Proving correctness w.r.t. specifications with hidden parts

Proving correctness w.r.t specifications with hidden parts

by

Jordi Farrés-Casals

LFCS Report Series ECS-LFCS-90-117

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

July 1990

Proving correctness w.r.t. specifications with hidden parts

Jordi Farrés-Casals[*]

July 11, 1990

Abstract

The task of proving the correctness of an implementation w.r.t. a formal specification is sometimes complicated by the use of auxiliary (*hidden*) functions and sorts within the specification which are needed for the specification but are not meant to be implemented.

Auxiliary sorts and functions are the normal way to express requirements in abstract model specifications. Algebraic specifications became popular as a way to define the elements of a system without representing them in terms of more primitive concepts, avoiding the definition of any extra structure. However, it has been shown that hidden functions are in general necessary for specifying computable functions [Maj 77, TWW 79].

In this paper we analyze general proving techniques for specifications with hidden parts and, in particular, an strategy which is complete when some side conditions are met.

1 Introduction

Many algebraic specifications of a system, especially if they are large, provide a lot of information describing functions and data types which are not interesting either to the user of the system or to the person implementing that system.

From a technical point of view, some of this auxiliary information is known as the **hidden part** of a specification and it has been shown necessary to overcome the lack of expressiveness of ordinary equational specifications.

From a methodological point of view, the hidden part of a specification *biases* the implementation: those implementations of the system based on an implementation of the hidden part are easier to verify. In such cases a change of *bias* can cause great disruption in the proof. For example, consider a `compiler` function

[*]Laboratory for Foundations of Computer Science, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, Scotland. Supported by a grant from the Ministerio de Educación y Ciencia.

which is specified by a simple interpreter and then implemented in a more efficient way without respecting any of the structures by means of which `compiler` was defined. The proof of correctness would be a great deal more difficult that the proof of correctness of a direct implementation of the interpreter.

In this paper we are interested in implementations which do not follow the bias of the specification, and their correctness proofs. This is a very common case in practice since following the specification bias may lead to inefficient implementations, as in the case of the compiler or more precisely in the example below.

1.1 Example

Suppose we want to add a function `mre: listnat -> nat` which computes the most repeated element in a list, to a specification *ListNat* of lists of natural numbers. Soon we realize that the available functions in *ListNat*, i.e. `car, cdr, emptylist, cons` and some operations on naturals, are not enough to directly define the new function. We proceed by defining an auxiliary function which counts the number of repetitions of a natural number in a list of natural numbers.

```
SP1= Enrich ListNat by
       Hidden
          count:  listnat, nat -> nat.
          ∀ x:nat. count(emptylist, x)= 0;
          ∀ x1,x2:nat; l:listnat.
           count(cons(x1,l),x2)= if x1=x2 then 1+count(l,x2)
                                          else count(l,x2)
       in
          mre:  listnat -> nat.
    (1)   ∀ x:nat; l:listnat.  count(l, mre(l)) >= count(l,x)
    end
```

Any efficient implementation of `mre` will not implement `count` directly but a function which computes simultaneously the frequencies of all elements in the list. In a stepwise refinement methodology, the following step leads in the direction of the desired implementation by defining `mre` in terms of a new function `frequencies: listnat -> set(nat x nat)` and some extra structure defining sets of pairs of natural numbers *SetPairsNat* (with sort name `set(nat x nat)`) and lists of pairs of natural numbers *ListPairsNat* (with sort name `list(nat x nat)`).

```
SP2= Enrich ListNat by
       Hidden
          Enrich SetPairsNat + ListPairsNat by
```

```
    sort:  set(nat x nat) -> list(nat x nat)
    frequencies:  listnat -> set(nat x nat).
    frequencies(emptylist)= emptyset;
    ∀ l:listnat; i,x:nat.
     if ((i,x)∈frequencies(l)
         and ∀ j:nat. (j,x)∈frequencies(l) ⇒ i≥j)
     then frequencies(cons(x,l))=frequencies(l)∪{(i+1,x)}

    ∀ l:listnat; x:nat.
     if ∄ i:nat.  (i,x)∈frequencies(l)
     then frequencies(cons(x,l))=frequencies(l)∪{(1,x)}
    ...
    { Axioms for sorting a set of pairs by the first
    element of the pair, in decreasing order }
    ...
in
    ∀ l:listnat.
    mre(l)= if l=emptylist then 0
                 else proj2(car(sort(frequencies(l))))
end
```

The function `frequencies` yields a set of pairs, where each pair contains a counter and an element of the argument list; for example

`frequencies([1,3,2,1])={(1,1),(1,3),(1,2),(2,1)}`

Here, `1` occurs twice in the argument list so the result contains both `(1,1)` (first `1`) and `(2,1)` (second `1`). We have `car(sort(frequencies([1,3,2,1])))=(2,1)` and so the `mre` of the list is `1`.

In the next refinement step `frequencies` can be made visible, i.e. forced to be implemented, or its definition may be changed so that redundant information is not included. Moreover picking the largest element need not be implemented using a sorting operation as in `car(sort(...))`. Eventually, after a few steps we obtain a correct and efficient implementation.

The above refinement step is proven correct by showing that the axiom (1) which defines `mre` in the specification *SP*1 can be deduced from the implementation, *SP*2, together with the definition of `count` (the hidden part of *SP*1): Informally the proof proceeds by proving that:

1. The frequencies of the elements of a list are in the set created by `frequencies`.

```
   ∀ l:listnat; x:nat.
     if count(l,x)>0 then (count(l,x),x)∈ frequencies(l)
```

2. The counters in the set created by `frequencies` denote at most the real frequencies.

   ```
   ∀ i,x:nat; l:listnat.
     if (i,x)∈ frequencies(l) then i≤count(l,x)
   ```

3. By definition of `sort` and `car` the term `car(sort(s))` yields the pair in `s` with the highest counter.

   ```
   ∀ s:set(nat x nat).  if s≠emptyset then
     ∀ i,y:nat. if (i,y)∈s then proj1(car(sort(s)))≥i
   ```

4. From the above theorems we conclude that for non-empty lists `count(x,l)` can be expressed in terms of `frequencies(l)`.

   ```
   (∃ x:nat. (count(l,x), x)=car(sorting(frequencies(l))))
     or (l=emptylist)
   ```

5. Hence, `mre(l)` is the most repeated element in `l` provided `l` is not an empty list, and zero when `l` is empty. Therefore, it is always the case that:

   ```
   ∀ x:nat; l:listnat.  count(l, mre(l)) ≥ count(l,x)
   ```

Apart from the technicalities of the proof, the interest of this example arises from the difficulties in generalizing this style of proof to a specification language with arbitrary use of hiding and justifying that it is sound.

The rest of the paper is divided in five sections. Section 2 gives a formal presentation of the problem and shows how little can be done in general. Section 3 enphasizes the particular structure of specifications such as *SP*1 and defines two general properties of this kind of specifications: *persistency* and *independence.* In section 4 the strategy is presented and proved sound for persistent hidden parts. In section 5 the strategy is proven complete specifications with an independent hidden part. Section 6 summarizes the results and draws some conclusions.

2 A specification language with hiding

We shall consider a specification to be a first order presentation with a part of the signature considered hidden. In fact, the restriction presentations to FOLEQ[1] is not needed but it helps to keeps things simple.

Our specification language is defined as follows:

[1]First order logic with equality.

Syntax

$$SP == \Phi_\Sigma |\ D_\iota SP \mid T_\iota SP \mid \Phi SP$$

Where Σ is a multi-sorted first order signature, Φ is a finite set of multi-sorted first order sentences, and ι is a signature inclusion between multi-sorted first order signatures.

A specification is well-formed if in Φ_Σ, Φ consists of sentences over Σ; in $D_\iota SP$ with $\iota : \Sigma 1 \hookrightarrow \Sigma 2$, SP is a specification over $\Sigma 2$; and in $T_\iota SP$ with $\iota : \Sigma 1 \hookrightarrow \Sigma 2$, SP is a specification over $\Sigma 1$.

The semantics of a well-formed specification SP is defined by a first order signature $Sig[SP]$ and a class of structures satisfying the specification $Mod[SP]$. Well-formed specifications and their semantics are defined recursively as follows:

Semantics

	$Sig[]$	$Mod[]$	Conditions
Φ_Σ	Σ	$\{A \in Mod[\Sigma] \mid A \models \Phi\}$	Φ defined over Σ
$D_\iota SP$	$\Sigma 1$	$\{A\|_\iota \mid A \in Mod[SP]\}$	$Sig[SP] = \Sigma 2,\ \iota : \Sigma 1 \hookrightarrow \Sigma 2$
$T_\iota SP$	$\Sigma 2$	$\{A \in Mod[\Sigma 2] \mid A\|_\iota \in Mod[SP]\}$	$Sig[SP] = \Sigma 1,\ \iota : \Sigma 1 \hookrightarrow \Sigma 2$
ΦSP	Σ	$\{A \in Mod[SP] \mid A \models \Phi\}$	Φ defined over Σ, $Sig[SP] = \Sigma$

Where $A|_\iota$ stands for the reduct model along the signature inclusion ι and $\models$ for the standard satisfaction relation between models and first order sentences.

Intuitively, an operator T_ι over a specification SP extends the signature of a SP with new symbols which can be arbitrarily interpreted whereas the old symbols must preserve their interpretation as in SP. An operator D_ι over a specification SP reduces its signature but it keeps the same meaning for the symbols which are left, in other words, it hides some symbols.

Two specification are equivalent is they have the same signature and class of models. Given this notion of equivalence, and since this specification language is a sublanguage of ASL [SW 83], a normalization result follows immediately from [Bre 89].

Fact 2.1 *Every specification SP is equivalent to a specification of the form $D_\iota \Phi$. Moreover, $D_\iota \Phi$ can be computed from SP.*

This fact shows that we can think of specifications as just presentations with a hidden part. This simple language is adequate to illustrate the main problems behind hiding and also their solution.

Second order logic is related to this specification language since D_ι can be seen as a generalization of a second order existential quantifier. The difference is that second order logic is single sorted and therefore $\exists^2$ cannot hide sorts whereas D_ι can.

Moreover, as in ASL, the underlying logic for the sentences and the nature of the models can be changed to another institution [GB 84, ST 88a] without affecting forthcoming results (see Conclusions).

Entailment & Proofs

Implementations of a specification $SP1$ are assumed to be specifications $SP2$ restricting the class of models, i.e. $Mod[SP2] \subseteq Mod[SP1]$. Therefore, we shall refer to them simply as antecedent and consequent, written $SP2 \models SP1$. Since all specifications can be reduced to their normal form, our task can be described as follows:

Task: $D_{\iota 2}\Phi 2 \models D_{\iota 1}\Phi 1$

Given two finite presentations $\Phi 1$ and $\Phi 2$ over signatures $\Sigma H1$ and $\Sigma H2$ which are two extensions of Σ, prove that all models over Σ obtained by restricting models of $\Phi 2$ can be obtained by restricting models of $\Phi 1$.

$$\begin{tikzcd}
\Sigma H1 & & \Sigma H2 \\
& \Sigma \arrow[ul, "\iota 1"] \arrow[ur, "\iota 2"'] &
\end{tikzcd}$$

Given such a task and considering that we only know how prove entailment between sentences over the same signature, there are only two *reasonable* things we may try naïvely to do: prove that $\Phi 1$ follows from $\Phi 2$ in an appropriate extension of both signatures, or prove that all visible consequences of $\Phi 1$ follow from $\Phi 2$. Unfortunately, the first approach only works for some trivial cases and the second one is unsound.

First naïve approach.

If we try the first one, we want to mix in our reasoning sentences over $\Sigma H1$ and over $\Sigma H2$ without confusing their auxiliary symbols; therefore, we consider the pushout signature ΣH. Now, if we prove that $\iota 1'(\Phi 2) \models \iota 2'(\Phi 1)$ (also written $\Phi 2 \models_{\Sigma H} \Phi 1$) then since $\Phi 2$ is finite (equivalent to a sentence), by the Craig interpolation lemma there exists a finite set of sentences Φ over Σ such that

$$\begin{tikzcd}
& \Sigma H & \\
\Sigma H1 \arrow[ur, "\iota 2'"] & & \Sigma H2 \arrow[ul, "\iota 1'"'] \\
& \Sigma \arrow[ul, "\iota 1"] \arrow[ur, "\iota 2"'] &
\end{tikzcd}$$

$$(2) \quad \Phi 2 \models_{\Sigma H2} \Phi \quad \textit{and} \quad \Phi \models_{\Sigma H1} \Phi 1$$

Turning around the argument, a refinement $D_{\iota 2}\Phi 2 \models D_{\iota 1}\Phi 1$, can only be proven correct in this fashion if there exists an intermediate flat specification Φ satisfying (2).

However, if this is case, the symbols in hidden part of $\Phi 1$ are rather trivial since their properties can be inferred from sentences in Φ which do not mention them. In specifications such as `mre` in section 1.1, a hidden part is not only needed in the specification but also in the implementation. In general, we expect the implementation to carry more auxiliary and hidden symbols that the specification, contrary to the cases which are provable using this strategy.

This approach offers surprisingly little. We might think that at least those implementations which proceed by implementing the hidden part used in the specification should be provably correct. For example, given the specification *SP*1 in section 1.1 we propose the following refinement:

```
SP2= Enrich ListNat by
        Hidden
           count:  listnat, nat -> nat
           ...
           { Axioms of count as in SP1}
           ...
        in
           mre:  listnat -> nat.
           ∀ x:nat; l:listnat.
            mre([])= 0;
            mre(cons(x,l))= if count(l,x)+1<count(l,mre(l))
                            then mre(l)
                            else x
     end
```

In this case the proof is easier because the implementation follows the *bias* of the specification, and we can directly prove that $\Phi 2 \models_{\Sigma H2} \Phi 1$. The difference with the approach just proposed is the existence of an inclusion $\iota 3$ (identity in this example) which is missing in the general case.

$$\begin{tikzcd}
\Sigma H1 \arrow[rr, "\iota 3"'] & & \Sigma H2 \\
& \Sigma \arrow[ul, "\iota 1"] \arrow[ur, "\iota 2"'] &
\end{tikzcd}$$

On the the other hand, the existence of $\iota 3$ does not guarantee that a refinement can be proved correct since the same symbols may have a different meaning in the two specifications.

Second naïve approach

More realistic is the second approach where we prove that:

$$\Phi 2 \models \Phi 1^{**}|_{\iota 1}$$

where $\_^{**}$ means closure of a set of sentences under semantic entailment and $\_|_{\iota 1}$ forgets the sentences which mention symbols not in Σ. Unfortunately, due to the difference in expressive power between flat theories (even if they are infinite) w.r.t.

presentations with hidden parts studied in [Maj 77, TWW 79, BBTW 81], this proof strategy is, in general, unsound [BHK 86, Far 89]. For example (from [BHK 86]), suppose we are asked to prove

$$Nat \models NonStandNat$$

where *Nat* is a specification of natural numbers including the standard interpretation of the natural numbers as a model, whereas *NonStandNat* specifies the class of the non-standard models for the natural numbers. It is well-known that all the first-order properties of the non-standard models are satisfied by the standard models of the natural numbers; however the refinement is incorrect.

In many cases this strategy is sound but then it happens that $\Phi1^{**}|_{\iota1}$ is infinite and may not be finitely presentable, making the proof very hard. Even worse, in a case where the strategy is sound and $\Phi1^{**}|_{\iota1}$ happens to be equivalent to a finite presentation Φ, we can prove that $\Phi2 \models \Phi$ but it will hard to prove the equivalence between Φ and $\Phi1^{**}|_{\iota1}$, in particular, it will be as difficult as the original task.

We conclude that given the general task of proving refinements between specifications with hidden parts we fail to prove cases such as the one in section 1.1, and indeed there are very few cases where refinement can be proven. In the following section we present some restrictions to the general case needed to successfully prove examples like the one in section 1.1.

3 Persistent and independent enrichments

The main difference between the general case, as presented in the last section, and the example, is the existence of a structure in the axioms of the example which allows us to distinguish some of them as *defining the hidden part* and some others as *using the hidden part in order to define a visible enrichment.* Therefore, we shall assume from now on that our task carries such a structure. This structure is formally defined below.

In general, we shall distinguish within the visible part, Σ, those visible symbols in terms of which the hidden enrichment is defined, $\Sigma0$, the inclusion along the hidden enrichment, $\iota1_h$, and finally the inclusion along the visible enrichment defined on *top* of the hidden enrichment, $\iota1_v$.

In the example *SP*1 in section 1.1, $\Sigma0$ is the signature of *ListNat*, $\iota1_h$ adds count, and $\iota1_v$ adds mre producing the whole signature $\Sigma H1$, which is then constrained by hiding count along $\iota1$.

$$\begin{tikzcd}
\Sigma H1 & \\
& \Sigma \arrow[ul, "\iota1"'] \\
\Sigma0H1 \arrow[uu, "\iota1_v"] & \\
& \Sigma0 \arrow[uu, "\iota_E"'] \arrow[ul, "\iota1_h"]
\end{tikzcd}$$

Such a structure of the signatures must be imposed upon the sentences so that, instead of having a single set of sentences $\Phi 1$ over $\Sigma H1$ and consequent $D_{\iota 1}\Phi 1$ as in the last section, we have three sets of sentences:

$\Phi 1_0$:	set of sentences over $\Sigma 0$.
$\Phi 1_h$:	set of sentences over $\Sigma 0 H1$ specifying the hidden part.
$\Phi 1_v$:	set of sentences over $\Sigma H1$ specifying a visible enrichment using the hidden part.

Therefore the consequent is:

$$D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0$$

Now, we must establish the conditions under which such a decomposition captures the informal ideas of *specifying the hidden part* and *using the hidden part for specifying a visible enrichment.* In order to do that, we define the concepts of *persistency* and *independence.*

Definition 3.1 *An* **enrichment** *by Σ_e-sentences Φ_e w.r.t. a signature inclusion $\iota_e : \Sigma \hookrightarrow \Sigma_e$ is the term $\Phi_e T_{\iota_e}$ which produces specifications over Σ_e when it is composed with specifications over Σ.*

Given a signature inclusion $\iota_e : \Sigma \hookrightarrow \Sigma_e$ and Σ_e-sentences Φ_e, an enrichment $\Phi_e T_{\iota_e}$ is **persistent** *w.r.t. a Σ-specification SP iff for all $A \in Mod[SP]$ there is a model $B \in Mod[\Phi_e T_{\iota_e} SP]$ which extends A, i.e. $B|_{\iota_e} = A$.*

Given signature morphisms ι_E, ι_v, ι and ι_h forming a pushout diagram as in the figure above, and sets of sentences Φ_h over $\Sigma 0 H1$ and Φ_v over $\Sigma H1$, an enrichment $\Phi_h T_{\iota_h}$ is **independent** *w.r.t. a $\Sigma 0$-specification SP and an enrichment $\Phi_v T_{\iota_v}$, iff for all $A \in Mod[(\Phi_h T_{\iota_h})SP]$ and $B \in Mod[D_\iota(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})SP]$ such that $B|_{\iota_E} = A|_{\iota_h}$, there exists $C \in Mod[(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})SP]$ such that $C|_{\iota_v} = A$ and $C|_\iota = B$.*

$$\begin{tikzcd}
C \arrow[d, "-|_{\iota_v}"'] \arrow[dr, "-|_{\iota}"] & \\
A \arrow[dr, "-|_{\iota_h}"'] & B \arrow[d, "-|_{\iota_E}"] \\
& A|_{\iota_h} = B|_{\iota_E}
\end{tikzcd}$$

Remarks

1. An enrichment to a presentation is a theory extension and a persistent enrichment is a particular kind of conservative extension.

2. If ι_h and ι_v are inclusions, we can restrict the definition of independence to the existence of inclusions ι and ι_E without loss of generality. Moreover, if the pushout of inclusions exists then inclusions ι and ι_E are unique given ι_h and ι_v.

   It worth noting, that inclusions are only used for the sake of simplicity. The results below hold on arbitrary signature morphisms, although persistency of an enrichment $\Phi_e T_{\iota_e}$ is generaly lost is ι_e is not injective.

3. Often, we omit the translations, T_ι, where this can be inferred from the context. Thus we write Φ_e for $\Phi_e T_{\iota_v}$, also we write $[\Phi_h]$ to denote the hidden enrichment and omit D_ι. Thus, $D_\iota(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})SP$ can be written as $\Phi_v[\Phi_h]SP$.

   We shall use this short notation in our explanations, though the full notation is kept in the results and their proofs.

4. The functor *Models*() mapping each signature Σ to its category of models, first order structures over Σ, is co-complete. Therefore, the pushout diagram of signatures induces a pullback diagram of models. Hence, C happens to be the pullback model of A and B w.r.t. $\_|_{\iota_h}$ and $\_|_{\iota_E}$. We shall call this model "amalgamed union" (generalizing the notation introduced in the amalgamation lemma [ST 88b] for algebras), denoted $A \oplus B$.

   Now, we can rephrase the definition of independent, by saying that: Φ_h *is independent w.r.t. SP and* Φ_v *iff the specification* $\Phi_v\Phi_h SP$ *is closed under amalgamed unions between models of* $\Phi_h SP$ *and models of* $\Phi_v[\Phi_h]SP$.

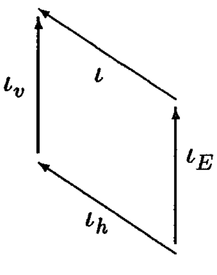
From an intuitive point of view, persistency of a hidden enrichment Φ_h w.r.t. a specification *SP* ensures that *SP* is not indirectly modified by some requirements of the hidden part, i.e. $[\Phi_h]SP$ is equivalent to *SP*. Independence of a hidden enrichment w.r.t. specification and a later visible enrichment means that the choice of a particular model for the hidden part does not rule out any of the possible solutions (visible models).

In next section we shall see how these conditions relate to correctness proofs, but we show first how independence is related to two other properties.

Independence versus persistency

We might think that a more *natural* condition upon a visible enrichment on top of a hidden enrichment is persistency, at least persistency w.r.t. the hidden symbols. This notion can be formalized as relative persistency.

Definition 3.2 *Given signature morphisms* ι_E, ι_v, ι *and* ι_h *forming a pushout diagram as in the figure, an enrichment* $\Phi_v T_{\iota_v}$ *is* **relatively persistent** *w.r.t. a previous enrichment* $\Phi_h T_{\iota_h}$ *and a specification SP, iff for all* $A \in Mod[(\Phi_h T_{\iota_h})SP]$ *for which there exists* $B \in Mod[D_\iota(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})SP]$ *such that* $B|_{\iota_E} = A|_{\iota_h}$, *there exists* $C \in Mod[(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})SP]$ *such that* $C|_{\iota_v} = A$.



This property states persistency of the second enrichment w.r.t. that part of the signature added during the first (hidden) enrichment. Technically, C extends A as in the definition of persistent enrichment, but now A is not an arbitrary model; hence, relative persistency is weaker that persistency. Intuitively, the axioms of Φ_v cannot further constraint the symbols to be hidden but they can add new constraints to the old visible symbols defined in SP.

It is not difficult to see that independence entails relative persistency.

Proposition 3.3 *If an enrichment $\Phi_h T_{\iota_h}$ is independent w.r.t. a specification SP and later enrichment $\Phi_v T_{\iota_v}$, then $\Phi_v T_{\iota_v}$ is relatively persistent w.r.t. $\Phi_h T_{\iota_h}$ and SP.*

Proof Trivial, by taking $A \oplus B$ as the extension C of A. □

But, independence is more than persistency of the hidden enrichment plus relative persistency of the visible enrichment. Consider the specification $D_\iota(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})\Phi$ with:

$$\begin{array}{ll} \iota_h = \{s; a, b\} \hookrightarrow \{s; a, b, h\} & \Phi = \{a \neq b\} \\ \iota_v = \{s; a, b, h\} \hookrightarrow \{s; a, b, h, v\} & \Phi_h = \emptyset \\ \iota = \{s; a, b, v\} \hookrightarrow \{s; a, b, h, v\} & \Phi_v = \{v = h\} \end{array}$$

Or more explicitly, it can be expressed by:

```
Enrich
   sorts s
   operations a,b:  s.
   axioms a≠b
   Hidden
      operations h:  s.
by
   operations v:  s.
   axioms v=h
end
```

Here, both enrichments are persistent but $\Phi_h T_{\iota_h}$ fails to be independent because $\{a = 1, b = 2, v = 1, h = 2\}$ is not a model of $(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})\Phi$ but it can be obtained as the amalgamated union of a model of $D_\iota(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})\Phi$, $\{a = 1, b = 2, v = 1\}$, and a model of $(\Phi_h T_{\iota_h})\Phi$, $\{a = 1, b = 2, h = 2\}$.

Since neither relative persistency nor persistency (as above) are enough to guarantee independence, we might think that persistency is weaker than independence as relative persistency has been shown to be. However, independence does not guarantee persistency. Consider for example:

$$\begin{array}{ll} \iota_h = \{s; a, b\} \hookrightarrow \{s; a, b, h\} & \Phi = \emptyset \\ \iota_v = \{s; a, b, h\} \hookrightarrow \{s; a, b, h, v\} & \Phi_h = \emptyset \\ \iota = \{s; a, b, v\} \hookrightarrow \{s; a, b, h, v\} & \Phi_v = \{v = a, v = b\} \end{array}$$

In this case the hidden enrichment is independent; i.e. given two models for $\{s; a, b, v\}$ and $\{s; a, b, h\}$, if $v = a = b$ in the first and both agree in $\{s; a, b\}$ then they can be combined into one $(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})\Phi$. But the visible enrichment it is not persistent since it requires $a = b$.

Summing up, independence and persistency are unrelated. However, relative persistency is a weak notion of persistency which is necessary for independence.

Independence versus persistent functors

It is very common to find cases like count where the hidden part is *totally* defined; i.e. $\Phi_h T_{\iota_h}$ is a persistent functor over the models of Φ. Then independence follows automatically:

Proposition 3.4 *Given a specification $(\Phi_h T_{\iota_h})SP$ such that for all $A \in Mod[SP]$ there exists exactly one (up to isomorphism) $B \in Mod[(\Phi_h T_{\iota_h})SP]$ such that $B|_{\iota_h} = A$, then for any visible enrichment $\Phi_v T_{\iota_v}$ and morphisms ι and ι_E forming a pushout diagram as in the figure, $\Phi_h T_{\iota_h}$ is independent w.r.t. SP and $\Phi_v T_{\iota_v}$.*

Proof For a given model B of $D_\iota(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})SP$, there is (up to isomorphism) a unique $A \in Mod[(\Phi_h T_{\iota_h})SP]$ such that $A|_{\iota_h} = B|_{\iota_E}$. On the other hand, by definition of D_ι there exists a $C \in Mod[(\Phi_v T_{\iota_v})(\Phi_h T_{\iota_h})SP]$ such that $C|_\iota = B$. Since $C|_{\iota_h;\iota_v} = A|_{\iota_h}$, by the uniqueness of A we conclude that C is also an extension of A, $C|_{\iota_v} = A$.
□

$$\begin{tikzcd}
C \arrow[d, "-|_{\iota_v}"'] \arrow[dr, "-|_{\iota}"] & \\
A \arrow[dr, "-|_{\iota_h}"'] & B \arrow[d, "-|_{\iota_E}"] \\
& A|_{\iota_h} = B|_{\iota_E}
\end{tikzcd}$$

The relation between persistent functors and amalgamated unions is not new. If we consider a specification language where specifications only denote isomorphic classes of models and hidden enrichments are required to be persistent functors (such as flat and parameterized specifications in the initial approach [EM 85]) the extension lemma establishes a similar connection.

4 Proving refinements with hiding - soundness

As we notice in section 1.1, some proofs of refinement can be carried out by *importing* the hidden part of the specification into the implementation and then proving that the visible part of the specification follow from the *enriched implementation.* The machinery introduced in the last section provides the means to formalize such a proof strategy.

In the following, our task is assumed to be proving

$$D_{\iota 2}\Phi 2 \models \Phi 1_v[\Phi 1_h]\Phi 1_0$$

instead of $D_{\iota 2}\Phi 2 \models D_{\iota 1}\Phi 1$ as it was proposed in section 2. Hiding is only required to be explicit in the consequent since explicit hiding in the antecedent restricts the task without getting any better results.

At our disposal, we have an inference system for first order logic with equality, $\models$, and the satisfaction lemma relating reducts to satisfaction, i.e. for all morphisms $\iota : \Sigma 1 \rightarrow \Sigma 2$, models A over $\Sigma 2$ and sentences φ over $\Sigma 1$,

$$A|_\iota \models \varphi \quad \text{iff} \quad A \models \iota(\varphi)$$

As said before, when morphisms are understood from the context $A \models \iota(\varphi)$ can be denoted by $A \models_{\Sigma 2} \varphi$.

Lemma 4.1 *Given presentations* $\Phi 1$ *over* $\Sigma 0$, $\Phi 1_h$ *over* $\Sigma 0 H1$, $\Phi 1_v$ *over* $\Sigma H1$ *and* $\Phi 2$ *over* $\Sigma H2$,

$$\frac{\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v \qquad \Phi 2 \models_{\Sigma H} \Phi 1_0}{D_{\iota 2}\Phi 2 \models D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0}$$

is a sound inference rule provided the signature morphisms are arranged according to the two pushout diagram to the right and $\Phi 1_h T_{\iota 1_h}$ *is a persistent enrichment w.r.t.* $\Phi 1_0$.

$$\begin{tikzcd}
 & \Sigma H & \\
\Sigma H1 \arrow[ur, "\iota 2'_h"] & & \Sigma H2 \arrow[ul, "\iota 1'_h"'] \\
 & \Sigma \arrow[ul, "\iota 1"'] \arrow[ur, "\iota 2"] & \\
\Sigma 0 H1 \arrow[uu, "\iota 1_v"] & & \\
 & \Sigma 0 \arrow[uu, "\iota_E"'] \arrow[ul, "\iota 1_h"] &
\end{tikzcd}$$

Proof For an inconsistent theory $\Phi 2$ the lemma holds trivially.

Otherwise, let A be a model of $D_{\iota 2}\Phi 2$; then there exists $B_2 \models \Phi 2$ such that $B_2|_{\iota 2} = A$, and $C \in Mod[(\Phi 1_h T_{\iota 1_h})\Phi 1_0]$ such that $C|_{\iota 1_h} = A|_{\iota_E}$. C is guaranteed to exist because, by the rule's premise $\Phi 2 \models_{\Sigma H} \Phi 1_0$ so $A|_{\iota_E} \models \Phi 1_0$ and, by persistency of $\Phi 1_h T_{\iota 1_h}$, $A|_{\iota_E}$ can be extended to an algebra C as required.

Now, we consider amalgamated unions $B_1 = A \oplus C$ and $G = B_1 \oplus B_2$ according to the diagram at the right.

$$\begin{tikzcd}
 & G \arrow[dl, "-|_{\iota 2'_h}"'] \arrow[dr, "-|_{\iota 1'_h}"] & \\
B_1 \arrow[dd, "-|_{\iota 1_v}"'] \arrow[dr, "-|_{\iota 1}"] & & B_2 \arrow[dl, "-|_{\iota 2}"] \\
 & A \arrow[dd, "-|_{\iota_E}"] & \\
C \arrow[dr, "-|_{\iota 1_h}"'] & & \\
 & A|_{\iota_E} &
\end{tikzcd}$$

By the satisfaction lemma it can be shown that $G \models_{\Sigma H} \Phi 2 \cup \Phi 1_h$. Then from the rule's premise $G \models_{\Sigma H} \Phi 1_v$, therefore $G|_{\iota 2'_h} \in Mod[(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0]$ and finally $A = G|_{\iota 1;\iota 2'_h} \in Mod[D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0]$. $\square$

Persistency is a sufficient condition for the soundness of the rule, but it is not necessary; i.e. a correct refinement

$$D_{\iota 2}\Phi 2 \models D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0$$

may satisfy the premises of the rule and $\Phi 1_h T_{\iota 1_h}$ not to be persistent w.r.t. $\Phi 1_0$.

The crucial point for considering persistency as an adequate condition has to do with its quantification, i.e. which sets of axioms (free variables of the rule) out of $\Phi 1_0, \Phi 1_h, \Phi 1_v$ and $\Phi 2$ are bound and which are free in the condition. For example, let

$\forall \Phi 1_0, \Phi 1_h, \Phi 1_v, \Phi 2.$

$Condition(\Phi 1_0, \Phi 1_h, \Phi 1_v, \Phi 2) \iff Rule(\Phi 1_0, \Phi 1_h, \Phi 1_v, \Phi 2)$

mean that *Condition* is a condition for soundness and completeness of the rule. In our case we have (by lemma 4.1 and lemma 4.2 below)

$\forall \Phi 1_0, \Phi 1_h.$

$Persistency((\Phi 1_h T_{\iota 1_h})\Phi 1_0) \iff \forall \Phi 1_v, \Phi 2.\ Rule(\Phi 1_0, \Phi 1_h, \Phi 1_v, \Phi 2)$

which means that persistency is adequate when we look for a condition on $\Phi 1_0$ and $\Phi 1_h$ independently of what $\Phi 1_v$ and $\Phi 2$ might be. From a methodological point of view, this is satisfactory since we would expect that specifications are written before their implementations and that inner enrichments are written before outer ones; thus, at the time of writing the specification of the hidden part, $\Phi 1_h$, only $\Phi 1_0$ is bound.

Some alternative sufficient conditions for soundness can be found by changing the bound variables; e.g.

$$\forall \Phi 1_h, \Phi 2.\ Persistency((\Phi 1'_h T_{\iota 1'_h})\Phi 2) \Longrightarrow \forall \Phi 1_0, \Phi 1_v.\ Rule(...)$$

where $\Phi 1_h = \iota 2'_h(\iota 1_v(\Phi))$. This condition is weaker than persistency of $\Phi 1_h T_\iota$ w.r.t. $\Phi 1_0$ since it only requires persistency of those models of the consequent which are fewer than the models of the antecedent. Looking at the proof of the lemma we realize that only persistency of $A|_{\iota_E}$ for $A \in Mod[D_{\iota_E;\iota 2}\Phi 2]$ is required. Therefore, persistency of $\Phi 1'_h T_{\iota 1'_h}$ w.r.t. $D_{\iota 2}\Phi 2$ guarantees soundness.

Unfortunately, this soundness conditions asks for requirements on the antecedent (implementation side), and that is not methodologically desired.

Considering only bound variables in the antecedent, we can produce a rule such as

$$\forall \Phi 1_0, \Phi 1_h, \Phi 1_v.\ ThPersistency((\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0) \Longrightarrow \forall \Phi 2.\ Rule(...)$$

where *ThPersistency* means that only persistency for those models of $\Phi 1_0$ which satisfy $(\Phi 1_v \cup \Phi 1_h \cup \Phi 1_0)^{**}|_{\Sigma 0}$ is needed. This also easy to prove because other models cannot be correct implementations and only persistency over correct implementations (models of $D_{\iota 2}\Phi 2$) is needed.

However, *ThPersistency* is not a necessary condition even under a proper quantification and in practice differs little of the clean model-theoretical definition of persistency.

Summing up, these sufficient conditions for soundness can be used for proving a refinement correct but are not adequate conditions to consider when designing specifications.

To conclude we can prove, as promised above, that persistency is indeed necessary w.r.t. the proper quantification.

Lemma 4.2 *Given finite presentations* $\Phi 1_0$ *over* $\Sigma 0$ *and* $\Phi 1_h$ *over* $\Sigma 0 H 1$,

$$\left(\forall \Phi 1_v, \Phi 2.\ \frac{\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v \qquad \Phi 2 \models_{\Sigma H} \Phi 1_0}{D_{\iota 2}\Phi 2 \models D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0}\right) \Rightarrow Persistency((\Phi 1_h T_{\iota 1_h})\Phi 1_0)$$

provided signatures and morphisms are arranged as in lemma 4.1.

Proof Given a non-persistent $\Phi 1_h T_{\iota 1_h}$ w.r.t. $\Phi 1_0$, there exists a model of $\Phi 1_0$ which cannot be extended to a model of $(\Phi 1_h T_{\iota 1_h})\Phi 1_0$, hence

$$\Phi 1_0 \not\models D_{\iota 1_h}(\Phi 1_h T_{\iota 1_h})\Phi 1_0$$

then if we choose $\Phi 2$ and $\Phi 1_v$ such that $\Phi 2 = (\iota_E; \iota 2)(\Phi 1_0)$ and $\Phi 1_v = \emptyset$ we get

$$D_{\iota 2}\Phi 2 \not\models D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0$$

while the premises of the rule hold trivially. □

So far, we have given an inference rule for proving refinements w.r.t. specifications with a persistent hidden enrichment. Now, we shall show that the rule is *good enough* for proving refinements w.r.t. independent specifications.

5 Proving refinements with hiding - completeness

In this section we show how the rule given in the last section is in general sufficient for proving refinements of independent specifications. Later the converse will be proven as well, confirming from a proof-theoretic standpoint that independence is not just intuitively reasonable but also very convenient.

Lemma 5.1 *Given a correct entailment*

$$D_{\iota 2}\Phi 2 \models (\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0$$

where $\Phi 1_h T_{\iota 1_h}$ *is independent w.r.t.* $\Phi 1_0$ *and* $\Phi 1_v T_{\iota 1_v}$*, then it is always the case that*

$$\Phi 2 \models_{\Sigma H} \Phi 1_0 \quad \text{and} \quad \Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v$$

provided signatures and morphisms are arranged as in lemma 4.1.

$$\begin{tikzcd}
& \Sigma H & \\
\Sigma H1 \arrow[ur, "\iota 2'_h"] & & \Sigma H2 \arrow[ul, "\iota 1'_h"'] \\
& \Sigma \arrow[ul, "\iota 1"] \arrow[ur, "\iota 2"'] & \\
\Sigma 0H1 \arrow[uu, "\iota 1_v"] & & \\
& \Sigma 0 \arrow[uu, "\iota_E"'] \arrow[ul, "\iota 1_h"] &
\end{tikzcd}$$

Proof The first entailment (1) $\Phi 2 \models_{\Sigma H} \Phi 1_0$ follows from the composition of the two following refinements with the proper extensions of signature:

$$D_{\iota 2}\Phi 2 \models D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0$$

$$D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0 \models T_{\iota_E}\Phi 1_0$$

In order to prove $\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v$, we assume the contrary (2) $\Phi 2, \Phi 1_h \not\models_{\Sigma H} \Phi 1_v$ and prove that the specification cannot be independent.

By (1) and (2) there exists a ΣH-algebra G such that

$$G \models_{\Sigma H} \Phi 1_0 \cup \Phi 2 \cup \Phi 1_h \qquad G \not\models_{\Sigma H} \Phi 1_v$$

Taking the appropriate reducts it is clear that

$$G|_{\Sigma 0H1} \in Mod[(\Phi 1_h T_{\iota 1_h})\Phi 1_0] \quad G|_{\Sigma} \in Mod[D_{\iota 2}\Phi 2]$$

$$G|_{\Sigma H1} \notin Mod[(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0]$$

and since the refinement is correct we can also conclude that

$$G|_{\Sigma} \in Mod[D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0]$$

$$\begin{tikzcd}
& G \arrow[dl, "-|_{\iota 2'_h}"'] \arrow[dr, "-|_{\iota 1'_h}"] & \\
G|_{\Sigma H1} \arrow[dr, "-|_{\iota 1}"] \arrow[dd, "-|_{\iota 1_v}"'] & & {} \arrow[dl, "-|_{\iota 2}"] \\
& G|_{\Sigma} \arrow[dd, "-|_{\iota_E}"] & \\
G|_{\Sigma 0H1} \arrow[dr, "-|_{\iota 1_h}"'] & & \\
& G|_{\Sigma 0} &
\end{tikzcd}$$

Now, taking the amalgamated union $G|_{\Sigma 0H1} \oplus G|_{\Sigma} = G|_{\Sigma H1}$ we come to a contradiction with the property of independence, since according to it $G|_{\Sigma H1}$ should be a model of $(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0$ but it is not. □

With lemmas 4.1 and 5.1 we have the basis for a sound and complete inference system for independent specifications with persistent hidden enrichments. Moreover, we give another lemma justifying the choice of independence as an *adequate* property w.r.t. the completeness of the rule, just as we gave lemma 4.2 to justify the choice of persistency w.r.t. soundness. We start with a new definition:

Definition 5.2 *A model A of a specification $D_\iota \Phi 1_0$ is* **abstract implementable** *iff there exists a presentation $\Phi 2$ such that $\Phi 2 \models D_\iota \Phi 1_0$ and $A \models \Phi 2$.*

Abstract implementability heavily depends on the expressiveness of the logical system used in specifications. In FOLEQ we cannot expect it to hold in general, but if we allow code as axioms, then all computable models are abstract implementable.

The following lemma about the adequacy of independence as a condition for completeness depends critically on the abstract implementability of one model.

Lemma 5.3 *Given finite presentations $\Phi 1_0$ over $\Sigma 0$, $\Phi 1_h$ over $\Sigma 0 H1$ and $\Phi 1_v$ over $\Sigma H1$,*

$$\left(\forall \Phi 2. \ \frac{D_{\iota 2}\Phi 2 \models D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0}{\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v \qquad \Phi 2 \models_{\Sigma H} \Phi 1_0} \right) \Rightarrow \mathit{Independence}((\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0)$$

provided signatures and morphisms are arranged as in lemma 4.1, and all models of $D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0$ are abstract implementable.

Proof Given a specification $(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0$ which is not independent, there exist A and C such that

$$C \in Mod[\Phi 1_h T_{\iota 1_h} \Phi 1_0] \qquad A \in Mod[D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0]$$

$$A|_{\Sigma 0} = C|_{\Sigma 0}$$

$$A \oplus C \notin Mod[(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0]$$

But, if A is abstract implementable there exists a presentation Φ over Σ such that

$$A \models_\Sigma \Phi \qquad \Phi \models D_{\iota 1}(\Phi 1_v T_{\iota 1_v})(\Phi 1_h T_{\iota 1_h})\Phi 1_0$$

and by the amalgamation lemma

$$A \oplus C \in Mod[(\Phi 1_h T_{\iota 1_h})\Phi]$$

hence,

$$A \oplus C \models_{\Sigma H} \Phi 1_h \cup \Phi \qquad A \oplus C \not\models_{\Sigma H} \Phi 1_v$$

Now, choosing $\Phi 2 = \Phi$ it happens that $\Phi 2, \Phi 1_h \not\models_{\Sigma H} \Phi 1_v$. □

$$\begin{tikzcd}
 & \Sigma H & \\
\Sigma H1 \arrow[ur, "\iota 2'_h"] & & \Sigma H2 \arrow[ul, "\iota 1'_h"'] \\
 & \Sigma \arrow[ul, "\iota 1"] \arrow[ur, "\iota 2"'] & \\
\Sigma 0 H1 \arrow[uu, "\iota 1_v"] & & \\
 & \Sigma 0 \arrow[uu, "\iota_E"'] \arrow[ul, "\iota 1_h"] &
\end{tikzcd}$$

The fact that A might be not abstract implementable does not matter for our purpose. We are only concerned with showing that no weaker condition than independence can be required in order to obtain completeness for the rule. Then the above lemma must be understood as saying: *Independence is the weakest condition for completeness, for an arbitrary logical system.*

6 Summary & Conclusions

The previous lemmas can be combined into a single theorem as follows:

Theorem 6.1 *Given a set of signatures and signature morphisms arranged in two pushouts as in the diagram below*

$$\begin{tikzcd}
 & \Sigma H & \\
\Sigma H1 \arrow[ur, "\iota 2'_h"] & & \Sigma H2 \arrow[ul, "\iota 1'_h"'] \\
 & \Sigma \arrow[ul, "\iota 1"'] \arrow[ur, "\iota 2"] & \\
\Sigma 0 H1 \arrow[uu, "\iota 1_v"] & & \\
 & \Sigma 0 \arrow[ul, "\iota 1_h"] \arrow[uu, "\iota_E"'] &
\end{tikzcd}$$

and variables $\Phi 1_0$ and $\Phi 2$ ranging over finite presentations over $\Sigma 0$ and $\Sigma H2$, and $\Phi 1_h$ and $\Phi 1_v$ ranging over enrichments along $\iota 1_h$ and $\iota 1_v$ respectively, then:
$\forall \Phi 1_0, \Phi 1_h.$

$$Persistency(\Phi 1_h \Phi 1_0) \Longleftrightarrow \left(\forall \Phi 1_v, \Phi 2. \ \frac{\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v \qquad \Phi 2 \models_{\Sigma H} \Phi 1_0}{D_{\iota 2} \Phi 2 \models \Phi 1_v [\Phi 1_h] \Phi 1_0} \right)$$

$\forall \Phi 1_0, \Phi 1_h, \Phi 1_v.$

$$Independence(\Phi 1_v \Phi 1_h \Phi 1_0) \Longleftrightarrow \left(\forall \Phi 2. \ \frac{D_{\iota 2} \Phi 2 \models \Phi 1_v [\Phi 1_h] \Phi 1_0}{\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v \qquad \Phi 2 \models_{\Sigma H} \Phi 1_0} \right)$$

provided all models of $\Phi 1_v [\Phi 1_h] \Phi 1_0$ are abstract implementable.

These results prompt a more restricted definition of specification where hidden parts must be explicitly defined in independent specifications by a persistent enrichment. Under these requirements, the above theorem provides a sound and complete inference system for refinements in the specification language.

Conclusions

1. All the results of the paper are valid for various logical systems. In fact, all proofs are at institution level [GB 84, Tar 86] requiring only of the existence of amalgamed union for the models [ST 88a] and of pushouts for the signatures; therefore, the results are valid for an arbitrary institution with a co-complete category of signatures and a co-continuous functor of models.

2. A sound and complete system for specifications with persistent and independent hidden enrichments can be obtained from a sound and complete inference system $\vdash$ for the underlying institution by adding the following rule:
$$\frac{\Phi 2, \Phi 1_h \vdash \Phi 1_v \qquad \Phi 2 \vdash \Phi 1_0}{D_{\iota 2}\Phi 2 \vdash \Phi 1_v[\Phi 1_h]\Phi 1_0}$$
In the context of a specification language such as ASL [SW 83, ST 88a], this kind of rule allows to overcome problems posed by the lack of an M-complete inference rule for *Derive* [Far 89] and to produce a reasonable inference system for entailment in ASL [Far 90].

3. In the case of first order logic, using hidden functions can be seen as using a second order existential quantifier. In fact, if the hidden part are only functions, the rule presented in this paper amounts to a weakening of the consequent - by importing the hidden part of the antecedent - and the application of the introduction rule for second order existential quantifiers.

   From there we can conclude that a allowing an arbitrary weakening we get a more general rule such as:
$$\frac{\Phi T_{\iota 1'_h}\Phi 2 \models T_{\iota 2'_h}\Phi 1}{D_{\iota 2}\Phi 2 \models D_{\iota 1}\Phi 1}$$
   where $\Phi T_{\iota 1'_h}$ is a persistent enrichment w.r.t. $\Phi 2$.

   The main difference with our proposal is that such a rule makes use of a new enrichment $\Phi T_{\iota 1'_h}$ which can be anything, therefore it cannot be automatized. In our case Φ is chosen to be an enrichment with the axioms of $\Phi 1_h$, that is, this paper considers the cases where Φ can be obtained from the specification by automatic means.

4. In cases such as the example given in section 1.1, our intuition matches the inference rule given. Nevertheless, the sentences for specifying structures such as natural numbers or lists are not just first order sentences but also some data constraints, that means that the logic used has no complete inference system.

   In these cases it is important to note that incompleteness arises in the underlying institution and not in the hiding.

Acknowledgements

I would like to thank those persons who read early drafts of the paper in the University of Edinburgh, those who listened to me at the first COMPASS workshop at Bremen, A.Tarlecki for very helpful comments and, particularly, Don Sannella for helping and correcting me for so long.

References

[Bre 89] R.Breu. A normal form for structured algebraic specifications. Internal report MIP-8917, Universität Passau, 1989.

[BBTW 81] J.A.Bergstra, et al. On the power of algebraic specifications. Proc. 10th Symp. on Mathematical Foundations of Computer Science. LNCS 118, Springer 1981, p. 193-204.

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

[EM 85] H.Ehrig, B.Mahr. Fundamentals of algebraic specification 1: Equations and initial semantics. Springer, 1985.

[Far 89] J.Farrés-Casals. Proving correctness of constructor implementations. Proc. 14th Symp. on Mathematical Foundations of Computer Science, Porabka-Kozubnik. LNCS 379, p. 225-235, 1989. Extended version in LFCS Report Series 89-72, University of Edinburgh, 1989.

[Far 90] J.Farrés-Casals. Verification in ASL and related specification languages. Draft of forthcoming Ph.D. thesis, University of Edinburgh.

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

[Maj 77] M.E.Majster. Limits of the algebraic specifications of abstract data types. ACM-Sigplan Notices 12 (1977), p.37-42.

[ST 88a] D.Sannella, A.Tarlecki. Specifications in an arbitrary institution. Information and Computation 76 (1988), p. 165-210.

[ST 88b] D.Sannella, A.Tarlecki. Towards formal development of programs from algebraic specifications: Implementations revisited. Acta Informatica 25 (1988), p. 233-281.

[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, 1983.

[Tar 86] A.Tarlecki. Bits and pieces of the theory of institutions. Proc. Intl. Workshop on Category Theory and Computer Programming, Guildford 1985, eds. D. Pitt, S. Abramsky, A. Poigné and D. Rydeheard, Springer LNCS 240, p.334-363, 1986.

[TWW 79] J.W.Thatcher, E.G.Wagner, J.B.Wright. Data type specification: Parameterization and the power of specification techniques. In SIGACT 10th Annual Symp. on the Theory of Computation, 1979. Also in, ACM TOPLAS 4, p. 711-732, 1982.