Coq, l'alpha et l'omega de la preuve pour B ?

Samuel Colin[1] & G. Mariano[2]

1: scolin@hivernal.org
2: Institut National de Recherches sur les Transports et leur Sécurité
georges.mariano@inrets.fr

Résumé

BiCoax, un plongement léger de la théorie de B en Coq, est la reprise d'une mise en œuvre réalisée précédemment avec PhoX. Cet article décrit les choix d'implémentation faits dans BiCoax et comment ces choix font écho, ou non, à la théorie de B.

1 Introduction

1.1 Objectifs

BICOAX[1] est une bibliothèque Coq [The Coq Development Team – LogiCal Project, 2006] pour la preuve de formules décrites dans la logique en théorie ensembliste constituant le corpus mathématique de base de la méthode formelle B. Ce travail reprend et continue les travaux similaires de [Rocheteau et al., 2004] qui avaient été réalisés pour un autre assistant de preuve, PhoX [Raffalli and Rozière,].

L'objectif originel de ce travail était de disposer d'un outil de preuve pour B sur la base d'un assistant de preuves génériques. Cet objectif s'est vu adjoindre un objectif secondaire de validation et de traçabilité de la «bible» de la méthode B, le BBook [Abrial, 1996]. En effet cet ouvrage présente très précisément les fondements mathématiques de B et cite de nombreuses propriétés et théorèmes sur les constructions mathématiques de B. Quelques-unes des propriétés présentées sont étayées d'une démonstration mathématique mais le nombre de ces démonstrations est anecdotique par rapport au nombre des propriétés énoncées.

À des fins d'intégrité, les fondements mathématiques de toute méthode formelle se devraient d'être validés par des outils lorsque cela est possible, ne serait-ce que pour s'assurer de l'absence d'incohérence dans ces fondements mathématiques. BiCoax s'inscrit dans cette démarche.

Après une rapide présentation des formalismes B et Coq concernés, en section 1.2, nous précisons les liens de BiCoax avec des travaux similaires en section 2. Nous décrivons nos choix d'implémentation en section 3 puis en déduisons un retour sur expérience en section 4. Le but de BiCoax étant de servir d'outil de preuve nous avons également commencé quelques expérimentations dans ce sens décrites en section 5. Nous résumons tous ces points en section 6.

1.2 Une (courte) présentation des formalismes impliqués

Cet article suppose que le lecteur est familier de la théorie des ensembles et de la notion de logique intuitionniste.

[1] BICOAX peut être récupéré via Subversion (`https://gna.org/svn/?group=brillant`) – le serveur est cependant souvent chargé – ou via une archive placée dans `http://download.gna.org/brillant/snapshots/`

1.2.1 La méthode B

B est une méthode formelle utilisée pour modéliser et raisonner sur des systèmes. Elle se caractérise par une démarche formelle, incrémentale et modulaire allant d'une modélisation mathématique initiale basée essentiellement sur la théorie des ensembles et la logique du premier ordre jusqu'à une génération automatique du code cible [Abrial, 1996]. Cette démarche est soutenue par la validation de conditions de vérifications, appelées obligations de preuves («OPs»), automatiquement générées.

Le principe qui sous-tend la construction d'un modèle B est l'expression de propriétés systémiques toujours vraies après chaque étape d'évolution du modèle. La vérification de la correction de ce modèle se ramène donc à la vérification de la préservation de ces propriétés, quelle que soit l'étape d'évolution retenue par le système.

Ces propriétés sont spécifiées dans la clause **`INVARIANT`** du modèle et l'évolution dans les méthodes de la clause **`OPERATIONS`**. La vérification d'un modèle B consiste en la vérification que chaque opération satisfait l'**`INVARIANT`** sous l'hypothèse que la précondition de l'opération et l'invariant soient vrais lors de l'appel de la méthode. Cette façon de faire est très proche de celle d'outils comme Why [Filliâtre and Marché, 2007].

L'une des forces, et non des moindres, de la méthode B réside dans sa progression incrémentale : le raffinement d'un modèle rend celui-ci moins indéterministe et plus précis de par l'introduction de constructions qui rapprochent B d'un langage de programmation usuel. Ce raffinement peut se répéter jusqu'à ce que le modèle puisse effectivement être traduit dans un langage de programmation donné.

La cohérence d'un raffinement doit elle aussi être vérifiée, cependant la différence cette fois est que les obligations de preuves portent directement sur la relation entre l'opération raffinée et son raffinement. Le raffinement peut donc être vu à la fois comme une opération de traduction et comme une opération de simplification de la vérification d'un modèle. Notons que c'est là une différence importante avec des formalismes comme Why qui ne dispose pas du raffinement, sinon par des moyens détournés.

Cette caractéristique, associée à la modularité explicite des modèles formels B, est une justification de la dénomination «méthode». B ne propose pas simplement un langage formel mais son utilisation constitue, de fait, une démarche méthodologique.

Industrialisation La méthode B a fait la preuve de son adéquation avec certains besoins industriels de par le développement d'applications complexes de taille réaliste comme le VAL de Roissy [Badeau and Amelot, 2005], pour ne citer que l'exemple le plus récent.

Support outillé Il existe des outils commerciaux tels B4free (`http://www.b4free.com`) ou l'Atelier B (`http://www.atelierb.eu`) qui permettent de générer et vérifier les OPs pour s'assurer de la cohérence d'un projet B. Notre but ici est de reproduire la partie «preuve» de cette étape de vérification via un assistant de preuves générique, la partie «génération des OPs» étant supposée réalisée par un autre outil (dans notre cas, l'outil BGOP de la plate-forme BRILLANT [Colin et al., 2005]).

1.2.2 Coq

Coq [The Coq Development Team – LogiCal Project, 2006] est un assistant de preuves basé sur le Calcul des Constructions équipé de types inductifs. Cela implique plusieurs points importants :

- La logique est de type intuitionniste, donc des propriétés comme le tiers exclu ou l'axiome du choix, caractéristiques de la logique de B, ne sont pas disponibles de base
- Les types sont des valeurs de première classe. Ainsi il est possible de construire des types de données associant des valeurs de type simple à des types, ou de rendre l'existence d'un type de données dépendant de la validité d'une proposition, par exemple
- Il est possible de construire des types de données exprimés sous forme de (plus petits) points fixes, leurs principes d'induction étant construits automatiquement.

L'exemple qui suit montre comment la propriété d'union entre deux ensembles est modélisée dans le module *Ensembles* de la bibliothèque standard :

Inductive *Union* (B C :*Ensemble*) : *Ensemble* :=
| *Union_introl* : $\forall$ x :U, *In* B x $\rightarrow$ *In* (*Union* B C) x
| *Union_intror* : $\forall$ x :U, *In* C x $\rightarrow$ *In* (*Union* B C) x.

Ce type de données peut être compris selon deux points de vue. Selon le point de vue propositionnel, *Union_introl* et *Union_intror* sont des axiomes qui décrivent les propriétés de l'union à partir de l'appartenance à un ensemble. Selon le point de vue des types, *Union_introl* est un élément de type *Union B C* et est paramétré par une propriété d'appartenance.

Les types étant des valeurs de première classe, ils sont organisés selon une hiérarchie de *sortes* :

Set est la sorte des types des programmes. Si la destination de ce type de données est d'être implémentable, on choisira plutôt cette sorte.

Prop est la sorte des propositions. Les formules logiques sont de cette sorte.

Type constitue le type de *Set* et *Prop* et constitue le reste de la hiérarchie des univers de types : un type de données construit sur la base d'un autre type de données habitant $Type_n$ habitera $Type_{n+1}$, mais ces indices, constitutifs de cette hiérarchie des sortes, seront cachés à l'utilisateur.

Une autre fonctionnalité importante de Coq dans notre contexte est la possibilité d'écrire des *tactiques* permettant d'automatiser des tâches de preuve répétitives. Grâce à des constructions qui rappellent un langage de programmation et la possibilité d'analyser la forme des buts à démontrer, il est possible d'associer à des buts donnés un cheminement de preuve spécifique utilisant des théorèmes et des tactiques déjà existantes.

2 Approches existantes de preuve pour B

Avant de traiter des travaux similaires aux nôtres, nous introduisons quelques notions pertinentes pour la compréhension de nos commentaires.

Par «plongement» nous entendons l'assimilation ou non de constructions logiques de B avec des constructions déjà existantes dans la logique cible retenue. Par exemple, l'utilisation des connecteurs logiques de l'assistant de preuves Coq formerait un plongement *superficiel* ou *léger* (shallow embedding). Une redéfinition de ces connecteurs constituerait plutôt un plongement *profond* (deep embedding).

Les domaines de B dont nous traitons comprennent plusieurs niveaux :
- les fondements mathématiques [Abrial, 1996, chapitres 1-3],
- les substitutions généralisées [Abrial, 1996, chapitres 4-5,9-10]
- la théorie des machines abstraites [Abrial, 1996, chapitres 6-8]
- le raffinement [Abrial, 1996, chapitres 11-13].

Dans une optique de validation, la couverture de chaque niveau correspond à des besoins complémentaires :

fondements La cohérence de la logique de B ainsi que du typage d'expressions B

substitutions La correspondance entre sémantique en transformateurs de prédicats et substitutions, ainsi que la cohérence du raffinement

machines La cohérence de la modularité de B, en machines abstraites et raffinées

Dans une optique d'outillage, la couverture se limite souvent aux fondements mathématiques, seuls ceux-ci étant nécessaires pour la vérification d'obligations de preuve.

Ces notions étant introduites, nous pouvons apporter des éclaircissements sur le rapport entre BiCoax et des travaux similaires.

2.1 Travaux similaires

Au regard des notions précédemment introduites, BiCoax est un plongement superficiel qui se limite aux fondements de B. Ce contexte étant précisé, nous présentons les travaux qui se rapprochent des nôtres ci-dessous en ordre chronologique.

A shallow embedding of Z in HOL [Bowen and Gordon, 1995] Z est un langage de spécifications formelles dont la méthode B s'est inspirée. Comme la méthode B, Z a ses fondements sur un langage du premier ordre et la théorie des ensembles, ce qui rend une comparaison avec BiCoax pertinente. La motivation de J. Bowen et de M. Gordon autour de Z et de HOL est proche de la nôtre : ils suggèrent HOL comme support de preuve automatique pour Z. Ils justifient le choix d'un plongement superficiel pour limiter les notations complexes.

Formalisation of B in Isabelle/HOL [Chartier, 1998] Isabelle/HOL est un méta-environnement de preuves dont l'ordre supérieur est hérité du couplage avec HOL. L'objectif de P. Chartier est de déterminer un prédicat d'ordre supérieur qui définit formellement la génération d'OPs d'une machine B et de démontrer sa validité. Le domaine de la méthode B qui est traité est par conséquent plus large que le nôtre puisqu'il concerne, outre les bases mathématiques de B, les substitutions généralisées et les machines abstraites. Il a donc réalisé un plongement profond du formalisme B dans Isabelle/HOL. À notre connaissance, ces travaux n'ont pas donné lieu à diffusion d'outils ou formalisation réutilisables.

Formalisation de la méthode B en Coq et PVS [Bodeveix et al., 2000] L'objectif de J.-P. Bodeveix, de M. Filali et de C.A. Muñoz est de formaliser les substitutions généralisées dans une logique d'ordre supérieur afin de valider les mécanismes propres à la méthode B dans un assistant à la preuve comme Coq ou PVS. Le plongement de B dans PVS est principalement un plongement profond qui est automatisé par l'outil PBS de Muñoz.

Type synthesis in B and the Translation of B to PVS [Bodeveix and Filali, 2002] L'objectif de J.-P. Bodeveix et de M. Filali dans cet article est de donner une sémantique fonctionnelle pour la méthode B. Cette traduction constitue un plongement superficiel similaire à celui de J. Bowen et M. Gordon [Bowen and Gordon, 1995] ou au nôtre quant aux bases mathématiques. Ce papier s'attache essentiellement aux conditions de typage du formalisme B. D'ailleurs, suite à ce papier, M. Filali a implémenté un vérificateur de types. Cet outil s'appuie d'ailleurs sur le *bparser* de la plateforme BRILLANT et avait été en retour intégré à la plateforme.

Validation des règles de base de l'Atelier B [Berkani et al., 2003] L'objectif de K. Berkani, C. Dubois, A. Faivre et J. Falampin est de démontrer la validité des règles du calcul des séquents de l'Atelier B – un logiciel commercialisé par Clearsy qui supporte le processus de développement d'un module B – règles qui n'ont pu être validées par le prouveur de l'Atelier B lui-même. Ils utilisent un plongement profond de B dans Coq afin de garantir la correction de règles de l'Atelier B.

Évaluation de l'extensibilité de PhoX : B/PhoX un assistant de preuves pour B. [Rocheteau et al., 2004] L'objectif de J. Rocheteau est de fournir un prouveur performant en termes de raisonnement équationnel sur une base générique. La mise en œuvre est basée sur le prouveur PhoX[Raffalli and Rozière,], dans lequel les constructions ensemblistes de B sont traduites. Comme la traduction de ces constructions est démontrée correcte, les preuves réalisées le sont aussi. Il est à noter que ces travaux sont les précurseurs directs de la bibliothèque BiCoax.

Why Would You Trust B ? [Jaeger and Dubois, 2007] L'objectif de É. Jaeger et C. Dubois est à plusieurs égards une réunion des objectifs des travaux ci-avant : fournir à la fois une validation de la théorie de B ainsi qu'un outil de preuve pour B. Pour mener à bien cette tâche, ils réalisent un plongement profond de B en Coq, y compris les connecteurs logiques. Outre de ne pas faire interférer la logique classique de B avec la logique intuitionniste de Coq, cela leur permet de proposer des procédures de décision. Ils sont aidés en cela par le choix d'une modélisation en indices de De Bruijn des variables quantifiées.

Synthèse Lorsque l'objectif est l'obtention d'un assistant de preuve performant, l'approche choisit usuellement le plongement superficiel. De ce point de vue [Jaeger and Dubois, 2007] fait figure d'exception. Lorsque l'objectif penche plutôt du côté de la validation d'éléments de la méthode B c'est alors le plongement profond qui est retenu. Le tableau 1 récapitule les différents points de convergence et de divergence de ces travaux.

	[Bowen and Gordon, 1995]	[Chartier, 1998]	[Bodeveix et al., 2000]	[Bodeveix and Filali, 2002]	[Berkani et al., 2003]	[Rocheteau et al., 2004]	[Jaeger and Dubois, 2007]	BiCoax
plongement								
Profond		$\star$	$\star$		$\star$		$\star$	
Superficiel	$\star$			$\star$		$\star$		$\star$
domaine de B								
Fondements	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$
Langage		$\star$	$\star$					
Machines		$\star$	$\star$					
objectifs								
Validation		?	$\star$		$\star$		$\star$	$\star$
Outillage	?			$\star$		$\star$	$\star$	$\star$

TAB. 1 – Synthèse des différents travaux

Le tableau 1 reste ambigü sur quelques points. En effet, si la couverture de B par [Bodeveix et al., 2000] va effectivement jusqu'aux machines B, la vérification de propriétés méta-théoriques ne peut se faire que jusqu'au niveau du langage de base. Concernant les objectifs de validation de [Jaeger and Dubois, 2007] et de BiCoax, ils sont différents. Alors que [Jaeger and Dubois, 2007] vise à une validation méta-théorique, nous visons seulement à une validation du BBook par la mise en œuvre des propriétés qui y sont énoncées.

Nous conclurons cette section en évoquant les outils commerciaux, tout du moins leur module de preuve intégré. Leur objectif est clair puisqu'il s'agit d'avoir à disposition un outil de preuve efficace en mode automatique sur un ensemble parfois très grand d'obligations de preuves. Leur fonctionnement se base sur le calcul des séquents et la saturation des prémisses jusqu'à trouver une contradiction. Bien que leur objectif soit plus directement pragmatique que les travaux sus-cités, ils fournissent néanmoins une base idéale pour jauger de l'efficacité d'autres outils de preuve.

2.2 Objectifs et caractérisation de BiCoax

Les objectifs premiers de BiCoax sont de fournir un outil de preuve pour B et de fournir une traçabilité des théorèmes de [Abrial, 1996] ce faisant. Il est donc en ce sens plus pertinent de le comparer avec [Rocheteau et al., 2004] et [Jaeger and Dubois, 2007].

BiCoax est basé sur une traduction des fichiers de BPhoX. À ce titre, le nombres de théorèmes du BBook vérifiés dans BiCoax a rapidement dépassé le nombre de ceux déjà vérifiés *via* BPhoX. Néanmoins l'implémentation BPhoX distinguait avec soin les théorèmes d'introduction de ceux d'élimination en vue d'aider le moteur de PhoX à parcourir les arbres de preuves. Ceci allié à la faible taille de PhoX font que l'outil résultant est a priori plus adapté à une utilisation pour la preuve en mode automatique.

BiCoax étant un travail initié récemment, la perspective immédiatement ultérieure à la complétion par rapport aux fondements mathématiques du BBook sera la mise en place de tactiques visant à une meilleure automatisation. Il est escompté que la perte initiale de performance lors du passage de PhoX à Coq soit rapidement compensée par l'efficacité des tactiques automatiques.

En dehors des considérations de performances, il est important de noter que BiCoax définit déjà plus de constructions B que ne le faisait BPhoX. Ainsi BiCoax est déjà plus pertinent pour la validation de projets que BPhoX, même si cette validation ne doit se faire pour le moment que de manière interactive.

Avec un objectf très similaire, la question de la ré-utilisation de BiCoq [Jaeger and Dubois, 2007] se pose naturellement. Plusieurs raisons nous ont néanmoins poussés à réaliser un nouveau développement :

– L'outil se doit d'être disponible et libre. Le but est de fournir un module de preuve à la plateforme BRILLANT. Or BiCoq n'est pas disponible au moment où cet article est rédigé. Nous ne savons pas, en outre, si la disponibilité future de BiCoq sera compatible avec les termes de licence de BRILLANT. Un re-développement restait donc le choix le plus prudent. Ce choix s'est avéré raisonnable, puisqu'obtenir une couverture complète ou presque du chapitre 2 de [Abrial, 1996] n'aura nécessité en définitive qu'un homme-mois, au plus.
– Le choix fait par [Jaeger and Dubois, 2007] d'utiliser des indices de De Bruijn fait sens dans un contexte de manipulation des formules. Cependant nous pensons que l'outil de preuve se doit de représenter le plus fidèlement possible les termes de l'OP à démontrer, ceci afin de faciliter les recherches de l'utilisateur sur une erreur dans les machines B (lorsqu'une preuve échoue). Les indices de De Bruijn ne facilitant pas cette traçabilité, nous avons préféré en rester à une approche plus classique.

Ces deux points ne préjugent cependant pas de l'efficacité intrinsèque de BiCoq. Il est donc tout à fait possible dans l'avenir de fournir via BRILLANT deux formats d'OPs, l'un pour BiCoax et l'autre pour BiCoq. Cette possibilité est plus floue concernant les autres travaux similaires, ceux-ci se basant sur des versions déjà anciennes de leurs outils de preuve respectifs.

3 Choix de mise en œuvre

Nous présentons dans un premier temps l'organisation globale de la bibliothèque puis nous nous intéressons ensuite aux particularités d'implantation.

3.1 Organisation globale de BiCoax

L'une des exigences portée sur BiCoax étant une traçabilité aussi fidèle que possible avec le BBook, nous avons choisi de respecter le plan fourni par le BBook :

Chapitre 1 Ce chapitre s'attache à la description de la logique propositionnelle du premier ordre, l'introduction des quantificateurs universel et existentiel, des paires ordonnées et une description basique des expressions

Chapitre 2 Il s'agit du chapitre qui introduit toutes les constructions ensemblistes et relationnelles de base, peuplant ainsi la catégorie des expressions

Chapitre 3 Ce chapitre construit à partir de la notion de point fixe, qu'il définit, des constructions pour les types de données (entiers naturels et relatifs) et des constructions de plus haut niveau comme les séquences et les arbres.

Le premier chapitre étant un chapitre assez court, limité à la vérification des théorèmes mentionnés dans le BBook, il est contenu dans un seul fichier.

Pour les chapitres 2 et 3, la séparation en fichiers suit globalement la règle «une section du livre = un fichier de définitions COQ», les exceptions étant les listings de propriétés, chacun étant représenté par un fichier spécifique.

3.2 Correspondances mathématiques de B à Coq

3.2.1 Notations Unicode

B dispose d'une notation ASCII des symboles qui avait été proposée à l'époque de sa conception. À ce moment Unicode n'était pas encore répandu comme il l'est maintenant. Cependant, il existe une évolution de B appelée event-B [Métayer et al., 2005] (ou «B événementiel») qui décrit une syntaxe Unicode pour B. Le point nous intéressant ici étant que si la syntaxe et la grammaire ont évolué, ce n'est pas le cas des fondements mathématiques ! Event-B utilise en effet les mêmes fondements mathématiques que B, à quelques rares constructions mathématiques supplémentaires près.

Or, il est possible en Coq de définir des notations pour les définitions introduites, ces notations pouvant être faites en Unicode. Nous avons donc choisi de définir un *Scope* de notation Event-B dans BiCoax qui reprend les priorités et la syntaxe décrites pour Event-B. L'intérêt immédiat est une meilleure lisibilité des formules lourdes en symboles. Par exemple, la notation suivante pour l'appartenance à un ensemble :

Notation "$x \in y$" := (*In y x*) (*at level* 11, *no associativity*) : *eB_scope*.

3.2.2 Constructions réutilisées

Logique classique et paires Puisque B est défini sur la logique classique, le fichier correspondant au chapitre 1 réutilise le module *Classical* de la bibliothèque standard de Coq. Globalement, la mise en œuvre BiCoax du chapitre 1 n'introduit aucune définition et se borne à la vérification des propriétés et théorèmes du chapitre correspondant. Il en va de même pour les paires ordonnées, qui reprennent la construction de couples de Coq.

Constructions ensemblistes de base La bibliothèque standard de Coq propose un module *Ensembles* avec, dans le même répertoire, des modules de théorèmes construits sur ce même module. Nous avons là encore utilisé directement ces constructions, parfois en spécifiant que les types de données sur lesquels ils étaient construits étaient des arguments implicites. Ceci allége l'utilisation future de ces constructions en laissant à Coq le soin de les inférer à partir du contexte d'utilisation.

Un autre argument en faveur de cette réutilisation est la définition même de l'appartenance à un ensemble en B. Dans le BBook, l'appartenance d'une variable x à un ensemble a est assimilée à la vérification d'une propriété A. Il en va de même dans le module *Ensembles* de Coq :

Definition *In* (A :*Ensemble*) (x :U) : Prop := $A\ x$.

Relations C'est à partir de ces constructions que nous avons dérivé de la bibliothèque standard de Coq. En effet en B, une relation est aussi un ensemble de couples donc la modélisation Coq correspondante est de type $S * T \to Prop$, avec S et T les types de données des éléments du couple alors que la bibliothèque *Sets* propose plutôt des relations de la forme $S \to S \to Prop$ qui opèrent sur des données de types identiques. Dans ce cas nous perdrions l'identification des relations à des ensembles ainsi qu'une certaine généralité dans la description des relations.

Constructions inductives complexes Bien que le chapitre 3 n'en soit encore qu'à ses débuts au moment où cet article est rédigé, nous avons pu constater qu'il était possible de réutiliser quelques définitions du module *Sets* à notre profit. Ainsi, plutôt que de baser la définition des ensembles finis sur la définition B en plus petit point fixe sur une fonction d'addition à un ensemble basée sur l'union, nous avons plutôt repris la définition d'*Approximant* d'un ensemble infini. Ce choix a des conséquences pratiques intéressantes dont nous faisons état en section 4. Bien que nous n'ayons pas encore abordé la modélisation de cette section, nous pensons que nous agirons de même pour la modélisation des entiers relatifs en réutilisant les définitions du module *ZArith*. Ce ne sera en revanche probablement pas le cas pour les séquences (assimilables à des listes) car celles-ci sont aussi des fonctions injectives, donc des relations et par conséquent des ensembles. Nous pourrons cependant proposer des coercions pour permettre à l'utilisateur d'exploiter le module *Lists* de la bibliothèque standard.

3.2.3 Constructions proposées

Lorsque nous proposons une définition, qu'elle soit réutilisée ou proposée, nous devons nous efforcer de considérer sa facilité d'utilisation et sa transparence par rapport au BBook. Nous avons donc équilibré l'un et l'autre au fur et à mesure de la mise en œuvre. Prenons par exemple la définition de la fonction partielle :

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

Elle dispose de deux définitions équivalentes en B :
- `{f | f ∈ A ↔ B ∧ (f⁻¹;f) ⊆ id(B) }`
- `{f | f ∈ A ↔ B ∧ ∀(x,y,z).(x,y,z ∈ A×B×B ∧ x↦y ∈ f ∧ x↦z ∈ f ⇒ y = z) }`

Cet exemple est représentatif de nos choix :
- Nous avons à chaque fois retenu la définition basée sur des arguments minimaux en terme de complexité. Il est par exemple difficile d'extraire la propriété d'unicité de l'image de la première définition

- Nous avons ôté le typage de x, y et z. En effet, qu'ils soient des éléments de A ou de B peut être déduit de leur appartenance à f. Ce «sur-typage» est nécessaire en B mais ne l'est pas avec Coq
- Nous avons préféré currifier les arguments à *pfun_intro* plutôt que les rassembler dans une conjonction. Ce choix est guidé par une certaine habitude d'utilisation des définitions inductives avec Coq car il évite d'ajouter des étapes de décomposition supplémentaires.

En résumé, les définitions que nous proposons se veulent plus faciles à manipuler pour l'utilisateur. Le danger est alors que ces définitions ne correspondent plus à ce qu'elles sont censées représenter. Notre mise en œuvre impose donc une vérification que les définitions proposées se ramènent aux définitions de B.

3.3 Les définitions modifiées sont équivalentes

Reprenons la définition de [Rocheteau et al., 2004] du plongement superficiel de B dans PhoX pour les termes, par exemple :

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

Pour être correct, le plongement doit assurer qu'il est équivalent de parler d'un terme complexe traduit ou du même terme dont chaque membre est traduit séparément. Nous nous sommes imposés de réaliser la vérification de cette équivalence pour chaque définition que nous introduisons dans BiCoax, à la suite de celle-ci.

Lorsqu'il s'agit d'une définition prédicative, nous utilisons l'équivalence logique de Coq. En revanche lorsqu'il s'agit d'une définition ensembliste, il s'agit d'une égalité. Nous devons dans ce cas faire appel à l'axiome d'extensionnalité sur les ensembles du module *Ensembles*. Cet axiome assimile l'égalité de Coq (définie comme l'égalité de Leibniz) à l'inclusion réciproque de deux ensembles. La validité de ces vérifications d'équivalence repose donc sur la validité de cet axiome.

Citons par exemple l'équivalence entre l'union du module *Ensembles* de la bibliothèque standard et la définition de l'union selon B :

Theorem *valid_union* : ∀ (*S* : Type) (*s t u* : *Ensemble S*), *Included s u* → *Included t u* →
(*Union s t*) = (*Comprehension* (fun *a* ⇒ *In u a* ∧ (*In s a* ∨ *In t a*))).
Proof.
intros S s t u H H0.
apply Extensionality_Ensembles ; [...]
Qed.

Ce théorème est conforme à la définition du BBook, puisqu'on y retrouve des contraintes d'inclusion qui, inutiles pour nous, sont nécessaires au typage de B. Le but est traduit en deux inclusions via l'utilisation de l'axiome *Extensionality_Ensembles*, elles-mêmes facilement démontrées en extrayant les propositions plus basiques qui composent ces définitions. Toutes les vérifications d'équivalence sont faites sur un modèle similaire.

Bien que nous n'en soyons pas encore là, nous pensons cependant que la vérification d'équivalence pour les entiers naturels et relatifs de Coq par rapport à ceux de B nécessitera l'utilisation de coercions idoines. Cela aura pour conséquence directe que les vérifications seront un peu moins visuellement «directes» que le théorème indiqué plus haut, par exemple.

3.4 L'axiome du choix ne change pas d'un iota

L'introduction de certains axiomes connus en Coq induit certains changements sur son comportement. Ainsi, l'axiome du tiers exclu permet de démontrer que deux preuves d'une même propriété sont égales. D'une manière similaire, l'introduction de l'axiome de description indéfinie alliée à l'imprédicativité de la sorte *Set* rendrait Coq incohérent (comme le rappelle P. Castéran dans [Castéran, 2007, introduction], par exemple).

Il se trouve que dans le cadre de notre modélisation, nous avons besoin de cet axiome, défini dans le module *ClassicalEpsilon* de Coq, pour définir l'axiome du choix de B :

Definition *epsilon* (*A* : Type) (*i* :*inhabited A*) (*P* : *A*→Prop) : *A*
 := *proj1_sig* (*classical_indefinite_description P i*).
Definition *Bchoice* := *epsilon*.

Sans entrer dans les détails, cet opérateur extrait un témoin d'une construction impliquant une preuve d'habitation de la propriété *P*. Il existe un autre opérateur dual qui extrait cette preuve.

Au vu des conséquences de l'introduction de cet axiome dans BiCoax, il devient alors *crucial* de ne pas activer l'imprédicativité de *Set* pour éviter de rendre Coq, et donc BiCoax, incohérent.

Notons également que sur la base de cet opérateur, nous avons introduit l'opérateur *iota* similaire à l'epsilon d'Hilbert, avec pour condition que la propriété identifie un élément unique. Nous avons pour ce faire repris la définition de P. Castéran [Castéran, 2007], qui l'utilise pour définir une théorie des fonctions partielles. Nous étions tentés de reprendre ce travail en B mais il s'avère que les fonctions partielles de B sont encore plus partielles qu'il n'y paraît, comme nous allons le voir en section 3.5. Cet opérateur *iota* est en effet ce dont nous avons besoin pour déterminer l'élément résultant de l'application d'une fonction à une variable.

3.5 Le cas particulier de l'application de fonction

L'opérateur de notre plongement superficiel qui s'éloigne le plus de sa définition B est l'application de fonction :
Definition app (U V :Type) (A : $Ensemble\ U$) (B : $Ensemble\ V$)(f : $Ensemble$ ($U{\times}V$)) (x :U)
  ($applicability$: In ($partial\_function\ A\ B$) $f \wedge In$ ($domain\ f$) x) :=
  $iota\ V$
    ($codomain\_unique\_inhabitation\ U\ V\ A\ B\ f\ x\ applicability$)
    (fun y :$V \Rightarrow f$ (x,y)).

La définition B de l'application impose comme condition que f soit au moins une fonction partielle et que l'élément auquel on l'applique appartienne à son domaine. Nous faisons donc de l'existence d'un tel élément un paramètre de l'application de fonction. De cet élément nous extrayons l'existence d'un élément unique habitant l'image de x par f qui est ensuite passée à l'opérateur de choix.

Pourquoi ne pas nous être contentés de la preuve d'habitation, plus simplement ? Il s'avère en fait que les obligations de preuve, lorsqu'elles impliquent des fonctions, contiennent bien plus rarement des éléments habitant l'image de l'application que des hypothèses d'appartenance au domaine et de partialité de fonctions. Ce choix sur la forme du «paramètre d'habitation» est donc un choix pragmatique en regard de l'objectif que nous nous sommes donnés.

En dehors de ces considérations sur le choix de la forme du paramètre d'habitation, il est à remarquer que ce paramètre est *nécessaire*. Ainsi, tout usage dans une formule d'une application de fonction devra être soumise à une contrainte d'existence de cette habitation. C'est la raison pour laquelle le fichier de propriétés d'égalités sur les applications de fonctions, *Equalities_evaluation* est si riche en quantifications existentielles. Pour certains opérateurs de base de B définis comme des fonctions, il est cependant possible de reconstruire ce paramètre, ceci éliminant alors le besoin d'une quantification existentielle.

Au lieu d'utiliser une quantification existentielle, nous nous étions posés la question d'utiliser une définition inductive paramétrée par la propriété à vérifier par l'application :
Inductive $app\_property$ (U V :Type) (A : $Ensemble\ U$) (B : $Ensemble\ V$)(f : $Ensemble$ ($U{\times}V$)) (x :U) (P : $V \rightarrow$ Prop) : Prop :=
  app_intro :
  $\forall$ ($applicability$: In ($partial\_function\ A\ B$) $f \wedge In$ ($domain\ f$) x),
  (P ($@app\ U\ V\ A\ B\ f\ x\ applicability$)) $\rightarrow$
  $app\_property\ U\ V\ A\ B\ f\ x\ P$.

Nous avons cependant préféré en rester à la quantification existentielle car :
- Cette définition inductive alourdirait le travail de traduction des obligations de preuves vers BiCoax
- Dans le cas de composition de fonctions, l'imbrication des existentielles est un peu plus intuitive qu'avec la syntaxe ci-dessus. Un exemple d'une telle imbrication se trouve dans le module *Equalities_evaluation* pour le théorème *Equality_laws_funct_02*

Enfin, nous avions mentionné en section 3.4 précédente le fait que nous ne pouvions pas réutiliser simplement les travaux de Pierre Castéran sur l'utilisation de fonctions partielles. En effet, les fonctions dont il est question sont partielles sur le type de données Coq ($A \rightarrow Prop$). Les fonctions partielles de B sont partielles sur leur domaine, donc un ensemble ($A \rightarrow (A \rightarrow Prop)$). Il ne serait donc pas possible pour nous de réutiliser les définitions de Pierre Castéran sans user de circonvolutions qui nuiraient à notre objectif de clarté/traçabilité.

Maintenant que les points majeurs de notre modélisation ont été abordés, nous en déduisons quelques retours sur expérience en section 4.

4 Retours de la mise en œuvre sur la théorie

4.1 Complétude du travail

En l'état actuel du développement, et modulo le besoin d'un deuxième passage pour vérifier les oublis, les premiers chapitres du BBook sont dans les états suivants :

Chapitre 1 Tous les théorèmes logiques sont (re)vérifiés. Nous avons fait l'impasse sur les théorèmes de substitutions, ceux-ci correspondant à l'usage de la β-réduction de Coq. Notons qu'il est intéressant de découvrir quels théorèmes nécessitent l'appel à des lemmes de logique classique.

Chapitre 2 Tout ce chapitre a été implementé, toutes les propriétés vérifiées.

Chapitre 3 Toutes les sections jusqu'à celle traitant des ensembles infinis incluse ont été implémentées, sans la preuve de (la moitié de la) définition de l'infinité selon Dedekind. Nous avons également proposé comme définition des entiers relatifs celle de *ZArith* sans la vérifier, à des fins de test (voir section 5).

Le tout porte le nombre de théorèmes à 726 pour le moment, pour à peu près un homme-mois de ressources.

Nous constatons donc que le BBook constitue une référence de qualité en ce qui concerne les concepts mathématiques de base.

En effet nous n'avons trouvé que trois erreurs, celles-ci portant sur les listings de propriétés dont il est peu probable que toutes aient été vérifiées aussi scrupuleusement qu'avec l'aide d'un assistant de preuve :

- Deux portent sur des lois d'appartenance (module *Membership_laws*, numérotées 33 et 34). Un contre-exemple pour l'une d'entre elles est fournie en commentaire,
- La troisième porte sur une égalité en lien avec l'image par une relation (module *Equalities_image*). L'erreur portait sur le typage d'une des hypothèses, nous pensons qu'il s'agissait plus probablement d'une coquille.

4.2 Remarques sur l'utilisabilité

Le fait de choisir des constructions inductives pour nos définitions a deux conséquences : une certaine uniformité des preuves et une meilleure efficactité.

Homogénéité des preuves Beaucoup des preuves simples se ressemblent. Comme il s'agit souvent d'extraire les arguments qui composent cette définition et de s'efforcer de les retrouver dans les hypothèses, les preuves sont souvent des combinaisons de *induction* et *constructor*, que l'on peut voir comme des tactiques d'élimination et d'introduction, respectivement. Cette ressemblance entre les preuves est en outre d'assez bon augure pour une future implantation de tactiques.

Supériorité des constructions inductives L'utilisation des constructions inductives de Coq est très supérieure à l'utilisation du principe d'induction proposé par B retranscrit tel quel dans BiCoax. La comparaison entre les deux cas de la preuve d'équivalence entre les ensembles finis selon B et les *Approximant* de Coq est plutôt éloquente à ce sujet (cf module *Bfinite_subsets*), si le nombre de commandes de preuve est une indication.

Les seules constructions qui s'utilisent plus difficilement sont l'image par une relation ainsi que l'application de fonction, dont nous relatons les particularités plus loin. L'ensemble vide pourrait également prétendre à ce titre, mais ses particularités sont aussi sémantiques, comme précisé en section 4.3.1.

L'image d'une relation Il s'agit d'une des rares définitions qui nécessite de fournir un élément d'où l'image est projetée lors des démonstrations. C'est la raison pour laquelle, plutôt qu'utiliser des tactiques comme *constructor* avec cette définition, nous en revenons plus souvent à l'utilisation de son constructeur (*image_intro*) directement. Sans être particulièrement encombrante, cette particularité brise un peu l'homogénéité des preuves.

L'application de fonction Nous craignions que la forme particulière de cette définition nous pose des problèmes, en particulier lors des preuves du module *Equalities_evaluation*. Au final, si les formules sont plutôt verbeuses, elles restent aussi simples à démontrer que d'autres.

Nous nous sommes aidés en cela de lemmes supplémentaires, dont en particulier un lemme démontrant l'égalité de deux images quelle que soit la façon dont ont été prouvées la partialité de la fonction et l'appartenance de l'élément projeté au domaine de la fonction. Ce lemme évoque un lemme similaire dans l'idée, le lemme *epsilon_inh_irrelevance* du module *ClassicalEpsilon*.

4.3 Le typage de B, l'inférence de types de Coq

Notre mise en œuvre a également permis de faire remonter à la surface quelques particularités du typage de B. Dans quelques cas particuliers nous avons également poussé l'inférence de types de Coq dans ses retranchements.

4.3.1 Ce ne sont pas les ensembles vides que vous cherchez

Il n'y a pas un seul ensemble vide mais plusieurs, comme indiqué dans le BBook[Abrial, 1996, section 2.3.3]. En effet, de par la définition des ensembles selon le module *Ensembles*, tous les ensembles sont basés sur un type donné. Ainsi l'ensemble vide des entiers n'est pas l'ensemble vide des booléens ni l'ensemble vide des couples d'entiers, par exemple. Si cette particularité est trop rapidement oubliée lors de l'implémentation d'un prouveur, elle peut facilement déboucher à une incohérence du prouveur, par exemple.

Nous avons tout de même gardé l'idée d'une notation unifiée ($\emptyset$)pour les ensembles vides, en explicitant à Coq d'inférer lui-même le type de l'ensemble vide manipulé :

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

En effet, comme nous l'avions mentionné plus tôt, les formules de B sont souvent «sur-typées» par rapport aux besoins réels des algorithmes usuels d'inférence de types. Dans la pratique, les formules utilisant cette notation nous ont jusqu'à présent donné raison (cf les modules *Equalities_**).

De plus, le fait d'utiliser Coq nous donne un point de vue d'ordre supérieur sur le formalisme. Dans la même section citée plus haut, J.-R. Abrial indique que le prédicat suivant ne passe pas la vérification de type : $\emptyset \in \{\emptyset, \{\emptyset\}\}$

La raison invoquée est tout à fait légitime dans un contexte de premier ordre, puisque le typeur est incapable de trouver un type de données auquel relier ces ensembles vides. Dans notre modélisation en Coq, il suffit pour prouver un tel prédicat de paramétrer le troisième ensemble vide par un type quelconque, donc un type paramètre du théorème décrivant cette formule, par exemple.

4.3.2 Une seconde vérification de type

À la lumière de la section 4.3.1, il est donc utile et souhaitable d'avoir un écho à ces particularités du typage de B. En effet, puisque que le type est assimilé à des prédicats (ensemblistes), la réciproque doit se retrouver lors de la génération d'OPs. En effet, l'invariant d'une machine B contient *a minima* le «typage» des variables qui y sont déclarées.

À ce titre, la vérification d'obligation de preuves pour cette partie de l'invariant, tant pour l'initialisation des variables que pour leur utilisation dans les opérations, est également une vérification de types. BiCoax permet donc d'obtenir, ne serait-ce qu'indirectement, des éclaircissements sur le typage orthodoxe de B.

4.3.3 Trop de typage alourdit les théorèmes

Pour beaucoup de définitions, nous avons ôté le «typage» supplémentaire des membres de ces définitions. Cela nous permettait d'avoir des définitions plus légères, restant toutefois équivalentes aux définitions originelles sous les conditions de définition données.

Une conséquence de cela est que de nombreux théorèmes deviennent démontrables sans avoir à ajouter de contraintes supplémentaires de typage. Il suffit de rejouer quelques preuves des modules *Equalities_** pour en avoir le cœur net : elles n'utilisent ces contraintes à aucun moment, ou peu s'en faut.

Pour que tous ces théorèmes soient (plus) utilisables dans un contexte de tactiques automatiques, il faudra donc éliminer toutes ces contraintes additionnelles. Ces théorèmes ont d'ailleurs déjà rempli leur rôle initial, qui était de vérifier leur validité par rapport au BBook.

Nous mentionnerons également, plutôt à titre de curiosité, les difficultés pour Coq à faire une inférence de types dans un cas particulier lors de nos expérimentations. En effet, l'assistant de preuves n'a pu effectuer cette inférence pour l'application d'un lemme d'égalité dans le cadre de l'application de fonctions, lors de la preuve du théorème de Knaster-Tarski pour la définition de plus grand point fixe de B, dans le module *Bfixpoints*. À la décharge de Coq, il s'agissait tout de même d'une inférence sur une construction non-triviale avec types dépendants.

5 Premiers résultats

L'évaluation de BiCoax comme outil de preuve pour des modèles B, au stade actuel du développement, est réalisée par un script qui applique automatiquement une transformation générique des obligations de preuves B en fichiers Coq. Chacun de ces fichiers comporte l'importation de la bibliothèque *Blib* élaborée dans BiCoax, la transcription dans la syntaxe Coq du but à prouver et l'appel à un embryon de tactique automatique (*firstorder* dans le cadre de cette expérimentation).

Cette transformation est simplement l'adaptation d'une démarche générique de conversion d'un format XML (les obligations de preuves B) en un format cible (les énoncés Coq) par une transformation XSLT spécifiée par une feuille de style (`bgop2bicoax`).

Cette feuille de style est l'adaptation de celle précédemment élaborée pour B/PhoX, avec pour cible le prouveur PhoX. Cette adaptation ne saurait être directe car PhoX et Coq ne partagent évidemment pas les mêmes syntaxes.

À l'heure actuelle, l'application du script de test a pour conséquence essentielle d'induire des corrections de la phase de transformation XML/XSLT et de la feuille de style. Les premiers essais n'ont pour l'instant pas permis de lever des incohérences dans les bibliothèques utilisées.

Expérimentation Les premiers tests consistent à «passer» BiCoax sur un développement B relativement simple et publiquement disponible, le «boiler» (la chaudière). Appliqué à ce modèle, le générateur d'obligations de preuve de BRILLANT génère 2335 preuves. Cette génération est aussi fidèle que possible à celle de l'Atelier B afin de faciliter les comparaisons ultérieures.

Si les résultats statistiques sont actuellement trop jeunes pour être pertinents, voici quelques observations expérimentales :

– les preuves qui réussissent correspondent à du typage. Comme indiqué précédemment, le typage étant parfois sur-spécifié, ces preuves ne sont pas difficiles. Elles sont réalisées en un temps moyen voisin de 45s. Ce temps est élevé, mais est aussi à ramener au fait que la tactique *firstorder* de Coq est encore expérimentale.
– un grand nombre de preuves échouent actuellement en raison de bugs dans la transformation XSLT ; ces bugs étant eux-même induits par des difficultés connues de la syntaxe B (notation unique pour les produits entier et cartésien, ambiguïté de certaines notations, ...). Les palliatifs à ces problèmes n'étant pas encore intégrés dans bicoax, une amélioration notable des résultats à (très) court terme est prévue.

La typologie des échecs de preuve rencontrés est la suivante :

	Projet 1	Projet 2
Nbr OP	2335	2563
Nbr BPhoX	1871	43
Nbr BiCoax	237	52
Erreur T1	1446	24
Erreur T2	310	2450
Erreur T3	175	8
Erreur T4	66	15
Erreur T5	25	1
Erreur T6	20	-
Erreur T7	13	11

TAB. 2 – Résultats de preuve

– 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`

La tableau 2 donne la répartition (à quelques unités près) des erreurs actuelles lors de l'application de bicoax. Le projet P1 est un projet de type cas d'étude, la structure du modèle et les expressions logiques impliquées sont relativement simples. Le projet P2 est en revanche une version d'étude préalable à un développement industriel. La structure et surtout les expressions sont plus complexes. Les obligations de preuves de P2 sont donc combinatoirement plus «verbeuses» que celles de P1, ce qui peut expliquer le nombre important d'erreurs T2. Les autres erreurs pertinentes nous indiquent des abus dans l'utilisation des arguments implicites aux constructions B dans la feuille de style, en particulier pour l'application de fonction (T1). Les erreurs T4 sont probablement liées à l'ambiguïté de syntaxe pour les paires ordonnées en B et les erreurs T6 à l'ambiguïté entre certains opérateurs B (multiplication et produit cartésien, par exemple).

6 Conclusion

Nous avons proposé un plongement léger des fondations mathématiques de B en Coq, en vue de fournir un prouveur pour B basé sur un assistant de preuves générique et en vue de valider par l'expérimentation les définitions et théorèmes du BBook. Nous avons réutilisé le plus possible de parties de la bibliothèque standard de Coq et choisi des constructions inductives plus simples à manipuler que les définitions usuelles B, lorsque cela était pertinent. Nous nous sommes également efforcés à chaque fois de vérifier l'équivalence entre la définition B et la définition que nous avions préféré retenir.

Le résultat, «BiCoax», est à l'heure actuelle un ensemble de modules couvrant complètement les deux premiers chapitres du BBook, le troisième chapitre étant en bonne voie. Le nombre de théorèmes démontrés s'élève à plus de 700. Cette mise en œuvre a également permis d'observer plus finement certaines particularités du typage de B.

Nous avons également, malgré la jeunesse de cette étude, mis en place des scripts pour évaluer la faisabilité et la pertinence de l'utilisation de BiCoax selon un mode plus automatique, d'une manière analogue à la génération et validation d'obligations de preuves avec les outils commerciaux supportant la méthode B et utilisés pour les projets industriels. Ce fonctionnement est également celui adopté par d'autres outils tels que Why [Filliâtre and Marché, 2007].

Cette expérimentation met en évidence les limites de transformations «simples» à l'aide du seul paradigme XML/XSLT. Celui-ci n'est pas nécessairement le formalisme le plus concis pour exprimer des traitements symboliques de conversion d'obligations de preuve. Ces difficultés pourraient être amoindries en anticipant les points délicats par adaptation des outils de génération des obligations de preuves. Néanmoins, pour la pérennité de cette adaptation, il serait utile d'envisager alors une véritable réflexion sur une standardisation éventuelle de ce format d'échange impliqué dans une approche multi-prouveurs.

Concernant BICOAX, les perspectives à court terme couvrent **(i)** la complétion de cette bibliothèque pour finalement couvrir les trois premiers chapitres du BBook, **(ii)** alléger les théorèmes d'hypothèses inutiles car non référencées par les preuves correspondantes et **(iii)** mettre en place des tactiques d'automatisation de preuves.

Les perspectives à plus long terme incluent **(i)** l'étude de la faisabilité d'une couverture, par BiCoax, des substitutions généralisées, ceci afin de pouvoir définir et valider la notion de *raffinement* de B et **(ii)** une étude similaire sur la *modularité* de B. Ces deux points nécessitent assurément de se placer à un ordre supérieur et un tel point de vue est peut-être plus pertinent dans le cadre d'un plongement profond. La question de la faisabilité dans le cadre d'un plongement léger reste cependant intéressante pour évaluer la facilité d'utilisation de Coq à des ordres supérieurs.

Nous pouvons ajouter en perspectives possibles l'utilisation des nombres réels, mathématiques ou en virgule flottante, comme nouveau type de données de base pour B : en effet il existe des modules pour ce faire, dans la bibliothèque standard et les contributions à Coq (`http://coq.inria.fr/contribs/Float.html`), respectivement. Il serait également intéressant d'étudier la jonction entre BiCoax et les travaux [Colin, 2006] d'extension de B à une logique temporelle appelée calcul des durées, celle-ci ayant également bénéficiée d'un plongement léger en Coq.

Références

[Abrial, 1996] Abrial, J.-R. (1996). *The B Book - Assigning Programs to Meanings*. Cambridge University Press.

[Badeau and Amelot, 2005] Badeau, F. and Amelot, A. (2005). Using B as a high level programming language in an industrial project : Roissy VAL. In Treharne, H., King, S., Henson, M. C., and Schneider, S., editors, *ZB05*, volume 3455 of *Lecture Notes in Computer Science*, pages 334–354. Springer.

[Berkani et al., 2003] Berkani, K., Dubois, C., Faivre, A., and Falampin, J. (2003). Validation des règles de base de l'Atelier B. In *AFADL'2003*, pages 121–136, IRISA Rennes – France. AFADL2003, IRISA, IRISA.

[Bodeveix and Filali, 2002] Bodeveix, J.-P. and Filali, M. (2002). Type synthesis in B and the translation of B to PVS. In Bert, D., Bowen, J. P., Henson, M. C., and Robinson, K., editors, *ZB'2002 – Formal Specification and Development in Z and B*, volume 2272 of *Lecture Notes in Computer Science (Springer-Verlag)*, pages 350–369, Grenoble, France. LSR-IMAG.

[Bodeveix et al., 2000] Bodeveix, J.-P., Filali, M., and Munõz, C. A. (2000). Formalisation de la méthode B en COQ et PVS. In *AFADL'2000*, pages 96–110, LSR/IMAG – BP 72 38402 Saint-Martin d'Heres Cedex – Grenoble – France. AFADL2000, LSR/IMAG, LSR/IMAG.

[Bowen and Gordon, 1995] Bowen, J. P. and Gordon, M. J. C. (1995). A shallow embedding of Z in HOL. In *Information and Software Technology*, pages 269–276.

[Castéran, 2007] Castéran, P. (2007). Utilisation en Coq de l'opérateur de description. In *Journées Francophones pour les Langages Applicatifs*, pages 30–44.

[Chartier, 1998] Chartier, P. (1998). Formalisation of B in Isabelle/HOL. In Bert, D., editor, *B'98 : The 2nd International B Conference*, volume 1393 of *Lecture Notes in Computer Science (Springer-Verlag)*, pages 66–82, Montpellier. B1998, LIRRM Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier, Springer-Verlag.

[Colin, 2006] Colin, S. (2006). *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.

[Colin et al., 2005] Colin, S., Petit, D., Rocheteau, J., Marcano-Kamenoff, R., Mariano, G., and Poirriez, V. (2005). BRILLANT : An open source and XML-based platform for rigourous software development. In *SEFM (Software Engineering and Formal Methods)*, Koblenz, Germany. AGKI (Artificial Intelligence Research Koblenz), IEEE Computer Society Press.

[Filliâtre and Marché, 2007] Filliâtre, J.-C. and Marché, C. (2007). The why/krakatoa/caduceus platform for deductive program verification. *Computer Aided Verification*, pages 173–177. `http://why.lri.fr`.

[Jaeger and Dubois, 2007] Jaeger, É. and Dubois, C. (2007). Why would you trust B ? In *LPAR, 14th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 288–302.

[Métayer et al., 2005] Métayer, C., Abrial, J.-R., and Voisin, L. (2005). RODIN deliverable 3.2 : Event-B language. `http://rodin-b-sharp.sourceforge.net`.

[Raffalli and Rozière,] Raffalli, C. and Rozière, P. *The PhoX Proof checker Documentation*. version 0.83.

[Rocheteau et al., 2004] Rocheteau, J., Colin, S., Mariano, G., and Poirriez, V. (2004). évaluation de l'extensibilité de phoX : B/PhoX un assistant de preuves pour B. In Ménissier-Morain, V., editor, *Journées Francophones des Langages Applicatifs (JFLA 2004)*, pages 37–54. INRIA.

[The Coq Development Team – LogiCal Project, 2006] The Coq Development Team – LogiCal Project (2006). *The Coq Proof Assistant Reference Manual – Version V8.1*. `http://coq.inria.fr/doc-eng.html`.

A Pourquoi ce titre ?

Le titre de cet article est plus pertinent qu'il n'y paraît :

- Il contient «alpha» car nous utilisons des identificateurs plutôt que des indices de De Bruijn comme le fait BiCoq
- Il contient «omega» car :
  - Il s'agit d'un hommage aux travaux de Pierre Castéran, travaux qui nous ont mis sur la voie quant à la modélisation délicate de l'application de fonctions, quand bien même nous réutilisons très (trop ?) peu ces travaux
  - Il s'agit d'une tactique spécifique de Coq qui sera très utile aux utilisateurs pour la preuve d'inéquations sur des termes en nombres entiers
- Il ne contient pas de «beta» car nous utilisons très peu les constructions fonctionnelles de Coq (`Fixpoint`).
- Il semble que parmi tous les travaux similaires pour B, c'est Coq qui prend la majorité quant aux prouveurs génériques utilisés, faisant de celui-ci un choix qui semble incontournable.

B Projets de tests

Le banc de test de BiCoax (et plus généralement de BRILLANT) est composé d'un ensemble de développement B accumulés «historiquement» dans les publications relatives à B.

Pour faire partie du banc de test BiCoax :

- un projet doit être accepté en version originale par l'Atelier B (v 4.0)
- la forme expansée par l'Atelier B est utilisée afin de contourner le problème du traitement non spécifié de la clause DEFINITIONS[2]

En cours

C La pile

La spécification

MACHINE
pile(nb_elements)
CONSTRAINTS
nb_elements : $\mathbb{N}$
$\wedge$ nb_elements $\geq$1
$\wedge$ nb_elements $\leq$10
VISIBLE_VARIABLES
la_pile, haut_de_pile
INVARIANT
la_pile : (1..nb_elements) $\rightarrow\mathbb{N}$
$\wedge$ haut_de_pile : $\mathbb{N}$
$\wedge$ haut_de_pile $\geq$0
$\wedge$ haut_de_pile $\leq$nb_elements

INITIALISATION
la_pile (1..nb_elements) $\rightarrow\mathbb{N}$
$||$ haut_de_pile := 0
OPERATIONS
drapeau $\longleftarrow$est_vide =
drapeau := bool(haut_de_pile = 0)
;
push(val) =
PRE
haut_de_pile <nb_elements
$\wedge$ val : $\mathbb{N}$
THEN
la_pile(haut_de_pile) := val
$||$ haut_de_pile := haut_de_pile + 1
END
;
pop =
PRE
haut_de_pile $\geq$1
THEN

[2]la version expansée d'un projet, générée par l'Atelier B, n'est cependant pas nécessairement acceptée par l'Atelier ! Le parenthésage systématique des variables «pointées» n'est pas accepté par l'analyseur B.

```
        haut_de_pile := haut_de_pile - 1
      END
      ;
    sp ⟵ sommet =
      PRE
        haut_de_pile ≠0
      THEN
        sp := la_pile(haut_de_pile - 1)
      END
    END
```

Une obligation de preuve BiCOAX

Le module COQ constitué par l'obligation de preuve générée pour l'«Op»ération «est_vide», numérotée «00007» Module Op__est_vide__00007

Require Import *Blib*.

Theorem *op* :
  ∀ *haut_de_pile la_pile nb_elements*, (
 (*In BN nb_elements*) →
  (*nb_elements* $\geq$ 1) →
  (*nb_elements* $\leq$ 10) →
  (*In* (*total_function* (*interval* 1 *nb_elements*) *BN*) *la_pile*) →
  (*In BN haut_de_pile*) →
  (*haut_de_pile* $\geq$ 0) →
  (*haut_de_pile* $\leq$ *nb_elements*)
 → (
 ((*haut_de_pile* = 0) → (*haut_de_pile* $\geq$ 0))))
 .
firstorder.
Qed.

Notes La machine comporte un invariant (conjonction de 4 termes), une initialisation et quatres opérations.

Bgop/BRILLANT génère 24 obligations de preuves. Pour chaque terme de l'invariant, une obligation est générée pour l'initialisation et pour chaque opération. L'opération «est vide» génère cependant 2 obligations de preuve en raison de l'évaluation booléenne qui la compose (c-a-d 2 cas considérés). Donc $4 \times (1+3+2) = 24$.

BiCOAX réalise 20 preuves (faciles) sur 24.

L'Atelier B (4.0beta) génère 9 obligations de preuves, dont deux restent à prouver.