Compositional structuring in the B-method: a logical viewpoint of the static context

Theo Dimitrakos, Juan Bicarregui, Brian Matthews, Tom Maibaum

Published version information

Citation: T Dimitrakos et al. 'Compositional structuring in the B-method: a logical viewpoint of the static context.' Lecture Notes in Computer Science, vol. 1878 (2000): 107-126. Is in proceedings of: International Conference of B and Z Users (ZB 2000), York, United Kingdom, 29-Aug-2000 to 02-Sep-2000.

DOI: 10.1007/3-540-44525-0_8

Compositional Structuring in the B-Method: A Logical Viewpoint of the Static Context

Theo Dimitrakos[1]*, Juan Bicarregui[1], Brian Matthews[1], and Tom Maibaum[2]

[1] ISE–CLRC, Rutherford Appleton Laboratory, OXON, OX11 OQX, U.K.
[2] Department of Computer Science, King's College, London, WC2R 2LS, U.K.

Abstract. The B-Method provides a collection of structuring mechanisms which support information hiding, modularisation and compositionality of module operations, although, in order to achieve compositionality and independent (parallel) refinement, sharing is restricted in B. In this paper we elaborate some non-interference and compositionality assumptions that underlie structuring mechanisms such as USES, SEES and IMPORTS and show how they may be violated by inducing emerging properties which alter the *context* of the used, seen or imported machine. We discuss how such situations can be avoided by considering necessary and sufficient conditions for logical conservativeness and modularisation. As proof obligations, these conditions ensure that the properties of the *context* of the seen, used or imported component are *conserved*, i.e. that they are preserved but not enriched. From a logical viewpoint, these proof obligations require that the uniform interpolant of the contextual extension axioms is implied by the base context.

1 Introduction

One strength of the B-Method [1] is its support for modular structuring. It provides a collection of structuring mechanisms which support information hiding, modularization and the compositionality of module operations, reuse and proof decomposition [10, 6, 7, 22]. In order to achieve compositionality and independent (parallel) refinement, sharing is restricted in B.

Important work in explaining and harmonizing the structuring mechanisms of B and their interplay with refinement has been done by Bert, Potet, and Rouzaud in [4] and [25, 26]. In [4], they focus on the INCLUDES and USES primitives which underlie the structuring of abstract specifications and in [25] they focus on the SEES and IMPORTS primitives which underlie the structuring of layered implementations. Mechanisms for supporting composition have also been proposed for extensions of B, as well as in methods closely related to B. Büchi and Back propose in [9] an extension of B with a compositional symmetric sharing mechanism based on roles describing rely/guarantee conditions. This mechanism applies, to sequential systems with shared components, rely/guarantee conditions that had

* Correspondence author. *e*mail: t.dimitrakos@rl.ac.uk, tel.: +44 1235 446387, fax: +44 1235 445381, WWW: http://www.itd.clrc.ac.uk/T.Dimitrakos

been developed by Jones [19] and Stølen [28] in order to reason about concurrent rather than modular sharing in a VDM-like logic and syntax. Z provides assembly primitives to facilitate the union of schemata with implicit sharing. In this context, modularity of refinement has also been considered in relation to the promotion of state and operations. (See for example [21] extending [33]). Most of these studies have focused on the *dynamic* part of the specification, i.e., the change of state values and the preservation of the invariant by interacting operations. They have paid less attention to the *static* part of the specification, i.e. the parameters, data types, auxiliary functions and permitted state space.

In this paper we elaborate some non-interference and compositionality assumptions that underlie structuring mechanisms such as USES, SEES and IMPORTS and show how they may be violated by inducing emerging properties and therefore altering the *static context* of the used, seen or imported machine. To avoid such violation, a set of contextual proof obligations related to the static part of the specification have to be considered. We provide a set of proof obligations which are associated with structuring the *static context* of an abstract machine, a refinement or an implementation in B. These proof obligations are necessary and sufficient to ensure that the properties of the (static) *context* of the seen, used or imported component are *conserved*, i.e. that they are preserved but not enriched.

In sections 3, 4, 5 we illustrate the need for such proof obligations by means of simple examples. In subsection 2.6 we explain their logical underpinnings and study their relation with a fundamental meta-logical property of the underlying logic, namely Interpolation. With the exception of importing an abstract machine in an implementation, these proof obligations about the static context of abstract machines, refinements and implementations in B have not been considered in the B-Book [1]. On the other hand, some of them have been implemented in release Beta 4.58 of the B-Toolkit [2]. Finally, in section 6 we summarise the results presented in this paper and discuss potential extensions of these results.

2 Background: The Structuring Primitives of B

The B-method has four composition primitives, each associated with a basic structuring mechanism. These are INCLUDES (and EXTENDS), USES, IMPORTS and SEES. We briefly discuss each in turn.

2.1 Incremental Specification: The INCLUDES and EXTENDS Primitives

The INCLUDES primitive can be understood as textual inclusion of a specification module with the additional constraint that the variables of the included component can only be modified indirectly through its own operations (*encapsulation*). This guarantees that the invariant of a consistent included component is preserved by the operations of the including component. The operations of the included component become visible at the interface of the including one

only if they are referenced in the PROMOTES clause; otherwise they are hidden. EXTENDS is a special case of INCLUDES where all operations are promoted.

INCLUDES provides a mechanism for structuring large specifications. It allows the abstract state to be decomposed into independent parts, each encapsulated by a separate included machine. The state of the including machine contains the states of all included components. Hence INCLUDES is *transitive* with respect to the knowledge of state variables. Subsequent inclusions are required to form a chain (i.e., no branching is allowed) to prevent sharing of variables, although *copies* of a machine can be included by renaming into disjoint name-spaces. However, names of SETS and CONSTANTS do *not* participate in the renaming [2]. To preserve the validity of proofs concerning the included machine, the INCLUDES primitive imposes the following *"syntactic"* restrictions:

- the (extension) signature of the including component and the signature of the included machines must be disjoint;
- the operations of the including component do not directly modify the variables of the included machines;
- The parallel substitution $S_1 \| S_2$ is well defined iff the variables modified by S_1 and S_2 are disjoint. Consequently, S_1 and S_2 cannot *both* call operations which modify state variables declared in the same included machine.

Bert, Potet and Rouzaud propose in [4] a new composition operator $\otimes$ which extends conservatively $\|$ by allowing S_1 and S_2 to modify shared variables in a compatible way. If the sets of variables modified by S_1 and S_2 are disjoint then $S_1 \otimes S_2 = S_1 \| S_2$. A similar composition operator is proposed by Dunne in [17].

2.2 Sharing Specification Text: The USES Primitive

The USES primitive can be understood as an intermediate step of a structured construction that provides a controlled form of sharing without an independent semantic content. A USES clause can only appear in abstract machines and always in a larger context. Any number of machines can use a shared machine. These using machines cannot be refined independently; USES is interpreted only in the final closure of the sharing construction. Consequently, all the using machines together with the used machine must be included into a common machine, which may then be refined. *All using machines have read-only access to the shared machine* and can reference the shared variables in their invariants. The construction guarantees (via proof) that the including machine does not invalidate the invariants of the using machines. USES is *not transitive*; the knowledge of the variables of the used machine does not transfer from the using machine to any other component.

The proof obligations of a USES M1 statement in machine M2 ensure that the operations defined in M2 preserve that part of its invariant *independent* of the variables of M1. The obligations ensuring the shared part of the invariant is respected are delayed until operations are promoted in a final closure. Operations in M1 which are not promoted never have such obligations discharged.

The following "syntactic" restrictions are also imposed.

– All (extension) signatures and the signature of used machines are disjoint;
– Operations of the using machine can only read variables of the used machines.

Bert, Potet and Rouzaud propose in [4] a modification of USES and INCLUDES which considers proof obligations sooner, independently of the final instantiations of the used machine. Their modification of USES is similar to an INCLUDES with no promoted operations and without instantiating parameters and it can be given an independent semantic content.

2.3 Reference-Only Sharing: The SEES Clause

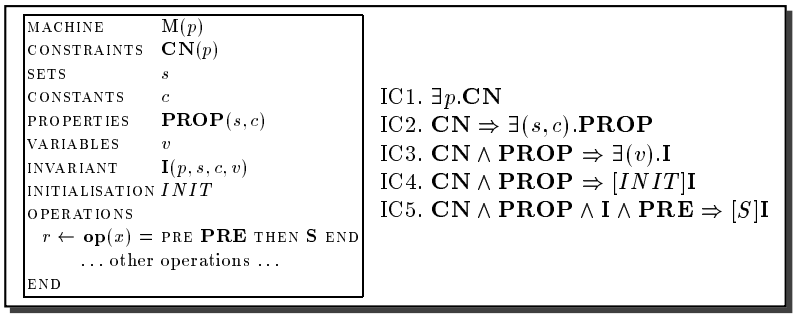
SEES allows the sharing of an abstract machine and can be used in another machine, a refinement or an implementation. The state of the seen machine can be consulted, but not modified, by the seeing components.

Most often, the SEES primitive is used in refinements and implementations for the purpose of sharing code and providing *reference-only* access to shared data, so machines that are imported once in a development can be seen elsewhere. The intention is that the code of the machine that is seen will be linked only once, thus establishing an *"one writer, many readers"* sharing scheme.

Another common use of the SEES primitive is for *sharing separately implemented*, system-wide types. The specification of such 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 particular case of this is to specify abstract (mathematical) data types defined using the SETS, CONSTANTS and PROPERTIES clauses of an operationless shared machine. Such machines can be seen by any other machine. Operationless 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. SEES is *not transitive*: the knowledge of the state variables of the seen machine does not transfer from the seeing machine to any other component. So, every refinement/implementation of a machine that sees M should also see M.

2.4 Layered Implementation: The IMPORTS Primitive

The IMPORTS primitive links implementations to abstract machines allowing a development to be structured into layers. It is this mechanism that allows B to handle large-scale developments by decomposing them into smaller separate developments. As Potet and Rouzaud mention in [25] IMPORTS is essentially the *"closed"* version of INCLUDES. This term refers to the *open-closed* duality principle (e.g. Meyer [24]). "Open" means building larger systems by extensions, e.g. when appending or amalgamating specification modules. "Closed" means building an encapsulated component available for blind use elsewhere, e.g. when linking independently constructed code modules. The distinction is also reflected in the difference between the *visibility rules* for INCLUDES and IMPORTS which



Notes:
(1) $\Phi(list)$ denotes the formula that is built from the primitive constructs of B and the identifiers in *list*.
(2) IC3 is not generated by the B-Toolkit. This is because either IC3 is covered by IC4 or otherwise the machine will not be implementable. (The B-Toolkit will also produce a warning at the analysis phase if no initialisation is specified.)
(3) As we elaborate in Remark 1, section 3, IC2 simplifies to the proof obligation $\exists(s, c).\mathbf{PROP}$.

Fig. 1. The general pattern of an abstract machine specification and the corresponding internal consistency proof obligations.

require that the included state variables are read-only to the operations of the including machine (*semi-hiding*) whereas the imported state variables are invisible to the operations of the importing implementation (*full-hiding*) and accessible only by calling the imported operations.

Following the appearance of some flaws in refinements in the presence of SEES and IMPORTS, Potet and Rouzaud present in [25], and Rouzaud extends in [26], a set of architectural conditions to guarantee the correctness of such refinements. However, these analyses do not consider the effect of adding new CONSTANTS and properties relating them to existing CONSTANTS. As we show, adding new CONSTANTS and properties may produce analogous correctness flaws, unless appropriate *context* related proof obligations are considered.

2.5 Internal Consistency of Abstract Machine Specifications

An apparent difference between the implementation of structuring in the B-Toolkit and the corresponding analysis in the B-Book consists in a collection of *context* related proofs that appear in the B-Toolkit but which are not considered in the B-Book. These proof obligations aim to establish the (logical) consistency of the specified component. The B-Book suggests a more "constructive" approach which consists in establishing the logical consistency of a component specification *a posteriori*, by exhibiting an implementation on a collection of pre-verified primitives.

A "flat" abstract machine specification is *internally consistent* if the proof obligations presented in Fig. 1 are satisfied. (They are called Proof Obligations

for Internal Consistency by Lano in [20]). IC1 asserts that there are possible machine parameter values that satisfy the specified constraints. IC2 asserts that, assuming the machine constraints on the machine parameters, there are SETS and CONSTANTS (i.e. auxiliary types, functions and predicates) satisfying the specified properties.[1] 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. If either of the first two proof obligations fail, then the machine cannot be instantiated and there is no executable system satisfying the specification. If the third obligation fails, then there is some acceptable actualization and evaluation (interpretation) of the machine-constants which does not expand to a meaningful machine. IC4 and IC5 give the base case and induction step of a proof that all reachable states satisfy the invariant.[2] We note that the last three proof obligations depend on the consistency of the (constant) context of the machine. If the machine is inconsistent they hold trivially.

In this paper we will focus our attention on the context of a specified component in B. The internal consistency of this context depends on the first three proof obligations: IC1, IC2 and IC3. Notice that these three proof obligations follow the same pattern: assuming the base theory of B, prove that a set of context axioms over a given a set of identifiers implies the uniform interpolant to this set of identifiers of some context extension axioms. As we will elaborate in subsection 2.6, such proof obligations establish that the *expansion* of the base mathematical language of the B-Method with the signature of the abstract machine M, and the simultaneous addition of the CONSTRAINTS, PROPERTIES and INVARIANT as axioms, result in a logical theory that is a *conservative extension* of the base theory of B. From a deduction point of view, such proof obligations are necessary and sufficient to guarantee the logical conservativeness of the extension.

2.6 Logical Background

In this subsection we provide a condensed description of some results from formal logic which we use in order to establish that the specific form of contextual proof obligations discussed in this paper may be used to guarantee

1. the internal consistency of abstract machines in B
2. the relative consistency of the static context of the machines that appear in the USES, SEES and IMPORTS primitives;

[1] For simplicity, we have omitted some obvious uniform expansions of the bodies of **CN** and **PROP** in the proof obligations. These are, in particular, additional constraints requiring that the parameters p are scalars and the finiteness property of *deferred* sets. In the case of enumerated sets the convention that their elements are distinct has to be appended to the hypothesis of every proof obligation where **PROP** appears.

[2] Note that the third proof obligation is *not* necessarily weaker than the fourth as the initialisation could be infeasible, e.g., initialisation is "any state where false".

3. the correctness of the extension of the static context specification in the case of SEES and IMPORTS.

In particular, we review the concepts of a *logical theory*, a *finite axiomatisation* of a logical theory, *theorem conservation* and *uniform interpolation*. Our intention is to identify some useful results from formal logic, interpret them into the logical language that is used for proof in B and then use them as facts. Note that some of these results essentially depend on the use of first order logic.

Logical Theories. Let γ be a sentence in first order logic. We write $\mathrm{ID}(\gamma)$ to denote the set of all symbol identifiers that appear in γ. We may also write $\gamma(s,c)$ to denote that the identifiers s and c may appear in γ. Let L be a set of identifiers. A *logical theory* T over L is a set of sentences T such that *(1)* T is $\Rightarrow$-closed and *(2)* $\mathrm{ID}(\chi) \subseteq L$, for all χ in T. A sentence $\gamma \in \mathsf{T}$ axiomatises T iff $\gamma \Rightarrow \chi$ for every $\chi \in \mathsf{T}$. If γ axiomatises T then we say that T is the logical theory of γ over L. Let T_1, T_2 be logical theories over L_1, L_2, respectively. Assume that $L_1 \subseteq L_2$. Then we say that "T_2 *extends* T_1" iff $\mathsf{T}_1 \subseteq \mathsf{T}_2$. In particular, we say that "T_2 extends T_1 conservatively" or "T_2 is a conservative extension of T_1" when, for all χ such that $\mathrm{ID}(\chi) \subseteq L_1$, $\chi \in \mathsf{T}_1$ iff $\chi \in \mathsf{T}_2$. Furthermore, if γ_1 axiomatises T_1 and γ_2 axiomatises T_2 then, by abusing the language, we say that γ_2 is conservative over γ_1 iff T_2 extends T_1 conservatively.

The following is an instance of the so-called "Modularisation Theorem" [29, 30, 16], which states that theorem conservation is stable under amalgamation of theory extensions in first order logic.

Proposition 1. *Assume that* $\mathsf{T}_1, \mathsf{T}_2, \mathsf{T}_3$ *are logical theories over* L_1, L_2, L_3*, respectively, such that (1)* $L_1 = L_2 \cap L_3$*, (2)* T_3 *extends* T_1 *and (3)* T_2 *extends* T_1 *conservatively. Let* T_4 *denote the* $\Rightarrow$*-closure of* $\mathsf{T}_2 \cup \mathsf{T}_3$ *over* $L_2 \cup L_3$*. Then* T_4 *extends* T_2 *and* T_4 *extends* T_3 *conservatively.*

In other words, if T_2 and T_3 share T_1 then the conservativeness of one extension T_2 over the shared T_1 implies the conservativeness of the amalgamation closure T_4 over the other (not necessarily conservative) extension T_3. Clearly, if both $\mathsf{T}_2, \mathsf{T}_3$ extend the shared T_1 conservatively then the amalgamation closure T_4 is conservative over both extensions T_2 and T_3. As we demonstrate by means simple examples in the following sections our interest in theorem conservation and Modularisation stems from the fact that conservative extensions ensure the absence of any emerging properties (viz. information flow) imposed on the context of a machine specification by a contextual extension.[3]

Uniform Interpolants. Let L be a set of identifiers and γ be a sentence. A sentence ϑ_o is called a *uniform interpolant* of γ for L iff *(1)* $\mathrm{ID}(\vartheta_o) \subseteq \mathrm{ID}(\gamma) \cap L$, *(2)* $\gamma \Rightarrow \vartheta_o$ and, *(3)* for every χ such that $\mathrm{ID}(\chi) \subseteq L$, if $\gamma \Rightarrow \chi$ then $\vartheta_o \Rightarrow \chi$.

[3] We use the term "emerging properties" in order to denote those sentences in the language of the logical theory describing the context of the component machine which are imposed by a contextual extension and which do not follow from the specification of the component machine alone.

Note that a uniform interpolant depends only on the assumption γ and the set of identifiers L that may appear in the conclusion χ of the implications $\gamma \Rightarrow \chi$. Clearly, whenever a uniform interpolant ϑ_o of γ for L exists, ϑ_o is unique up to logical equivalence and stronger than any other interpolant ϑ such that $\gamma \Rightarrow \vartheta \Rightarrow \chi$. In this paper we use the following results about uniform interpolants.

Proposition 2. *Let γ_1, γ_2 be sentences such that*

1. $\mathrm{ID}(\gamma_1) \subseteq \mathrm{ID}(\gamma_2)$. *Then γ_2 is conservative over γ_1 iff γ_1 is a uniform interpolant of γ_2 for $\mathrm{ID}(\gamma_1)$.*
2. $L_0 = \mathrm{ID}(\gamma_1) \cap \mathrm{ID}(\gamma_2)$. *Assume that γ_2 has a uniform interpolant ϑ_o for L_0. Then $\gamma_1 \wedge \gamma_2$ is conservative over γ_1 if and only if $\gamma_1 \Rightarrow \vartheta_o$.*

Extension by Constants. First order logic, in its usual formulation, is well known to possess ordinary (Craig) interpolation [11] but it lacks uniform interpolation [13, 14] as a global meta-logical property. This means that there are some first order sentences γ and some first order alphabets L such that γ does not have a uniform interpolant for L. However, there are cases where uniform interpolants not only exist but they also have a distinguished meta-form. A typical example consists in pairs of a sentence γ and a sub-alphabet L such that all symbols in the difference $\mathrm{ID}(\gamma) - L$ are constants. In those cases the uniform interpolant is produced by abstracting away the constant symbols in $\mathrm{ID}(\gamma) - L$ using a prefix of fresh existentially quantified variables. For example, if γ is $\forall x.pr(c, x)$ and $L = \{pr\}$ then ϑ_o will be $\exists y \forall x.pr(y, x)$. (See also [31] where similar proof obligations are used in order to verify the conservativeness of extensions that introduce new function symbols by means of pre- and post-conditions.)

Skolem Normal Form. It is well known [18] that, for every first order sentence γ and alphabet $L_0 \subseteq \mathrm{ID}(\gamma)$, there is an existential (Σ^1_1) second order sentence Θ_0 such that $\mathrm{ID}(\Theta_0) \subseteq L_0$ and Θ_0 has the same *first order* logical consequences as γ over L_0. This is a consequence of the Skolem Normal Form (SNF) theorem. (See for example chapter 4 of Enderton's text-book on logic [18].) As is explained in [18] the above use of SNF also holds for a blend of second order grammar and first order deduction, avoiding therefore the necessity to resort to the full deductive power of second order logic. This blend of second order grammar and first order deduction is called "general" second order logic in [18] (as opposed to "absolute" second order logic which uses second order deduction) and it is reducible to an axiomatic enrichment of unsorted first order logic. In a few words, the key difference between "absolute" second order logic and "general" second order logic is the following. In "absolute" second order logic, certain basic notions such as *membership*, *being a subset*, *set comprehension*, etc., are fixed primitives at the level of models. In contrast, "general" second order logic avoids appealing to a fixed notion such primitives and is consequently reducible to first order logic enriched with relativisation predicates, (a first order presentation of) set

theoretic membership and a comprehension scheme[4]. A well known reduction of "general" second order logic to first order logic was given by Enderton in [18].

Such first order interpretations of "general" second order logic give an analogous result with the use of a deductive presentation of set theoretic membership in the base theory of B.[5] On the one hand, when defining a component in B, one gives first order context specifications axiomatically. The function and predicate identifiers are given as parameters or in the CONSTANTS clause and the axiomatisation is provided in the CONSTRAINTS and the PROPERTIES clauses[6]. The (relativised) quantifiers used in this case are quantifying over individuals like the usual first order quantifiers. On the other hand, in a proof obligation, one can quantify over a CONSTANTS identifier by replacing all the occurrences of that identifier by a new (first order) existential quantifier-bounded variable. This may be viewed as first order interpretation of a "general" second order quantifier. Note that the above neither assumes nor implies that the employed first order set theory possesses uniform interpolation. For the existential first order sentences that play the role of uniform interpolants are the result of encoding second order SNF sentences and not ordinary first order sentences: the variables in their existential prefix are not relativised over a domain of individuals.

Uniform Interpolants in B. The following are some useful results about the use of uniform interpolants in the B-Method. As we already mentioned in this section, they merely depend on the fact that a deductive presentation of set theory is incorporated in the base theory of B.[7]

Proposition 3. *Let* $L' = \{c, c_1, \ldots, c_n\}$ *be a set of (*ABSTRACT_*)*CONSTANTS *identifiers and let* $L = L' - \{c\}$. *Assume that* $\mathbf{P}(c, c_1, \ldots, c_n)$ *axiomatise the context of a component with context signature* L'. *Then*

1. $\exists c.\mathbf{P}$ *is the uniform interpolant of* $\mathbf{P}$ *for the (sub)signature* L.
2. *For all* φ *such that* $\mathrm{ID}(\varphi) \subseteq L$, $\mathbf{P} \Rightarrow \varphi$ *if and only if* $(\exists c.\mathbf{P}) \Rightarrow \varphi$.

Notice that the CONSTANTS identifier c does not appear in φ.

---

[4] I.e., $\exists X.X(x_1, \ldots, x_n) \Leftrightarrow \varphi(x_1, \ldots, x_n)$, where variables $x_1 \ldots x_n$ are first order and free whereas X is a second order variable that does not appear in the formula φ.

[5] The set theoretic foundation of B consists in a subtheory of ZFC which focuses on reasoning with arbitrarily large finite sets and distinguishes cofinite sets by means of a constant 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 [1]). Notably, the notion of an ordered pair is defined in B outside set theory, and type-checking all set theoretic statements ensures that ordinal numbers are not dealt with in a specification.

[6] Further restriction apply of course in order to ensure that the SETS are finite, the actualisations of the parameters are scalars, etc.

[7] For simplicity we focus on CONSTANTS identifiers. One can easily extend these results to parameters, SETS and parameter CONSTRAINTS by appending the (definitions of) the enumeration for defined sets and the finiteness condition for deferred sets, etc.

Proposition 4. *Let* $\mathbf{P}$ *axiomatise the context of a component with context signature* $L = \{c_1, \ldots, c_n\}$. *Let* $\Delta\mathbf{P}(c_1, \ldots, c_n, c'_1, \ldots, c'_k)$ *be the extension axiom specifying the properties of a contextual extension with (extension) signature* $\Delta L = \{c'_1, \ldots, c'_k\}$. *The following are equivalent.*

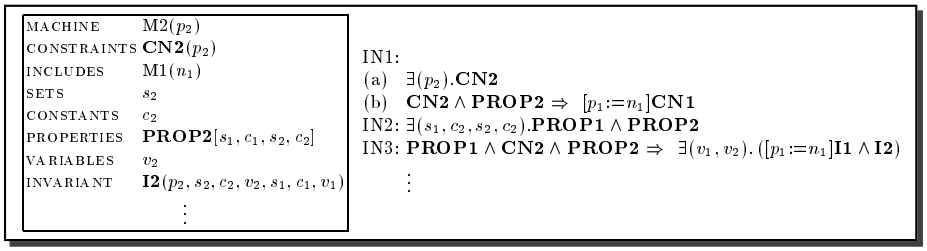
1. $\mathbf{P} \wedge \Delta\mathbf{P}$ *is conservative over* $\mathbf{P}$.
2. *For every* φ *such that* $\mathrm{ID}(\varphi) \subseteq L$, $\mathbf{P} \wedge \mathbf{\Delta P} \Rightarrow \varphi$ *if and only if* $\mathbf{P} \Rightarrow \varphi$.
3. $\mathbf{P} \Rightarrow \exists(c'_1, \ldots, c'_k).\Delta\mathbf{P}$.

We note that the above **Proposition** 4(3) can be seen as a proof obligation in order to establish that $\mathbf{P} \wedge \mathbf{\Delta P}$ is conservative over $\mathbf{P}$. A typical example of such a proof obligation from the B-Book [1] is the context-related proof obligation of IMPORTS (page 599 of the B-Book). The latter can be seen as the first order reduction of the SNF-style (second order) formula describing the uniform interpolant of the conjunction of the formulae in the PROPERTIES clauses of the implemented refinement sequence to the context signature of the imported machine. (See also section 5.) For the rest of the paper, our main focus will be on this kind of proof obligation which we show to be generally useful, and often essential, for ensuring relative consistency, relative completeness of presentation, and proof modularity for the (static) context of several composition primitives that are used in a development with the B-Method.

3 Incremental Specification

As we explained in subsection 2.6, theorem conservation introduces a relative consistency and a relative completeness argument – as follows. Firstly, the consistency of the extended logical theory reduces to the consistency of the base theory. If the *"Internal Consistency"* proof obligations IC1–IC4 illustrated in Fig. 1 hold then the logical theory describing the static part of machine M is consistent if and only if the base theory of B is consistent. Secondly, the context of M does not say anything more about the built-in constructs of B, other than what is already deducible from the base theory. Furthermore,

- **IC3** guarantees that the addition of state variables v, to refer to the possible state values, and the invariant $\mathbf{I}$, to specify them, extends the theory describing the parameters and CONSTANTS of M, conservatively. That is, from the base theory of B, it follows that $\mathbf{CN} \wedge \mathbf{PROP} \wedge \mathbf{I} \Rightarrow \varphi$ iff $\mathbf{CN} \wedge \mathbf{PROP} \Rightarrow \varphi$ for every φ such that the state variables v do not appear in φ.
- **IC2** guarantees that the axiomatisation of the SETS and CONSTANTS is conservative over the logical theory generated by the axiomatisation of the CONSTRAINTS on the parameters (by proving the uniform interpolant of the formula of **PROP** for the base language.) That is, assuming the base theory of B, $\mathbf{CN} \Rightarrow \varphi$ iff $\mathbf{CN} \wedge \mathbf{PROP} \Rightarrow \varphi$ for every first order sentence φ such that s, c do not appear in φ.
- **IC1** guarantees that the theory of the constraints on the actualization parameters extends conservatively (and is relatively consistent with) the base theory of B.



Note: For simplicity and clarity of presentation we assume that (in proof obligations) $\mathbf{CN}_i$ and $\mathbf{PROP}_i$ embody the (default) assumptions about the type of formal parameters and of SETS and CONCRETE_CONSTANTS identifiers that appear in their signature (i.e., that parameters are finite sets or scalars, that deferred sets are finite, etc).

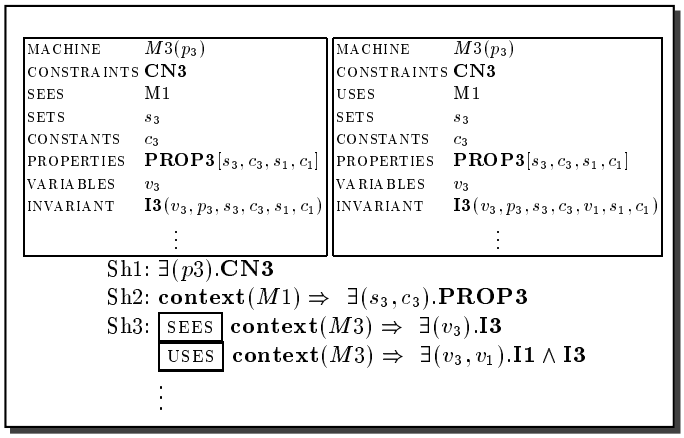
Fig. 2. The general form of the context-related clauses and proof obligations for the INCLUDES and EXTENDS primitives.

Remark 1. IC2 can be further simplified to $\exists(s,c).\mathbf{PROP}$. This is because of the following reasons. First, B requires that (abstract) parameters do not appear in **PROP**; thus proving the uniform interpolant of **PROP** over the base theory of B guarantees that the extension $\langle\langle s,c\rangle, \mathbf{PROP}\rangle$ is conservative. Second, IC1 establishes that the extension of the base theory by $\langle\langle p,s,c\rangle, \mathbf{CN}\rangle$ is conservative. Third, the "Modularisation Theorem" (Proposition 1) ensures that the amalgamated sum $\langle\langle p,s,c\rangle, \mathbf{CN} \wedge \mathbf{PROP}\rangle$ conservatively extends both the logical theory describing the parameter constraints and the base theory of B.

Note that when large specifications are structured incrementally, the logical theory of the static part of the whole specification need not be conservative over the theories of the static part of its components. As we illustrate in the following subsection, it is *often* the case that (the logical theory describing the static part of) a compound machine is *not* conservative over some or all of its components. Whatever the case is, both components and compound specifications extend the base theory of B conservatively.

3.1 Extending Specifications

In the case of EXTENDS and INCLUDES the *static context* related proof obligations take the form that is provided in Fig. 2, which is similar to those of a "flat" machine over an extended context. The only new form of proof obligations is about the actualization of the parameters of the included machine. This proof obligation (Fig. 2–IN1.(b)) validates the constraints of the included machine in the context of the including machine. The logical theory that describes the static part of $M2$ is equivalent to that of a "flat machine" with parameters p_2, constraints **CN2**, sets s_1, s_2, CONSTANTS c_1, c_2, properties **PROP1** $\wedge$ **PROP2** and invariant $[p_1{:=}n_1]$**I1** $\wedge$ **I2**. We emphasise that IN2 is weaker than an obligation of the type **PROP1** $\Rightarrow \exists(s,c).$**PROP2**. For INCLUDES and EXTENDS, the



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

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

static context of $M2$ does *not* have to be conservative over $M1$. The last point underpins the conception of INCLUDES and EXTENDS as the "open" structuring assemblies that facilitate information disclosure. They allow the designer to reuse the *specification text* of the included machine, thereby specifying the context of the including machine as an enrichment of the context of the included one.

4 Sharing Components

In the case of SEES and USES, the context-related proof obligations take the form provided in Fig. 3.[8] (We omit the parameters of the component machine and the associated constructs for simplicity. These parameters are not instantiated via the USES and SEES primitives.) 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,

1. in the case of SEES, the logical theory of the 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 $M1$;

[8] If, following the philosophy of the B-Book, USES is considered as an auxiliary syntactical mechanism whose sole purpose is to facilitate sharing of specification text (as opposed to an actual composition primitive between machine specifications) then one might like to replace **I1** $\wedge$ **I3**$(p_3, v_3, s_3, c_3, v_1, s_1, c_1)$ with **I3**$'(p_3, v_3, s_3, c_3, s_1, c_1)$ (where **I3**$'$ is the subformula of the invariant where the state variable v_1 of the used machine $M1$, to be shared, does not appear) and thus postpone the proof for the shared part of the invariant until the closure is formed.

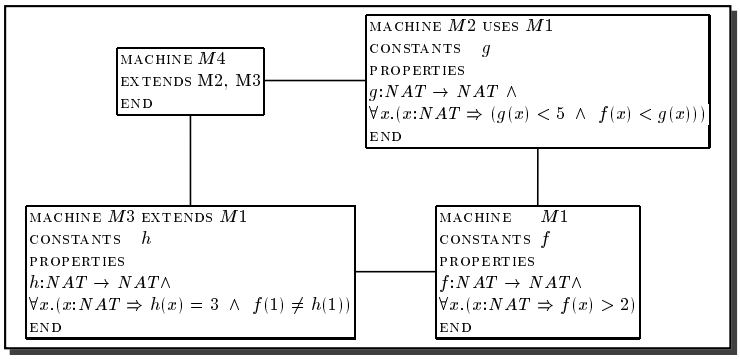


Fig. 4. The abstract machine specifications of Example 1.

2. in the case of USES, the logical theory of the context of $M3$ is conservative over the logical theories describing the stateless fragment of $M3$, and the stateless fragment of $M1$.
   Note that it is not conservative over the whole context of $M1$ because the USES-invariant may strengthen the invariant over the state variables of the used machine $M1$.

In the following paragraphs, we discuss the impact of these context-related proof obligations in the cases of USES and SEES separately.

The Impact of the Context-Related Proof Obligations on USES . Establishing Sh2 in this case guarantees that the underlying theory of the CONSTANTS of the using machine is *conservative* over the corresponding theory of the used machine. Further, the *Modularisation* property of first-order theory presentations (Proposition 1) guarantees that the union of the corresponding theories, underlying the sharing mechanism of USES, will preserve consistency. Consequently, no further consistency proofs about the union are necessary. This is demonstrated in the following example.

Example 1. Consider the following abstract machines $M1$, $M2$, $M3$ and $M4$ provided in Fig. 4. The abstract machine M1 is shared; it is used by M2 and extended by M3. When the closure machine M4 is formed no further proof obligations about the context are needed. (Indeed no constant-context proof obligations are generated by the B-Toolkit.) Because of the Modularisation property (Propositions 1 and 2), the proof obligation establishing the conservativeness of the context of $M2$ over the context of $M1$ guarantees the conservativeness of the (constant) context of $M4$ over the context of $M3$. Consequently the internal consistency of $M3$ guarantees the internal consistency of $M4$. Of course in this particular example the proof obligation related to the conservativeness of $M2$ USES $M1$ cannot be discharged. So the potential of a consistency problem related to the incremental specification in the design is detected when it first appears.

It is worth noting that, unlike SEES, the conservativeness requirement between the constant contexts that underlies USES is not seen as fundamental. It facilitates proof modularity, presentation clarity and it is consistent with the architectural concept of "one writer, many readers" but it can be omitted. For example Bert, Potet and Rouzaud propose in [4] an alternative version of SEES which is very similar to an INCLUDES that does not instantiate parameters and does not promote any operation. Although they do not consider CONSTANTS in their development, one can imagine a variant of USES where the constant context of the using machine is not conservative over the constant context of the used machine. In such a case, one would not be able to employ the Modularization property and the consistency of the closure should be re-established (or delayed until the implementation). Since only the closure machine is refined and later implemented, the conservativeness of the constant context for USES can be conceived to be a matter of taste and style of development. Though, as we illustrate in this paper, the above argument does not apply in the cases of SEES and IMPORTS where the conservativeness between the static contexts of the seeing/importing component and the seen/imported machine *is* required.

The Impact of the Context-Related Proof Obligations on SEES. For SEES, establishing Sh2 guarantees that the underlying theory about the CONSTANTS of the seeing machine is *conservative* over the theory of the seen machine. Therefore the properties of the seeing machine cannot impose any further *"emerging"* properties on the CONSTANTS of the seen machine. In other words, there is no information flow from the seeing machine to 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* 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 (non-empty state space) 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,

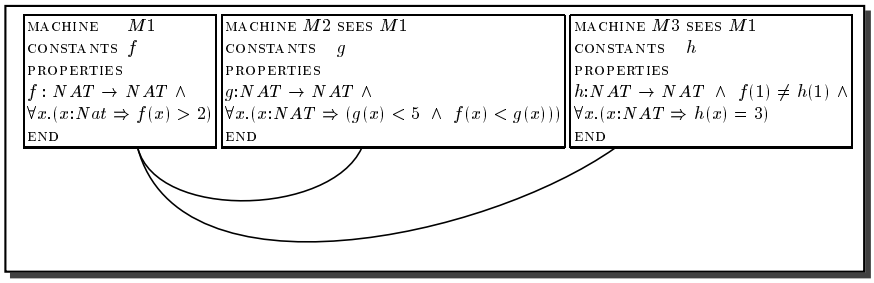


Fig. 5. The abstract machines specifications of Example 2.

ensures the conservativeness of **I3**.[9] Example 2 demonstrates the importance of considering these context-related proof obligations associated with the SEES primitive.

Example 2. Consider the following example where one machine M1 is seen by two other machines M2 and M3 as presented in Fig. 5. 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 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. As we explain in section 2.6 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 Layered Implementation (IMPORTS)

In the case of IMPORTS (in an IMPLEMENTATION) the context related proofs have the form provided in Figure 6. Imp1 establishes, in analogy to INCLUDES, the

[9] Because of the Modularisation property (Propositions 1 and 2), 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.

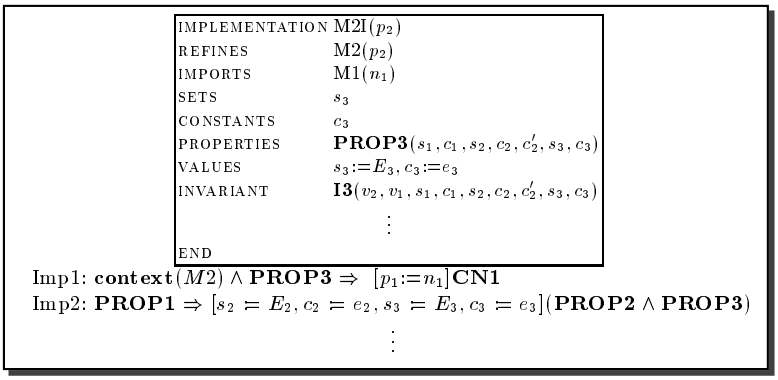
correctness of the instantiation $p_1{:=}n_1$ of the imported machine $M1(p_1)$. We note that the assumption in Imp1 does *not* embody **PROP1** although imported constant identifiers in c_1 may appear in the assumption via **PROP3**. As we elaborate in the sequel, this is important for establishing the conservativeness of the properties of **context**$(M3)$ over the properties of the imported machine $M1$. Imp2 proves the uniform interpolant of the implementation instance of the refined properties in the static language of imported machine. On the one hand, the sentence $\mathbf{PROP}2 \wedge \mathbf{PROP}3$ axiomatises the static context of the (resulting machine of the) refinement. On the other hand, the instance

$$[s_2{:=}E_2, c_2{:=}e_2, s_3{:=}E_3, c_3{:=}e_3]\mathbf{PROP}1 \wedge \mathbf{PROP}2 \wedge \mathbf{PROP}3$$

which is produced by the evaluation of the refined sets and (concrete) CONSTANTS and the embodiment of imported properties **PROP**1, axiomatises the static context of the implementation. Hence, Imp2 establishes that the properties of implementation context extend conservatively the properties of the imported context and, by Modularisation (Proposition 1), that **context**$(M2I)$ is conservative over **context**$(M1(n_1))$. But this alone is not enough. In order to ensure that no "emerging" properties are imposed over the (static) context of $M1$ by the implementation $M2I$ one has to establish that the instantiated **context**$(M1(n_1))$ does not affect **PROP1**. Because n_1 may depend on some constants c_2, c_3 in the contextual extension signature of $M2I$ which are then related to c_1 via **PROP3**, there is an indirect channel through which "emerging"properties about c_1 can flow from $M2I$ to $M1$. However, the conservativeness of **context**$(M1(n_1))$ over **PROP**1 follows by taking Imp1 into account. Imp1 and Imp2 together guarantee that if **context**$(M1(n_1)) \Rightarrow \varphi(c_1)$ then **PROP**$1 \Rightarrow \varphi(c_1)$.

We note that, among the contextual conservativeness requirements we discuss in this paper, only Imp2 is considered in the B-Book (page 599). Indeed, the conservativeness of this extension is fundamental. Because the imported machine $M1$ is implemented independently in a layered development, the only (abstract) properties considered in the implementation of $M1$ are **PROP1** (i.e., those specified in the PROPERTIES clause of $M1$). If any *emerging* properties about the static context of $M1$ were allowed, these properties would not be considered in the implementation of $M1$. The latter could allow the validation of an implementation $M1I$ of a behaviour that is weaker than that assumed by $M2I$, in which case the correctness of each implementation layer individually would not guarantee the correctness of the overall development. This is illustrated in Example 3.

Example 3. Consider the implementation presented in Figure 7 where the abstract machine to be refined is $M2$ which specifies an operation op_2 such that op_2 always returns $TRUE$, and the imported machine is $M1$ which specifies an operation op_1 such that op_1 always returns $FALSE$. Both machines $M1$ and $M2$ are clearly internally correct but the property given in the implementation is inconsistent with those inherited from the refinement sequence and the imported machine. Consequently, the proof obligation related to the preservation of the invariant in $M2I$ holds trivially:



Notes:
(1.) c_2' are the (abstract) CONSTANTS of $M2$ which are *not* given concrete values via the implementation and E_i, e_i are concrete scalar values.
(2.) Both proof context-related obligations Imp1 and Imp2 are proposed by the B-Book.

Fig. 6. The general form of the context-related clauses and proof obligations for the IMPORTS primitive.

$\bigwedge$ **HYP** $\Rightarrow$ $\exists(v_2).(v_2 : NAT \wedge bool(fun_2(v_2) = v_2) = bool(fun_1(v_1) = v_1))$ where
HYP = {**CN**$M2I$,**PROP**$Bool\_TYPE$,**PROP**$M1$,**PROP**$M2$,**PROP**$M2I$,I1,I2,I$M2I$,**assign**($M3I$),**PRE**(op_2)}

This is because the conjunct $\bigwedge$**{PROP**Bool_TYPE, **PROP**M1, **PROP**M2, **PROP**M2I} in the assumption is self-contradictory (asserting $0 = 1$). The incorrectness of the implementation is caught in the validation of the context related proof obligation: $\bigwedge$**{PROP**$M1$,**PROP**$Bool\_TYPE$}$\Rightarrow$ $\exists(fun_2).\bigwedge$**{PROP**$M2$,**PROP**$M2I$} which fails by, for example, reducing the proof to the validity of $0 = 0 + 1$.

We note that while testing the above example in the B-Toolkit, the *incorrect* proof obligation of Figure 8 was generated by the tool. The tool generated proof obligation is incorrect because it trivialises the conservativeness argument by bounding the occurrences of fun_2 *and* fun_1 in the conclusion with an existential quantifier and thus producing a formula that is strictly weaker than the uniform interpolant. In the correct form of this proof obligation fun_1 appears unbounded (as the same constant in the conclusion and the assumption). The above mentioned proof obligation reads *" there exist* fun_1 *and* fun_2 *such that* fun_2 *has those properties and* fun_1 *is equal to* fun_2 *"* which is of course valid. (Because it reduces to the internal consistency of the abstraction $M2$.)

6 Conclusion

In this paper we have discussed how the conservativeness between the static context of components in B can be established by means of proof obligations which have a common (meta-)form. From a deduction perspective, the common (meta-)form of these proof obligations consists in establishing that the

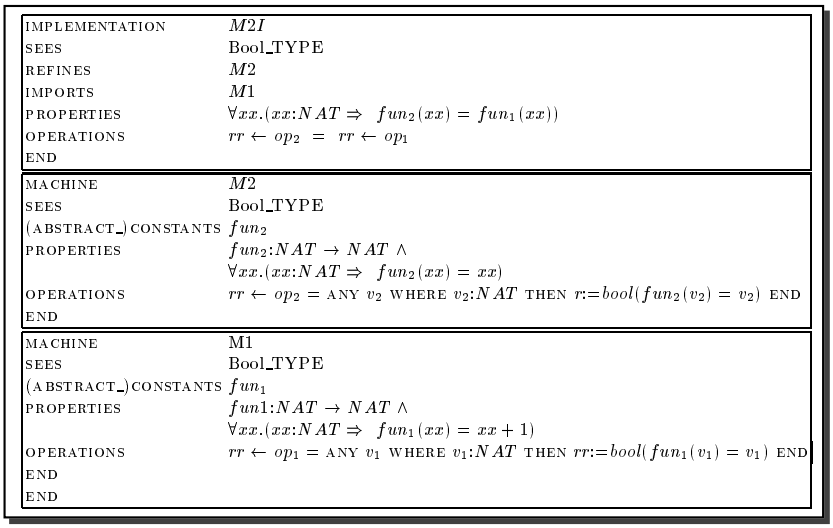


Fig. 7. The implementation of Example 3.

```
cst(M2I$1) ⇒ ( ctx(M1) & ctx(Bool_TYPE) ⇒ (#(fun1 , fun2).( !xx.(xx : NAT ⇒ fun1(xx) = fun2(xx)) &
                                                              fun2 : NAT ⟶ NAT & !xx.(xx : NAT ⇒ fun2(xx) = xx) )))
```

Fig. 8. A tool-generated incorrect contextual proof obligation for the implementation of Figure 7, Example 3.

uniform interpolant of the contextual extension axioms is implied by the base context. Furthermore, in order to produce the above mentioned uniform interpolant, one can simply *abstract away* the identifiers of the extension signature from the extension axioms by replacing all their occurrences with new existential quantifier-bounded variables. Hence, the above mentioned proof obligations take the following general form, where $M1$ is the component specification, $M2$ is the compound specification and $c'_1, \ldots, c'_n$ are the identifiers in the contextual extension signature, $\mathbf{context}(M1)$ is the conjunction of the context-related properties of the component and $\Delta\mathbf{context}(M2)$ is conjunction of the properties of the contextual extension, i.e., $\mathbf{context}(M1) \Rightarrow \exists(c'_1, \ldots, c'_n).\Delta\mathbf{context}(M2)$

From a logical viewpoint, these proof obligations can be seen as a set of necessary and sufficient conditions for establishing the conservativeness of the contextual extension. By requiring such a contextual extension to be conservative, one guarantees the relative consistency, relative completeness of presentation and the absence of information flow from the context of the compound specification to the context of the component specification. By establishing the conservativeness of the compound context over the context of the component these proof obligations are

1. essential in order to ensure the correctness of the "consultation only" sharing architecture in the case of SEES and the compatibility of layered implementation in the case of IMPORTS;
2. useful in order to facilitate proof modularity and orthogonality of incremental specification in the case of sharing specification modules via the USES primitive.

Instances of such proof obligations are generated by the B-Toolkit (release Beta 4.58) for the validation of the contextual extensions that are associated with the USES, SEES, or IMPORTS primitives, as well as ensuriing that (the static context) of a component specification in B is internally consistent (following the Proof Obligations for Internal Consistency provided by Lano in [20]). We also noted that only in the case of IMPORTS (in an IMPLEMENTATION) some form of conservativeness validation by means of appropriate uniform interpolants has been explicitly considered in the B-Book. However, the context-related proof obligation generated by the B-Toolkit for IMPORTS is not sufficient and needs to be corrected.

We plan to investigate the potential of producing analogous proof obligations to ensure non-interference and to control state sharing and information flow in the dynamic part of component specifications in B. This may involve developing an appropriate (polymodal) formalism (c.f. [3, 14]) to model the correlation between sequences of general substitutions and state transitions and then reduce it to classical logic enriched with a deductive presentation of set theoretic membership (c.f [8, 12]). Non-interference and absence of information flow are known to be related with bisimulation between the abstract state spaces [5, 27] while the existence of interpolants is known to be equivalent with entailment along bisimulation in various (poly)modal logics [8, 12]. Furhtermore, an equivalence between the stability of theorem conservation under amalgamation of theory extensions and some variants of interpolation has been established in [16] for various families of reflexive, transitive and monotonic entailment relations.

References

1. J.R. Abrial. *The B-Book : Assigning Programs to Meanings.* C.U.P., 1996.
2. B-CORE (UK) Ltd. The b-toolkit. 1999. URL: http://www.b-core.com.
3. J. Barwise, D. Gabbay, and C. Hartonas. On the logic of information flow. *Bulletin Of The IGPL*, 3 (1):7–49, 1995.
4. D. Bert, M-L. Potet, and Y. Rouzaud. A study on Components and Assembly primitives in B. In H. Habrias, ed., *First Conference on the B-Method*, 1996.
5. J.C. Bicarregui. Non-Interference, Security and Bisimulation: explorations into the roles of Read and Write frames. CLRC-RAL, 1998.
6. J.C. Bicarregui and *et al.* Formal Methods Into Practice: case studies in the application of the B Method. *I.E.E. Transactions on Software Engineering*, 1997.
7. J.C. Bicarregui, J.Dick, B.Matthews, and E.Woods. Making the most of formal specification through animation, testing and proof. *Sci. of Comp. Prog.*, 1997.
8. J. van Benthem. Modality, bisimulation and interpolation in infinitary logic. *ANNALSPAL: Annals of Pure and Applied Logic*, 96, 1999.

9. M. Buchi and B. Back. Compositional Symmetric Sharing in B. In *FM'99 – Formal Methods*, volume I of *LNCS*, pages 431–451. Springer, Septermber 1999.
10. D. Clutterbuck, J.C. Bicarregui, and B.M. Matthews. Experiences with proof in formal development. In H. Habrias, ed., *First Conference on the B-Method*, 1996.
11. W. Craig. Three uses of the Herbrand-Getzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic XXII*, pages 269–285, 1957.
12. G. D'Agostino, A. Montanari, and A. Policriti. A set-theoretic translation method for (poly)modal logics. *Lecture Notes in Computer Science*- 900, 1995.
13. Th. Dimitrakos and T.S.E. Maibaum. Notes on refinement, interpolation and uniformity. In *ASE'97, 12th IEEE Int. Conf.*, 1997.
14. Theodosis Dimitrakos. *Formal support for specification design and implementation.* PhD thesis, Imperial College, March 1998.
15. Theodosis Dimitrakos. Parameterising specifications on diagrams. In *ASE'98, 13th IEEE Int. Conf.*, 1998.
16. Theodosis Dimitrakos and Tom Maibaum. On a generalised modularisation theorem. *Information Processing Letters*,74(1-2):65-71, 2000.
17. S. Dunne. The Safe Machine: A New Specification Construct for B. In *FM'99 – Formal Methods*, volume I of *LNCS*, pages 472–489. Springer, Septermber 1999.
18. H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.
19. Cliff B. Jones. Accomodating interference in the formal design of concurrent object-based programs. *Formal Methods in System Design*, 8(2):105-122, March 1996.
20. Kevin Lano. *The B Language and Method.*. Springer-Verlag, 1996.
21. P.J. Lupton. Promotin Forward Simulation. In J.E. Nicholls, editor, *Z User Workshop*, pages 27–49. Springer-Verlag, Oxford 1990.
22. B. Matthews, B. Ritchie, and J. Bicarregui. Synthesising structure from flat specifications. In *2nd International B Conference*, LNCS, 1998.
23. M.C. Mere and P.A.S. Veloso. Definition-like extensions by sorts *Bulletin of the IGPL*, 3:579-595, 1995.
24. B. Meyer. *Object Oriented Construction.* Prentice-Hall, 1988.
25. M-L. Potet and Y. Rouzaud. Composition and Refinement in the B-Method. In D. Bert, editor, *Second B International Conference*, pages 46–65, 1998.
26. Yann Rouzaud. Interpreting the B-Method in the Refinement Calculus. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods*, vol. I, 1999.
27. 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.
28. Ketil Stølen. *Development of Parallel Programs on Shared Data-Structures.* PhD thesis, University of Manchester, 1990. Available as a technical report UMCS-91-1-1.
29. Władysław M. Turski and Thomas S. E. Maibaum. *The Specification of Computer Programs.* Addison-Wesley, 1987.
30. P.A.S. Veloso and T.S.E. Maibaum. On the modularisation theorem for logical specifications. *Information Processing Letters 53*, pages 287–293, 1995.
31. P.A.S. Veloso and S.R.M. Veloso. On extensions by function symbols: coservativeness and comparison. Tech. Report. COPPE/UFRJ. 1990. (See also [23, 32])
32. P.A.S. Veloso and S.R.M. Veloso. Some remarks on conservative extensions: a Socratic dialogue. *Bulletin of the EATCS*, vol. 43, 1991.
33. J.C.P. Woodcock. Mathematics as a Management Tool: Proof Rules for Promotion. In *CSR Sixth Annual Conference on Large Sofware Systems. Bristol*, 1989.