BiCoax,
a proof tool traceable to the BBook *

Samuel Colin[1] & Georges Mariano[2]

[1] Univ Lille Nord de France, UVHC
LAMIH, CNRS UMR 8530
F-59313 Valenciennes Cedex 9, France
scolin@hivernal.org

[2] Institut National de Recherches sur les Transports et leur Sécurité
georges.mariano@inrets.fr

Abstract. We introduce BiCoax, a shallow embedding of B set-theoretic first-order logic into the Coq proof assistant. This tool aims at validating the mathematical foundations of B described in the BBook and providing the B community a proof tool matching those foundations. While this is still a work in progress, BiCoax has become usable for mundane proof work in B projects.

1 Introduction

BiCoax[3] is a Coq[25] library for proving formulas defined in the first-order set-theoretic logic of the B formal method. This work is initially based upon similar efforts done by Rocheteau [23] with another proof assistant named PhoX [22] and it is available as part of the Brillant platform (https://gna.org/projects/brillant).

The original aim of this work was to provide a B *proof tool* based on a generic proof assistant, for proving B Proof Obligations. This aim was completed with a secondary objective of *validating* the mathematical foundations of B described in the BBook[1]. This book indeed introduces very precisely these mathematical foundations and claims many of their properties, sometimes providing a proof along. Unfortunately, the number of these proofs is reduced in comparison of the number of claimed properties.

Ideally, the design of any formal method shall be supported with tools when possible, e.g. for avoiding inconsistencies introduced by mere human errors. BiCoax subscribes to this approach: the use of a tool helped with the discovery a few simple typographic errors as well as minor semantic mistakes in the BBook.

---

* The present work has been partially supported by the ANR-SETIN project: TACOS : Trustworthy Assembling of Components: frOm requirements to Specification, the European Community, the Délégation Régionale à la Recherche et à la Technologie, the Ministère de l'Education Nationale, de la Recherche et de la Technologie, the Région Nord-Pas de Calais, the Centre National de la Recherche Scientifique. The authors gratefully acknowledge the support of these institutions.

[3] BiCoax can be downloaded through Subversion (https://gna.org/svn/?group=brillant) or through a tarball placed in http://download.gna.org/brillant/snapshots/

As for providing a proof tool for the B method, BiCoax implements all operators and most theorems up to the middle of the third chapter of [1], namely the introduction of integers. This means that the missing constructs are sequences and trees, which are not as frequently used in B projects as function constructs, for instance. As a consequence, we think that BiCoax is already mature enough for being used as a proof tool in the development of mundane B projects.

After justifying the need for our implementation with respect to similar work in Sec. 2, we give a short introduction to the involved formalisms in Sec. 3. We describe our implementation choices in Sec. 4 and their consequences on the theoretical side in Sec. 5. We present a feedback of some manual and automated experiments with our tool in Sec. 6. We conclude with the various perspectives in Sec. 7.

2 Related work

2.1 Objectives

As stated in the introduction, the goal of BiCoax is to provide a proof tool for B based on the first reference on B theory [1]. It is possible to try re-implementing a proof tool in a given programming language, but using a generic proof assistant provides us with the ability to double-check the theorems already established on paper. This gives us our second goal of tracking the inconsistencies of the BBook. We left aside other objectives such as the automation of proofs, although we hope that our choice of a proof tool will help make our task easier when these objectives are given a higher priority.

Moreover, two additional parameters must be chosen when implementing through another formalism: the kind of embedding and what levels of B will be implemented. The embedding can be either *shallow* or *deep*. For B, an embedding usually will be shallow if it reuses the logic of the target formalism directly (such as the first-order logic of Coq) and it will be deep if the syntactic structure of B is modeled again in the target formalism. The implementable levels of B can be divided into the following categories: (i) mathematical foundations [1, chapters 1-3], (ii) generalized substitutions [1, chapters 4-5,9-10], (iii) theory of abstract machines [1, chapters 6-8] and (iv) refinement [1, chapters 11-13].

For a proof tool, the sole mathematical foundations are sufficient and the embedding is often, but not exclusively, a shallow one. With all the previous items we can now characterize BiCoax: it is a shallow embedding of the B set-theoretic logic in Coq. As such it reuses directly the first-order logic connectors of Coq with the axiom of excluded middle and the axiom of choice. At the time of writing, BiCoax covers the first three chapters of the BBook but the integers, the sequences and the trees, i.e. the last half of the third chapter is yet to be implemented. Let us now describe more precisely similar work.

2.2 Similar work

[23] presented a comprehensive view of other efforts directed at the validation of B or the creation of a proof tool for B. We shall sum up here the description of these efforts and amend this description with more recent work.

	Bowen [14]	Chartier [16]	Bodeveix [12]	Bodeveix [13]	Berkani [9]	Rocheteau [23]	Jaeger [19]	BiCoax
formalism	HOL	Isabelle/HOL	PVS	PVS	Coq	PhoX	Coq	Coq
embedding								
Deep		$\star$	$\star$		$\star$		$\star$	
Shallow	$\star$			$\star$		$\star$		$\star$
B level								
Foundations	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$
Language		$\star$	$\star$					
Machines		$\star$	$\star$					
Objectives								
Validation		?	$\star$		$\star$		$\star$	$\star$
Tool	?			$\star$		$\star$	$\star$	$\star$

Table 1. B implementations at a glance

Table 1 sums up the existing implementations of B, what embedding they use, what levels of B they implement and whether the implementation is proposed to validate the semantics of B or to provide a proof tool. Note that we did not include "refinement" because it is also taken care of at the "Machines" level.

We also included in Table 1 an implementation of Z, because the mathematical foundations of B and Z are very similar. It turns out that the various implementations are done with usually mature proof assistants: HOL, Isabelle/HOL, PVS and Coq. The only exception is PhoX, which did not meet the same adoption despite an efficient equational reasoning engine.

Let us firstly give more details at implementations that are more than five years old. Bowen & Gordon [14] propose a shallow embedding of Z in HOL with HOL viewed as a proof tool. They justify the choice of a shallow embedding for avoiding too complex notations. The goal of Chartier [16] is the derivation of a predicate for defining formally PO generation and its validation. His implementation is a deep embedding realized with Isabelle/HOL and it supports not only the foundations of B but also the generalized substitutions and the representation of abstract machines. Bodeveix & al. [12] have a similar but less ambitious goal, as their validation involves only the generalized substitutions. This time the deep embedding is done with PVS and automated with the PBS tool by Muñoz. The work of Bodeveix & Filali [13] concerns the type-checking of B, this time with a shallow embedding in PVS. This work gave way to a typechecker which was later integrated into the BRILLANT platform. Berkani & al. [9] proposed a deep embedding of B into Coq for validating the logic rules of the prover of the AtelierB commercial tool. To the best of our knowledge, all the works we just cited have no usable tool available, either because it does not exist anymore or because the proof assistants they are based upon have evolved too much.

More recent implementations consist of B/Phox [23], BiCoq [19] and BiCoax. Rocheteau [23] introduces a shallow embedding of the foundations of B in PhoX: the set constructs are translated into PhoX[22], this translation being proved correct. This work is now abandoned because its direct successor is actually BiCoax: the very first working source code of BiCoax is a translation of the PhoX constructs into Coq.

We are left with comparing BiCoax with BiCoq [19]: the objective of Jaeger & Dubois can be seen as a gathering of all the objectives of the previously cited works. Their goal is to validate the theory of B and propose a proof tool for B. They realize a deep embedding of B into Coq. Choosing a deep embedding avoids the interference of the classical logic of B with the more intuitionistic logic of Coq and allows the proposition of decision procedures. Their implementation also has the peculiarity of using De Bruijn indices for quantified variables. [19] does not make it clear what parts of [1] are covered, although we can infer that they include the first two chapters.

With respect to all the works presented here, we can see that the most relevant work to compare our efforts with is BiCoq [19]. The most recurring question in comments about earlier presentations of BiCoax was about the reuse of BiCoq: why not develop on the base of BiCoq ? The first answer is purely practical: several attempts to contact the authors of BiCoq before and during the development of BiCoax were unsuccessful, thus we simply did not have BiCoq at disposal. We then were deprived of the most straightforward option of reusing BiCoq.

The second answer is about the durability and the availability of BiCoax. We wanted to avoid the fall into oblivion from which all the previous work, B/PhoX [23] excepted, seem to have suffered. We also wanted to include BiCoax in the tool suite of the BRILLANT platform, which requires the release of BiCoax under open-source license terms. As a result, BiCoax is now part of the BRILLANT platform and is available for anyone to try. It is also durable as it is proposed in the same conditions as B/PhoX, which was the only work we could reuse in the end. Moreover, using Coq as an underlying proof tool makes the durability of BiCoax linked to the effective durability of Coq.

In light of the previous comparisons, we thus make the strong but independently verifiable claim that BiCoax is the *most complete* academic tool for proving B or event-B projects, *complete* to be read in the sense of “matching the BBook”.

We shall conclude this section by mentioning the provers of B commercial tools. Their main objective is very pragmatic: having an efficient automated proof tool for a possibly big number of proof obligations. They function mostly by saturation of premises until a contradiction in the hypotheses is found. They also include mechanisms for the users to specify their own decision procedures. For instance, the prover of the B-toolkit includes a tactic language comparable to the tactic language of Coq, although much simpler. Those commercial provers are also based upon an adapted B set-theoretic logic, hence comparing them with academic tools is somewhat less relevant. What makes these tools interesting is their efficiency: they are thus a good choice for evaluating the performance of B academic proof tools.

3 A short presentation of the involved formalisms

3.1 The B and event-B formal methods

B is a formal software development method used to model and reason about systems [1]. The B method has proved its strength in industry with the development of complex real-life applications such as the Roissy VAL [7]. The principle behind building a B model is the expression of system properties which are always true after each evolution step of the model. The verification of a model correctness is thus akin to verifying the preservation of these properties, no matter which step of evolution the system takes. A successor of B called event-B[21] takes the modelling further in allowing the reasoning about systems in an event-based fashion. The verification step of an event-B model is done in a similar way as B with taking into account deadlock problems due to the fact the formalism is event-based.

In both cases, the verification entails the generation of so-called *proof obligations* (POs) which are set-theoretic first-order logic formulas to be proved in the context of the B or event-B theory. PO generation for B is supported by several tools like B4Free[6], , AtelierB[4] , the B-toolkit[5] or the BGOP of our BRILLANT [18] platform. For event-B, the only tool is Rodin[24]. The proof is also supported by the tools mentioned previously except for the BGOP, whose only task is PO generation. Proof in BRILLANT shall be supported by BiCoax.

There are a few differences in the theory of B and event-B, though. Event-B gives different definitions for some of the basic constructs,supposedly compatible with B. As BiCoax is primarily about validating the BBook, we shall focus in this document on B alone, although we might occasionally mention event-B to state how our implementation fares or shall fare in an event-B context.

3.2 Coq

Coq [25] is a proof assistant based on the calculus of inductive constructions. This implies the following important points:

- The logic is an intuitionistic one, hence properties such as excluded middle or the axiom of choice, fundamental in the logic of B, are not available immediately. Coq fortunately provides modules in its libraries for such axioms
- Types are first-class citizens. It is possible to build datatypes associating basic values with types or to make the existence of a datatype rely upon the validity of a proposition, for instance.

Types are also organized along a hierarchy of *sorts*:

Set is the sort of program types. If this datatype is supposed to be implementable, this sort shall be preferred. Natural numbers and integers belong to `Set`, for instance

Prop is the sort of propositions, hence logical formulas

Type is the sort of `Set` and `Prop` and it constitutes the rest of the type universe hierarchy: a datatype built upon another datatype of $Type_n$ will inhabit $Type_{n+1}$, although these indices are hidden to the user.

The following example shows how the union property of two sets is modeled in the ENSEMBLES module of Coq's standard library:

```
Inductive Union (B C:Ensemble) : Ensemble :=
  | Union_introl : ∀ x:U, In B x → In (Union B C) x
  | Union_intror : ∀ x:U, In C x → In (Union B C) x.
```

This datatype can be interpreted two ways. In the propositional perspective, Union_introl and Union_intror are axioms describing the properties of the union from the notion of set belonging. In the type theory perspective, Union_introl is an element of type **Union B C** and it is parametrized by a (set belonging) property.

Another important functionality of Coq in our context is the ability to write *tactics* for automating repetitive proof tasks. Thanks to constructs reminiscent of a programming language and to pattern-matching, it is possible to associate specific proof steps to goals of a known shape. These proof steps can involve existing tactics and theorems.

4 Implementation choices

After a short presentation of the targetted organization of our library and of the design choices made before implementation, we exhibit its particularities.

4.1 Organization and design choices

Ideally, it should be possible to trace BiCoax theorems and definitions back to the corresponding entry of the BBook. We thus chose to split BiCoax the same way as the part of the BBook we model. The BCHAPTER1 module introduces the theorems of first-order classical logic (we reuse the logical connectives of Coq), the BCHAPTER2 introduces basic and derived set constructs and BCHAPTER3 introduces high-level constructs. In order to manage code size, we chose to have one file for each section. There are two noticeable exceptions: the first chapter and properties listings. The first chapter holds in one single file because no definitions are introduced and proofs are short. The properties listings of the BBook follow the "one table = one file" policy. Finally each definition or theorem shall refer in a comment the corresponding page, section and table row of the BBook, when applicable. Table 2 sums up what sections of the BBook are modelled, into how many and which files.

When implementing the various definitions of the BBook, we used the following guideline: if a matching definition exists in the standard library of Coq, we use it. If not, we implement it preferably as an inductive definition: this constitutes more of a stylistic choice which is consistent with the module of the standard library we reuse the most. In all cases, BiCoax is initially based on B/PhoX[23] which means that the equivalence or the equality of each translated operator shall be assessed. Hence each BiCoax module containing definitions will also contain theorems showing the equality or the equivalence of the definition with the corresponding BBook definition.

BiCoax might also be used by non-specialists in the future: POs can be big, hence readability of formulas is an important concern. Coq allows the definition of additional Unicode notations and event-B introduced Unicode notations for B constructs [21].

Chapter	BBook section	Filled-in files	files
1	All	1	BCHAPTER1
2	Basic set constructs	2	BBASIC, BINCLUSION_PROPS
	Derived constructs	2	BDERIVED_CONSTRUCTS{,_PROPS}
	Relations	1	BRELATIONS
	Functions	1	BFUNCTIONS
	Catalogue of properties	13	..._LAWS, EQUALITIES_...
3	Generalized Intersection/Union	1	BGENERALIZED_UNION_INTER
	Fixpoints	1	BFIXPOINTS
	Finite sets	1	BFINITE_SUBSETS
	Infinite sets	1	BINFINITE_SUBSETS
	Natural numbers	13	BNATURALS{_BASICS,...}
	Integers	1	BINTEGERS{_BASICS,...}
	Finite sequences	0	BSEQUENCES
	Finite trees	0	BTREES
	Labelled trees	0	BLABELLED_TREES
	Binary trees	0	BBINARY_TREES
	Well-founded relations trees	0	BWELL_FOUNDED

Table 2. Global organization

We thus decided to introduce a notation scope that can be activated when necessary. This notation scope also follows the guidelines of [21] for associativity and priority of notations. Some event-B notations are in a "private" zone of Unicode because no corresponding symbol exists officially: in that case we decided to reuse other similar-looking symbols. Here follows an example of Unicode notation for set belonging:

`Notation` "x ∈ y" := (In *y x*) (`at` *level* 11, *no associativity*): *eB_scope*.

Many definitions are parametrized by the types of the sets they manipulate. In that case we declared these type parameters implicit so that Coq infers them. This design choice makes the formulas feel less crowded.

We shall now detail what parts of Coq were reused and when not, what changes we introduced w.r.t. the definitions of the BBook.

4.2 From B to Coq

Reused constructions. BiCoax is a shallow embedding, hence we reuse some parts of Coq for our benefit.

The most basic connectors are directly reused, i.e. logical connectors (conjunction, disjunction, negation, etc) and pairing. On a related note, reusing the pairing of Coq shielded us from discovering the fact that the injectivity of pairing is never showed in B, as noticed by Jaeger in the French version of [20] at the end of Sec. 8.2.

We reused the definitions of the Coq's standard library ENSEMBLES module for basic set constructs: set belonging, inclusion, union, intersection, empty set, set difference. This reuse is also justified by the fact that set belonging in this module match very closely set belonging in B. Let U be a datatype and A a value of type (Ensemble$U$): A is actually a function of U to propositions ($U \rightarrow Prop$). Set belonging is defined as :

`Definition` In (A:Ensemble) (x:U) : `Prop` := $A\ x$.
This means that for the ENSEMBLES module, belonging to A is the same as verifying the A predicate, which matches exactly the B definition of set belonging.

We also imported the CLASSICALEPSILON module for the choice operator as well as related constructs and tactics proposed by Castéran [15]. We could not reuse the notion of relation of the ENSEMBLES module because it lacks the property being itself a set (of couples), hence we modeled relational and functional constructs in the "B way" upon the basic set constructs mentioned above.

Earlier communications about BiCoax raised the question of why we decided to reuse the ENSEMBLES module instead of FSETS. This is because this module requires an ordered datatype upon which sets of this datatype can be built. In B, the only basic datatype is that of the BIG abstract set: as we have no information about the order of its elements, it was thus not possible to use FSETS. Let us note here that it would be possible to do so with event-B as it defines the natural numbers as a basic datatype, which we know are an ordered type.

Our rule of thumb for definitions is to use inductive constructions when possible. The third chapter of the BBook introduces at its beginning the notion of fixpoint and makes almost systematic use of it for defining the rest of B constructs: this is where our choice of Coq becomes the most fruitful as inductive (and thus fixpoint-based) constructs are pervasive in Coq. As a consequence, all inductive constructs of B are translated into inductive definitions. The existing inductive definitions we reused were the notion of finite sets as the **Approximant** of the ENSEMBLES module, the notion of natural number and all arithmetic operators for natural numbers of the ARITH module.

Proposed constructions When not reusing definitions of the standard library of Coq, we had to balance ease-of-use and traceability to the BBook of the definitions we proposed. While it is not the place here to present all definitions we introduced, we think that the definition of partial function sums up what challenges choosing a definition poses. The BBook suggests two equivalent definitions of partial function:

- $\{f \mid f \in A \leftrightarrow B \wedge (f^{-1};f) \subseteq id(B)\}$
- $\{f \mid f \in A \leftrightarrow B \wedge$
  $\forall(x,y,z).(x,y,z \in A\times B\times B \wedge x\mapsto y \in f \wedge x\mapsto z \in f \Rightarrow y = z)\}$

The second definition make explicit the image unicity property for functions, which is often used in proofs involving partial functions. This property can of course be deduced from the first form of the definition, but it makes it less immediately usable for the end-user. In the second definition, the type information about x, y and z seems redundant: it should be deducible from the belonging of $x \mapsto y$ and $x \mapsto z$ to f. These remarks lead us to the final shape of the corresponding BiCoax definition:

`Inductive` **partial_function** ($U\ V$: `Type`) (A: Ensemble U) (B: Ensemble V): Ensemble (Ensemble (U*V)) :=
  pfun_intro : ∀ (f: Ensemble (U*V)),
    In (**relation** $A\ B$) f
  → (∀ (x: U) (y: V), In f (x,y) → (∀ (z: V), In f (x,z) → y=z))
  → **partial_function** $U\ V\ A\ B\ f$.

The use of a curryfication of pfun_intro arguments instead of a conjunction also removes one or two additional decomposition steps in proofs later on.

In the end, we believe it to be easy for the end-user to manipulate the various definitions. The danger is then that the definitions we propose do not match the BBook definitions anymore. This imposes on us the verification that our definitions are equivalent to the BBook definitions.

4.3 Validity of modified constructs

Let us look at the definition of shallow embedding of a B term in PhoX as defined in [23], where † is a translation function:

$$(f\ t_1 \cdots t_n)^\dagger \equiv f^\dagger\ t_1^\dagger \cdots t_n^\dagger$$

For the embedding to be correct, it must guarantee that it is the same to handle a translated complex term or to handle the same term for whom each subterm is translated. When the embedding is a deep one, this verification can be assisted by the tool used for it. Because our embedding is a shallow one, we can also do such a verification but it is meaningful only for connectors directly reused: for instance, proving with Coq that $(P \leftrightarrow Q) \leftrightarrow (P \leftrightarrow Q)$ is trivially verified.

Fortunately this meaninglessness is limited to logical connectives. For other constructs, the verification, even if done within Coq itself, is more meaningful. When the definition is predicative we use the logical equivalence of Coq and for terms we used Coq's Leibniz equality, whether they be sets or other constructs. Here we advise the reader to see e.g. the verification of the validity of set union in the BDERIVEDCONSTRUCTS module.

When dealing with datatypes the verification becomes a bit more convoluted as the operators defined upon this datatype must also be transformed. We thus introduce homomorphic functions for translating one datatype into the other and we verify that one operator in the one datatype is isomorphic with the corresponding operator in the other datatype. For instance, in the BNATURALS* modules the homomorphisms are nat_of_bbN and bbN_of_nat and for all arithmetic operators (except logarithm at the time of writing) the isomorphism is verified.

All the design choices we presented here gave us a clear guideline for implementing the first part of the BBook and we present some theoretical results and remarks in section 5.

5 Theoretical results

So far BiCoax amounts to about 27000 lines, or 768KB. It contains 1163 theorems and lemmas and was written with 3 estimated person-months. What has been implemented is split between:

Chapter 1 All non-trivial theorems about first-order logic with predicates have been proved. This part is somewhat trivial but what makes it most interesting is seeing which theorems actually require the excluded middle for being proved

Chapter 2 All this chapter was implemented and all properties were proved

Chapter 3 All sections up to and excluding integers have been implemented. The only theorems left unproved at the moment are the Dedekind-infinity theorem for infinite sets and the properties about natural logarithm, for which we also proposed a definition purely based on the nat datatype of Coq.

At this point, we found very few mistakes in the BBook, which makes it a solid reference for basic mathematical concepts. The mistakes include:

- What we think are typographical errors, e.g. a u instead of a v in a hypothesis
- False properties, the ones numbered 33 and 34 in MEMBERSHIP_LAWS. Given their location, they are most likely copy/paste errors
- Property 2.5.1, where the right-to-left implication is actually false. This mistake most probably occurred because one of the hypotheses needed for proving the definition was overlooked (which the use of a tool would not allow). As a consequence the proof of theorem 3.5.3 is wrong, while the theorem itself is true: we did the proof differently, because the theorem 3.5.3 is needed after
- Properties of addition: the codomain of the addition was deduced to be $\mathbb{N}$ from theorem 3.5.3, while it can only be deduced to be $\mathbb{P}(BIG)$. Many definitions depend on addition, hence blindly following the theorems would have induced too many chances. As a consequence we decided to define the addition upon $(\mathbb{N} \lhd \texttt{succ})$, which leads to the desired properties, instead of `succ`.

We put counterexamples and more detailed explanations in an ERRATA file distributed along with BiCoax. The following sections present interesting or difficult points pertaining to our implementation.

5.1 Axioms in BiCoax and dependability

The dependability of BiCoax can be attributed to the trustworthiness of the following items: Coq, the axioms we included and the non-inconsistency brought by the introduced axioms. Coq has existed for about two decades and many academic and industrial users trust it, hence we will not discuss it further here.

The very fundamental axioms we needed and thus included are: the excluded middle (EM), the constructive indefinite description (epsilon) and set extensionality (Set_ext). The derived axioms we introduced so far are the infinity of the *BIG* set and the axioms related to integer negation for defining the negative part of the set of integers in the BBook. If BiCoax were to be inconsistent, it would thus come from the use of these axioms which are necessary for our implementation.

Without entering into details, we know that the excluded middle, while making a strong assumption about the decidability of statements, does not cause inconsistencies by itself. Used with the axiom of choice, it implies proof-irrelevance [8], which is still not inconsistent. As epsilon can be seen as a weaker form of the axiom of choice, it is then very possible that proof-irrelevance is present in our implementation. We also know that epsilon allied with the impredicativity of Coq's Set sort leads to inconsistency [15, introduction]. As of version 8.0, Set is predicative by default hence we also avoided inconsistency here.

As a consequence, the other places from where inconsistencies might originate are the derived axioms coming from B: it can thus be said that BiCoax can be considered as dependable as B. We however do not plan to scrutinize these axioms in the near future to look for potential inconsistencies.

5.2 Pitfalls of function application

As reckoned by Jaeger [19, section 3], implementing function application in a shallow embedding is a tricky exercise. According to the BBook, functional application requires the choice operator (epsilon) and a relation having the property that the image of any of its element is unique (the functional property). As we used Castéran's [15] *unique choice* iota operator, unicity becomes an intrinsic property of the image.

Definition app (*U V*:Type) (*A*: Ensemble *U*) (*B*: Ensemble *V*)(*f*: Ensemble (*U**V)) (*x*:U) (*applicability*: In (**partial_function** *A B*) *f* ∧ In (**domain** *f*) *x*) :=
  iota *V*
    (codomain_unique_inhabitation *U V A B f x applicability*)
    (fun *y*:V ⇒ *f* (*x*,y)).

This might be the definition the most foreign to its B origin, as the problem of its soundness in Coq comes much into play: it requires an additional parameter which is a proof that the datatype of the codomain is inhabited. Fortunately such a proof can actually be deduced from the the mandatory well-definedness side-condition of functional application. The definition above is thus what we think is the best trade-off between Coq soundness and friendliness to the end user, who is likely to have already seen proofs of well-definedness in other B tools. Its use in formulas requires an additional proof of existence of well-definedness. This explains why the EQUALITIES_EVALUATION module contains so many existential quantifications.

5.3 The types of B and Coq

As expected, BiCoax exhibits some of the peculiarities of B typing. For instance, any set must be based on a given datatype: hence there is not one but several empty sets [1, section 2.3.3]. Overlooking this fact when implementing a prover "from scratch" might lead to inconsistencies. We still could make it look like there is only one empty set without breaking typechecking thanks to Coq's notation mechanism:

Notation "∅" := (**Empty_set** _) (at *level* 10, *no associativity*): *eB_scope*.

Coq is indeed able to infer type parameters when they are simple enough: this permitted us to propose a unified notation for the empty set.

We also removed in some definitions the "B typing" parts, i.e. the predicates stating to what sets the bound variables belong. Indeed, these predicates are sometimes redundant (see e.g. the partial function definition of Sec. 5.2). Systematically proving the equivalence of the new definition with the corresponding BBook definition comforted us in our action.

5.4 Natural numbers and arithmetic operations in BiCoax

Implementing the part of the BBook introducing natural numbers faced us with a dilemma: reusing the convenient Coq's natural numbers or strictly following the BBook. In order not to sacrifice usability, we decided to do both, finding easy-to-overlook peculiarities in the process. For using both datatypes, we had to propose an isomorphism in the form of two homomorphims: bbN_of_nat is a recursive function with a straightforward definition similar to the B fixpoint construct for natural numbers. The nat_of_bbN homomorphism is simply the cardinal of the B natural, which is a subset of BIG.

We proved that B arithmetic operations and Coq's functions are equivalent, sometimes *under conditions*. Coq functions are indeed total w.r.t. their arguments (e.g. $0 - 1 = 0$ for natural numbers in Coq) while B operations are partial ($x - y$ is defined under the condition that $x \geq y$). This suggests that for any other tool implementing B naturals, the partiality of these operations may have been overlooked: hence some formulas might be provable when they should not.

As a conclusion for the theoretical side of BiCoax, we can state that our work, while not ground-breaking, is useful. It helped the trimming down of previously uncovered mistakes of the BBook. It has also helped and will help in stating how B types and structures match their "programming-like" counterparts as we did for arithmetic operations on natural numbers. We shall now present how BiCoax fares on the more practical side.

6 Experimental results

Experimentations with BiCoax were twofold: using it in a purely automated way for determining what the tools in the upper part of the toolchain are expected to provide and interactively with B toy projects for getting a better idea of the completion of BiCoax.

6.1 At the end of the B toolchain

We inherited from B/Phox, the ancestor of BiCoax, a way of generating PhoX proof obligations from B POs described in an XML format. We adapted the XSL stylesheet for doing so to Coq instead of PhoX, resulting in the `bgop2bicoax` XSL stylesheet. The transformation from a PO to BiCoax is straightforward: after inserting the importing of the BLIB module of BiCoax, the formula is translated to a Coq theorem using the B constructs defined in BiCoax. A tactic call is inserted, followed by a proof-saving command. Nowadays this part of the toolchain is mostly used for correcting the PO-to-BiCoax transformation step.

Our first tests were with a publicly available B project, the boiler, in a "case-study" flavor and an "industrial" flavor. The PO generator of BRILLANT issues 2335 POs for the case-study boiler and 2563 for the industrial boiler. PO generation is as close to the atelierB as possible so as to make subsequent comparisons more relevant. In a nutshell, successful proofs (see Tab. 3) correspond to typing proofs and are realized in 45s on average with the (still experimental) `firstorder` Coq tactic. Failures come mostly from B disambiguation errors (such as cartesian product and multiplication) or XSLT transformation errors (missing constructors, etc). The various errors we encountered are as follows:

T1: `Cannot infer an instance for the implicit parameter U of app`
T2: `The reference ... was not found in the current environment`
T3: `Cannot infer a term for ...`
T4: `',' expected after [binder_list] (in [constr:binder_constr])`
T5: `Cannot infer a type for ...`
T6: `The term "..." has type "..." while it is expected to have type "..."`
T7: `Attempt to save an incomplete proof`

	Case-study	Industrial
Nbr PO	2335	2563
Nbr BPhoX	1871	43
Nbr BiCoax	237	52
Error T1	1446	24
Error T2	310	2450
Error T3	175	8
Error T4	66	15
Error T5	25	1
Error T6	20	-
Error T7	13	11

Table 3. Proof results

The discrepancy of T1 and T2 errors in the proof results seems to be a consequence of the more complex shape of expressions in the industrial version of the boiler. In the end, the use of BiCoax in an automated fashion seems possible but will require corrections in the upper part of the toolchain in the near future.

6.2 Using BiCoax interactively

It is our belief that the interface to proof tools should be easy to use interactively, because on non-trivial B projects the end-user will spend most of his/her proving-time issuing commands to the prover (the automated part is invisible, as expected). Hence interactive proof shall be made easy, which was also part of our motivation in using Coq as a proof tool.

As a quick test of the usability of BiCoax, we tried to prove two B toy projects, the "LittleExample" project appearing in [1, Sec. 11.2] or [19] and a bounded stack, with no further assumptions than the POs translated manually. As a result, the proofs were successful and took no more than half an hour for each project. They are present in the BMISC module of BiCoax along with the corresponding B projects in comment. It is this somewhat unexpected success that turned our opinion to the idea that BiCoax is almost ready for mundane proof tasks. Here follows a top-down proof tree of the demonstration of the PO assessing that the `read` operation preserves the invariant of the "LittleExample" machine:

$$
\begin{array}{cl}
\vdash \forall(n,y).y \in FIN(\mathbb{N}_1) \land n \in \mathbb{N}_1 \Rightarrow (y \cup \{n\}) \in FIN(\mathbb{N}_1) & \\
\hline
n : Z, y : (Ensemble\,Z), H : (y \in FIN(\mathbb{N}_1) \land n \in \mathbb{N}_1) \vdash (y \cup \{n\}) \in FIN(\mathbb{N}_1) & \text{intros} \\
\hline
n : Z, y : (Ensemble\,Z), H : (\dots) \vdash (\{n\} \cup y) \in FIN(\mathbb{N}_1) & \text{rewrite commutativity\_1} \\
\hline
n : Z, y : (Ensemble\,Z), H : (\dots) \vdash (n,y) \in (\mathbb{N}_1 \times FIN(\mathbb{N}_1)) & \text{apply augmented\_set\_in\_finite\_sets} \\
\hline
\dots, H : (y \in FIN(\mathbb{N}_1) \land \dots) \vdash y \in FIN(\mathbb{N}_1) \qquad \dots, H : (\dots \land n \in \mathbb{N}_1) \vdash n \in \mathbb{N}_1 & \text{constructor} \\
\hline
 & \text{intuition} \quad \text{intuition}
\end{array}
$$

We perceived that our experience with proving the above projects could be better though: some very trivial theorems were missing, such as the non-emptiness of a singleton, and tactics could have helped us solving repetitive proof tasks. In the future, such interactive sessions shall give us directions for automating the proof of B projects, maybe to the point of proposing specialized tactics for domain-specific B projects.

Because the projects we proved are not very complicated, we could not illustrate here the full extent of using other tactics provided by Coq. For instance, when proving equality of arithmetic equations we can use the `ring` tactic: it normalizes arithmetic terms and attempts to prove their equality. This tactic is actually defined for structures that form a ring (hence here, natural numbers) but it could be applied to any other ring (e.g. one formed with set operations). Another high-level tactic called `omega` follows the same idea for mixes of equations and inequations in Presburger arithmetic.

7 Conclusion

We have proposed a shallow embedding of the mathematical foundations of B into Coq, in order to provide a proof tool for B based on a generic proof assistant and in order to validate the definitions and theorems of the BBook through implementation. When relevant, we proposed handier alternative definitions, either ours or from Coq's library. We systematically proved the equivalence with the corresponding B definitions in the process, to the extent allowed by a shallow embedding.

Our implementation covers the first two and a half chapters of the BBook up to and almost including the integers, which amounts to about 1100 theorems. This implementation helped us uncover minor BBook mistakes not documented elsewhere. It also brought more focus on confusing or overlooked parts of B. For instance, the difference between "B typing" as set belonging and typing in the sense of type theory is hopefully clearer. Moreover we exhibited potential pitfalls of the partiality of B arithmetic operators (subtraction, division and natural logarithm) which might have been overlooked in other implementations of B.

Experimental results showed that an automated use of BiCoax is feasible but it mostly requires changes in the upper part of the B toolchain. The ease, initially unexpected, with which we proved two B toy projects leads us to believe that BiCoax is almost ready for the proof of mundane B projects.

The short-term perspectives include what we know or perceive BiCoax lacks: the sections of the BBook left to be implemented, trivial theorems not present in the BBook and tactics for repetitive proof tasks. Long-term perspectives include the implementation of several of the numerous B extensions that appear in the literature. Such extensions consist for instance in the introduction of other datatypes in B or the extension of B with a temporal logic [17].

Acknowledgements We would like to thank Arnaud Lanoix for proof-reading earlier versions of this paper.

References

1. Jean-Raymond Abrial. *The B Book - Assigning Programs to Meanings*. Cambridge University Press, August 1996.
2. AFADL2000, LSR/IMAG. *Approches Formelles dans l'Assistance au Développement de Logiciels*, LSR/IMAG – BP 72 38402 Saint-Martin d'Heres Cedex – Grenoble – France, January 2000. LSR/IMAG.
3. AFADL2003, IRISA. *Approches Formelles dans l'Assistance au Développement de Logiciels*, IRISA Rennes – France, January 2003. IRISA.

4. AtelierB. http://www.atelierb.eu.
5. B-Toolkit. http://www.b-core.com.
6. B4Free. http://www.b4free.com.
7. F. Badeau and A. Amelot. Using B as a high level programming language in an industrial project: Roissy VAL. In *Formal Specification and Development in Z and B*, volume 3455 of *LNCS*, pages 334–354. Springer-Verlag, 2005.
8. Franco Barbanera and Stefano Berardi. Proof-irrelevance out of excluded-middle and choice in the calculus of constructions. *Journal of Functional Programming*, 6:519–526, 1996.
9. Karim Berkani, Catherine Dubois, Alain Faivre, and Jérôme Falampin. Validation des règles de base de l'Atelier B. In *AFADL'2003* [3], pages 121–136.
10. Didier Bert, editor. *B'98 : The 2nd International B Conference, Recent Advances in the Development and Use of the B Method*, volume 1393 of *Lecture Notes in Computer Science (Springer-Verlag)*, Montpellier, April 1998. B1998, LIRRM Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier, Springer-Verlag.
11. Didier Bert, Jonathan P. Bowen, Martin C. Henson, and Ken Robinson, editors. *ZB'2002 – Formal Specification and Development in Z and B*, volume 2272 of *Lecture Notes in Computer Science (Springer-Verlag)*, Grenoble, France, January 2002. LSR-IMAG.
12. J.-P. Bodeveix, Mamoun Filali, and C. A. Munõz. Formalisation de la méthode B en COQ et PVS. In *AFADL'2000* [2], pages 96–110.
13. Jean-Paul Bodeveix and Mamoun Filali. Type synthesis in B and the translation of B to PVS. In Bert et al. [11], pages 350–369.
14. Jonathan P. Bowen and Michael J. C. Gordon. A shallow embedding of Z in HOL. In *Information and Software Technology*, pages 269–276, 1995.
15. Pierre Castéran. Utilisation en Coq de l'opérateur de description. pages 30–44, January 2007.
16. Pierre Chartier. Formalisation of B in Isabelle/HOL. In Bert [10], pages 66–82.
17. Samuel Colin. *Contribution à l'intégration de temporalité au formalisme B : utilisation du calcul des durées en tant que sémantique pour B*. PhD thesis, Université de Valenciennes et du Hainaut-Cambrésis, October 2006.
18. Samuel Colin, Dorian Petit, Jérôme Rocheteau, Rafaël Marcano-Kamenoff, Georges Mariano, and Vincent Poirriez. BRILLANT : An open source and XML-based platform for rigourous software development. In *SEFM (Software Engineering and Formal Methods)*, Koblenz, Germany, September 2005. AGKI (Artificial Intelligence Research Koblenz), IEEE Computer Society Press.
19. Éric Jaeger and Catherine Dubois. Why would you trust B ? In *LPAR, 14th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 288–302, 2007.
20. Éric Jaeger and Thérèse Hardin. A few remarks about formal development of secure systems. In *HASE '08: Proceedings of the 2008 11th IEEE High Assurance Systems Engineering Symposium*, pages 165–174, Washington, DC, USA, 2008. IEEE Computer Society.
21. C. Métayer, J.-R. Abrial, and L. Voisin. RODIN deliverable 3.2: Event-B language, May 2005. http://rodin-b-sharp.sourceforge.net.
22. Christophe Raffalli and Paul Rozière. *The PhoX Proof checker Documentation*. version 0.83.
23. Jérôme Rocheteau, Samuel Colin, Georges Mariano, and Vincent Poirriez. évaluation de l'extensibilité de phoX: B/PhoX un assistant de preuves pour B. In Valérie Ménissier-Morain, editor, *Journées Francophones des Langages Applicatifs (JFLA 2004)*, pages 37–54. INRIA, 2004.
24. Rigorous Open Development Environment for Complex Systems. http://www.event-b.org/platform.html.
25. The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.2*, 2008. http://coq.inria.fr/doc-eng.html.