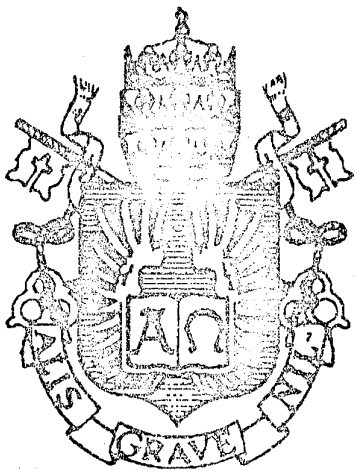


PUC

Monografias em Ciência da Computação
nº 17/92

On the Modularisation Theorem for Logical Specifications: its role and proof

Paulo A. S. Veloso

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Monografias em Ciência da Computação, Nº 17/92
Editor: Carlos J. P. Lucena Março, 1992

On the Modularisation Theorem for Logical Specifications: its role and proof *

Paulo A. S. Veloso

* This work has been sponsored by the Secretaria de Ciência e Tecnologia da Presidência da República Federativa do Brasil and SERC, United Kingdom.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente,225 — Gávea
22454970 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br
techrep@inf.puc-rio.br (for publications only)

ON THE MODULARISATION THEOREM FOR LOGICAL SPECIFICATIONS: its role and proof

Paulo A. S. VELOSO

ABSTRACT

This paper examines the role played by the Modularisation Property and presents a proof of the Modularisation Theorem for logical specifications, i. e. those presented by a set of first-order sentences of a (possibly many-sorted) language.
The importance of (some version of) the Modularisation Property in this context has been noted by several researchers, for it may be regarded as involving the preservation of modular structure under refinements. The Modularisation Theorem amounts to a basic logical tool guaranteeing this preservation, in particular providing both composite implementations and instantiated specifications in a natural and direct manner.
The proof of the Modularisation Theorem presented here is based on two central ideas: Craig's Interpolation Lemma from logic and the concept of kernel of an interpretation.

Key words:

Formal specifications, software development, formal methods, Modularisation Property, formal logic, Craig's Interpolation Lemma, interpretation, conservative extension, implementation, parameterised specifications.

SOBRE O TEOREMA DA MODULARIZAÇÃO PARA ESPECIFICAÇÕES LÓGICAS: seu papel e demonstração

Paulo A. S. VELOSO

RESUMO

Este trabalho examina o papel desempenhado pela Propriedade da Modularização e apresenta uma demonstração do Teorema da Modularização para especificações lógicas, i. e. aquelas descritas por um conjunto de sentenças de primeira ordem de uma linguagem (possivelmente poli-sortida).
A importância da Propriedade da Modularização (em alguma versão) nesse contexto tem sido apontada por vários pesquisadores, uma vez que ela pode ser vista como envolvendo a preservação de estrutura modular sob refinamentos. O Teorema da Modularização vem a ser uma ferramenta lógica básica garantindo essa preservação, em particular, fornecendo maneiras naturais e simples para se compor implementações e para se instanciar especificações parametrizadas.
A demonstração do Teorema da Modularização apresentada aqui se baseia em duas idéias centrais: o Lema da Interpolação de Craig, da lógica, e o conceito de núcleo de uma interpretação.

Palavras chave:

Especificações formais, desenvolvimento de programas, métodos formais, Propriedade da Modularização, lógica formal, Lema da Interpolação de Craig, interpretação, extensão conservativa, implementação, especificações parametrizadas.

ACKNOWLEDGEMENTS

Research reported herein is part of an on-going research project in collaboration with Prof. T. S. E. Maibaum. Partial financial support from British and Brazilian agencies is gratefully acknowledged. The hospitality and support of the Dept. of Computing, Imperial College of Science, Technology and Medicine, and the Dept. of Informatics, Pontifícia Universidade Católica do Rio de Janeiro were very helpful. Collaboration with Tom Maibaum and Martin Sadler was instrumental in sharpening many ideas. Also, José Fiadeiro, Sheila Veloso and Haydée Poubel are to be thanked for several helpful discussions.

CONTENTS

1. INTRODUCTION

This paper examines the role played by the so-called Modularisation Property and gives a proof of the Modularisation Theorem for logical specifications. By a logical specification we mean a formal specification presented by a set of first-order sentences of a (possibly many-sorted) language. In this context, the importance of Modularisation Property stems from its use in composing implementations and in instantiating parameterised specifications. The Modularisation Theorem is a basic logical tool providing in a natural and direct manner both composite implementations and instantiated specifications.

The motivations for the logical approach to formal specifications come from two related sources. On the one hand, logical axioms employ the language and concepts related to program verification [Manna '74]; on the other hand, the logical formalism accommodates 'liberal' specifications, which provide flexibility for specifying what one wishes without forcing over-specification [Maibaum+Veloso '81; Maibaum+Turski '84; Veloso+Maibaum+Sadler '85; Maibaum '86]. In addition, this logical approach has been instrumental in extending some of these ideas to problem solving [Veloso+Veloso '81; Veloso '88].

The Modularisation Property of logical specifications is very useful in the context of formal specifications, particularly for dealing with program development by stepwise refinement and abstract data types [Turski+Maibaum '87; Maibaum+Veloso+Sadler '84; Veloso '87]. The importance of (some version of) the Modularisation Property in this context has been noted by several researchers. The Modularisation Theorem may be regarded as guaranteeing the preservation of modular structure under refinements.

In the context of logical specifications, the central role played by the Modularisation Property was pointed out by T. S. E. Maibaum, and connections between the latter and Craig's Interpolation Property have been exploited by M. R. Sadler in early formulations of the Modularisation Theorem [Maibaum+Sadler '85].

In this paper we discuss the importance of the Modularisation Property for stepwise refinement of logical specifications and prove the Modularisation Theorem for such specifications. We also briefly discuss the role played by Craig's Interpolation Property in this context. The core of this paper is in section 3, where we present and prove the Modularisation Theorem.

This paper is structured into a main text followed by a few appendices. The main texts gives the general ideas and lines of argument whereas some more technical details are left for the appendices.

The main text discusses the role of the Modularisation Property and presents a proof of the Modularisation Theorem and comments on the importance of Craig's Interpolation Property for modularisation. Its structure is as follows.

Section 2 examines the Modularisation Property and discusses its importance in the formal development of specifications and programs in a stepwise manner. We start by reviewing the concept of implementation as an interpretation into a conservative extension. We then go on to explain how the Modularisation Property is instrumental in composing such implementations in a natural and direct manner. Next, we review some basic ideas concerning parameterised logical specifications and indicate the central role of the Modularisation Theorem for instantiation of parameters.

Section 3 presents and proves the Modularisation Theorem for the simple case of logical specification in one-sorted languages. We first show the construction involved in the Modularisation Theorem for this simple case. Then, we prove it by relying on some lemmas, whose proofs are left for appendix 2.

Section 4 deals with the extension of these ideas to many-sorted specifications and the role played by Craig's Interpolation Property for modularisation. We first indicate how these ideas can be adapted to many-sorted specifications by formulating them in simple categorical terms. Then we comment on the role of Craig's Interpolation Property for modularisation, indicating in which sense they are actually equivalent.

Finally, section 5 presents some concluding remarks and comments.

The appendices present the versions of Craig's Interpolation Lemma used and contain some details involved in the proofs of the claims on which our proof of the Modularisation Theorem rests.

We adopt usual notation and terminology. For more information concerning logical aspects the reader is referred to standard textbooks, for instance [Enderton '72; Ebbinghaus+Flum+Thomas '84; Shoenfield '67; van Dalen '89]. For the few simple categorical concepts employed in section 4, some useful references are [Arbib+Mannes '75; Goldblatt '79].

2. THE ROLE OF THE MODULARISATION PROPERTY

In this section we shall informally introduce the Modularisation Property. We shall also point out its importance in the formal development of specifications and programs in a stepwise manner [Maibaum+Turski '84]. This importance arises mainly in two situations, namely in composing implementations and in instantiating parameterised specifications.

2.1 Implementations of Logical Specifications

Let us start by considering implementation of abstract data types [Ehrig+Mahr '85; Maibaum+Veloso+Sadler '84]. One has an abstract specification A which one wishes to implement on a (more concrete) specification C. For this purpose, one has to provide on top of C some support for the abstract concepts of A. One account of what is involved is as follows [Turski+Maibaum '87; Veloso '87].

Let us examine more closely what is involved in implementing an abstract specification A in terms of another one, C. The result will be a module representing objects of A in terms of those of C, and operations and predicates of A by means of procedures using operations and predicates of C. We can abstract a little from the actual procedure texts by replacing them by specifications of their input-output behaviours. These amount to (perhaps incomplete) definitions of the operations and predicates of A in terms of those of C and can be regarded as axioms involving both the symbols of A and of C. Similarly, the representation part gives rise to axioms capturing (some of) the so-called representation invariants [Guttag '77].

Now, let us describe this situation in terms of theories presented by axioms.

One extends the concrete specification C by adding symbols to correspond to the abstract ones in A, perhaps together with some auxiliary symbols. Since one does not wish to disturb the given concrete specification C, this extension B should not impose any new constraints on C. This can be formulated by requiring the extension $B \supseteq C$ to be *conservative* (or non-creative) in the sense that B adds no new consequence to C in the language of the latter.

One then wishes to correlate the abstract symbols in A to corresponding ones in B. But, the properties of A are important, for instance in guaranteeing the correctness of an abstract program supported by A. Thus, in translating from A to B, one wishes to preserve the properties of A as given by its axioms. This can be formulated by requiring the

translation $i : A \rightarrow B$ to be an *interpretation* of theories in the sense that it translates each consequence of A into a consequence of B.

We thus arrive at the concept of an *implementation* of A on C as an interpretation i of A into a conservative extension B (sometimes called a *mediating specification*) of C. This is depicted as a triangle below in figure 1.

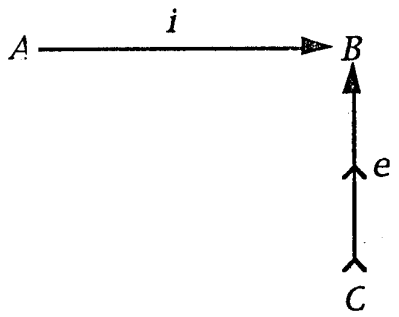


Figure 1: The implementation triangle.

It is interesting to notice that the mediating specification in an implementation triangle as above may be regarded as a solution to a problem with conflicting goals, in the following sense. First, consider the problem of interpreting A. Clearly, if one can interpret A into B, then so can one into any specification stronger than B. That is, the stronger the specification B, the easier it is to interpret A into it. Now, consider the problem of extending C conservatively. Clearly, if B is a conservative extension of C, then so is any extension of C that is weaker than B. So, now the weaker the specification B, the more likely it is to be a conservative extension of C. Thus, the mediating specification B in an implementation triangle has to be both strong enough to interpret A and weak enough to be a conservative extension of C.

2.2 The Role of Modularisation in Composing Implementations

In stepwise development it is highly desirable to be able to compose refinement steps in a natural way. Let us consider the situation depicted in figure 2. Here, one has a first implementation of A on C (with mediating specification B) and a second implementation of C on D (with mediating specification E).

Now, one would wish to compose these two implementations, in an easy and natural manner, so as to obtain a composite implementation of A directly on D. An immediate question that arises is: what would its mediating specification be?

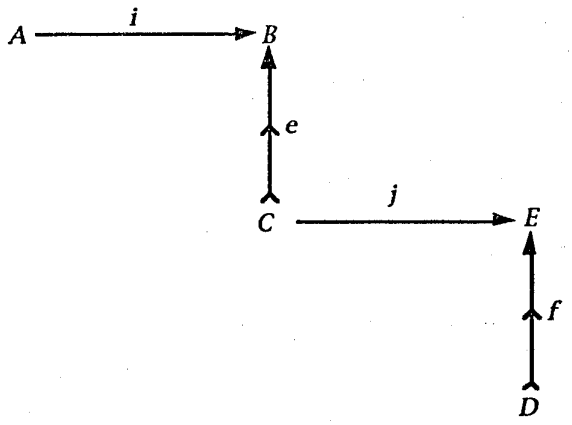


Figure 2: Composition of implementations.

This is where Modularisation Property comes into play. For, it will allow one to obtain such a mediating specification M, together with the required interpretation k of B into M and a conservative extension g of E into M. In other words, it will enable one to complete the square, thereby obtaining a composite implementation of A directly on D, consisting of a composite interpretation of A into M together with a composite conservative extension of D into M, as illustrated in figure 3.

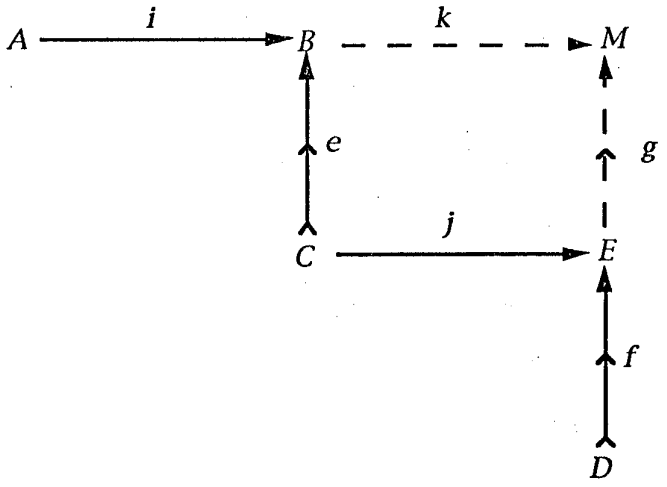


Figure 3: Composite implementation.

Here it is worthwhile noting that this composition mimics exactly what a programmer does by simply putting together the corresponding modules. This is possible because we do not require that the given mediating symbols be eliminated in constructing the composite one.

2.3 Parameterised Logical Specifications

One of the long standing research goals in work on formal specifications is the provision of standard building blocks from which larger specifications might be constructed and that may be re-used in different situations. In particular, the structuring of a specification into a "context" and "parameter" has been found to be particularly useful. The idea is that the context can be plugged into different situations by appropriate choice of values (instances) for the parameters. Such structured specifications are called parameterised (or generic) specifications [Ehrig+Mahr '85].

Let us consider a simple example: *SEQ[DATA]* (sequences of, as yet, unspecified values), where *DATA* is the formal parameter and *SEQ* is what we referred to above as the context. Thus, DATA should be a part of *SEQ[DATA]*. One may visualise this situation as in figure 4.

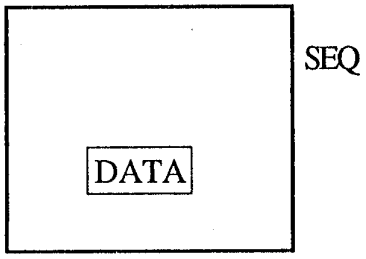


Figure 4: Parameterised data type *SEQ[DATA]*.

Now, one would like to instantiate *DATA* by various actual parameters to get 'normal' specifications. So, if *NAT* and *INT* are specifications of natural numbers and integers, respectively, then *SEQ[DATA* **replaced by** *NAT]* and *SEQ[DATA* **replaced by** *INT]* should give specifications for sequences of naturals and sequences of integers, respectively. One wishes to instantiate *DATA* in *SEQ[DATA]* by *NAT* to obtain *SEQ[NAT]* by 'replacing' the formal parameter *DATA* by the actual parameter *NAT*. Our intuition tells us that *SEQ[NAT]* should look like figure 5.

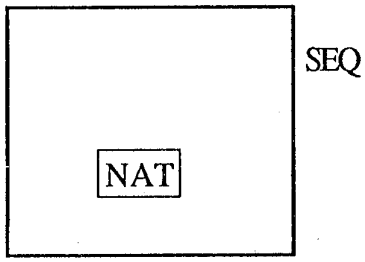


Figure 5: Instantiated data type *SEQ[NAT]*.

That is, one just replaces the *DATA* part by *NAT* within the context *SEQ*. Thus, for each given specification for *DATA*, *SEQ[DATA]* produces an instantiated version, like *SEQ[NAT]*.

The above intuition suggests regarding *SEQ[DATA]* as a function on specifications. Indeed, this is the idea underlying the semantics of a parameterised specification as a (partial) function from models to models, or as a (partial) function from specifications to specifications.

Another viewpoint is provided by considering the properties of the specifications. First, one should expect *SEQ[NAT]* to inherit all the properties of *SEQ[DATA]* that concern only sequences, such as

$$tail(cons(x,l)) = l$$

Also, *DATA* is supposed to be a part of *SEQ[DATA]*, but not an arbitrary one, in the following sense. In going from *DATA* to *SEQ[DATA]*, one would not expect to gain any more knowledge about *DATA*. In other words, no new constraints on *DATA* are placed by the addition of the context *SEQ[ ]*. (This is not to say that every parameter is appropriate for every context.)

2.4 The Role of Modularisation in Instantiation of Parameters

The simple tools of conservative extension and interpretations between theories provide us with a quite straightforward account of parameterisation. The parameterised types have specifications which are essentially the same as those of normal types. Thus *SEQ[DATA]* should be a specification just like *SEQ[NAT]*. Their meanings (theories) are the same as for normal specifications and not (partial) functions between specifications or models. Instantiation of a formal parameter by an actual parameter rests on a straightforward application of the Modularisation property [Maibaum+Veloso+Sadler '85].

Indeed, a specification S is said to be *parameterised* by a sub-specification X (called *parameter*) whenever S is a conservative extension of X, and a *parameter instantiation* is an interpretation $p : X \rightarrow Y$. We thus have a situation, depicted in figure 6, similar to the one encountered in composing implementations.

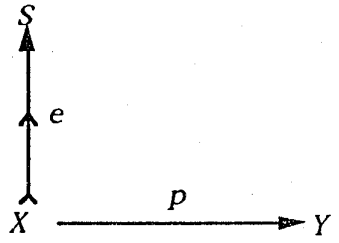


Figure 6: Parameterised specification with parameter instantiation.

Once again the Modularisation Property comes into play. It will enable us to complete the square, thereby yielding the resulting instantiated specification, as illustrated in figure 7.

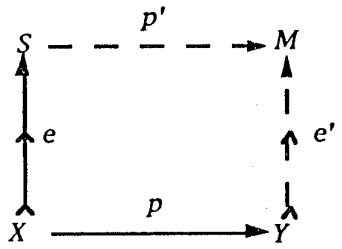


Figure 7: Instantiated specification resulting from figure 6.

Here it is worthwhile noting that the construction of this instantiated specification mimics exactly what was suggested above in figures 4 and 5. Moreover, the instantiated specification T is still a conservative extension of the actual argument Y.

3. THE MODULARISATION THEOREM AND ITS PROOF

In this section we shall formulate the Modularisation Theorem and indicate its proof. It can be formulated in categorical terms, but for the sake of simplicity, we shall adopt another approach. We shall first formulate the Modularisation Theorem, and prove it in simple terms, for one-sorted specifications. In the next section we shall indicate how it can be expressed, in categorical terms, for many-sorted specifications, and indicate how the proof idea can be adapted to this more general case.

3.1 The Modularisation Construction: simple version

Consider the following situation. We have:

specifications P, Q and R, with R consistent;
a conservative extension $e : P \subseteq Q$, and
an interpretation $f : P \to R$.

This situation is depicted in figure 8 below.

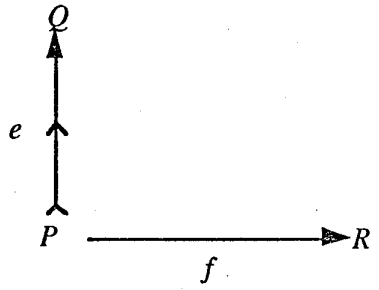


Figure 8: The situation of the Modularisation Theorem: a conservative extension and an interpretation of a specification.

Without loss of generality, we may also assume that the underlying languages L_Q and L_R, of Q, and R have no symbols in common.

Let A consist of the new symbols added to the language L_P of P to form that of Q. Consider the following construction.

We first form a new language L by adding A to the language L_R of R.
We then extend the translation f from L_P to L_R to a map g from L_Q to the new language L.
Now, we use this map g to translate specification Q, thereby obtaining a new specification $g(Q)$ in the language L.
Finally, we construct specification R as the union $S = g(Q) \cup R$.

This construction is illustrated in figure 9 below.

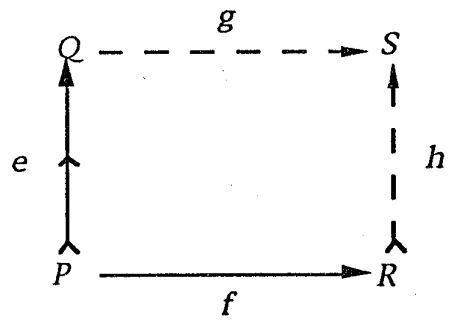


Figure 9: The construction of the Modularisation Theorem: completing the square with a specification.

Theorem: *The Modularisation Theorem* (simple version)

Given

specifications P, Q and R, with R consistent,

a conservative extension $e : P \subseteq Q$, and

an interpretation $f : P \to R$;

the above construction yields

a conservative extension S of R, and

an interpretation $g : Q \to S$.

Thus, the Modularisation Theorem completes the square with the construction of a new specification S satisfying the conflicting requirements mentioned in 2.1, namely it extends R conservatively and interprets Q.

Also, notice that the above construction actually yields a 'reasonable' presentation for S from the given situation, provided that it is 'reasonable' as well. For instance, it constructs a finite specification for S provided that the given ones for P, Q and R are so.

3.2 Proof of the Modularisation Theorem: simple version

First of all, since R is assumed to be consistent, notice that so are:

P, because it can be interpreted into R; and

Q, because it is a conservative extension of P.

Also, by construction, S is specified by $g(Q) \cup R$. Thus, we clearly have

g interprets Q into S, and

S.. is an extension of R.

Thus, it only remains to prove that

the extension $h : R \subseteq S$ is conservative.

The proof idea is actually simple:

all would be fine if g (and f) happened to be bijective;
we shall resort to Craig's Interpolation Lemma to take them as surjective, so to speak;
all that remains is to make them injective (this is the purpose of the kernel to be introduced below).

The version of Craig's Interpolation Lemma we shall use appears in Appendix 1.

Define the kernel of f to consist of the biconditional sentences of the language of P such that f identifies both components. More precisely

$$K(f) = \{ (\phi \leftrightarrow \psi) \in Sent(L_P)) \,/\, f(\phi) = f(\psi) \}.$$

We now construct two new specifications, namely $P' = P \cup K(f)$ and $Q' = Q \cup K(f)$.

Notice that this does not change their languages: $L_{P'} = L_P$ and $L_{Q'} = L_Q$.

Claim 1:

a) Q' is a conservative extension of P'.

b) f interprets P' into R and g interprets Q' into S.

c) $f(P') = f(P)$ and $g(Q') = g(Q)$.

d) P' and Q' are consistent.

In view of claim 1, we may replace P by P' and Q by Q'.

Claim 2:

The symbols of L_Q that g translates to symbols of L_R are exactly the translations under f of the symbols of L_P, that is

$$f(L_P) = g(L_Q) \cap L_R.$$

Claim 3

a) For each sentence θ of L_Q, if $g(Q') \vDash g(\theta)$, then $Q' \vDash \theta$

i.e., the interpretation $g : Q' \rightarrow g(Q')$ is faithful (conservative).

b) In particular, $g(Q')$ is consistent.

Now, consider a sentence α of L_Q such that α is in Cn S.

Since

$f(L_P) = g(L_Q) \cap L_R$, by claim 2, and
$g(Q')$ is consistent, by claim 3.b,

we can apply Craig's Interpolation Lemma.

It yields a set J of sentences of $f(L_P)$ such that

(i) $g(Q') \models \tau$, for each sentence τ of J, and

(ii) $R \cup J \models \alpha$.

Claim 4

$J = f(I)$, for some set of sentences I of L_P.

Hence, (i) yields, since g extends f,

(iii) $g(Q') \models g(\sigma)$, for each sentence σ of I.

From (iii), by claim 3.a, we have, for each sentence σ of I,

$Q' \models \sigma$, whence, since σ is in L_P, $P' \models \sigma$.

Thus, since, by claim 1.b, f interprets P' into R, we have

$R \models f(\sigma)$, for each sentence σ of I, i.e.,
$R \models \tau$, for each sentence τ of J, in view of claim 4.
Therefore, because of (ii),
$R \models \alpha$.

This establishes the conservativeness of the extension $h : R \subseteq S$, thereby concluding the proof of the Modularisation Theorem.

The proofs of these claims appear in Appendix 2.

4. THE MODULARISATION THEOREM: COMMENTS AND EXTENSIONS

In this section we indicate how to adapt the previous construction and proof of the Modularisation Theorem for specifications in many-sorted languages. We shall also briefly comment on the relationship between the Modularisation Property and Craig's Interpolation Property.

4.1 The Modularisation Theorem: many-sorted, categorical version

The construction of the Modularisation Theorem can be described as an amalgamated sum: a pushout rectangle. Since this rectangle involves an extension, it turns out to be a pullback as well. This viewpoint is particularly appropriate for the case of specifications in many-sorted language.

A *many-sorted language* consists of an alphabet of symbols, a set of sorts and a declaration assigning to each symbol its profile of sorts of arguments and results. A *translation* between such languages is required to preserve declarations. A *many-sorted specification* consists of a set of sentences of such a many-sorted language.

The construction of the Modularisation Theorem can then be adapted to many-sorted specifications as follows.

We are given:

specifications T_0, T_1 and T_2, with T_2 consistent;

a conservative extension $e : T_0 \subseteq T_1$, and

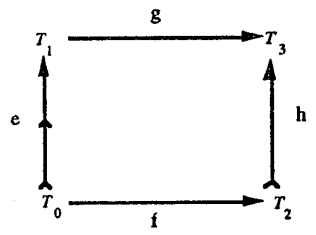
an interpretation $f : T_0 \to T_1$.

We first form language L_3 as the pushout of the language diagram underlying the given specifications. The construction of pushout of languages parallels the one given in [Ehrich '82] for algebraic languages. This yields language translations $g : L_1 \to L_3$ and $h : L_2 \to L_3$, with h being a language extension.

We then use these maps to translate the given specification of T_1 into the new language L_3 and construct specification T_3 as before: $T_3 = g(T_1) \cup T_2$.

Now, it is not difficult to see that these four theories do form a pushout diagram. Moreover, since T_1 is an extension of T_0, T_3 turns out to be an extension of T_2 as well and the language diagram underlying these specifications is seen to be a pullback.

Theorem: *The Modularisation Theorem* (many-sorted version)
Consider a pushout rectangle of theories



with T_2 consistent and $e : T_0 \subseteq T_1$ a conservative.
Then, $h : T_2 \subseteq T_3$ is a conservative extension as well.

The proof of this many-sorted version of the Modularisation Theorem follows basically the same idea as before. The only difference is that we have to resort to a many-sorted version of Craig's Interpolation Theorem (see Appendix 3) and claim 2 now becomes:

The induced rectangle of language inclusions

$$\begin{array}{ccc} g(L_1) & \subseteq & L_3 \\ \cup & & \cup \\ f(L_0) & \subseteq & L_2 \end{array}$$

is a pullback.

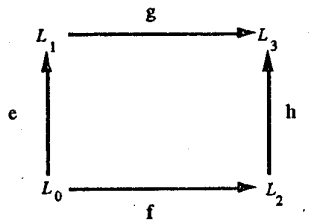
4.2 The Role of Craig's Interpolation Property for Modularisation

We have seen that Craig's Interpolation Property plays a crucial role in establishing the Modularisation Theorem. It has been pointed out that a version of this property is, not only sufficient, but also, in a certain sense, necessary for the modularisation property [Sadler '84 ; Maibaum+Fiadeiro+Sadler '90].

The above connection has a somewhat 'global' nature. But, a somewhat stronger, and more 'local', connection can be established as we shal now indicate.

Both the Modularisation Theorem and Craig's Interpolation Lemma have to do with theories over languages arranged in a certain pattern.

In order to take a closer look at the relationship between these two properties we shall consider a pushout rectangle $\mathcal{R}$ of language translations



where $e : L_0 \subseteq L_1$ is an extension of L_0 to L_1.
Thus, we also have:

$h : L_2 \subseteq L_3$ is an extension of L_2 to L_3; and

the rectangle $\mathcal{R}$ of language translations is a pullback.

Now, Craig's Interpolation Property is concerned with theories over languages L_1 and L_2 and the 'common' sub-language L_0, in the sense that one can find a set of interpolant sentences in this common sub-language L_0.

On the other hand, the Modularisation Property is concerned with theories over languages L_1, L_2 and L_3, where one wishes the property of conservative extension to be preserved.

Now, given a language extension $e : L_0 \subseteq L_1$ and a theory T_1 over L_1, the so called *restriction* $T_1|L_0 = \mathrm{Cn}T_1 \cap Sent(L_0)$ gives a theory over L_1 that T_1 extends conservatively [Shoenfield '67, p. 95, exercise 9]. (Notice that this restriction is the pullback of the inclusions $j : T_1 \subseteq Sent(L_1)$ and $e : Sent(L_0) \subseteq Sent(L_1)$.)

By means of the construction of restriction one can establish a 'local' connection between theories with Craig's Interpolation Property and theories with the Modularisation Property. In this sense, having Craig's Interpolation Property is a necessary and sufficient condition for the modularity of a diagram.

5. CONCLUSIONS

This paper has examined the role played by the Modularisation Property and presented a proof of the Modularisation Theorem for logical specifications, i. e. those presented by a set of first-order sentences of a (possibly many-sorted) language. The motivations for this logical approach to formal specifications come from two related sources: being close to concepts and notation related to program verification and accommodating 'liberal' specifications, which provide flexibility without forcing over-specification [Maibaum+Veloso '81; Veloso+Maibaum+Sadler '85; Turski+Maibaum '87].

The importance of (some version of) the Modularisation Property in this context has been noted by several researchers, for it may be regarded as involving the preservation of modular structure under refinements. The Modularisation Theorem amounts to a basic logical tool guaranteeing this preservation, in particular providing both composite implementations and instantiated specifications in a natural and direct manner.

We have started in section 2 by reviewing the Modularisation Property and discussing its importance in the formal development of specifications and programs in a stepwise manner, both for implementation and for instantiation. The Modularisation Theorem itself has been dealt with in two stages, in order to single out the main ideas. In section 3 we have presented and proved the Modularisation Theorem for the simple case of logical specification in one-sorted languages; and in section 4 we have indicated how these ideas can be adapted to many-sorted specifications by formulating them in simple categorical terms, in addition to commenting on the role of Craig's Interpolation Property for modularisation.

Section 2 has been intended to situate the Modularisation Property for formal development of specifications and programs in a stepwise manner and indicate its importance in this context. We have first reviewed the idea of implementation triangle (an interpretation into a conservative extension). It then becomes quite clear why the Modularisation Property is instrumental in composing such implementations in a natural and direct manner. Next, we have reviewed the basic ideas of parameterised logical specification as a conservative extension and parameter instantiation as an interpretation. Once again, the central role of the Modularisation Theorem for defining the result of parameter instantiation is immediately clear.

The core of this paper is section 3, where we have presented and proved the Modularisation Theorem for the simple case of logical specification in one-sorted languages. We have first shown the construction involved in the Modularisation Theorem for this simple case. Then, we have established it by relying on some lemmas, whose proofs have been left for appendix 2.

Here two central ideas deserve some comments. As we have already mentioned, the proof idea is actually simple: one would wish the interpretations involved to be bijective. Now, Craig's Interpolation Lemma allows us to assume them as surjective, so to speak. (We shall come back to the central role played by Craig's Interpolation Property.) The other important idea is the introduction of the kernel. It allows us to replace the given source theories by stronger ones, without changing the target theories, but yielding the payoff that the interpretations now become faithful as desired.

Section 4 has concentrated on the extension of these ideas to many-sorted specifications and the role played by Craig's Interpolation Property for modularisation. We have indicated how these ideas can be easily adapted to many-sorted specifications by formulating them in simple categorical terms. Then we have commented on the role of Craig's Interpolation Property for modularisation, indicating in which sense they are actually equivalent.

We shall conclude with two general remarks. The first one concerns Craig's Interpolation Property: one should bear in mind that there are a few versions of this property, which may be appropriate for distinct specification formalisms [Maibaum+Sadler '85; Maibaum '86; Rodenburg + van Glabbeek '88]. The second one has to do with the technique employed in our proof of the Modularisation Theorem: the central idea of kernel is readily seen to be connected to the concept of quotient, a point to be examined in more detail in a report in preparation.

APPENDICES

APPENDIX 1: CRAIG'S INTERPOLATION LEMMA (simple version)

The version of Craig's Interpolation Lemma we use can be stated as follows.

Theorem: *Craig's Interpolation Lemma*

Assume that we have first-order languages such that $L_0 = L_1 \cap L_2$, and consider sets of first-order sentences T_1, of L_1, and T_2, of L_2, such that T_1 is consistent.

Given any sentence ϕ of L_2 such that $T_1 \cup T_2 \models \phi$, there exists a set of sentences I of L_0 (called *interpolants*) such that

(i) $T_1 \models \sigma$ for every $\sigma \in I$ and

(ii) $T_2 \cup I \models \phi$.

APPENDIX 2: PROOFS OF THE CLAIMS

Claim 1:

a) Q' is a conservative extension of P'.

b) f interprets P' into R and g interprets Q' into S.

c) $f(P') = f(P)$ and $g(Q') = g(Q)$.

d) P' and Q' are consistent.

Proof.

a) follows from the fact that the sentences in K(f) added to P and Q are all in L_P.

Indeed, consider a sentence θ of L_P such that $Q' \models \theta$.
Then, by compactness, for some finite conjunction χ of sentences of K(f), $Q \cup \{ \chi \} \models \theta$, whence $Q \models (\chi \rightarrow \theta)$.
Now, since $(\chi \rightarrow \theta)$ is a sentence of L_P, by conservativeness we have $P \models (\chi \rightarrow \theta)$, whence $P' \models \theta$.

b) and c) follow because whenever $(\phi \leftrightarrow \psi)$ is in K(f), then $f(\phi) = f(\psi)$ and g acts identically on ϕ and ψ.

d) The consistency of P' follows from the assumed consistency of R and from part b, whence the consistency of Q' follows from part a.

QED

Claim 2:

The symbols of L_Q that g translates to symbols of L_R are exactly the translations under f of the symbols of L_P, that is

$f(L_P) = g(L_Q) \cap L_R$.

Proof.

This follows because, by construction, g acts identically on the symbols of L_Q that are not in L_P.

QED

Claim 3

a) For each sentence θ of L_Q, if $g(Q') \vDash g(\theta)$, then $Q' \vDash \theta$

i.e., the interpretation $g : Q' \rightarrow g(Q')$ is faithful (conservative).

b) In particular, $g(Q')$ is consistent.

Proof.

First, form an auxiliary presentation T by adding to Q' the following set of sentences:

for each symbol t in $f(L_P)$ choose a symbol s in L_P such that $t = f(s)$ and add the corresponding equivalence sentence: $(t \leftrightarrow s)$
{ e.g. for a unary predicate symbol add $\forall x\ (\ t(x) \leftrightarrow s(x)\ )$,
and for a unary function symbol add $\forall x \forall y\ (\ y=t(x) \leftrightarrow y=s(x)\ )$ }.

(i) For each atomic formula μ of L_P, since Q' includes K(f), we have:

$T \vDash (f(\mu) \leftrightarrow \mu)$.

(ii) For every sentence ρ of L_Q, since, g is the identity on symbols of L_Q not in L_P, we have, by induction:

$T \vDash (g(\rho) \leftrightarrow \rho)$.

(iii) T is an extension of $g(Q')$.

Because, for each axiom η of Q:
$T \vDash \eta$, since $Q' \subseteq T$; and
$T \vDash (g(\eta) \leftrightarrow \eta)$, by *(ii)*.
Thus, $T \vDash g(\eta)$.

(iv) T is a conservative extension of Q'.

Because, each new symbol f(s) added to Q' to form T is introduced by a corresponding definition in terms of a (single) symbol s of L_P.

a) Consider a sentence θ of L_Q, such that $g(Q') \vDash g(\theta)$.

Then, since, by *(iii)*, T extends $g(Q')$, $T \vDash g(\theta)$.
Also, by *(ii)*, $T \vDash (g(\theta) \leftrightarrow \theta)$.
Hence, $T \vDash \theta$, with θ a sentence of L_Q.
Thus, by *(iv)*, $Q' \vDash \theta$.

b) Claim 1.d and *(iv)* yield the consistency of T, whence, by *(iii)*, that of $g(Q')$.

QED

Claim 4

$J = f(I)$, for some set of sentences I of L_P.

Proof.

This follows from claim 2, since J is a set of sentences of $f(L_P)$.

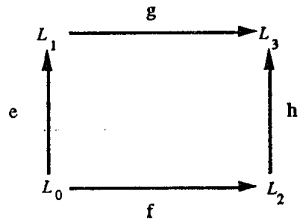
QED

APPENDIX 3: CRAIG'S INTERPOLATION LEMMA (many-sorted version)

The many-sorted version of Craig's Interpolation Lemma we use can be stated as follows.

Theorem: *Craig's Interpolation Lemma (many-sorted version)*

Assume that the following rectangle of language extensions



is a pullback and consider sets of first-order sentences T_1, of L_1, and T_2, of L_2, such that T_1 is consistent.

Given any sentence ϕ of L_2 such that $T_1 \cup T_2 \models \phi$, there exists a set of sentences I of L_0 (called *interpolants*) such that

(i) $T_1 \models \sigma$ for every $\sigma \in I$ and

(ii) $T_2 \cup I \models \phi$.

REFERENCES

Arbib, M. ; Mannes, E. - *Arrows, Structures and Functors : the Categorical Imperative.* Academic Press, New York, 1975.

Ebbinghaus, H. D. ; Flum, J. ; Thomas, W. - *Mathematical Logic.* Springer-Verlag, Berlin, 1984.

Enderton, H. B. - *A Mathematical Introduction to Logic.* Academic Press; New York, 1972.

Ehrich, H.-D. - "On the theory of specification, implementation and parameterization of abstract data types". *J. ACM*, vol. **29** (no. 1), 1982; p. 206 - 227.

Ehrig, H. ; Mahr, B. - *Fundamentals of Algebraic Specifications, 1: Equations and Initial Semantics.* Springer-Verlag, Berlin, 1985.

Goldblatt, R. - *Topoi: the Categorial Analysis of Logic.* North-Holland, Amsterdam, 1979.

Guttag, J. V.- "Abstract data types and the development of data structures". *Comm. ACM*, vol. **20** (no. 6), 1977; p. 396 - 404.

Maibaum, T. S. E. - "The role of abstraction in program development". Kugler, H.-J. (ed.) *Information Processing '86.* North-Holland, Amsterdam, 1986, p. 135 - 142.

Maibaum, T. S. E. ; Fiadeiro, J. L. ; Sadler, M. R. - "Stepwise program development in Π-institutions". Dept. of Computing, Imperial College, London; draft, Nov., 1990.

Maibaum, T. S. E. ; Sadler, M. R. - "Requirements on Logics for Specification". Dept. of Computing, Imperial College, London; draft.

Maibaum, T. S. E. ; Sadler, M. R. - "Axiomatising Specification Theory". Kreowski, H.-J. (ed.) *Recent Trends in Data Type Specification.* Springer-Verlag, Berlin, 1985, p. 171 - 177.

Maibaum, T. S. E. ; Turski, W. M. - "On what exactly is going on when software is developed step-by-step". *7th Intern. Conf on Software Engin.*. IEEE Computer Society, Los Angeles, 1984 ; p. 528 - 533.

Maibaum, T. S. E. ; Veloso, P. A. S. - A logical theory of data types motivated by programming. Imperial College of Science and Technology, Dept. Computing, Tech. Rept. 81 / 28 , Nov. 1981 ; ii + 35 p .

Maibaum, T. S. E. ; Veloso, P. A. S. ; Sadler, M. R. - "Logical specification and implementation". Joseph, M. ; Shyamasundar, R. (eds.) *Foundations of Software Technology and Theoretical Computer Science* { LNCS 181 }. Springer-Verlag, Berlin, 1984 ; p. 13 - 30.

Maibaum, T. S. E. ; Veloso, P. A. S. ; Sadler, M. R. - "A theory of abstract data types for program development : bridging the gap?". Ehrig, H. ; Floyd, C. ; Nivat, M. ; Thatcher, J. (eds.) *Formal Methods and Software Development ; vol. 2: Colloquium on Software Engineering* {LNCS 186}. Springer-Verlag, Berlin, 1985 ; p. 214 - 230.

Manna, Z. - *The Mathematical Theory of Computation.* McGraw-Hill; New York, 1974.

Rodenburg, P. H. ; van Glabbeek, R. J. - "An interpolation theorem in equational logic". Centre for Mathematics and Computer Science, Amsterdam, Res. Rept. CS - R8838, 1988.

Sadler, M. R. - "Mapping out specification". Alvey Workshop on Formal Aspects of Specifications, Swindon; position paper, Oct., 1984.

Shoenfield, J. R. - *Mathematical Logic.* Addison-Wesley, Reading, 1967.

Turski, W. M ; Maibaum, T. S. E. - *The Specification of Computer Programs.* Addison-Wesley, Wokingham, 1987.

van Dalen, D. - *Logic and Structure.* Springer-Verlag; Berlin, 1989 (2. ed., 3. pr.).

Veloso, P. A. S. - *Verificação e Estruturação de Programas com Tipos de Dados.* Edgard Blücher, São Paulo, 1987.

Veloso, P. A. S. - "Problem solving by interpretation of theories". Carnielli, W. A. ; Alcântara, L. P. (eds.) *Methods and Applications of Mathematical Logic* { Contemporary Mathematics, vol. 69 }. American Mathematical Society, Providence, 1988 ; p. 241 - 250.

Veloso, P. A. S. - "A computing-like example of conservative, non-expansive, extension". Dept. of Computing, Imperial College of Science, Technology and Medicine, Res. Rept. DoC 91/36, October, 1991.

Veloso, P. A. S. ; Veloso. S. R. M. - "Problem decomposition and reduction: applicability, soundness, completeness". Trappl, R.; Klir, J. ; Pichler, F. (eds.) *Progress in Cybernetics and Systems Research.* Hemisphere, Washington, DC, 1981; p. 199 - 203.

Veloso, P. A. S. ; Maibaum, T. S. E. ; Sadler, M. R. - "Program development and theory manipulation". *3rd Intern. Workshop on Software Specification and Design.* IEEE Computer Society, Los Angeles, 1985 ; p. 228 - 232.