigma(\sigma^{-1}\Delta)\} =$
  $= \{A \in |Alg(\Sigma)| \mid A|_\sigma \models \sigma^{-1}\Delta\}$
  applying the reduct functor to both sides

  $$Mod[D_\sigma SP] \subseteq \{A \in |Alg(\Sigma)| \mid A|_\sigma \models \sigma^{-1}\Delta\}|_\sigma = Mod[\sigma^{-1}\Delta]$$

therefore the rule is sound.

On the other hand, let Δ be a complete theory w.r.t. SP. Assume that Δ_{inf} is not complete w.r.t. $D_\sigma SP$, then there exists a sentence $\varphi \in Th[D_\sigma SP]$ such that $\varphi \notin \Delta_{inf}$. But, by definition of Th and the satisfaction condition $\sigma(\varphi) \in Th[SP]$, therefore $\sigma(\varphi) \in \Delta$ and $\varphi \in \Delta_{inf}$. Hence, the rule is complete.

It must be noted that there exist classes of algebras which can be defined using A and D but which are not axiomatizable in first order logic (see [BHK 86] for examples and discussion), therefore no M-complete and sound rule can be expected in general. However, there would exist one for some kinds of institutions.

Fact 5.17 *For an injective σ and in an institution where the conjecture below holds this is an M-complete inference rule.*

Conjecture 5.18 *For an injective σ, theory Δ and specification SP,*

$$Mod[\Delta] \subseteq Mod[T_\sigma SP] \Rightarrow Mod[\sigma^{-1}\Delta] \subseteq Mod[SP].$$

Proof Let be Δ a M-complete theory w.r.t. SP. By definition of D and T,

$$Mod[\Delta] \subseteq Mod[SP] \;=\; Mod[T_\sigma D_\sigma SP]$$

applying the conjecture

$$Mod[\sigma^{-1}\Delta] \subseteq Mod[D_\sigma SP].$$

- $SP \vdash \varphi \Rightarrow A_\Phi SP \vdash \varphi$ *and* $A_\Phi SP \vdash \varphi$ (if $\Phi \vdash \varphi$)

Fact 5.19 *This is a sound and M-complete inference rule.*

Proof For every M-complete Δ for SP, the inferred theory $\Delta_{inf} = Cl(\Delta \cup \Phi)$ has models $Mod[\Delta_{inf}] = \{A \in Mod[\Delta] |\!|\ A \models \Phi\}$. Considering the definition of A_Φ and the M-completeness of Δ it follows that $Mod[\Delta_{inf}] \subseteq \{A \in Mod[SP] |\!|\ A \models \Phi\} = Mod[A_\Phi SP]$.

Analogously, and considering the inclusion in the opposite way the inference rule can be proved M-sound and therefore sound.

- $\epsilon_\Sigma \vdash$ *Tautologies on* Σ

Fact 5.20 *The set of tautologies on Σ form an exact theory w.r.t. ϵ_Σ.*

Proof Trivial.

Sound inference rules

- $SP \vdash \varphi \Rightarrow M_\sigma SP \vdash \varphi$

  Fact 5.21 *This is a sound inference rule.*

  Proof For all sound theories Δ for SP, since M is a selector, $Mod[M_S SP] \subseteq Mod[SP] \subseteq Mod[\Delta]$.

- $eq \vdash \varphi \Rightarrow Q_{eq} SP \vdash \varphi$

  Fact 5.22 *This is a sound inference rule.*

  Proof By soundness of equational logic, $eq \vdash t_1(X) = t_2(X) \Rightarrow \forall A \in Mod[eq].\ \forall \nu : X \to |A|.\ \nu^\#(t_1(X)) = \nu^\#(t_2(X))$ and by construction of the minimal congruence for the relation of equivalence $\equiv_{eq}$ and algebra A, $(\nu^\#(t_1(X)), \nu^\#(t_2(X))) \in \sim_{eq}$. Therefore, in the quotient algebra, $A/\sim_{eq}$, for all valuations $\nu_\sim : X \to |A/\sim_{eq}|$, both $\nu^\#_\sim(t_1(X))$ and $\nu^\#_\sim(t_2(X))$ yield the same value (partition class); hence, $t_1(X) = t_2(X)$ holds in $A/\sim_{eq}$ and, in general, for every model of $Q_{eq}SP$.

- $(SP \vdash \varphi)$ and $(\varphi$ *has no negation w.r.t.* $SP) \Rightarrow Q_{eq} SP \vdash \varphi$

  Fact 5.23 *This is a sound inference rule.*

  Proof It is enough to show that for all algebras s.t. $A \models \varphi$, where φ is a sentence built up exclusively from the equality predicate and connectives: $\wedge, \vee, \forall$ and $\exists$, it holds that $\forall \sim$: *congruence on* A. $(A/\sim) \models \varphi$. This can be proved by induction on the structure of φ:

  $A \models t_1 = t_2$. For all algebras $\nu_A(t_1) = \nu_A(t_2)$, by definition of quotient, $\nu_{A/\sim}(t_1) = [\nu_A(t_1)]$ and consequently $\nu_{A/\sim}(t_1) = \nu_{A/\sim}(t_2)$ and then $(A/\sim) \models t_1 = t_2$.

  $A \models \exists x : s.\ F(x)$. By definition of satisfaction of $\exists$, there exists a $a \in |A|_s$ such that taking a valuation $\nu : x \mapsto a$ we obtain $\nu^\#(F(x)) = true$. By the induction hypothesis and substituting x by a constant a_c of value a in A, $A \models F(a_c) \Rightarrow (A/\sim) \models F([a_c])$ which implies $(A/\sim) \models \exists x : s.\ F(x)$.

  Similarly we can write more cases for the other connectives.

- $(SP \vdash \varphi)$ and $(\varphi$ *has no existentials w.r.t.* $SP) \Rightarrow R_S SP \vdash \varphi$

  Fact 5.24 *This is a sound inference rule.*

Proof It is enough to show that for all algebras $A \models \varphi$, where φ is a sentence whose prenex form does not contain existential quantifiers, it holds that $\forall A' : \textit{subalgebra of } A.\ A' \models \varphi$. This can proved by induction on the structure of φ (assumed in prenex form):

$A \models t_1 = t_2$ where t_1, t_2 are ground terms. By definition of subalgebra, for all $f : s_1, ..., s_n \to s$ in Σ and $b_1 \in |A'|_1, ..., b_n \in |A'|_n$, $f_A(b_1, ..., b_n) = f_{A'}(b_1, ..., b_n)$; therefore, constants have the same value in A and A', and so do the ground terms generated from them, hence $A' \models t_1 = t_2$.

$A \models t_1 \neq t_2$. As in the previous case ground terms preserve the same values in subalgebras, so A and A' agree either in equalities or inequalities of ground terms.

$A \models \forall x : s.\ F(x)$. By definition of satisfaction, $\forall a \in |A|_s.\ A \models_{\nu_a} F(x)$ where $\nu_a : x \mapsto a$. By definition of subalgebra $|A'|_s \subseteq |A|_s$ then $\forall a \in |A'|_s.\ A \models_{\nu} F(x)$ which by induction hypothesis yields $\forall a \in |A'|_s.\ A' \models_{\nu} F(x)$, hence $A' \models \forall x : s.\ F(x)$.

Similarly we can write more cases for the other connectives.

- $eq \vdash \varphi \Rightarrow F_\sigma^{eq} SP \vdash \varphi$

Fact 5.25 *This is a sound inference rule.*

Proof By definition, for $\sigma : \Sigma \to \Sigma'$, $F_\sigma^{eq} : Alg(\Sigma) \to Mod[\langle \Sigma', eq \rangle]$. Then, it holds that $Mod[F_\sigma^{eq} SP] \subseteq Mod[\langle \Sigma', eq \rangle]$ and since $Mod[\langle \Sigma', eq \rangle] = Mod[Cl(eq)]$ it follows that all semantic consequences of eq hold in F_σ^{eq}. By soundness of equational logic the rule is sound.

Note The same proof can be obtained by proving a transformation rule $F_\sigma^{eq} SP = Q_{eq} F_\sigma SP$ and then applying the appropriate rule already given for Q.

- $(SP \vdash \varphi)$ and $(F_\sigma^{eq} SP$ *sufficiently complete and consistent*$) \Rightarrow F_\sigma^{eq} SP \vdash \sigma(\varphi)$

Fact 5.26 *This is a sound inference rule.*

Proof Sufficient completeness and consistency mean that $(Free_\sigma^{eq})|_\sigma$ is isomorphic to a quotient of A and to subalgebra of A at the same time, therefore by the homomorphism theorem we conclude that

$$\forall A \in Mod[SP].\ (Free_\sigma^{eq} A)|_\sigma \cong A$$

and by the satisfaction condition $(Free_\sigma^{eq} A)|_\sigma \models \varphi \Leftrightarrow Free_\sigma^{eq} A \models \sigma(\varphi)$.

- (t is a new sort) and $(A_{eq}SP \not\models a =_t b) \Rightarrow F^{eq}_{\sigma}SP \vdash a \neq_t b$
  (a is a new term) and $(A_{eq}SP \not\models a =_s b) \Rightarrow F^{eq}_{\sigma}SP \vdash a \neq_s b$
  (Note: In fact the first rule is a particular case of the second.)

  Fact 5.27 *This is a sound inference rule.*

  Proof Trivial by considering the construction of a free extension $Free^{eq}_{\sigma}A$ as an initial object [Tar 86].

  Note Although non-provability can be proved in some cases it is particularly interesting to use the technique presented in [B 87] which involves transforming the free extension into another one where such a proof is easy; this is perfectly consistent with our approach.

- $SP \vdash Gx : s.\ F(x) \Rightarrow M_S SP \vdash \forall x : s.\ F(x)$
  where $Gx : s$ universally quantifies over the values of sort s which can be expressed as terms without variables of sort s (ground terms if SP is reachable).

  Fact 5.28 *This is a sound inference rule.*

  Proof Let $\forall x : s.F(x)$ be false in $M_S SP$, then by definition of satisfaction there must be a model of $A \in Mod[M_S SP]$ and a value $a \in |A|_s$ such that $A \not\models_{(x \mapsto a)} F(x)$. However by definition of M_S, A has to be reachable on sort s thus there exists a term t_a whose value in A is a and therefore $Gx : s.\ F(x)$ could not be true for SP.

  Note This rule is essentially infinitary therefore, analogously to what happends to arithmetic, the rule is complete but it can only be implemented using induction techniques leading, in practice, to an imcomplete system.

Preserving-submodels rules

- $\forall L : selector;\ \iota : morphism;\ SP : spec.\ \ D_\iota LS \subseteq D_\iota SP$

  Fact 5.29 *This is a preserving-submodels rule.*

  Proof Trivial by definition of constructor and monotonicity of D.

- $\forall \sigma, \iota : morphisms.\ \ D_{\sigma;\iota} T_\sigma SP \subseteq D_\iota SP$

  Fact 5.30 *This is a preserving-submodels rule.*

Proof If σ is injective then $D_\sigma T_\sigma SP = SP$, hence the rule holds. If σ is not injective then $D_\sigma T_\sigma SP$ contains those models of SP where the carriers corresponding to the sorts identified by σ were identical, hence the rule holds.

- $\sigma \cdot \sigma_1 = \sigma_c \Rightarrow \ \forall SP : spec.\ D_{\sigma_1}(D_\sigma SP) \subseteq D_{\sigma_c} SP$

  Fact 5.31 *This is a preserving-submodels rule.*

  Proof Immediate by composing the two reduct functors $D_{\sigma_1} D_\sigma = D_{\sigma \cdot \sigma_1} = D_{\sigma_c}$.

Conserving-submodels rules

We present rules of this kind only for the constructors since we already have preserving-submodels rules for the other SBO's and they entail automatically the existence of a conserving-submodels rule.

Proposition 5.32 *For any preserving-submodels rule for a SBO ξ w.r.t. a signature morphism ι,*

$$\forall SP : spec.\ D_{\iota'}(\xi SP) \subseteq D_\iota SP$$

where ι' is the suitable morphism by which both terms have the same signature, then for any specification C

$$D_{\iota'} \xi T_\iota C \subseteq C$$

is a conserving-submodels rule for ξ w.r.t. ι and C.

Proof The conserving-submodels rule is the instantiation of the preserving-submodels rule with the specification $T_\iota C$ and simplified by the fact that $D_\iota T_\iota C \subseteq C$.

In the following some conserving-models rules for the constructors are given:

- (s is a new sort) or (C *is closed under subalgebras*) $\Rightarrow D_{\sigma_c} R_S T_{\sigma_c} C \subseteq C$

  Fact 5.33 *This is a conserving-submodels rule.*

  Proof In the case where s is a new sort and since R only changes the carrier of s, the rest remains unchanged so $\forall SP_1 : spec.\ D_{\sigma_c} SP_1 = D_{\sigma_c} R_S SP_1$.

  In the case where C is closed under subalgebras, first we realize that for every algebra $A \in T_{\sigma_c} C$, $(R_S A)|_{\sigma_c}$ is a subalgebra of $A|_{\sigma_c}$ since $R_S A$ is a subalgebra of A and the reduct functor preserves subalgebras. Then, since $A|_{\sigma_c} \in C$ and since C is closed under subalgebras, $(R_S A)|_{\sigma_c} \in C$.

- (σ_c *forgets eq*) or (C *is closed under quotients*) $\Rightarrow D_{\sigma_c} Q_{eq} T_{\sigma_c} C \subseteq C$

  Fact 5.34 *This is a conserving-submodels rule.*

Proof By definition, if σ_c *forgets eq*, then $D_{\sigma_c}(Q_{eq}T_{\sigma_c}C) = D_{\sigma_c}T_{\sigma_c}C \subseteq C$.

In the case where C is closed under quotients, it is enough to show that for all algebras $A \in T_{\sigma_c}C$, $(A/{\sim_{eq}})|_{\sigma_c} = (A|_{\sigma_c})/{\sim_{eq}|_{\sigma_c}}$. And this holds because a quotient is equivalent to the application of a particular homomorphism and by definition functors (and reducts in particular) commute w.r.t. the application of morphisms. Finally, since $A|_{\sigma_c}$ is in C its quotient will also be in C.

- $((\sigma_c$ *forgets extension* $F_\sigma)$ or $(F^{eq}_\sigma T_{\sigma_c}C$ *is sufficiently complete*$))$ and $((\sigma_c$ *forgets eq*$)$ or $(F^{eq}_\sigma T_{\sigma_c}C$ *is consistent*$)$ or $(C$ *is closed under quotients*$))$ $\Rightarrow D_{\sigma\cdot\sigma_c}(F^{eq}_\sigma T_{\sigma_c}C) \subseteq C$

  This rule enumerates some trivial cases where F is persistent and the case where F does not need to be consistent since C is closed under quotients.

Fact 5.35 *This is a conserving-submodels rule.*

Proof For all algebras $A \in T_{\sigma_c}C$, if either F^{eq}_σ is sufficient complete or σ_c forgets the extension F_σ, there exists a congruence $\sim$ such that $(Free^{eq}_\sigma A)|_{\sigma\cdot\sigma_c}$ $\cong (A|_{\sigma_c})/{\sim}$; i.e. it is sufficiently complete w.r.t. the part of the signature we are interested in. In the first case by definition of sufficient completeness and considering the top reduct $|_{\sigma_c}$ we obtain $\sim = (\sim_{eq}|_{\sigma\cdot\sigma_c})$. In the case where σ_c forgets the extension F_σ, we consider the standard decomposicion of $Free^{eq}_\sigma$ into $Free_\sigma$ and a quotient by $\sim_{eq}$; by definition of forgetting a free extension the first step is ignore, then $Free^{eq}_\sigma A \cong A/\sim_{eq}$ and applying the reduct we obtain $\sim = (\sim_{eq}|_{\sigma\cdot\sigma_c})$ again.

If C is closed under quotients, then since $A|_{\sigma_c} \in C$, $(A|_{\sigma_c})/{\sim} \in C$.

If F^{eq}_σ is consistent, by definition A is a subalgebra of $(Free^{eq}_\sigma A)|_\sigma$. Reduct functors preserve subalgebras , thus $A|_{\sigma_c}$ is a subalgebra of $(A|_{\sigma c})/{\sim}$ and this is only possible in the trivial case where $\sim$ is the identity. Therefore, $(A|_{\sigma c})/{\sim} = A|_{\sigma_c}$ which belongs to C.

Finally, if σ_c forgets *eq* then, by construction, $\sim$ is the identity again.

Commutativity rules for M Due to rule 11 we do not need to bother examining commutativity of M w.r.t. A, U and M itself. Therefore only two rules are needed.

- (No new operations for s in σ) $\Rightarrow T_\sigma M_S SP = M_{\sigma(s)}T_\sigma SP$

Fact 5.36 *This is a commutativity rule of reachability requirements*

Proof For all algebras $A \in Mod[M_S SP]$, A is already reachable on s and it will necessarily be so after applying T_σ since no operation is forgotten thus $T_\sigma M_S SP \subseteq M_{\sigma(s)} T_\sigma SP$. Conversely, for all algebras $A \in M_{\sigma(s)} T_\sigma SP$, $A|_\sigma$ should already be reachable on s since $Lang(s, SP)$ is not expanded and without new operations no new elements in $|A|_s$ can be reached; hence $A|_\sigma \in M_S SP$.

- (No new operations it for $\sigma^{-1}s$ in σ) $\Rightarrow D_\sigma M_S SP = M_{\sigma^{-1}(s)} D_\sigma SP$

Fact 5.37 *This is a commutativity rule of reachability requirements*

Proof By definition we know that $\sigma(Lang(s, D_\sigma SP)) = Lang(\sigma(s), T_\sigma D_\sigma SP)$, which may be simplified to $Lang(\sigma(s), SP)$. Then, every reachable algebra $A \in SP$ yields a reachable $A|_\sigma \in D_\sigma SP$ since the operations considered are the same (up to renaming); hence $D_\sigma M_S SP \subseteq M_{\sigma^{-1}(s)} D_\sigma SP$. And the opposite holds trivially since a D_σ never makes reachable those sorts which were not reachable before.

Inheritance rules for M

- $T_\sigma M_S SP = M_{\sigma(s)} T_\sigma M_S SP$

Fact 5.38 *This is an inheritance rule of reachability requirements*

Proof The rule asserts that the models of $T_\sigma M_S SP$ are reachable on $\sigma(s)$. This can be proven by contradiction assuming $A \in T_\sigma M_S SP$ has a proper subalgebra A' only distinct on sort $\sigma(s)$. Since reduct functors preserve subalgebras (proper subalgebras in this case because s is not a forgotten sort), then $A'|_\sigma$ is a proper subalgebra of $A|_\sigma \in M_S SP$ only distinct on sort s, and consequently $A|_\sigma \in M_S SP$ would not be reachable on $\sigma(s)$.

- $Q_{eq} M_S SP = M_S Q_{eq} M_S SP$

Fact 5.39 *This is an inheritance rule of reachability requirements*

Proof The rule asserts that the models of $Q_{eq} M_S SP$ are reachable on s. This can be proven by contradiction assuming $A/\sim_{eq} \in Q_{eq} M_S SP$ has a proper subalgebra A' only distinct on sort s. We can define another algebra A'' identical to A but with carrier $|A''|_s = \{a \in |A|_s \mid [a] \in |A'|_s\}$, which is a proper subalgebra of $A \in M_S SP$ in contradiction with the definition of M_S.

- $R_S SP = M_S R_S SP$

Fact 5.40 *This is an inheritance rule of reachability requirements*

Proof Immediate since $\forall A.\ Reach_S(Reach_S(A)) = Reach_S(A)$ therefore all algebras in $R_S SP$ are preserved by M_S.

- (S_1 and S_2 are junk-independent in SP) $\Rightarrow R_{S_1} M_{S_2} SP = M_{S_2} R_{S_1} M_{S_2} SP$

Fact 5.41 *This is an inheritance rule of reachability requirements*

Proof Assume the rule does not hold, then it exist an algebra $A \in Mod[M_{S_2} SP]$ and a sort $s \in S_2$ such that elements in $|A|_s$ is only reachable through junk in $|A|_{S_1}$. However, if that were the case $Reach_{S_1}(Reach_s A) \neq Reach_{S_1 \bigcup \{s\}} A$ and junk-independence would not hold.

- $F^{eq}_\sigma M_S SP = M_{\sigma(s)} F^{eq}_\sigma M_S SP$

Fact 5.42 *This is an inheritance rule of reachability requirements*

Proof First of all by recalling the equivalence $F^{eq}_\sigma SP = Q_{eq} F_\sigma SP$ and considering that the reachability requirements have been proved to commute with Q, we restrict this proof to the case $F_\sigma M_S SP = M_{\sigma(s)} F_\sigma M_S SP$.

The rule asserts that $F_\sigma M_S SP$ is reachable on $\sigma(s)$. This can proven by contradiction assuming that $A \in F_\sigma M_S SP$ has a proper subalgebra A' only distinct on sort $\sigma(s)$. Since F_σ is a constructor, there exists $B \in M_S SP$ such that $A = Free_\sigma B$. $Free_\sigma$ cannot introduce non-reachable elements to the carrier $|B|_s$ because if that were the case there would exist another (smaller) extension of B, A', without non-reachable elements such that no homomorphism $h : Free_\sigma B \rightarrow A'$ would exist, violating the freeness of the extension. Finally, if $Free_\sigma$ does not introduce new non-reachable elements in s, those in A should already exist in B; hence contradicting the reachability of $B \in M_S SP$.

- (s is new sort) $\Rightarrow F^{eq}_\sigma SP = M_s F^{eq}_\sigma SP$

Fact 5.43 *This is an inheritance rule of reachability requirements*

Proof Considering the construction of a free extension $Free^{eq}_\sigma A$ as an initial object [Tar 86] it is clear that any new value must be finitely generated from constants and the old carriers in A, therefore any new sort has a reachable carrier.

5.5 Example:

The proof of $SetNat \overset{\kappa}{\rightsquigarrow} BagNat$ left open in section 2.3 and informally sketched in section 4.3 looks as follows:

1. Nat is a shared submodel.

   Since the specification branch only contains T and selectors, and all these preserve submodels (rule 7 instanciated by the cases 5.29 and 5.30), Nat is a submodel of $SetNat$.

   It is the same in the implementation branch up to the constructor (rule 10 instanciated by the preserving-submodels rules 5.29-5.30 turned into conserving-submodels rules as shown in 5.32). In the constructor, we can apply the conserving-submodels rule for the quotient since equational specifications are closed under quotients (see 5.34), and for the free extension since it is consistent and sufficiently complete (see 5.35). And finally the derive does not affect the subsignature corresponding to Nat so the preserving-submodel rule applies (see 5.31) and therefore Nat is also a submodel of $\kappa(BagNat)$.

2. M_{set} is satisfied.

   M_{bag} is inherited (rule 9) through free extensions and quotients (see 5.39 and 5.42) and the derive does not remove any constructor of *bag*, therefore M_{set} $(=M_{\sigma^{-1}(bag)})$ is satisfied by $\kappa(BagNat)$ (see 5.37).

3. The axioms of Set are satisfied.

   One by one we can prove the axioms of Set from $\kappa(BagNat)$. In this example we need the equation supplied by the quotient, the translation through the derive of the axioms in Bag, the translation of the axioms in the free extension, and the translation of a couple of inferred theorems for *bag*'s. For example:

   $\forall a : Nat, S : set.\ isempty(add(a, S)) = false$ is just the translation of a Bag axiom: $\sigma^{-1}(\forall a : Nat, S : bag.\ isempty(add(a, S)) = false)$ and we know that $SP \vdash \sigma(t) \Rightarrow D_\sigma SP \vdash t$ (see 5.15).

   $\forall a : Nat.\ a \in \emptyset = false$ can be deduced from $\{(\forall x : Nat.\ succ(x) > 0)\ ;\ (\forall a : Nat, S : bag.\ a \in S = count(a, S) > 0)\ ;\ (\forall a : Nat.count(a, \emptyset) = 0)\ \}$ for *bag* (see 5.19) and then translated to *set* (see 5.15).

   $\forall a : Nat, S : set.\ add(a, add(a, S)) = add(a, S)$ is satisfied since it belongs to *eq* in the quotient and we know that $Q_{eq}SP \vdash eq$ (see 5.22).

6 Final Remarks

6.1 Generalizing to an arbitrary institution

So far the work has been presented in the context of a particular institution, namely first order logic with equality. Throughout the paper some simplifications have been mentioned in the context of the equational institution, but some of the results are more general. Some of these generalizations are:

Arbitrary institution.- The original work from [ST85] which includes the definition of A, T, U, D and M_σ[5], and the proof rules for them mentioned in section 3.3, are institution independent. Moreover the characterization of these proof rules as M-complete or complete given in this paper (facts 5.14-5.20) is also institution independent (generalizing some considerations about hidding treated in [BHK 86]). The rules for the same operations on preserving submodels for T, U and D (facts 5.29-5.31) are directly institution independent. However, the rules which mention M_S should be transformed to suitable rules for M_σ.

Algebraic institution.- In an algebraic institution, that is considering algebras to be the models and algebraic signatures but disregarding the kind of sentences used, M_S is already definable. The rules concerning M_S dealing with submodels and all those concerning its commutativity and inheritance are directly valid. At the same time R_S makes sense since the minimal subalgebra of any algebra is unique. Summing up all rules are valid except those which explicitly use equations.

Algebraic institution with equality.- If the logic used in our algebraic institution is a conservative extension of equational logic then Q_{eq} and F_σ^{eq} are definable, and their inference rules are valid.

Apart from generalizing to an arbitrary institution, another important point should be noted. Analogously to what is done in [ST 86] when defining implementations, here the two-step strategy is presented in two layers. The first one defines a strategy independently of the particular SBO's, and the second one completes the strategy with rules for every SBO. This is comparable to the definition of constructor implementation with a separate list of possible constructors; the first definition is independent of the specification language considered and the second one is specific to it. Summing up, the specification language is independent of the logic used in the axioms (institution independent), and the implementation notion and the proving strategy are independent of the concrete SBO's.

[5] M_σ is more general than M_S, since it does not require the existence of sorts but only the existence of signature morphisms, which are available in any institution. Both SBO's have the same generality in an algebraic institution.

6.2 Extensions

The work presented can be continued in three main ways.

1. Try to make the proof method effective; that means, sort out how to compute the side conditions (presented as premises) of the rules.

   These side conditions are syntactic requirements such as 't is a new term', or semantic ones which can be replaced by stronger syntactic requirements as for instance 'junk-independent', or merely semantic requirements such as 'closed under subalgebras' or '$S \not\models \varphi$'. The work to be done is to find suitable syntactic requirements in order to be able to apply all the rules.

2. Extend the two-step strategy to handle some more cases. For instance, we do not know what to do when an M and some axioms cannot be pulled all the way to the end of the specification branch. However, proofs are still feasible if we can reproduce the same situation in the implementation branch, so we have to generalize matching below the end node of each branch.

3. Related to the previous point, extend the transformation rules to a more complete calculus of structured specifications, and investigate to what extent it is complete.

4. Use this technique in the context of abstractor implementations.

5. Explore possible simplifications when we substitute ASL by a more pragmatic workbench such as PLUSS (a specification language based on ASL) and consider only the reusability relation (a special case of constructor implementation).

6.3 Related work

Among all the work done in a similar area, and apart from the papers already mentioned as the basis of this work [ST 85, ST 86] this paper is particularly related to four earlier ones.

In first place the rules we give in section 5.1 are related to the transformations for ASL given in [SW 83]. The rules presented here can be considered as a selection of those for ASL which suit our proof purposes. For the sake of our goal, we have gone deeper in a few of the equivalences searching for side conditions where a general result was not possible, whereas in [SW 83] the main concern was to make ASL more understandable and only general results were presented.

In [EWT 82] some transformation rules and canonical forms are shown for a language similar to ours. In this case a *general (complete)* strategy looks obvious: reduce both branches to their canonical forms and compare them. But such a result is questionable for two reasons. First the language in [EWT 82] is a language

of constraints where basic specifications with axioms (presentions) are ignored, so that undecidability is avoided and canonical forms for the language of constraints are effectively computable, but comparing constraints is not comparing specifications. In the second place our language has a few more SBO's than the one in [EWT 82], in particular some involving equations which entails undecidability in their comparison. In this case, canonical forms would be non-effectively computable therefore any *complete* strategy is bound to be inapplicable.

In [BHK 86] an extensive treatment is given to a specification language whose SBO's are basically A_Φ, U, T_σ and D_σ with interesting results on the expression power of such a language when used with first order logic, coditional or equational logic. It differs from this work on the language considered, which is a sublanguage of ours, and on the purpose, which is the study of its expression power instead of a method for proving implementations.

Finally this work is related to that in [SB 83], where theorems are inferred from specifications represented by graphs. Our work can be seen as a new instance of the same problem for an extended specification language and with a more specific interest in proving implementations instead of plain theorems.

Strongly related to this paper is the recent work undertaken by D.Sannella and A.Tarlecki in [ST 88]. The main concern is finding correctness requirements for a framework of stepwise program development using Extended-ML[6]. Although both works are close, [ST 88] leaves open the problems related to non-trivial structure and especially addresses the issue of behavioral equivalence which has been ignore in this paper when restricting to constructor implementations (point 4 in extensions).

Acknowledgements

The author is grateful to Andrzej Tarlecki, Fernando Orejas, Cristine Choppy, Rod Burstall and Stuart Anderson for their reading of eralier drafts, to those attending the 6th Workshop on ADT for fruitful discussions and, especially, to Don Sannella for introducing me to the topic and for helping me closely throughout my work.

[6]Specification language using the module facilities of ML and semantically defined in terms of ASL.

7 References

[BHK 86] J.A.Bergstra, J.Heering, P.Klint. Module algebra. Centrum voor Wiskunde en Informatica, Report CS-R8617, 1986.

[Ber 87] G.Bernot. Good functors... are those preserving philosophy!. In proceedings of Summer Conference on Category Theory and Computer Science, Edinburgh 1987. LNCS 283, p. 182-195.

[BG 77] R.Burstall, J.Goguen. Putting theories together to make specifications. Proc. 5th International Joint Conference on Artificial Intelligence 1977. Volume Two, p. 1045-1058.

[B 87] R.Burstall. Inductively Defined Functions in Functional Programming Languages. Report CSR-230-87, Dept. of Computer Science, Univ. of Edinburgh.

[EKMP 82] H.Ehrig, H.-J.Kreowski, B.Mahr, P.Padawitz. Algebraic implementation of abstract data types. Theoretical Computer Science 20 (1982) p. 209-263.

[EWT 82] H.Ehrig, E.Wagner, J.Thatcher. Algebraic specifications with generating constraints. In 10th ICALP 1983, Barcelona. LNCS 154, p. 188-202.

[GB 80] J.Goguen, R.Burstall. CAT, a system for the structured elaboration of correct programs from structured specifications. SRI International, Technical Report CSL-118, 1980.

[GB 84] J.Goguen, R.Burstall. Introducing Institutions. Proc. Workshop on Logic of Programs. LNCS 140. Springer 1984. p. 221-256.

[GTW 76] J.Goguen, J.Thatcher, E.Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In Current Trends in Programming Methodology. Vol. 4. Prentice Hall 1978. p. 80-149.

[KM 87] D.Kapur, D.R.Musser. Proof by consistency. Journal of Artificial Intelligence 31 (1987), p. 125-157.

[SB 83] D.Sannella, R.Burstall. Structured theories in LCF. Proc. 8th Colloq. on Trees in Algebra and Programming. L'Aquila, Italy. LNCS 159 (1983), p. 377-391.

[ST 85] D.Sannella, A.Tarlecki. Specifications in an arbitrary institution. Information and Computation 76 (1988), p. 165-210.

[ST 86] D.Sannella, A.Tarlecki. Towards formal development of programs from algebraic specifications: Implementations revisited. Acta Informatica 25 (1988), p. 233-281.

[ST 88] D.Sannella, A.Tarlecki. Toward formal development of ML programs: foundations and methodology. Extended abstract in *Proc. Colloq. on Current Issues in Programming Languages*, Joint Conf. on Theory and Practice of Software Development (TAPSOFT), Barcelona, March 1989, Springer Lecture Notes in Computer Science, to appear.

[SW 83] D.Sannella, M.Wirsing. A kernel language for algebraic specification and implementation. Proc. Intl. Conf. on Foundations of Computation Theory, Borgholm, Sweden. Springer LNCS 158, p. 413-427.

[Tar 86] A.Tarlecki. On the existence of free models in abstract algebraic institutions. Theoretical Computer Science 37, p. 269-304.