PARTIAL COMPOSITION AND RECURSION OF MODULE SPECIFICATIONS

Francesco Parisi-Presicce
Department of Mathematics
University of Southern California
Los Angeles, California 90089-1113

ABSTRACT

The basic interconnections of module specifications (union, composition and actualization) were studied in earlier papers. Here we introduce partial composition and partial actualization of module specifications, describe the connection with their "total" counterpart and prove that the result of successive partial compositions (or actualizations) is independent of the order. We also introduce a recursive construction first of a single module and then of two modules "recursively calling" each other. A connection between these two recursions is established, along with compatibility properties with the basic constructions and the expected fixed point equation at the semantical level.

1. INTRODUCTION

The algebraic approach to the formal specification of data types has been the most investigated one (/LZ 75/, /GTW 78/, /WPPDB 83/, /Ga 83/ and many others), although there have been other interesting approaches in more general settings (/BMM 79/).
The module specification introduced in earlier papers (/EW 85/, /BEPP 86/) is a formalization of a notion which is central to the modular approach to the development of large software systems (/Pa 72/, /WE 85/). It combines the main ideas of parametrized specification and of implementation of abstract data types along with the notion of information hiding, treated in /GM 82/ by adding an export interface to a data type to represent its visible part.

An abstract module consists of four parts: an export interface, with the operations visible outside the module, an import interface, representing the operations to be provided to the module, a parameter part, shared by the interfaces, and a body, containing both interfaces and providing an implementation of the sorts and operations of the export interface in terms of those of the import. All four parts are described by algebraic specifications (see /EWT 83/, /EFPB 86/ for extensions), the first three with loose semantics, while that of the body is the free construction over import algebras. Our notion of module reflects in part the structure of Ada packages and Modula-2 modules, both consisting of a "declarative" part, with the list of sorts and operations visible outside and either the list of those to be imported (Modula-2) or the name of another module whose export operations are needed in the body (Ada), and an "implementation" part, with the module's own data type and defined operations. In both languages, the interfaces are purely syntactical, not allowing semantical conditions on the sorts and operations, as we do or as permitted in OBJ2 (/FGJM 85/) and in Extended ML (/ST 85a/). More detailed discussions on the relationship between our module concept and Ada and Modula-2 can be found in /BEFP 86/, /EW 86/.

The interconnection mechanisms for building modules from other modules are an integral part of a stepwise modular development of software systems (/BG 77/). In previous papers, we have introduced four basic operations on module specifications: union (/BPP 85/), composition (/EW 85/, /BEPP 86/), actualization (/EW 85/, /PP 85/) and extension (/BEPP 86/). We have also shown that these operations are compatible (/PP 86/, /EFP 86/), guaranteeing that the correctness of a stepwise refinement strategy for module specifications is independent of the order in which the building operations are carried out.

In this paper, we introduce two new operations on module specifications: partial composition/actualization, and recursion. Partial composition allows us to compose a module, whose import consists of two distinguishable parts I1 and I2, with another module which provides the data described, say, in I1, postponing the decision for I2 to a later time. This operation is proved to be well defined both syntactically and semantically and to produce the same result regardless of whether the successive partial compositions are carried out first through I1 and then through I2 or vice versa. Similar results are obtained for partial actualization. The single recursion construction defines a new module $rec_f(M)$ from a given M, when the import of M is intended to be provided by the export of M itself. The mutual recursion operation provides a new module from two module specifications "recursively calling each other". The two constructions are syntactically correct but, unlike the other operations, additional conditions are required to guarantee their semantical correctness. Single recursion is shown to be compatible with union, composition, actualization and the "submodule" partial order.

The paper is organized as follows: section 2 contains a review of the basic notions of module specification, its semantics and basic operations, along with a summary of their compatibility. Section 3 introduces partial composition and partial actualization, relates them to their total counterpart and shows how successive partial compositions (or actualizations) are equivalent to a union followed by a total composition (or actualization). Section 4 defines single recursion using coequalizers, and shows how its semantic satisfies a fixed point equation. Mutual recursion, defined independently, is shown to be related in a natural way to single recursion. Some conclusions are drawn in section 5.

2. MODULE SPECIFICATION AND THEIR BASIC OPERATIONS

We assume some familiarity with the basic notions of algebraic specification SPEC = (S, OP, E) and of specification morphism $f = (f_s, f_{op})$: SPEC1 → SPEC2. We use Alg(SPEC) to denote the category of SPEC-algebras and SPEC-homomorphisms. Any specification morphism f : SPEC1 → SPEC2 defines a forgetful functor V_f : Alg(SPEC2) → Alg(SPEC1) whose left adjoint F_f : Alg(SPEC1) → Alg(SPEC2) is called the free functor associated with f.

The category CATSPEC of specifications and specification morphisms is closed under the pushout construction (/EM85/) and $\mathrm{SPEC1} +_{\mathrm{SPEC0}} \mathrm{SPEC2}$ denotes the pushout object of fj : SPEC0 → SPECj, j = 1,2, when the specification morphisms are obvious from the context. For any pushout $\mathrm{SPEC3} = \mathrm{SPEC1} +_{\mathrm{SPEC0}} \mathrm{SPEC2}$, any SPEC3-algebra A3 (resp. SPEC3-homomorphism h3) is the amalgamated sum $\mathrm{A1} +_{\mathrm{A0}} \mathrm{A2}$ (resp., $\mathrm{h1} +_{\mathrm{h0}} \mathrm{h2}$) of SPECi-algebras Ai (resp., SPECi-homomorphisms hi). Given pushout specifications SPEC3 and SPEC3' and functors Fi : Alg(SPECi) → Alg(SPECi'), i = 0,1,2, we use $\mathrm{F1} +_{\mathrm{F0}} \mathrm{F2}$ to denote the functor F3: Alg(SPEC3) → Alg(SPEC3') defined by $\mathrm{F3}(\mathrm{A1} +_{\mathrm{A0}} \mathrm{A2}) = \mathrm{F1}(\mathrm{A1}) +_{\mathrm{F0(A0)}} \mathrm{F2}(\mathrm{A2})$. For more details, see /EM85/.

2.1 Definition (Module Specification and Semantics)

A module specification M is a four-tuple (PAR, EXP, IMP, BOD) of algebraic specifications along with four specification morphisms i, v, s and e (s and e injective) making the following syntactical diagram commute

$$\begin{tikzcd}
\mathrm{PAR} \arrow[r, "e"] \arrow[d, "i"'] & \mathrm{EXP} \arrow[d, "v"] \\
\mathrm{IMP} \arrow[r, "s"] & \mathrm{BOD}
\end{tikzcd}$$

The semantics of M is SEM = $V_v \cdot F_s$.
The restricted semantics is RSEM = $R_e \cdot$ SEM, where R_e : Alg(EXP) → Alg(EXP) is given by $R_e(E) = \cap\ \{E' \in \mathrm{Alg(EXP)}:\ E' \subset E,\ V_e(E') = V_e(E)\}$.

A semantical condition is imposed on the free functor F_s, which is required to be strongly persistent, i.e. that $V_s\ (F_s(A)) = A$ for all IMP-algebras A. Sometimes (in particular when dealing with composition of module specifications) we will add the requirement that F_s preserves injective morphisms and call it, in this case, strongly conservative.

Interpretation The specifications IMP and EXP represent the import and export interfaces, respectively, PAR is the shared parameter part and BOD is the body of the module intented to contain an implementation of the EXP operations using the IMP operations. The semantics SEM is a transformation from IMP-interface algebras to EXP-interface algebras and the strong persistency guarantees that the PAR part of the IMP-algebra is not modified by this transformation. The restriction functor R_e reduces the carrier of the EXP-algebra SEM(A) to those data reachable from its parameter part.

2.2 Definition (Submodule and Union)

A module specification M0 = (PAR0, EXP0, IMP0, BOD0) is a submodule specification of M1 = (PAR1,EXP1, IMP1, BOD1) if there exists a four-tuple m = (m_P, m_E, m_I, m_B) of injective specification morphisms such that

i) each square of the following diagram commutes

$$\begin{tikzcd}
PAR0 \arrow[r, "e0"] \arrow[d, "m_P"'] & EXP0 \arrow[r, "v0"] \arrow[d, "m_E"'] & BOD0 \arrow[d, "m_B"'] & IMP0 \arrow[l, "s0"'] \arrow[d, "m_I"] & PAR0 \arrow[l, "i0"'] \arrow[d, "m_P"] \\
PAR1 \arrow[r, "e1"'] & EXP1 \arrow[r, "v1"'] & BOD1 & IMP1 \arrow[l, "s1"] & PAR1 \arrow[l, "i1"]
\end{tikzcd}$$

ii) if V_Q is the forgetful functor associated with m_Q, then $V_B \cdot F_{s1} = F_{s0} \cdot V_I$ and $V_E \cdot R_{e1} = R_{e0} \cdot V_E$.

We write $M0 \leq_m M1$ and call $m : M0 \rightarrow M1$ a Mod-morphism. Given $M0 \leq_{mj} Mj$, $j = 1,2$, the union $M1 +_{M0} M2$ of M1 and M2 with respect to M0 is the pushout object of the morphisms $mj: M0 \rightarrow Mj$ in the category of module specifications and Mod-morphisms (/PP86/). Each component of $M1 +_{M0} M2$ is the pushout specification of the corresponding components of M0, M1 and M2.

The operation of actualization of a module specification M= (PAR, EXP, IMP, BOD) consists of "replacing" the parameter part PAR by a (parametrized) specification PS1 = (Par1, ACT1) with j: Par1 → ACT1 via a parameter passing morphism h: PAR → ACT1.

2.3 Definition (Actualization)

Given a module specification M = (PAR, EXP, IMP, BOD), a parametrized specification PS1 = (Par1, ACT1) and a parameter passing specification morphism h: PAR → ACT1, the actualization of M by PS1 via h, denoted by act_h (PS1, M), is the module specification (Par1, EXP1, IMP1, BOD1), where $EXP1 = ACT1 +_{PAR} EXP$, $IMP1 = ACT1 +_{PAR} IMP$ and $BOD1 = IMP1 +_{IMP} BOD$ as in the following diagram

$$\begin{tikzcd}
 & PAR \arrow[rr] \arrow[dr] \arrow[d] & & EXP \arrow[dr] \arrow[d] & \\
Par1 \arrow[rr] & IMP & ACT1 \arrow[rr] \arrow[dd] & BOD & EXP1 \arrow[dd] \\
 & & & & \\
 & & IMP1 \arrow[rr] & & BOD1
\end{tikzcd}$$

$$\begin{tikzcd}
 & PAR \arrow[rr] \arrow[dr] \arrow[dd] & & EXP \arrow[dr] \arrow[dd] & \\
Par1 \arrow[rr] & & ACT1 \arrow[rr] \arrow[dd] & & EXP1 \arrow[dd] \\
 & IMP \arrow[rr] \arrow[dr] & & BOD \arrow[dr] & \\
 & & IMP1 \arrow[rr] & & BOD1
\end{tikzcd}$$

The third basic operation on module specifications is that of composition, where the import interface of a module specification is "matched" with the export interface of another one. The "unused" interfaces will provide two of the components of the composite module specification.

2.4 Definition (Composition)

The composition $M1 \bullet_h M2$ of two module specifications Mj = (PARj, EXPj, IMPj, BODj), j = 1,2, with interface morphism $h = (h_P, h_E)$, where h_P: PAR1 → PAR2 and h_E: IMP1 → EXP2 are specification morphisms such that $e2 \cdot h_P = h_E \cdot i1$, is the module specification M3 = (PAR3, EXP3, IMP3, BOD3) as in the diagram

$$\begin{tikzcd}
 & PAR3 \arrow[r, phantom, "="] & PAR1 \arrow[r] \arrow[d] \arrow[ddl, "h_P"'] & EXP1 \arrow[r, phantom, "="] \arrow[d] & EXP3 \\
 & & IMP1 \arrow[r] \arrow[d, "h_E"] & BOD1 \arrow[dd] & \\
 & PAR2 \arrow[r] \arrow[d] & EXP2 \arrow[d] \arrow[r, phantom, "(1)"] & & \\
IMP3 \arrow[r, phantom, "="] & IMP2 \arrow[r] & BOD2 \arrow[r] & BOD3 &
\end{tikzcd}$$

where (1) is a pushout in CATSPEC.

For each of the three operations, the semantics of the resulting module specification can be expressed directly in terms of those of the arguments. The semantics and restricted semantics of $M1 +_{M0} M2$ are $SEM1 +_{SEM0} SEM2$ and $RSEM1 +_{RSEM0} RSEM2$, respectively (/BPP85/). The semantics of act_h (PS1, M) is $id_A +_{id_P} SEM$, with id the appropriate identity functors, and a similar characterization of the restricted semantics holds if either h factors through Par1l(/PP85/) or the semantics of PS1 is taken into account (/PP86/, /EFPB86/). Denoting by V_h the forgetful functor of h_E, the semantics of $M1 \cdot_h M2$ is $SEM1 \cdot V_h \cdot SEM2$ while the restricted semantics is $RSEM1 \cdot V_h \cdot RSEM2$, if F_{s1} is strongly conservative (/EW85/, /BEPP86/).

The compatibility of these operations on module specifications is necessary to guarantee that the order in which these operations are applied does not effect the final system. This allows the restructuring of large systems foi reasons of efficiency and gives more flexibility in updating specifications due to changes in system requirements. The interaction of these module interconnections has been studied elsewhere (/EW85/, /BEPP86/, /PP86/, /EFP86 /).

2.5 Theorem (Compatibility of the Basic Operations)

1) The operations of union, actualization and composition are monotone in each of their arguments with respect to the "submodule" partial order.

2) For i = 0, 1, 2, let Mi = (PARi, EXPi, IMPi, BODi) be module specifications, PSi = (Pari, ACTi) with ji: Pari → ACTi parametrized specifications and hi: PARi → ACTi parameter passing morphisms. If $M0 \leq_{mi} Mi$ and (qi_P, qi_A): PS0 → PSi are such that $ji \cdot qi_P = qi_A \cdot j0$ and $qi_A \cdot h0 = hi \cdot mi_P$, then

$$act_{h1 +_{h0} h2}(PS1 +_{PS0} PS2, M1 +_{M0} M2) = act_{h1}(PS1, M1) +_{act_{h0}(PS0,M0)} act_{h2}(PS2,M2)$$

3) For i = 0, 1,2, let Mi = (PARi, EXPi, IMPi, BODi) and Ni = (PARi', EXPi', IMPi', BODi') be module specifications and $hi = (hi_P, hi_E)$ interface morphisms from Mi to Ni. If $M0 \leq_{mi} Mi$ and $N0 \leq_{ni} Ni$ and $ni_E \cdot h0_E = hi_E \cdot mi_I$, then

$$(M1 +_{M0} M2) \bullet_{h1 +_{h0} h2} (N1 +_{N0} N2) = (M1 \bullet_{h1} N1) +_{(M0 \bullet_{h0} N0)} (M2 \bullet_{h2} N2).$$

4) Let M1 and M2 be module specifications with an interface morphism $h = (h_P, h_E)$ from M1 to M2 and PS1 = (Par1, ACT1) a parametrized specification with a parameter passing morphism h1: PAR1 → ACT1. Then there exist PS2 =(Par1, ACT2) and h2: PAR2 → ACT2 such that

$$act_{h1}(PS1, M1 \bullet_h M2) = act_{h1}(PS1, M1) \bullet_{h+_{id}id} act_{h2}(PS2, M2).$$

3. PARTIAL COMPOSITION AND ACTUALIZATION

In this section, we are going to investigate two somewhat different ways of combining module specifications. Suppose we have a module specification M, whose import interface can be decomposed as the union $IMP1 +_{IMP0} IMP2$ of two subspecifications sharing a common part IMP0, and another module specification M1 whose export interface provides the operations described in IMP1. Given such a "matching", is it possible to compose the two modules now, postponing the matching of the remaining part IMP2 of the import interface? Under what conditions is such a composition well defined and how does it relate to the composition defined in the previous section?

3.1 Definition (Partial Composition)
Let M = (PAR, EXP, IMP, BOD) be a module specification with $PAR = PAR1 +_{PAR0} PAR2$ and $IMP = IMP1 +_{IMP0} IMP2$, M' = (PAR', EXP', IMP', BOD') another module specification and $h1 = (h1_P, h1_E)$ an interface morphism with $h1_P$: PAR1 → PAR' and $h1_E$: IMP1 → EXP' satisfying $e' \cdot h1_P = h1_E \cdot i1$. If there exists a specification morphism k: IMP0 → IMP' such that IMP0 → IMP' → BOD' = IMP0 → IMP1 → EXP' → BOD', then the <u>partial composition</u> $M \bullet^{p}_{h1} M'$ <u>of M and M' w.r.t. h1</u> is the module specification (PAR, EXP, $IMP' +_{IMP0} IMP2$, $BOD' +_{IMP1} BOD$) as in the following diagram

$$\begin{tikzcd}
 & & PAR0 \arrow[dl] \arrow[dr] & & \\
 & PAR1 \arrow[dr] \arrow[ddd, "i1"] \arrow[dddddl, "h1_P"'] & & PAR2 \arrow[dl] & \\
 & & PAR \arrow[rr] & & EXP \arrow[ddd] \\
 & & IMP0 \arrow[dl] \arrow[dr] \arrow[ddddll] & & \\
 & IMP1 \arrow[dr] \arrow[dd, "h1_E"] & & IMP2 \arrow[dl] \arrow[dddd] & \\
 & & IMP \arrow[rr] & & BOD \arrow[dd] \\
PAR' \arrow[r] \arrow[d] & EXP' \arrow[d] & & & \\
IMP' \arrow[r] \arrow[drrr] & BOD' \arrow[rrr] & & & BOD'' \\
 & & & IMP'' &
\end{tikzcd}$$

The condition on h1 is exactly the one required in the (full) composition of Definition 2.4.

The existence and property of the morphism k state that the two "subimports" IMP1 and IMP2 can share only a specification which is preserved basically unchanged, from the import to the export of M'. The pushout property of IMP" guarantees the existence of a specification morphism s" : IMP" $\rightarrow$ BOD" while the universal property of PAR defines i": PAR $\rightarrow$ IMP". This universal property also guarantees the commutativity v"$\cdot$ e = PAR $\rightarrow$ EXP $\rightarrow$ BOD" = PAR $\rightarrow$ IMP $\rightarrow$ BOD" = s" $\cdot$ i" of the diagram of $M \bullet^{p}_{h1} M'$ by proving that PARj $\rightarrow$ PAR $\rightarrow$ EXP $\rightarrow$ BOD" = PARj $\rightarrow$ PAR $\rightarrow$ IMP" $\rightarrow$ BOD" for j = 1,2,.
The existence of the morphism k is necessary (in the basic algebraic case treated here) to insure a consistent handling of the shared subimport IMP0. The requirement on k could be dropped by allowing constraints along with the basic specifications (see/EFPP86/), thereby restricting subsequent compositions to module specifications not in conflict with the already matched IMP0.

Instead of proving that the resulting module specification satisfies the semantical conditions of definition 2.1, we now show how to relate partial composition with the operations defined in the previous section. The basic idea is that leaving IMP2 unchanged is equivalent to composing it with a module specification which behaves like the identity.

3.2 Main Lemma

Let M, M', h1 and k be as in definition 3.1, MIj = (PARj, IMPj, IMPj, IMPj) and $h = h1 +_{id} id$. Then

$$M \bullet^{p}_{h1} M' = M \bullet_{h} (M' +_{MI0} MI2)$$

The first immediate consequence of the Main Lemma is that partial composition is well defined, that is, that the resulting four-tuples of specifications and specification morphisms satisfy the conditions in 2.1.

3.3 Theorem

Let Mj = (PARj, EXPj, IMPj, BODj), j = 3,4, and M = (PAR, EXP, IMP, BOD) be module specifications with $PAR = PAR1 +_{PAR0} PAR2$ and $IMP = IMP1 +_{IMP0} IMP2$. For j = 3,4, let hj be an interface morphism and kj: IMP0 $\rightarrow$ IMPj a specification morphism such that the partial composition of M and Mj is defined as in 3.1. Then

$$M \bullet_{h3 +_{id} h4} (M3 +_{MI0} M4) = (M \bullet^{p}_{h3} M3) \bullet^{p}_{h4} M4.$$

3.4 Corollary

With the notation of the previous Theorem, $(M \bullet^{p}_{h3} M3) \bullet^{p}_{h4} M4 = (M \bullet^{p}_{h4} M4) \bullet^{p}_{h3} M3.$

We should point out that there are some compatibility properties enjoyed by partial composition. Their formulation and proofs can be reconstructed in a straigthforward manner using Thm. 2.5 and Lemma 3.2.

For the remaining part of this section, we investigate the analog of partial composition for actualization. If we are given a module specification whose parameter part is the union of two subspecifications, we can actualize only one of the subspecifications, postponing the choice of the remaining parameter part. The results parallel those of partial composition, including the relationship between partial and "total" actualization and the effect of successive partial actualizations.

3.5 Definition (Partial Actualization)

Let M = (PAR, EXP, IMP,BOD) be a module specification with $PAR = PAR1 +_{PAR0} PAR2$, PS1 = (Parl,ACT1) a parametrized specification with f1: Parl → ACT1 and $h1_A$: PAR1 → ACT1 a parameter passing specification morphism. If there exists a specification morphism $h1_P$: PAR0 → Parl such that PAR0 → Parl → ACT1 = PAR0 → PAR1 → ACT1, then the partial actualization $pact_{h1}$ (PS1, M) of M by PS1 w.r.t. h1 is the module specification (PAR', EXP', IMP', BOD') where, in the diagram

$$\begin{tikzcd}
 & PAR0 \arrow[r] \arrow[d] \arrow[ddl, "h1_P"'] & PAR2 \arrow[d] & \\
 & PAR1 \arrow[r] \arrow[d, "h1_A"] & PAR \arrow[r] \arrow[d] & EXP \arrow[d] \\
Parl \arrow[r, "f1"] & ACT1 & IMP \arrow[r] & BOD
\end{tikzcd}$$

$IMP' = IMP +_{PAR1} ACT1$, $\quad$ $EXP' = EXP +_{PAR1} ACT1$

$PAR' = PAR2 +_{PAR0} Parl$, $\quad$ $BOD' = BOD +_{IMP} IMP'$

The new import interface is obtained by replacing PAR1 in IMP with ACT1; similarly for the new export interface. The new parameter is the union of the untouched subparameter PAR2 with the parameter Parl introduced by the actualization with PS1. Their shared part PAR0 is not duplicated and could represent basic standard specifications, such as bool and nat, which are going to be shared by every specification and to be left unchanged by every module in the system.

The universal property of PAR' induces a unique specification morphism i': PAR ' → IMP', compatible with the existing morphisms. Similarly, we can obtain a unique e': PAR' → EXP' and it can be shown, using the uniqueness of the induced morphism from PAR' to BOD', that PAR' → IMP' → BOD' = PAR' → EXP' → BOD'. As was the case for partial composition, we can relate partial actualization to (total) actualization, thereby inferring the semantical correctness of the construction above.

3.6 Lemma

Let M, PS1 and h1 = ($h1_P$, $h1_A$) be as in Definitiion 3.5 and, abusing the notation, let PARj be the parametrized specification (PARj, PARj) with id: PARj → PARj, j = 0,2. Then

$$pact_{h1}(PS1, M) = act_{h1_A + _{id} id} (PS1 +_{PAR0} PAR2, M)$$

Since the free construction of the (totally) actualized module specification is strongly persistent or conservative if the original free construction is, partial actualization is semantically correct in view of the above Lemma. The next result shows that successive partial actualizations by PS1 and PS2 yields the same module as the total actualization by PS1 $+_{PAR0}$PS2. As a corollary, we obtain the commutativity of repeated partial actualization.

3.7 Theorem

Let M = (PAR, EXP, IMP, BOD) be a module specification with PAR = PAR1 $+_{PAR0}$PAR2, and, for j = 1,2, PSj = (Parj, ACTj) a parametrized specification and hj = (hj_P, hj_A) a parameter passing morphism such that the partial actualization of M by PSj w.r.t. hj is defined. Then

$$pact_{h2}\,(PS2, pact_{h1}(PS1,M)) = act_{\,h1+_{id}h2}(PS1 +_{PAR0} PS2, M).$$

4. RECURSION OF MODULE SPECIFICATIONS

The operations of union, actualization and composition are the basic mechanisms to build module specifications from other module specifications. The construction which we are going to introduce next is be motivated by looking at the interface morphism f in the composition $M \bullet_f M'$ as a "call" of the export of M' by M. A "recursive call" is then represented by an interface morphism from (part of) the import of a module specification M to the export of the same M, in such a way that the parameter part is left unchanged. The effect of such a recursive call should leave the parameter part and the export interface unchanged, remove the "domain" of the recursive call f from the import interface and identify within the body each operation *op* of IMP with its counterpart f(*op*) of EXP. To make this informal discussion precise, we need to review the notion of coequalizer (/HS 73/).

Given two morphisms f,g: A → B in a category CAT, the coequalizer of f and g, denoted by Coeq(f, g), is a pair (C, k), with k: B → C a morphism in CAT, such that $k \cdot f = k \cdot g$ and for any m: B → D in CAT satisfying $m \cdot f = m \cdot g$, there is a unique morphism n : C → D in CAT such that $m = n \cdot k$. (We will at times abuse the terminology and refer to the object part C as the coequalizer of f and g).

Coequalizers are unique up to isomorphism and k is always an epimorphism (/HS 73/). In the category of sets, C is the set of equivalence classes of B generated by the pairs (f(a), g(a)) for $a \in A$ and k is the canonical projection sending each element of B into its equivalence class. If f=g, then Coeq(f,g) = (B, id_B). It is not too hard to show that any pair of specification morphisms f,g: SPEC1 → SPEC2 has a coequalizer in CATSPEC. First construct S3 and OP3 in the category of sets and then use k = (k_S, k_{op}) to "translate" E2 into E3. Given a SPEC2 - algebra A satisfying $V_f(A) = V_g(A)$, there is always a SPEC3 - algebra B such that $V_k(B) = A$. The construction of B is similar to that of amalgamation (/BPP 85/, /EM 85/) and can be summarized as follows. Since k is an epimorphism, for any $s \in S3$, there is $s' \in S2$ such that $s = k(s')$: define $B_s = A_{s'}$. There is no ambiguity in the definition, since if we also have $s = k(s'')$ for some $s'' \in S2$, then there exists, by definition of coequalizer, $s1 \in S1$ such that $f(s1)=s'$ and $g(s1)=s''$. But then, by assumption, $A_{s'}= (V_f(A))_{s1}=(V_g(A))_{s1}=A_{s''}$. Similarly for $op \in OP3$. Such an algebra B is unique, since V_k:Alg(SPEC3)→Alg(SPEC2) is the equalizer of V_f and V_g and thus a monomorphism.

We are now ready to define recursion over a single module specification. Later in this section we will also discuss the case of two modules "recursively calling" each other.

4.1 Definition (Single Recursion)

Given a module specification $M=(PAR1+_{PAR0}PAR2, EXP, IMP1+_{IMP0}IMP2, BOD)$ and a specification morphism $f: IMP2 \rightarrow EXP$ such that

(a) $PAR2 \rightarrow IMP2 \rightarrow EXP = PAR2 \rightarrow PAR1+{}_{PAR0}PAR2 \rightarrow EXP$ and

(b) $IMP0 \rightarrow IMP2 \rightarrow EXP \rightarrow BOD = IMP0 \rightarrow IMP1+_{IMP0}IMP2 \rightarrow BOD$,

the recursion of M over f, denoted by $rec_f(M)$, is the module specification $M1= (PAR1+_{PAR0}PAR2, EXP, IMP1+_{PAR0}PAR2, BOD1)$ as in the following diagram

$$\begin{tikzcd}
PAR1 + {}_{PAR0}PAR2 \arrow[rr, "e"] \arrow[d, "i1 +_{id}id"'] & & EXP \arrow[d, "v"'] \arrow[dr, "v'"] & \\
IMP1 +_{PAR0}PAR2 \arrow[r, "id +_{i0}i2"'] \arrow[rrr, bend right, "s'"'] & IMP1 +_{IMP0}IMP2 \arrow[r, "s"'] & BOD \arrow[r, "k"'] & BOD1
\end{tikzcd}$$

where $(BOD1,k) = Coeq(v\cdot f, s\cdot j2)$, with $j2: IMP2 \rightarrow IMP1+_{IMP0}IMP2$ the canonical inclusion. The conditions (a) and (b) are similar to those imposed in the definition of partial composition.

It is easy to show that the diagram commutes and that s' is again injective. If f is such that $IMP2 \rightarrow EXP \rightarrow BOD=IMP2 \rightarrow IMP1 + {}_{IMP0}IMP2 \rightarrow BOD$, then the only effect of the construction is to remove from the import interface the part of IMP2 not in PAR2. This agrees with our definition, since if $v\cdot f=s\cdot j2$, then $k=id_{BOD}$.

The semantical conditions on the new module specification need not be satisfied in general as the following example shows. For simplicity, it is understood that all specifications contain BOOL with the appropriate if-then-else operator. Let $M=(PAR, EXP, IMP1 +_{PAR}IMP2, BOD)$ be given by the following

```
PAR =    sort   nat
         opns   0:  → nat
                SUCC:   nat → nat

IMP1 = PAR +   opn  + : nat nat → nat
                eqn  0 + x = x
                     SUCC(x) + y = SUCC(x+y)

IMP2 = PAR +   opn g: nat → nat

EXP = PAR +    opn G : nat → nat
```

BOD = IMP ∪ EXP +   **eqn** G(x) = if x = 0 then SUCC(0) else g(SUCC (x))

Let f: IMP2 → EXP be the identity on PAR and $f_{op}(g) = G$. Then
$rec_f(M)$= (PAR, EXP, IMP1, BOD1) where
BOD1=IMP1 ∪ EXP +
eqn G(x) = if x = 0 then SUCC(0) else G(SUCC(x))

and the specification morphisms are the obvious inclusions. Then the free functor associated with s':IMP1→BOD1 is not strongly persistent. The problem is similar to that of termination of recursively defined functions (completeness). From the point of view of Term Rewriting Systems (/PD85/), persistency of the free functor $F_{s'}$ is equivalent to proving confluency and termination of the expanded TRS obtained by adding (in the above example) the equation G(x)=g(x) and removing terms containing g from the set of normal forms.

The following theorem establishes the correspondence between the semantics of M and that of $rec_f(M)$. For simplicity, we restrict our attention to the case where PAR1=PAR0 and IMP1=IMP0 in def. 4.1.

4.2 Theorem (Fixed Point Property)
Let M=(PAR, EXP, IMP, BOD) , f:IMP→EXP a specification morphism satisfying $f \cdot i = e$ and M1=$rec_f(M)$=(PAR, EXP, PAR, BOD1).

a) If F_i and F_k are strongly persistent, then the free functor $F_{s'}$ is strongly persistent
b) If F_k is strongly persistent, then $SEM \cdot V_f \cdot SEM1 = SEM1$
c) If, in addition, F_s is strongly conservative, then $RSEM \cdot V_f \cdot RSEM1 = RSEM1$

With one additional assumption , the recursive construction preserves the submodule partial order.

4.3 Theorem (Submodule Compatibility)
Let M = (PAR, EXP, IMP, BOD), M'= (PAR', EXP', IMP', BOD'), $M \leq_m M'$, f: IMP → EXP and f': IMP' → EXP' be such that $m_E \cdot f = f' \cdot m_I$. If F_k and $F_{k'}$ are strongly persistent and $V_I \cdot F_{i'} = F_i \cdot V_{P'}$ then $rec_f(M) \leq rec_f(M')$.

We can show that the operation of single recursion is compatible with union, actualization and composition. All three compatibilities are based on different interpretations of the following Lemma.

4.4 Lemma
Let hj : A0 → Aj, kj : B0 → Bj, j=1,2, and mj, nj : Aj → Bj, j = 0, 1, 2, be such that kj · n0 = nj · hj and kj · m0 = mj · hj, j = 1,2. Then

$$Coeq(m1+_{m0}m2, n1+_{n0}n2) = Coeq(m1,n1)+_{Coeq(m0, n0)}Coeq(m2,n2).$$

We now state the compatibility of single recursion with the three basic operations. For simplicity, for union and actualization we restrict our attention to the case considered in Theorems 4.2 and 4.3.

4.5 __**Theorem**__ (Compatibility with Union)

Let M0 ≤ Mj, j = 1,2 and fj: IMPj → EXPj, j = 0,1,2, compatible specification morphisms which are the identity on PARj. Then

$$rec_{f1\ +_{f0}\ f2}\ (M1 + _{M0}M2) = rec_{f1}(M1) +_{rec_{f0}(M0)} rec_{f2}(M2).$$

4.6 __**Theorem**__ (Compatibility with Actualization)

Let M = (PAR, EXP, IMP, BOD), f:IMP → EXP a specification morphism with f • i = e, PS = (Par, ACT) a parametrized specification and h:PAR → ACT a parameter passing morphism. Then $act_h\ (PS, rec_f\ (M)) = rec_{\ f\ +_{\ id} id}\ (act_h\ (PS, M))$.

The third compatibility property we have investigated is that with partial composition, in which the IMP2 part of the import interface is matched by f with the export of the same module while the IMP1 part is matched via h with the export interface of another module. The next result states that the order in which these two operations are carried out is immaterial.

4.7 __**Theorem**__ (Compatibility with Partial Composition)

Let M = (PAR1+$_{PAR0}$PAR2, EXP, IMP1+$_{IMP0}$IMP2, BOD) be a module specification and f : IMP2 → EXP a specification morphism such that rec$_f$(M) is defined. Let M' = (PAR',EXP', IMP', BOD') be another module specification and h=(h$_P$,h$_E$) an interface morphism satisfying the conditions in definition 3.1. Then

$$rec_f(M \bullet_h^p M') = rec_f(M) \bullet_h^p M'$$

In Theorem 4.5, the single recursion over a union is the union of the corresponding single recursions __provided__ that each specification morphism fj is within the same module. We want to consider next the case of two module specifications recursively "calling" each other as allowed, for example, in Ada (/B184/). Suppose that we have two module specifications M1 and M2 with imports IMP1+$_{IMP0}$IMP1' and IMP2+$_{IMP0}$IMP2', and exports EXP1 and EXP2, respectively, and we want to define a new module consisting of M1 and M2 only, where M2 provides the import IMP1' of M1 and M1 the IMP2' part of the import of M2. In the new module, IMP1' and IMP2' are no longer in the import interface, while IMP1 and IMP2 are. The new body should contain not only the bodies of M1 and M2, but also the information that part of the import of M1 uses M2 and viceversa. Let us make this precise.

4.8 __**Definition**__ (Mutual recursion)

Let Mj = (PARj+$_{PAR0}$PAR0, EXPj, IMPj+$_{IMP0}$IMPj', BODj), j=1,2, be module specifications and let f1 : IMP1' → EXP2 and f2 : IMP2' → EXP1 be specification morphisms such that

(a) IMP0 → IMP1'→ EXP2 → BOD2 = IMP0 → IMP2' → BOD2 and

(b) IMP0 → IMP2' → EXP1 → BOD1=IMP0 → IMP1'→ BOD1.

Then the __mutual recursion of M1 and M2 via f1 and f2__, denoted by rec$_{f1,f2}$(M1,M2), is the module specification (PAR1+$_{PAR0}$PAR2 , EXP1+$_{IMP0}$EXP2, IMP1+$_{IMP0}$IMP2, BOD*) where BOD* is the colimit object of

$$\begin{tikzcd}
 & \mathrm{IMP1'} \arrow[r] \arrow[rrd] & \mathrm{EXP2} \arrow[r] & \mathrm{BOD2} \arrow[rd] & \\
\mathrm{IMP0} \arrow[ru] \arrow[rd] & & & & \mathrm{BOD^*} \\
 & \mathrm{IMP2'} \arrow[r] \arrow[rru] & \mathrm{EXP1} \arrow[r] & \mathrm{BOD1} \arrow[ru] &
\end{tikzcd}$$

and the specification morphisms are all induced by the appropriate universal properties. The assumption of f1 and f2 guarantees that the module specification MI0 = (PAR0, IMP0, IMP0, IMP0) shared by M1 and M2 is not modified by the recursive calls. In particular, it implies that the interface morphisms (id_{PAR0}, f1) and (id_{PAR0},f2) provide a well defined partial composition of M1 with M2, and of M2 with M1, respectively, in the sense of definition 3.1. The verification that the syntactical diagram of rec $_{f1,f2}$(M1,M2) commutes is very tedious but direct. The strong persistency of the free functor from Alg(IMP1$+_{IMP0}$IMP2) to Alg(BOD*) cannot be guaranteed in general for reasons similar to the single recursion case. The construction above can be generalized slightly to allow an arbitrary parameter PAR, the same for both IMP1' and IMP2', in place of PAR0, provided that f1 and f2 are compatible with PAR → EXPj.

It is not surprising that mutual and single recursion are closely related. The situation is similar to that of two recursive definitions, say of functions F and G, where F is defined by a polynomial in G alone and viceversa. The components of the solution of this recursive system can be also obtained by substituting one polynomial in the other one and then solve the single recursive definition in the only unknown function variable left.

4.9 Theorem (Mutual vs. single recursion)
Let Mj and fj be as in definition 4.8 and let
MEj=(PARj, EXPj, EXP1$+_{IMP0}$EXP2, EXP1$+_{IMP0}$ EXP2) , j=1,2, with the obvious specification morphisms.Then

$$ME1 \bullet rec_{f1,f2}(M1,M2) = rec_{f2}(M1 \bullet^{p}_{f1} M2) \text{ and}$$
$$ME2 \bullet rec_{f1,f2}(M1,M2) = rec_{f1}(M2 \bullet^{p}_{f2} M1)$$

The previous Theorem allows us to exploit Theorems 4.5, 4.6 and 4.7 to obtain the compatibility of mutual recursion with union, composition and actualization.

5. CONCLUSION

In this paper we have introduced two partial interconnections of module specifications: partial composition and partial actualization. Although motivated by the intuitive idea of establishing interconnections as other module specifications become available, it is shown that both partial operations can be expressed in terms of their total counterpart and union (3.2, 3.6). Exploiting the algebraic laws expressing the compatibility of the basic operations, we have shown that successive partial compositions (or actualizations) are equivalent to a total composition (or actualization) with a union (3.3, 3.7). As a by product, the order in which partial compositions are carried out becomes immaterial.

A more interesting construction mechanism, *recursion*, was introduced in section 4. Unlike the other operations on module specifications, where the semantical correctness is a direct consequence of that of its arguments, the recursion construction (whether single or mutual) does not guarantee the strong persistency of its free functor, unless additional conditions are imposed on the original module. Lacking these, generating or logical constraints (/EWT 83/, /EFPB 86/) should be used to restrict the class of import algebras. Along the same lines, while the semantics of the basic operations can be proved to be compositional and can be explicitly described in terms of those of the arguments (/EW 85/, /BPP 85/), the semantics of $rec_f(M)$ is related implicitly to that of M by a fixed point equation. Notice that the equation is at a semantical level and not syntactical: in CATSPEC (as in all categories) the emphasis is on the morphisms, not on the objects. So, by composing M with $rec_f(M)$ using f, the "information" that the module M is the same is lost. Even though our definition of BOD1 in 4.1 was motivated by an intuitive idea of the effect of "self reference" via f, we can find in /EL 83/ that, in order to obtain the fixed point equation 4.2(b), the use of the coequalizer is, more or less, forced. Coequalizers are also used in the approach to the algebraic solution of recursive equations found in /BG 81/. It should be pointed out that our solution for recursive interconnections of modules is different than in /Bl 84/, where partial orders within each algebra and on the set of algebras are introduced with the functors required to be monotone.

Although the semantical funtors SEM and RSEM can be viewed as an observational behavior of the module specification, other notions of "behavioral" semantics of modules are being analyzed, giving rise to different "levels" of observability. We will discuss them in a forthcoming paper, along with comparisons with other observability concepts, such as in /GGM 76/, /Rei 81/, /GM 82/, /ST 85b/.

ACKNOWLEDGEMENTS

I am grateful to E. Astesiano, E.K. Blum, H. Ehrig, W. Fey, H, Hansen, M. Lowe and B. Mahr for many helpful discussions. Many thanks also to Annette Mosley for her excellent job of typing an unreadable manuscript in record time. This research was supported in part by the National Science Foundation under Grant DCR-8406920.

REFERENCES

/AMRW85/ Astesiano E., Mascari G.F., Reggio G., Wirsing M., On the Parametrized Algebraic Specification of Concurrent Systems, CAAP 85, LNCS 185(1985) 342-358

/BG 81/ Benson D.B., Guessarian I., Algebraic Solutions to Recursion Schemes, L.I.T.P. Tech.Rep. 81-66. Univ. Paris VII, Dec. 1981

/BMM79/ Bertoni A., Mauri G., Miglioli P.A., A Characterization of Abstract Data as Model-Theoretic Invariants, ICALP 79, LNCS 71(1979) 26-37

/B1 84/ Blum, E.K., An Abstract System Model of Ada Semantics, TRW Technical Report, Aug. 1984.

/BEPP86/ Blum E.K., Ehrig H., Parisi-Presicce F., Algebraic Specification of Modules and their Basic Interconnections, to appear in JCSS 86.

/BPP 85/ Blum, E.K., Parisi-Presicce, F., The Semantics of Shared Submodule Specifications, Proc. TAPSOFT 85 Vol. 1, LNCS 185 (1985) 359-373.

/BG 77/ Burstall R.M., Goguen J.A., Putting Theories together to make Specifications. Proc. 5th Intern. Joint Conf. on Artif. Intell., Cambridge 1977, 1045-1058.

/EL 83/ Ehrich H.-D., Lipeck U., Algebraic Domain Equations, Theoret. Comp. Sci. 27 (1983) 167-196.

/EFP 86/ Ehrig, H., Fey, W., Parisi-Presicce, F., Distributive Laws for Composition and Union of Module Specifications for Software Systems, Proc.IFIP TC2 Work. Conf. on Program Specification and Transformation, Bad Tolz, April 1986.

/EFPB 86/ Ehrig, H., Fey, W., Parisi-Presicce, F., Blum, E.K., Algebraic Theory of Module Specifications with Constraints, Proc. Math. Found. of Comp. Sci, LNCS 233 (1986) 59-77.

/EKTWW 81/ Ehrig, H., Kreowski, H.-J., Thatcher, J.W., Wagner, E.G., Wright, J.B., Parameter Passing in Algebraic Specification Languages, Proc. Aarhus Workshop on Prog. Spec., 1981, LNCS 134 (1982) 322-369.

/EM 85/ Ehrig, H., Mahr, B., Fundamentals of Algebraic Specifications 1: Equations and Initial Semantics, EATCS Monographs on Theoret. Comp. Sci. Vol 6, Springer-Verlag, 1985.

/EWT 83/ Ehrig H., Wagner, E.G., Thatcher, J.W., Algebraic Specifications with Generating Constraints, ICALP 83, LNCS 154 (1983) 188-202.

/EW 85/ Ehrig, H., Weber, H., Algebraic Specification of Modules, Proc IFIP Working Conference on Formal Models in Programming, Vienna 1985.

/EW 86/ Ehrig H., Weber, H., Programming in the Large with Algebraic Module Specifications, Proc. IFIP Congress '86, Dublin, Sept 1986.

/FGJM 85/ Futatsugi, K., Goguen, J.A., Joannaud, J.-P., Meseguer, J., Principles of OBJ2, 12th ACM POPL, New Orleans, 1985, 52-66.

/Ga 83/ Ganzinger, H., Parametrized Specifications: Parameter Passing and Implementation, ACM TOPLAS 5, 3 (1983).

/GGM 76/ Giarratana, V., Gimona, F., Montanari, U., Observability Concepts in Abstract Data Type Specifications, 5th Symp. Math. Found. of Comp. Sci. 1976, LNCS 45 (1976) 576-587.

/GM 82/ Goguen, J.A., Meseguer, J., Universal Realization, Persistent Interconnection and Implementation of Abstract Modules, ICALP 82, LNCS 140 (1982) 265-281.

/GTW 78/ Goguen, J.A., Thatcher, J.W., Wagner, E.G., An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types, in Current Trends in Prog. Method., IV: Data Structuring (R.T. Yeh, Ed.), Prentice Hall, New Jersey (1978) 80-149.

/HS 73/ Herrlich, H., Strecker, G.E., Category Theory, Allyn and Bacon Inc., Boston, 1973.

/LZ 75/ Liskov, B.H., Zilles, S.N., Specification Techniques for Data Abstraction, IEEE Trans. on Soft. Eng., Vol SE-1, No. 1(1975) 7-19.

/PD 85/ Padawitz, P., Parameter Preserving Data Type Specifications, Proc. TAPSOFT 85 Vol 1, LNCS 185 (1985) 323 - 341.

/PP 85/ Parisi-Presicce, F., Union and Actualization of Module Specifications: Some Compatibility Results, Techn. Report, Univ. of Southern California, 1985, to appear in JCSS.

/PP 86/ Parisi-Presicce, F., Inner and Mutual Compatibility of Basic Operations on Module Specifications, Proc. CAAP 86, LNCS 214 (1986) 30-44. Full version: Techn. Rep. 86-06, Techn. Univ. Berlin, April 1986.

/Par 72/ Parnas, D.L., A Technique for Software Module Specification with Examples, Comm. ACM 15, 5(1972) 330-336.

/Rei 81/ Reichel, H., Behavioral Equivalence - A unifying concept for initial and final specification methods, Proc. 3rd Hungarian Comp. Sci. Conf., Budapest, 1981, 27-39.

/ST 85a/ Sannella, D., Tarlecki, A., Program Specification and Development in Standard ML, 12th ACM POPL, New Orleans, 1985, 67-77.

/ST 85b/ Sannella, D., Tarlecki, A., On Observational Equivalence and Algebraic Specification, CAAP 85, LNCS 185 (1985) 308-322.

/SW 83/ Sannella, D., Wirsing, M., A Kernel Language for Algebraic Specification and Implementation, Internal Report No. CSR-131-83, Univ. Edinburgh, 1-44.

/WE 86/ Weber, H., Ehrig, H., Specification of Modular Systems, IEEE Trans. Soft. Eng., June 1986.

/WPPDB 83/ Wirsing, M., Pepper, P., Partsch, H., Dosch, W., Broy, M., On Hierarchies of Abstract Data Types, Acta Inform. 20 (1983) 1-33.