Interpolation in practical formal development

Juan Bicarregui , Theo Dimitrakos , Dov Gabbay , Tom Maibaum

Published version information

Citation: J Bicarregui et al. 'Interpolation in practical formal development.' Logic Journal of the IGPL, vol. 9, no. 2 (2001): 231-244.

DOI: 10.1093/jigpal/9.2.231

Interpolation in practical formal development

Juan Bicarregui and Theo Dimitrakos*
CLRC Rutherford Appleton Laboratory, OX11 0QX, UK
Dov Gabbay and Tom Maibaum
King's College,London, WC2R 2LS, U.K.

Abstract

Interpolation (together with completeness and decidability) has become one of the standard properties that logicians investigate when designing a logic. In this paper, we provide strong evidence that the presence of interpolants is not only cogent for scientific reasoning but has also important *practical implications* in computer science. We illustrate that interpolation in general, and *uniform splitting interpolants*, in particular, play an important role in applications where formality and modularity are invoked. In recognition of the fact that common logical formalisms often lack uniform interpolants, we advocate the need for developing general methods to (re)engineer a specification logic so that (at least) some critical uniform interpolants become available.

Category: *position paper* Topics: *Applications, Logics*

1 Introduction

The (formal or informal) specification of a system such as a software artefact can be conceived as a *finite presentation* of a theory (in the sense of natural science) over a certain reality. As Cengarle and Haeberer also note in [15] the role of logic in computer science is analogous to the role of calculus in physics.[1] However, computer science is not a conventional natural science. A physicist or a chemical engineer, on the one hand, usually apply well established, existing general theories based on *laws of nature* and rarely recourse to creating new theories for an application. The creation of new theories, on the other hand, (as well as the extension, amalgamation, update and revision of existing theories) is an everyday activity for a computer scientist. Logic can be useful for accommodating the construction of semantic models for the theories developed by a computer scientist, and for providing mechanisms that amalgamate, update or revise this semantics in a way that matches the amalgamation, update

*Correspondence author. t.dimitrakos@rl.ac.uk, *www.itd.clrc.ac.uk/Person/T.Dimitrakos*

[1] It is unrealistic to assert that computer science is reducible to logic, as it is illusory to maintain that physics is reducible to calculus.

or revision of the computer scientist's theories.[2] In order for logic to be successful in supporting computer science, methods for enhancing useful deductive systems with meta-logical properties that facilitate efficiency in scientific reasoning, modular axiomatic presentation and proof decomposition have to be pursued further. In this paper we advocate that such meta-logical properties have to include interpolation, in general, and the possession of certain uniform splitting interpolants, in particular.

A summary of original key results relating interpolation to modularization and other practically useful mechanisms such as proof decomposition and presentation modularity is presented in section 2.

As a motivating case study, we provide an overview of the formal support required by an industrial strength formal method in subsection 5. Finally in section 6 we describe a general approach to overcoming the difficulties caused by the potential absence of some critical uniform interpolants in a specification logic and emphasise the need for developing concrete methods to support it. This will require a closer collaboration between information systems engineers, theoretical computer scientists and logicians.

2 Interpolation in specification theory

In this section we provide a summary of key results linking interpolation to modularization, information hiding, and other practically useful concepts such as presentation modularity and proof decomposition in the verification of complex system designs.

In addition to the work of logicians, interpolation has, sporadically, gained the attention of some groups of researchers in formal methods and theoretical computer science. Some of these groups have contributed useful results – most of them related to applications of interpolation-like properties in modular specification.

First, interpolation is the most critical condition for engineering a sound and complete compositional calculus to reason *about specifications* over a given algebraic specification formalism. Algebraic specification [43, 49, 42] are defined over any logical system which can be represented as an *Institution* [28]. For each of these formalisms there is a sound compositional calculus for reasoning about specifications (cf. [13] and [14]). Amongst the conditions that make such a calculus complete (with respect to the underlying logical system) the most critical is a generalized interpolation property [13].

Second, an equivalence between a generalisation of a "splitting" variant of interpolation and a generalisation of the so called *"modularization"* property (ie., stability of theorem conservation under amalgmation) has been established in [20] for every (structured) family of reflexive, transitive and monotonic entailment relations. Such an equivalence was earlier conjectured in [35] (see also [32]

[2] This semantics can be conceived as defining a correspondence (interpretation) between the computer scientist's theory which can be rigorous but need not be formal and a logical theory over a deductive system which is formal.

non-uniform	For all ($\alpha \in L_1$ and $\beta, \varphi \in L_2$) there is $\vartheta \in L_0$ such that
uniform	For all ($\alpha \in L_1$ and $\beta \in L_2$) there is $\vartheta \in L_0$ such that for all $\varphi \in L_2$
if $e'(\alpha), i'(\beta) \Rightarrow i'(\varphi)$ then $\alpha \Rightarrow e(\vartheta)$ and $i(\vartheta), \beta \Rightarrow \varphi$ where $L_1 - i' \rightarrow L_3 \leftarrow e' - L_2$ is the amalgamation (pushout) of $L_1 \leftarrow e - L_0 - i \rightarrow L_2$	
pushout	(e, i) is an arbitrary pair of translations.
simple	(e, i) are both language expansions, $L_3 = L_1 \cup L_2$, $L_0 = L_1 \cap L_2$.
ordinary	β does not appear in the premises.
splitting	β is a secondary assertion.
common	$\alpha, \beta, \varphi, \vartheta$ are sentences.
generalised	α, β, ϑ are sets of sentences, φ is a sentence.
arrow	$\Rightarrow$ denotes an implication connective.
turnstile	$\Rightarrow$ denotes entailment.

Figure 1: The various variants of interpolation emphasise on different aspects of a common meta-logical property.

and [33]). Detailed proofs for unsorted first order logic were given in [47] and in [48]. An analogously strong correspondence between ordinary interpolation and amalgamation of models (which leads to the stability of conservativeness in the amalgamation of a span of conservative extensions - a special case of Modularisation) had been observed earlier in [36], where an equivalence was shown to hold for *every* (propositional) super-intuitionistic logic. (See also [25, 26, 39].) Finally, an equivalence between pushout interpolation, "splitting" interpolation and ordinary interpolation for compact entailments with a deduction theorem and finite syntax[3] has also been shown in [20]. A taxonomy of these variants of interpolation for reflexive, monotonic and transitive entailment relations over a finite syntax is depicted in Figure 1.

Third, modularization facilitates

1. the composibility of a form of refinement for axiomatic specification, c.f. [46, 48, 17];

2. the correct instantiation of (multi)parametric axiomatic specifications of data types, c.f. [19];

3. the mixed-variance refinement of parameterisations which unfolds to a (contravariant) parameter instantiation followed by a (covariant) non-parameterised refinement c.f. [18] discussing [44];

4. the proof of correctness for various operations of module algebras, c.f.[40] discussing an earlier version of [8];

5. the orthogonality of presentation when sharing specification text following an "one writer, many readers" architecture in state based specification

[3] Where grammatical expressions correspond to inductively defined finite strings of symbols and translations enjoy a mono-epi factorisation. See [18] and [20] for further details.

languages such as B, Z and VDM, c.f. [21] discussing [2, 30] and the contextual proof obligations generated by B-Toolkit [6];

6. the promotion of data refinement by some schemata in the Z formal method, c.f. [21] discussing [34, 50].

3 Localisation and uniform interpolants

In many of the studies that we refer to in section 2, the following facts about the employed interpolants have not been considered.[4].

1. *Interpolation can be relativised over syntactical loci*, thus allowing the study of interpolants in calculi that do not posses interpolation globally.

2. *Uniform interpolants are more appropriate for use in modular specification* than ordinary (splitting) interpolants.
   This is because uniform interpolants depend only on the assertions and the shared language; they are independent of the conclusion. As such, they assist in decomposing finite axiomatisations as ordinary interpolants assist in decomposing derivations.

Possessing ordinary (splitting) interpolation facilitates various activities related to specification modularity. These activities include the modular structuring, the orthogonal specification of components and component refinements, the promotion of valid refinements from the components to the composite specification. Furthermore, a uniform presentation of the interpolants facilitates the specification of submodules, the generation of proof obligations which are sufficient for validating the correctness of refinements and the correctness of those structuring mechanisms that allow building a subsystem from encapsulated components which are available for blind use elsewhere. (See also chapters 4 and 5 of [18] and [20].) There is not much use in assisting in "correctly" composing implementations, if one is not sufficiently supported in detecting, verifying, classifying or constructing "correct" implementations. Possessing interpolants does not always provide adequate support for designing and developing formal specifications, unless a uniform presentation of these interpolants is available. Possessing (at least some critical) uniform interpolants is therefore important.

There is no need, though, to require that the employed deductive system possesses (uniform) interpolation globally. For reasons which depend on the notation that is used for specification and on the application at hand, it is very common that the user provided specifications utilise only a fragment of the expressive power provided by the underlying deductive system.[5] It therefore suffices that the employed deductive system provides sentences which *act* as

[4] These facts were first indicated, recently, in [18] and [17, 20]. Their practical implications in specification theory were indicated in [17, 18] for logical/algebraic specification. They are further exemplified in [21] for state based specifications.

[5] As a syntax in mathematical logic often accommodates only a fragment of what can be stated in the underlying model theory.

(uniform) interpolants for derivations in this fragment. As we illustrate in the following section there are interesting cases where the above mentioned sentences interpolate entailments between sentences in a fragment of the employed logic while being outside this fragment themselves.

4 Further practical applications

In the following paragraphs we indicate potential applications of interpolants for controlling non-interference in concurrent and object-based specification and controlling information in the hierarchical design of secure systems (in the sense of confidentiality) using process based models. Unfortunately, in most of these cases the role of interpolants has not received enough attention and a common syntactic form of the critical interpolants has not been yet identified. Though, both non-interference between (sub-)operations that share state and no-information-flow between high and low security components are related to the presence of a (power) bisimulation between the corresponding operations or traces, c.f. [11] and [41]. Notably, a strong connection between entailment along bisimulation and interpolation has recently been brought into attention , c.f. [9, 37, 3]. In particular, Barwise and van Benthem investigated in [5] some weaker forms of interpolation for infinitary logics and proved that the existence of such an interpolant for a given pair of formulae (α, β) is equivalent to that α *entails* β along potential isomorphisms (viz. bisimulations) on the common language. We intend to further investigate the potentially latent relationship between the existence of such interpolants on the shared part of two component specifications and non-interference in concurrent and state based specification.

Non-interference between (sub-)operations provides a key mechanism in the compositional development of operations that share state within a module [11]. The read and write frames in VDM implicit operations definitions can be understood as constraints on the read and write accesses which can be made by valid implementations [10]. A strong relation between non-interference and the existence of a (strong) bisimulation between (sub-)operations has been established in [11]. Non-interference is equally important in concurrent system specification. A mechanism based on roles describing rely/guarantee conditions has been developed by Jones [29] and Stølen [45] in order to reason about concurrent rather than modular sharing in a VDM/LPF-like framework.

Interpolation may also assist in the detection of information flow between a High security system and a Low security subsystem or interface. In general, provided that High is a specification of a security critical system and Low is the publicly available specification of a (visible) subsystem, High is secure if there is no visible behaviour of the system which betrays a property that was not meant to be revealed. Formally, this amounts to assuring that every interpolant of High to the visible language is deducible from the specification of Low. Then High can reveal at most what is already deducible by analysing Low. If, in particular, a process-based or trace-based approach is employed then the interpolant sentences describe the impact of the High system's computations to the context

of the Low subsystem. This impact captures information that flows from High to Low. By establishing that all these interpolants are potential behaviours of Low, one establishes that the overall system does not betray to the intruder any distinguishable behaviour, i.e. that there is no information flow from High to Low.

The usefulness of interpolants in assisting to identify testing equivalence can emerge from a similar viewpoint. The philosophy is strikingly similar: in [1] tests can be understood as encapsulating all possible attacks, and equivalence under testing establishes that no such attack can succeed in distinguishing a real system from an ideal one. As is argued in [41] tests can alternatively be considered to encapsulate all possible ways in which information may flow from High to Low (as a result of High-communication, Low-elicitation, some collusion involving shared data, etc.) and equivalence under testing establishes that no such way can succeed.

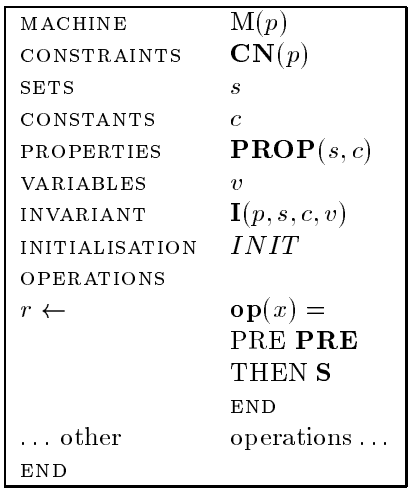
We further explain the role of interpolants and modularization in supporting the promotion of property enrichment from a component to a subsystem (i.e., an aggregate of components) and in controlling interference by means of a case study that is provided in section 5. This case study focuses on the internal structure of elementary specification units and on the (external) compositional structuring assemblies that are used to specify shared data in the B-Method. The choice of the B-Method as an example of a useful application of interpolation has been made merely for two reasons. First, it has been taken-up by the industry with considerable success[6] and there are commercial tools to support it ([4] and [6]). Second, the uniform splitting interpolants appear without much disguise, almost in their common $\sum_1^1$ (or $\prod_2^1$) meta-forms.

5 Case Study: *Interpolation in Industrial Strength Formal Methods*

Typically, an industrial strength formal method is expected to be equipped with an unambiguous notation, a formal semantics, and a collection of tools which support specification construction, layered implementation and proof of correctness. The built-in constructs of the method are realised (interpreted) into a logical theory over a chosen deductive system, which we may call the *"base theory"* (BTh). The B-Method [2, 30] is a typical example of an industrial strength formal method which is supported by commercial tools. The basic theory of the B-Method is the amalgamated union of a first order set theory, a first order theory of natural numbers and a first order theory of *generalised substitutions.*[7]. In order to facilitate specification, it is also equipped with a col-

[6] The safety critical components of the first driverless metro in Paris have been developed using the B-Method in all phases (c.f. the **Météor** project [7]).

[7] This set theory is a subtheory of ZFC which focuses on reasoning with arbitrarily large finite sets and a single infinite set BIG. Most importantly, this set theory *lacks* the Replacement Axiom, the Pairing Axiom and the Foundation Axiom and uses a specific instance of the Axiom of Choice (c.f. Chapter 2 of [2]). The notion of an ordered pair is defined in B outside set



IC1. $\exists p.\mathbf{CN}$
IC2. $\mathbf{CN} \Rightarrow \exists(s,c).\mathbf{PROP}$
IC3. $\mathbf{CN} \wedge \mathbf{PROP} \Rightarrow \exists(v).\mathbf{I}$
IC4. $\mathbf{CN} \wedge \mathbf{PROP} \Rightarrow [INIT]\mathbf{I}$
IC5. $\mathbf{CN} \wedge \mathbf{PROP} \wedge \mathbf{I} \wedge \mathbf{PRE} \Rightarrow [S]\mathbf{I}$

IC1 asserts that there are possible machine parameter values that satisfy the specified constraints.
IC2 asserts that, assuming the machine constraints on the machine parameter, there are SETS and CONSTANTS (i.e. auxiliary types, functions and predicates) satisfying the specified properties.
IC3 asserts that, assuming the specified constraints on the machine parameters and the properties of the constant identifiers, there is at least one machine state satisfying the invariant.
IC4 and IC5 give the base case and induction step of a proof that all reachable states satisfy the invariant.

Figure 2: The general pattern of an abstract machine specification and the corresponding internal consistency proof obligations.

lection of pre-specified data types such as (an arbitrarily large finite fragment of) integers, enumerable sets, sequences and trees. In [21] we provide an elaborate analysis of the compositional structuring assemblies of the B-Method from a logical view-point. This analysis exposes a wide use of uniform interpolants in the proof obligations generated by the B-Toolkit in order to prove the internal consistency of the B specification units, on the one hand, and the correctness of some structuring assemblies of the B-Method, on the other hand. In this section we review some selected examples where interpolants are most relevant. See [21] for further details.

5.1 Internal consistency of specification units

A typical elementary component specification in B, called an "abstract machine" (AM), has the general form depicted in Figure 2. The formal semantics of AM are given by an extension of BTh which incorporates a *static* part specifying the parameters, data types, auxiliary functions and possible states, and a *dynamic*

theory, and type-checking all set theoretic statements ensures that ordinal numbers are not dealt with, and that statements of the form $\exists x.x \in x$ (or their negation) are not a part of the allowed discourse. The employed first order theory of natural numbers it proves Peano's axiomatisation of arithmetic (c.f. Section 3.5.2 of [2]), but defines 0 as BIG - BIG, the successor using BIG and choice, and induction using Knaster and Tarski's fixpoint theorem (c.f. Chapter 3 of [2]). Generalised substitutions give rise to a concept similar to Dijkstra's "predicate transformers" [22].

part specifying the initialisation of the AM and the operations by means *generalised substitutions* that relate "before" and "after" state similarly to predicate transformers (c.f. [30] and Chapter 5 of [2]). An elementary AM specification is *internally consistent* if the proof obligations presented Figure 2 are satisfied. The notion of internal consistency is analogous to the notion of soundness in logic; if these proof obligation can be discharged then the AM specification has some "meaningful" model in B, c.f. [30]. As we elaborate in [21], the creation of a formal semantics for an AM specification over the underlying deductive system corresponds to a chain of extensions of BTh. Proof obligations IC1–IC3 ensure that each extension in the chain is conservative. This is achieved by requiring that the uniform interpolant of the extension axioms is in the theory to be extended. The conservativeness of these extensions introduces a relative consistency and a relative completeness argument. Firstly, every theory in the chain is consistent iff its predecessor is consistent. Hence, the logical consistency of the semantics of an AM is reduced to the logical consistency of BTh. Secondly, they ensure that the logical theory describing the semantics of an AM is constructed in a modular fashion.[8]

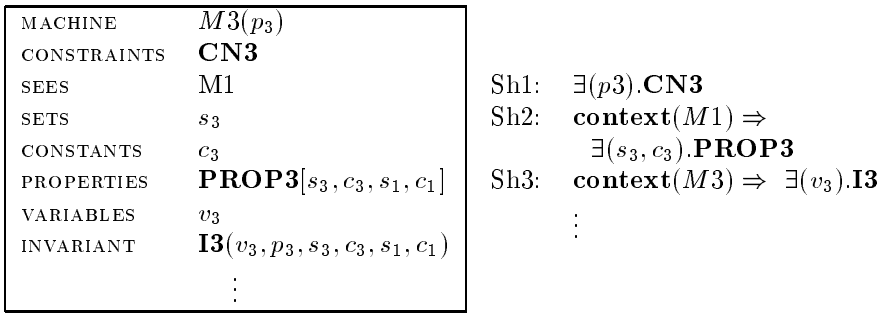
5.2 System-wide data types and inquiry-only sharing

Very often, when large specifications are structured incrementally, the semantics of a compound specification describing a subsystem need not be conservative over the semantics of the components. Indeed, various assembly primitives such as INCLUDES, EXTENDS and REFINES amount to non-conservative extensions denoting a (non-conservative) transition from one chain of conservative extensions of BTh to another, c.f. [21] . However, the modeling of certain architectures, where a subsystem is allowed to "read" or to "inquire" but not to "write" or implicitly modify the state of a component, invokes some form of (occasionally partial) conservativeness. More generally, "closed" assemblies involve some from of partial conservativeness between the underlying (chains of conservative extensions of) logical theories whereas conservativeness (between such chains) is not an issue with "open" assemblies.[9]

SEES is a typical example of a structuring assembly where no implicit or explicit modification of the component specification is premited. SEES allows the sharing of an AM and can be used in AM a refinement or an implementation. The state of the seen component AM can be consulted, but not modified, by the seeing module. Most often, the SEES primitive is used in refinements and implementations for the purpose of sharing code and providing *inquiry-only* access to shared data. The intention is that the code implementing the AM that is seen

[8] That is, all properties about the built-in primitives of the method are those provable from BTh, all properties about the parameters are provable from BTh+**CN**, all properties about the static context are provable from BTh+**CN** + **PROP** and all global properties about the machine state are provable from BTh+**CN** + **PROP** + **I**.

[9] These terms refer to the *open-closed* duality principle (e.g. Meyer [38]). "Open" means building larger systems by extensions, e.g. when appending or amalgamating abstract specification modules. "Closed" means building an encapsulated component available for blind use elsewhere, e.g. when linking independently constructed code modules.



Note: We write **context**$(M3)$ as a shorthand for **context**$(M1) \wedge$ **CN3** $\wedge$ **PROP3**.

Figure 3: The general form of the context-related clauses and proof obligations for the SEES and USES primitives.

will be linked only once, thus establishing an *"one writer, many readers"* sharing scheme. Another common use of SEES is for *sharing separately implemented*, system-wide data types. The specification of such data types is provided in a *stateless* machine. Importing such a machine only once in a development, and seeing it many times, ensures that a single copy of code will be present in the final product. A special case of this is to specify abstract (mathematical) data types defined using static context of an operationless shared machine. Such machines can be seen by any other machine. Such machines may not need implementing; they provide a library of useful mathematical concepts that will ease the specification of algorithms and architectures, and can be "programmed away" during the development.

In the case of SEES, the static context-related proof obligations take the form provided in Fig. 3. Sh2 guarantees that the logical theory describing the properties of the constant fragment of $M3$ is conservative over the corresponding component of $M1$. This is obtained by proving the *uniform interpolant* $\exists(s_3, c_3).$**PROP3**(s_1, c_1, s_3, c_3) of **PROP3** in the sublanguage generated by the CONSTANTS of $M1$. In addition, Sh3 guarantees that the logical theory of the static context of $M3$ is conservative over the logical theories describing the stateless fragment of $M3$, the stateless fragment of $M1$, and the whole context of $M1$. Therefore the properties of the seeing machine cannot impose any further *"emerging"* properties on the CONSTANTS of the seen machine. This is fundamental for the following reasons.

1. The seen machine $M1$ may be consulted from machines other than $M2$ and any *emerging properties* from $M2$ may have unpredictable side-effects on the operation of those machines by implicitly enriching their properties clause with the potential of creating conflicts or inconsistencies.

2. If $M1$ is enriched (viz. refined) in a development, such a modification takes place on the only shared copy of $M1$ and the only properties about s_1, c_1 taken into consideration are those specified within $M1$; any *emerging*

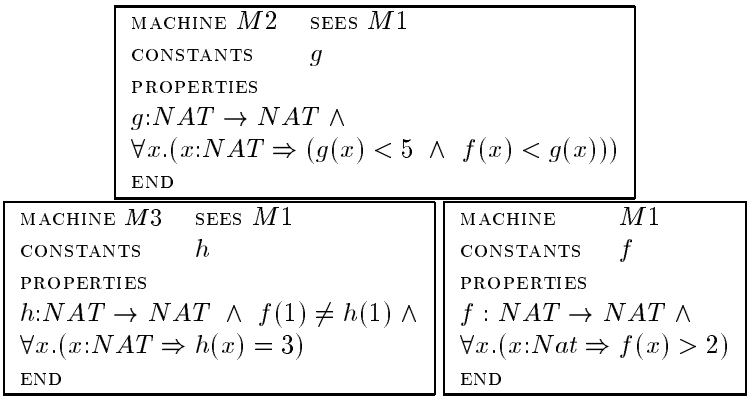


Figure 4: The specification examples of subsection 5.2.

properties cannot be considered.

3. The seen machine $M1$ will be implemented separately and in such an implementation only the properties about the sets and CONSTANTS s_1, c_1 are considered. If *emerging* properties on the CONSTANTS of $M1$ had been allowed these will not be considered by the implementation therefore causing incompatibilities in parallel development.

Given that Sh2 holds, then Sh3 is equivalent to the corresponding proof obligation of a machine $M3'$ whose context is the result of the enrichment of $M3$ with the CONSTANTS and properties of $M1$. Notice that the parameters p_1 and the state variables v_1 of the seen machine do not appear in **I3**. This "hiding principle", together with the context-related proof obligations, ensures the conservativeness of **I3**.[10]

The following example demonstrates the importance of the context-related proof obligations associated with the SEES primitive. Assume that AM M1 is seen by two other AMs M2 and M3 as presented in Fig. 4. Although each of M2 and M3 extend M1 in a consistent way, they induce contradictory emerging properties on M1. On the one hand, M2 *alters* the context of M1 by forcing the f to accept only one possible model interpretation, namely the constant function $f(x) = 3$. On the other hand, M3 *alters* f by accepting only those model interpretations where $f(1) > 3$.

In fact, both M2 and M3 are ill defined. Because they implicitly *modify* the static context of M1 by imposing (in this example conflicting) emerging properties. In order to avoid such side-effects when a machine M SEES a machine

[10] Because of the modularization property, the implication $\mathbf{context}(M3) \Rightarrow (\mathbf{I1} \Rightarrow \exists(v_3).\mathbf{I3})$ reduces to $\mathbf{context}(M3) \Rightarrow \exists(v_3).\mathbf{I3}$, by the definition of $\mathbf{context}(M3)$ (Fig. 3) and because $\mathbf{context}(M1) \Rightarrow \exists(v_1).\mathbf{I1}$, by the assumption that $M1$ is internally consistent, and v_1 does not appear in **I3** by the syntactical conditions of SEES. In fact, with an analogous argument one can also drop **CN1** from the assumption in Sh3-SEES.

M1, the context of M must be conservative on the context of M1. That is, all sentences about the SETS and CONSTANTS identifiers of M1 that are provable in the context of *seeing* machine M should also be provable in the context of the *seen* machine M1. The latter is the case if and only if the sentences $\exists g.(g{:}NAT \rightarrow NAT \wedge \forall x.(x{:}NAT \Rightarrow (g(x) < 5 \ \wedge \ f(x) < g(x))))$ and, respectively, $\exists h.(h{:}NAT \rightarrow NAT \wedge \forall x.(x{:}NAT \Rightarrow h(x) = 3) \wedge f(1) \neq h(1))$ follow from the context axioms of M1. Clearly, in this example, none of the above mentioned proof obligations can be discharged.

5.3 Using interpolants to control information flow

Whenever the absence of any emerging properties from a subsystem to a component is required, the automatically generated proof obligations (which validate the conservativeness of the associated extension) follow the same pattern: $\mathsf{BTh} \vdash \mathbf{context} \rightarrow \Theta(\mathbf{extension})$ where **context** is the formula describing the axiomatisation of the component, **extension** is the conjunction of the extension axioms and $\Theta(\mathbf{extension})$ is the uniform interpolant of **extension** in the language of the component. As is elaborated in [21], such uniform interpolants exist in the logic that provides formal semantics to the B-Method. In a few words, this is because the incorporation of a deductive presentation of set theoretic membership, and appropriate relativization predicates, allow all the identifiers declared in the static context signature of a component to be treated as set theoretic constants (possibly indexed by state). So, $\Theta(\mathbf{extension})$ can be understood as the first order reduction of a general second order $\sum_1^1$ sentence: the quantifiers inside **context** and **extension** quantify over individuals whereas the existential quantifiers in the prefix of $\Theta(\mathbf{extension})$ quantify over identifiers.[11] Such sentences are *syntactically* and *ontologically* different from the first order existential sentences that can appear in a user-provided AM specification. On the one hand, they are not relativised to a domain of individuals and, on the other hand, they cannot appear as user provided axioms in a specification; their purpose in a formal development is to facilitate feasibility and non-interference proofs. See [21] for further details. If the conservation of all behaviours described by an operation is to be established then the corresponding uniform interpolant $\Theta(op)$ can be conceived as the first order reduction of a $\prod_2^1$ (i.e. universal-extistential) sentence where the "before" state extension is abstracted away by a universally quantified variable and the "after" state extension is abstracted away by an existentially bounded variable.

[11] The main idea behind this reduction is that, unlike the usual ("absolute") second order logic where basic notion such as "being a subset" are fixed primitives in the model theory, general second order logic avoids appealing to a fixed notion of such primitives and is thus reducible to first order logic enriched with (a first order presentation of) set theoretic membership and appropriate relativisation predicates. General second order generates a recursive enumerable set of valid sentences which is a proper subset of the non-enumerable set generated by "absolute" second order logics. See [21, 18] and Chapter 4 of [23] for further details.

6 (Re)engineering logics to support formal design

So far in this paper, we referred to a variety of applications of interpolation in computer science and emphasised some cases where uniform (splitting) interpolants were the protagonists. In recognition of the fact that a variety of, otherwise useful and well understood, logics lack uniform interpolants, we conclude this position paper by outlining a general approach to alleviating the potential problems caused by the use of such formalisms in the absence of uniform interpolation and advocate the need for developing methods that will assist in realising such an approach.

The obvious question to ask if one's favourite formalism does not possess a crucial modularity property is how to "fix" it. In general, there are two obvious possibilities: either extend the formalism to one which has some form of the property, or alternatively, restrict the specifications which are "acceptable" to a subclass which enjoy this property. Let us consider the first design choice (expansion) and view the appropriate restriction of "acceptable" specifications retrospectively. First, we need to place uniform interpolation in the perspective of the uniform extension of logical systems. A simple case of such an extension is obtained when extending an entailment $\vdash_{\mathsf{SP}}$ conservatively to an entailment $\vdash_{\mathsf{DES}}$ by adding new logical connectives and without changing the corresponding (extralogical) alphabets.[12] Now, assume that $\vdash_{\mathsf{SP}}$ lacks *uniform* (splitting) interpolation and that $\vdash_{\mathsf{DES}}$ provides uniform interpolants for derivations between $\vdash_{\mathsf{SP}}$sentences. One can use the simpler and perhaps more efficient $\vdash_{\mathsf{SP}}$ for presenting the formal semantics of the specifications and reasoning with requirements and recourse to $\vdash_{\mathsf{DES}}$ in order to generate and attempt to discharge proof obligations that are related to modular structuring as well as the control of interference and any information flow from subsystems to components.

In order to realise such a layered model of logical support computer scientists and logicians need to establish need to jointly investigate *concrete* strategies for extending commonly used specification logics to logics enhanced with useful meta-logical properties such as the possession of uniform splitting interpolants for derivations in the "specification logic" fragment. The results of a preliminary research towards a methodology for enhancing a calculus with a uniform presentation of interpolants whenever these exist are described in chapter 6 of [18]. This approach extends $\vdash_{\mathsf{SP}}$ to $\vdash_{\mathsf{DES}}$ by introducing new logical operators which combine a dynamic modality denoting (extra-logical) language expansion and generalised quantifiers which can abstract away the atoms introduced along the language expansion. Given an alphabet L_0 and a sentence α in an extended alphabet $L_1 \supseteq L_0$, the intention is to enable stating using the source alphabet L_0 a common premise ϑ_0 of all L_0-theorems that are forced by α. This common premise is obtained by abstracting away all atoms that have been introduced along the (extra-logical) language expansion[13]. Further on-going research fo-

[12] The term "conservatively" does not mean that $\vdash_{\mathsf{DES}}$ is necessarily reducible to $\vdash_{\mathsf{SP}}$ but that for all sentences φ, ψ where new connectives do not appear $\varphi \vdash_{\mathsf{SP}} \psi$ iff $\varphi \vdash_{\mathsf{DES}} \psi$.

[13] The approach of [18] presents similarities to the schematic variables discussed in [31] and Feferman's view (expressed in [24], page 8) that the extension of schemata beyond an original

cuses on the generalisation of the algorithm for constructing interpolants in goal directed proof systems which is outlined in chapter 2 of [27]. Our aim is to develop an algorithmic method which will

1. succeed in producing an common interpolant of a given derivation, if an interpolant exists and

2. otherwise provide useful information about any missing connectives whose introduction can restore interpolation locally.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 1999.

[2] J.R. Abrial. *The B-Book : Assigning Programs to Meanings.* Camb. Univ. Press, 1996.

[3] C. Areces and M. de Rijke. Interpolation and bisimulation in temporal logic. In *Proceedings of WoLLIC'98. Workshop of Logic, Language, Information and Computation*, pages 15–21, São Paulo, Brazil, July 1998. IME/USP.

[4] Steria Méditerranée. Atelier B. S.A.V. Steria, BP 16000, 13791 Aix-en-Provence cedex 3. France.

[5] J. Barwise, J.F.A.K. van Benthem. Interpolation, Preservation and Pebble Games. *Journal of Symbolic Logic 64(2)*, pages 881-903, 1999.

[6] B-CORE (UK) Ltd. The B-toolkit. Kings Piece, Harwell, Oxon OX11 0PA, U.K. 1999. URL: http://www.b-core.com.

[7] P. Behm, P. Benoit, and J.M. Meynadier. Météor: A Successful Application of B in a Large Project. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods*, volume I, pages 369–387, 1999.

[8] J.A. Bergstra, J. Heering, and P. Klint. Module algebra. volume 37, pages 335–372, 1990.

[9] Johan van Benthem. Modality, bisimulation and interpolation in infinitary logic. *ANNALSPAL: Annals of Pure and Applied Logic*, 96, 1999.

[10] J.C. Bicarregui. Algorithm Refinement with Read and Write Frames. In Woodcock and Larsen, editors, *FME'93*. Sringer-Verlag, 1993. LNCS 670.

[11] J.C. Bicarregui. *Intra-Modular Structuring in Model-Oriented Specification: Expressing Non-Interference with Read and Write Frames.* PhD thesis, University of Manchester, 1995. (UMCS-95-10-1).

[12] T. Borzyszkowski. Correctness of a logical system for structured specifications. *Recent Trends in Algebraic Development Techniques, WADT'97 Seleted Papers. Lecture Notes in Computer Science 1376*, pages 107–121, 1998.

[13] T. Borzyszkowski. Completeness of the logical system for structured specifications. *Lecture Notes in Computer Science 1589*, pages 16–30, 1999.

[14] M.V. Cengarle *Formal Specification with High-Order Parameterization.* PhD thesis, Institute for Informatics, Ludwig-Maximilians Unversity, Munich, 1994.

[15] M.V. Cengarle and A.M. Haeberer. Towards an epistemology-based methodology for verification and validation testing. Institut für Informatik Ludwig-Maximilians-Universität München Oettingenstr. 67, 80538 München, Germany, January 2000.

fixed language *"corresponds to a more fundamental view of what it means to accept some axiomatic principles."*

[16] Răzvan Diaconnescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gérard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge University Press, 1993.

[17] Th. Dimitrakos and T.S.E. Maibaum. Notes on refinement, interpolation and uniformity. In *Automated Software Engineering–ASE'97, 12th IEEE International Conference*, 1997.

[18] Theodosis Dimitrakos. *Formal support for specification design and implementation.* PhD thesis, Imperial College, March 1998.

[19] Theodosis Dimitrakos. Parameterising (algebraic) specifications on diagrams. In *Automated Software Engineering–ASE'98, 13th IEEE International Conference*, pages 221–224, 1998.

[20] Theodosis Dimitrakos and Tom Maibaum. On a generalised modularisation theorem. *Information Processing Letters*,74(1-2):65-71, 2000.

[21] Th. Dimitrakos, J.C. Bicarregui, B.M. Matthews, T.S.E. Maibaum. Compositional Structuring in B: A Logical Analysis of the Static Part. In *ZB'2000. Proceedings of the first International Conference of B and Z Users.* LNCS. Springer-Verlag. To appear.

[22] E.W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[23] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[24] Solomon Feferman. Reflecting on incompleteness. *Journal of Symbolic Logic 56*, pages 1-49, 1991.

[25] D.M. Gabbay. Sematnic proof of the craig interpolation theorem for intuitionistic logic and extensions. In R.O. Gandy and C.M.Y. Yates, editors, *Logic Colloquium'69*, Amsterdam, 1971. North-Holland.

[26] D.M. Gabbay. Craig's interpolation theorem for modal logics. In W. Hodges, editor, *Porceedings of Logic Conference, London, 1970*, volume 255 of *Lecture Notes in Mathematics*, pages 11–127, Berlin, 1972. Springer-Verlag.

[27] Dov Gabbay and Nicola Olivetti. *Goal-Directed Proof Theory.* Pre-print.

[28] J.A. Goguen, R.M. Burstall. Institutions: Abstract Model Theory for Specification and Programming, Journal of the *ACM 39(1)*, pages 95-146, 1992.

[29] Cliff B. Jones. Accomodating interference in the formal design of concurrent object-based programs. *Formal Methods in System Design 8(2)*, pages 105-122, March 1996.

[30] Kevin Lano. *The B Language and Method. A Guide to Practical Formal Development.* Springer-Verlag, 1996.

[31] Shaughan Lavine. *Understanding the Infinite.* Harvard University Press, second edition 1998.

[32] Renardel de Lavalette. Modularisation, parameterisation and interpolation. *J. Inf. Process. Cybern. EIK* 25(5/6), pages 283–292, 1989.

[33] Renardel de Lavalette. Logical semantics of modularisation. In *CSL: 5th Workshop on Computer Science Logic.* LNCS, Springer-Verlag, 1991.

[34] P.J. Lupton. Promotin Forward Simulation. In J.E. Nicholls, editor, *Z User Workshop*, pages 27–49. Springer-Verlag, Oxford 1990.

[35] T.S.E. Maibaum and M.R. Sadler. Axiomatising specification theory. In H.J. Kreowski, editor, *Recent Trends in Data Type Specification*, pages 171–177. Springer-Verlag, Berlin, 1985.

[36] L.L. Maksimova. Craig's interpolation theorem and amalgamable varieties. *Soviet Math. Dokl.* 18(6), pages 1550–1553, 1977.

[37] Maarten Marx and Carlos Areces. Failure of interpolation in combined modal logics. *Notre Dame Journal of Formal Logic*, 1999. To appear.

[38] B. Meyer. *Object Oriented Construction.* Prentice-Hall, 1988.

[39] A. M. Pitts. Amalgamation and interpolation in the category of Heyting algebras. *Jour. Pure and Appl. Algebra* 29, pages 155–165, 1983.

[40] P.H. Rodenburg and R.J. van Galbbeek. An Interpolation Theorem in Equational Logic. Technical Report CS-R8838, Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands, 1988.

[41] P.Y.A Ryan and S.A. Schneider. Process algebra and non-interference. In *PCSFW: Proc. of The 12th Computer Security Foundations Workshop*. IEEE Comp. Soc. Press, 1999.

[42] D. Sannella, S. Sokołowski, A. Tarlecki. Toward Formal Development of Programs from Algebraic Specifications: Parameterisation Revisited,*Acta Informatica 29(8)*, pages 689–736, 1992.

[43] D. Sanella and A. Tarlecki. Specifications in an arbitrary instituion", Journal of *Information and Computation*, pages 165–210, 1998

[44] Yellamraju V. Srinivas. Refinement of parameterized algebraic specifications. In *IFIP TC2 Working Conference on Algorithmic Languages and Calculi*. Chapman & Hall, February 1997.

[45] Ketil Stolen. *Development of Parallel Programs on Shared Data-Structures*. PhD thesis, University of Manchester, 1990. Available as a technical report UMCS-91-1-1.

[46] Władysław M. Turski and Thomas S. E. Maibaum. *The Specification of Computer Programs*. Addison-Wesley, 1987.

[47] P.A.S. Veloso. A New, Simpler Proof of the Modularisation Theorem for Logical Specifications. *Bulletin of the IGPL 1(1)*, pages 3–12, 1993.

[48] P.A.S. Veloso and T.S.E. Maibaum. On the modularisation theorem for logical specifications. *Information Processing Letters 53*, pages 287–293, 1995.

[49] Martin Wirsing. Algebraic Specification: Semantics, Parameterization and Refinement. *Formal Description of Programming Concepts*. IFIP, State-of-the-Art Reports, Springer 1991.

[50] J.C.P. Woodcock. Mathematics as a Management Tool: Proof Rules for Promotion. In *CSR Sixth Annual Conference on Large Sofware Systems. Bristol*, 1989.