Évaluation de l'extensibilité de PhoX B/PhoX un assistant de preuves pour B

Jérôme ROCHETEAU[1] & Samuel COLIN[1,2] & Georges MARIANO[1] & Vincent POIRRIEZ[2]

1 : INRETS-ESTAS
20, rue Élisée Reclus
59 650 Villeneuve d'Ascq, France
{jerome.rocheteau,samuel.colin,georges.mariano}@inrets.fr
2 : Université de Valenciennes du Hainaut Cambrésis
59 313 Valenciennes Cedex 9, France
{samuel.colin,vincent.poirriez}@univ-valenciennes.fr

Résumé

PhoX est un assistant à la preuve en logique d'ordre supérieur. Nous étendons PhoX par un ensemble de théories afin de l'utiliser comme outil de démonstration pour la méthode de développement formel B. Notre construisons ainsi un plongement du formalisme B dans PhoX. Nous définissons une traduction des obligations de preuves B par des énoncés de PhoX et nous montrons que toutes les preuves du formalisme B sont conservées avec PhoX.

Introduction

Nos motivations initiales sont de tester et si possible valider l'extensibilité du logiciel de démonstration PhoX et, d'autre part, de compléter une plate-forme d'outils écrits en OCaml pour l'expérimentation de la méthode B. Il s'agit de la plate-forme Brillant qui ne possède pas encore de moyen d'effectuer de preuves de programmes. L'attrait de PhoX réside dans sa « légèreté » qui n'est pas obtenue au prix d'une limitation de la logique d'ordre supérieure implémentée[1]. L'expressivité du langage et le pouvoir de démonstration sont comparables à ceux des systèmes Coq [Coq03], Hol98 [Gordon et al.93], Isabelle/HOL [Nipkow et al.02] et PVS [Shankar et al.93]. Notre projet est d'établir une théorie d'ordre supérieur qui est équivalente à la théorie de B afin de développer un assistant à la preuve interactif et automatique. Nous appelons cette théorie du nom du projet : B/PhoX. Plus précisément, nous définissons une traduction d'un langage du premier ordre dans un langage d'ordre supérieur qui vérifie qu'une formule du premier ordre est un théorème de B si et seulement si sa traduction est un théorème de B/PhoX.

Nous pourrions penser qu'une telle traduction est triviale le premier ordre étant un cas particulier de l'ordre supérieur. Mais nous omettrions alors ce qui fait la spécificité d'un langage d'ordre supérieur. Les langages d'ordre supérieur sont des langages applicatifs : la notion de fonction y est fondamentale ; et ce contrairement aux langages du premier ordre. Une autre différence concerne la formalisation des objets mathématiques. La théorie des ensembles est nécessaire pour définir des objets mathématiques dans un langage du premier ordre. Ces objets y sont construits par des *points fixes* et aucun enrichissement du langage ou aucun nouvel axiome n'est nécessaire. Nous sommes ainsi assurés que ces constructions sont correctes si la théorie des ensembles l'est. En revanche, la construction d'ordre supérieur des objets mathématiques par des *types de données* exige que le langage soit enrichi et

[1]La preuve de complétude de la logique d'ordre supérieur est donnée dans [David et al.01, théorème 6.5.7].

que la théorie soit étoffée. Nous devons par conséquent maîtriser cette double asymétrie quantitative (enrichissement du langage) et qualitative (langage applicatif) pour construire notre plongement.

Après avoir brièvement présenté la méthode B dans la section 1, nous définissons ce qu'est une traduction et nous établissons à quelles conditions elle constitue un plongement dans la section 2. Nous construisons alors un plongement des preuves de B dans celles de B/PhoX dans la section 3 après avoir présenter successivement ces deux théories. Dans la section 4, nous comparons notre étude aux travaux qui concernent B ou des méthodes formelles similaires et des environnements de preuves pour des logiques d'ordre supérieur. Enfin, nous appliquons B/PhoX à une étude de cas classique dans la section 5 ; il s'agit du modèle B dit « de la chaudière ».

1. La méthode B

La méthode B définie par J.-R. Abrial dans le B-Book [Abrial96] est un processus de construction de programmes conformes à leurs spécifications. La méthode B permet de générer un code exécutable sécurisé et prouvé à partir d'expressions mathématiques formelles d'une théorie des ensembles du premier ordre[2]. Ces expressions mathématiques sont rassemblées dans un composant B appelé *machine* B. Elles sont ensuite raffinées dans un ou plusieurs autres composants B appelés *raffinement(s)* de la machine B. Après tous les raffinements successifs, nous trouvons le dernier composant dit d'*implémentation* B. L'ensemble de ces composants forme alors un *module* B. Le code généré par la méthode B dans un langage de programmation impératif est *a priori* sécurisé et prouvé, c'est-à-dire conforme aux spécifications des composants. Les *obligations de preuve* d'un module B expriment la cohérence des machines B et des raffinements par rapport à leurs spécifications. Ces obligations de preuve sont formulées dans une logique du premier ordre. La démonstration de ces obligations de preuve se base sur la logique de Floyd-Hoare et la preuve de programme impératif auxquelles sont classiquement associées une sémantique de pré/post-conditions.

$$
\begin{array}{ll}
\text{MACHINE} & \mathcal{M} \\
\text{VARIABLES} & y \\
\text{INVARIANT} & y \in \mathbb{F}(\mathbb{N}) \\
\text{INITIALISATION} & y := \emptyset \\
\text{OPERATIONS} & \\
\multicolumn{2}{c}{\mathtt{lire}(n) = \text{PRE } n \in \mathbb{N} \text{ THEN } y := y \cup \{n\} \text{ END}} \\
\multicolumn{2}{c}{m \leftarrow \mathtt{maximum} = \text{PRE } y \neq \emptyset \text{ THEN } m := \max(y) \text{ END}} \\
\text{END} &
\end{array}
$$

FIG. 1 – Un petit exemple de machine B

Prenons l'exemple de la machine $\mathcal{M}$ de la figure 1 qui est inspiré du « petit exemple » dans [Abrial96, section 11.2]. Elle définit une variable y qui représente un ensemble fini d'entiers et dont la valeur initiale est l'*ensemble vide*. Les seules manipulations autorisées sur cet ensemble sont l'opération `lire` qui ajoute un entier à la variable de la machine et l'opération `maximum` qui retourne l'entier le plus grand contenu dans y. Pour que cette machine soit conforme à sa spécification, il faut montrer, d'une part, que l'initialisation vérifie l'invariant et, d'autre part, que toute opération préserve l'invariant. Pour l'initialisation, nous devons monter que l'ensemble vide est une partie finie des entiers naturels, formellement $\emptyset \in \mathbb{F}(\mathbb{N})$. L'obligation de preuve de l'opération `lire` est $\forall y, n\ (y \in \mathbb{F}(\mathbb{N}) \wedge n \in \mathbb{N} \Rightarrow y \cup \{n\} \in \mathbb{F}(\mathbb{N}))$ et celle de l'opération `maximum` est $\forall y\ (y \in \mathbb{F}(\mathbb{N}) \wedge y \neq \emptyset \Rightarrow y \in \mathbb{F}(\mathbb{N}))$.

L'originalité de la méthode B réside dans le processus de raffinement. La machine $\mathcal{N}$ de la figure 2 constitue un raffinement de la machine $\mathcal{M}$ précédente. Le processus de raffinement s'achève lorsque toutes les pré-conditions « PRE...THEN » ont été remplacées par « BEGIN » comme ce qui se fait

[2]Nous parlerons, par la suite, de *B* ou de la *théorie de B* pour la théorie mathématique qui fonde la méthode B.

pour l'opération `maximum`. La cohérence des raffinements est vérifiée par des obligations de preuve spécifiques. D'une part, un raffinement est cohérent si son invariant est vérifié par les initialisations de la machine et du raffinement. Par exemple, l'obligation de preuve de l'initialisation du raffinement $\mathcal{N}$ est $0 = \max(\emptyset \cup \{0\})$. D'autre part, un raffinement est cohérent si les invariants de la machine et du raffinement sont préservés par toutes les opérations. Ainsi, $\forall y, z, n\ (y \in \mathbb{F}(\mathbb{N}) \wedge z = \max(y \cup \{0\}) \wedge y \neq \emptyset \Rightarrow z = \max(y \cup \{0\}) \wedge z = \max(y))$ est l'obligation de preuve de raffinement pour l'opération `maximum` de la machine $\mathcal{N}$.

REFINEMENT	$\mathcal{N}$
REFINES	$\mathcal{M}$
VARIABLES	z
INVARIANT	$z = \max(y \cup \{0\})$
INITIALISATION	$z := 0$
OPERATIONS	
	`lire`$(n) =$ PRE $n \in \mathbb{N}$ THEN $z := \max(\{z, n\})$ END
	$m \leftarrow$ `maximum` $=$ BEGIN $m := z$ END
END	

FIG. 2 – Le raffinement $\mathcal{N}$ de la machine $\mathcal{M}$

2. Plongement du premier ordre dans l'ordre supérieur

Nous supposons dans ce qui suit qu'un langage d'ordre supérieur contient la sorte de base θ qui représente la catégorie des formules et la sorte de base τ qui représente celle des termes. Une formule est alors une expression de sorte θ et un terme est une expression de sorte τ.

Une *traduction* d'un langage du premier ordre dans un langage d'ordre supérieur associe,
- à chaque fonction f d'arité n, une expression close $f^\dagger$ de sorte $\tau_1 \to \cdots \to (\tau_n \to \tau)$;
- à chaque relation R d'arité n, une expression close $R^\dagger$ de sorte $\tau_1 \to \cdots \to (\tau_n \to \theta)$;
- à chaque connecteur C d'arité n, une expression close $C^\dagger$ de sorte $\theta_1 \to \cdots \to (\theta_n \to \theta)$;
- à chaque quantificateur Q, une expression close $Q^\dagger$ de sorte $(\tau \to \theta) \to \theta$.

Son prolongement aux termes, aux formules[3] et aux ensembles de formules est défini par :

$$\begin{array}{rclrcl}
(x)^\dagger & \equiv & x & (C\ F_1 \cdots F_{|C|})^\dagger & \equiv & C^\dagger\ F_1^\dagger \cdots F_{|C|}^\dagger \\
(f\ t_1 \cdots t_{|f|})^\dagger & \equiv & f^\dagger\ t_1^\dagger \cdots t_{|f|}^\dagger & (QxF)^\dagger & \equiv & Q^\dagger \lambda x F^\dagger \\
(R\ t_1 \cdots t_{|R|})^\dagger & \equiv & R^\dagger\ t_1^\dagger \cdots t_{|R|}^\dagger & \Gamma^\dagger & \equiv & \{F^\dagger \mid F \in \Gamma\}
\end{array}$$

Nous avons montré dans [Rocheteau et al.03] que toute traduction d'un terme du premier ordre est une expression d'ordre supérieur de sorte τ et toute traduction d'une formule du premier ordre est une expression d'ordre supérieur de sorte θ.

Pour qu'une traduction d'un langage du premier ordre dans un langage d'ordre supérieur constitue un plongement entre une logique du premier ordre et une logique d'ordre supérieur, il faut démontrer qu'il existe une preuve du séquent $\Gamma \vdash F$ si et seulement s'il existe une preuve du séquent $\Gamma^\dagger \Vdash F^\dagger$.

Un *plongement* ou un morphisme est, en effet, défini dans [David et al.01, définition 2.4.3] comme étant une fonction entre deux structures (X, R_X) et (Y, R_Y) qui vérifie que tous les éléments de X sont en relation par R_X si et seulement si leurs images sont en relation par R_Y. Notre plongement est une fonction $()^\dagger$ entre les formules de B et les formules de B/PhoX qui vérifie qu'une formule F est une conséquence d'un ensemble de formules Γ dans une logique du premier ordre, ce que nous notons

[3]Nous notons plus simplement QxF la formule d'ordre supérieur $Q\lambda xF$. L'abstraction d'une variable est effectuée par le lieur λ et non le quantificateur.

$\Gamma \vdash F$, si et seulement si la formule $F^\dagger$ est une conséquence de l'ensemble $\Gamma^\dagger$ dans une logique d'ordre supérieur, ce que nous notons $\Gamma^\dagger \Vdash F^\dagger$. C'est ce que récapitule la figure 3.

Nous précisons ici les différents usages que nous faisons de l'égalité :
- l'égalité entre deux formules F et G d'un langage du premier ordre (resp. $\mathcal{F}$ et $\mathcal{F}$ d'un langage d'ordre supérieur) est désignée par $F = G$ (resp. par $\mathcal{F} = \mathcal{G}$) ;
- l'égalité syntaxique qui définit une abréviation F' d'une formule F, qu'elle soit du premier ordre ou d'ordre supérieur, est désignée par $F' \hat{=} F$;
- l'égalité introduite par la traduction entre une formule F du premier ordre et une formule $\mathcal{F}$ d'ordre supérieur est désignée par $F \equiv \mathcal{F}$.

$$
\begin{tikzcd}
\text{B} & \Gamma \arrow[r, "\vdash"] \arrow[d, "\equiv"] & F \arrow[d, "\equiv"] \\
\text{B/PhoX} & \Gamma^\dagger \arrow[r, "\Vdash"] & F^\dagger
\end{tikzcd}
$$

FIG. 3 – Le principe du plongement de B dans PhoX

3. Plongement de B dans PhoX

Nous procédons à la construction de notre traduction en trois étapes qui correspondent à la présentation des fondements mathématiques de B dans [Abrial96] et de B$^\#$ qui est une évolution de B décrite dans [Abrial03]. Il s'agit :
1) de la définition de la logique des prédicats avec égalité et paire ordonnée ;
2) de la théorie des ensembles ;
3) de la construction des objets mathématiques.

Nous présentons au préalable la logique cible qu'est PhoX.

Nous scindons la présentation de la logique de PhoX en deux sections. La première correspond aux fondements de la logique d'ordre supérieur. La seconde concerne les extensions de PhoX aux définitions des types de données pour le formalisme B.

3.1. La logique d'ordre supérieur de PhoX

PhoX est une logique d'ordre supérieur composée, de la sorte `prop`,
- de la constante infixe $\Rightarrow$ de sorte $\mathtt{prop} \to (\mathtt{prop} \to \mathtt{prop})$,
- de la constante $\forall_\alpha$ de sorte $(\alpha \to \mathtt{prop}) \to \mathtt{prop}$ pour toute sorte α ;
- de la constante infixe $=_\alpha$ de sorte $\alpha \to (\alpha \to \mathtt{prop})$ pour toute sorte α ;
- de la constante ε_α de sorte $(\alpha \to \mathtt{prop}) \to \alpha$ pour toute sorte α.

Le système de sortes de PhoX est similaire au système de types polymorphes « à la ML ». Une seule définition de constantes ou d'abréviations concernant différentes sortes d'éléments est nécessaire.

La constante ε représente une fonction de « choix » sur un ensemble qui renvoie l'unique élément d'un ensemble. C'est le sens de la règle de description définie, une version *affaiblie* de l'axiome du choix :

$$\frac{\Gamma \Vdash F\ x \qquad \Gamma \Vdash \forall y\ (F\ y \Rightarrow x = y)}{\Gamma \Vdash \varepsilon\ (\lambda z\ F\ z) = x}\ \text{def}$$

Les preuves sont aussi obtenues par les règles d'introduction et d'élimination des constantes logiques :

$$\frac{}{\Gamma, F \Vdash F}\,\text{ax} \qquad \frac{\Gamma, F \Vdash G}{\Gamma \Vdash F \Rightarrow G}\Rightarrow_i \qquad \frac{\Gamma \Vdash F \qquad \Gamma \Vdash F \Rightarrow G}{\Gamma \Vdash G}\Rightarrow_e$$

$$\frac{\Gamma \Vdash F}{\Gamma \Vdash \forall x F}\forall_i \star \qquad \frac{\Gamma \Vdash \forall x F}{\Gamma \Vdash [t/x]F}\forall_e \qquad \frac{}{\Vdash \chi = \chi} =_i \qquad \frac{\Gamma \Vdash t = u \qquad \Gamma \Vdash [t/x]F}{\Gamma \Vdash [u/x]F} =_e$$

La condition $\star$ impose que x ne soit une variable libre d'aucune formule de Γ. Outre, ces règles usuelles d'un système de déduction naturelle, sont ajoutés l'axiome d'extensionnalité selon lequel deux fonctions sont égales si à chaque point elles le sont et la loi de Peirce pour obtenir la logique classique :

$$\frac{\Gamma \Vdash \forall x\ (R\ x = S\ x)}{\Gamma \Vdash R = S}\,\text{ext} \qquad \frac{\Gamma, P \Rightarrow Q\ \Vdash P}{\Gamma \Vdash P}\,\text{peirce}$$

Les règles ci-dessus définissent les fondements de PhoX. Le plongement d'une logique L dans PhoX consiste à coder les expressions de L par des constantes ou des abréviations en PhoX, tout en vérifiant que les propriétés de L sont conservées dans PhoX[4]. La consistance de d'une telle extension est alors garantie par la consistance des fondements de PhoX. Par exemple, nous pouvons définir les autres symboles logiques comme le vrai, le faux, la négation, la conjonction, la disjonction, l'équivalence et le quantificateur existentiel par des abréviations de formules composées uniquement du symbole d'implication ou du quantificateur universel[5] :

– le vrai	$\top$	$\hat{=}$	$\forall K\ (K \Rightarrow K)$
– le faux	$\bot$	$\hat{=}$	$\forall K\ K$
– la négation	$\neg F$	$\hat{=}$	$\forall K\ (F \Rightarrow K)$
– la conjonction	$F \wedge G$	$\hat{=}$	$\forall K\ ((F \Rightarrow (G \Rightarrow K)) \Rightarrow K)$
– la disjonction	$F \vee G$	$\hat{=}$	$\forall K\ ((F \Rightarrow K) \Rightarrow G)$
– l'équivalence	$F \Leftrightarrow G$	$\hat{=}$	$(F \Rightarrow G) \wedge (G \Rightarrow F)$
– le quantificateur existentiel	$\exists x\ F$	$\hat{=}$	$\forall K\ (\forall x\ (F \Rightarrow K) \Rightarrow K)$

Les règles d'inférence pour ces abréviations sont dérivées des règles précédentes de PhoX :

$$\frac{}{\Vdash \top}\top_i \qquad \frac{\Gamma \Vdash F \qquad \Gamma \Vdash \top}{\Gamma \Vdash F}\top_e \qquad \frac{\Gamma \Vdash \bot}{\Gamma \Vdash F}\bot_e \qquad \frac{\Gamma, F \Vdash \bot}{\Gamma \Vdash \neg F}\neg_i \qquad \frac{\Gamma \Vdash \neg F \qquad \Gamma \Vdash F}{\Gamma \Vdash \bot}\neg_e$$

$$\frac{\Gamma \Vdash F \wedge G}{\Gamma \Vdash F}\wedge_e^1 \qquad \frac{\Gamma \Vdash F \wedge G}{\Gamma \Vdash G}\wedge_e^2 \qquad \frac{\Gamma \Vdash F \qquad \Gamma \Vdash G}{\Gamma \Vdash F \wedge G}\wedge_i$$

$$\frac{\Gamma \Vdash F}{\Gamma \Vdash F \vee G}\vee_i^1 \qquad \frac{\Gamma \Vdash G}{\Gamma \Vdash F \vee G}\vee_i^2 \qquad \frac{\Gamma \Vdash F \vee G \qquad \Gamma, F \Vdash H \qquad \Gamma, G \Vdash H}{\Gamma \Vdash H}\vee_e$$

$$\frac{\Gamma \Vdash F \Rightarrow G \qquad \Gamma \Vdash G \Rightarrow F}{\Gamma \Vdash F \Leftrightarrow G}\Leftrightarrow_i \qquad \frac{\Gamma \Vdash F \Leftrightarrow G}{\Gamma \Vdash F \Rightarrow G}\Leftrightarrow_e^1 \qquad \frac{\Gamma \Vdash F \Leftrightarrow G}{\Gamma \Vdash G \Rightarrow F}\Leftrightarrow_e^2$$

$$\frac{\Gamma \Vdash F}{\Gamma \Vdash \exists x F}\exists_i \qquad \frac{\Gamma \Vdash \exists x F \qquad \Gamma, F \Vdash G}{\Gamma \Vdash G}\exists_e *$$

La condition $*$ est la même que pour la règle d'introduction du quantificateur universel.

[4]Nous pouvons enrichir PhoX avec de nouvelles règles à condition d'avoir démontré au préalable la clôture universelle de la formule $(F_0 \Rightarrow \ldots \Rightarrow F_m \Rightarrow F) \Rightarrow (G_0 \Rightarrow \ldots \Rightarrow G_n \Rightarrow G) \Rightarrow (H_0 \Rightarrow \ldots \Rightarrow H_k \Rightarrow H)$. pour obtenir la règle

$$\frac{\Gamma, F_0, \ldots, F_m \Vdash F \qquad \cdots \qquad \Gamma, G_0, \ldots, G_n \Vdash G}{\Gamma, H_0, \ldots, H_k \Vdash H}$$

[5]Notons que nous pouvons définir la relation d'égalité $t = u$ comme une abréviation de « l'égalité de Leibniz » qui est la formule $\forall X\ (X\ u \Rightarrow X\ t)$ selon laquelle deux objets sont indiscernables s'ils vérifient les mêmes propriétés.

3.2. Ensembles et types de donnée en PhoX

La construction d'ordre supérieur des objets mathématiques consiste à définir un type de donnée et à vérifier des théorèmes relatifs sur ce type de donnée. Un *type de donnée* pour la sorte α est une expression close F de sorte $\alpha \to \mathtt{prop}$ qui vérifie, d'une part, qu'il existe une expression c de sorte α telle $F\ c$ est un théorème et que la formule $F\ x$ est un théorème pour toute expression x de sorte α ; formellement, si $\Gamma \rhd x : \alpha$ alors $\Vdash F\ x$.

Nous présentons d'abord la notion d'ensemble d'ordre supérieur, puis nous construisons le type de donnée du produit cartésien parce qu'il est nécessaire aux définitions des relations et des fonctions qui viennent ensuite. Nous nous limitons enfin aux types de donnée des nombres naturels et des suites finies parce qu'ils constituent des exemples significatifs de ce genre de construction et qu'ils forment les extensions essentielles de PhoX à B.

Les ensembles Les ensembles sont de simples abréviations d'expressions de sorte $\alpha \to \mathtt{prop}$, c'est-à-dire qu'un ensemble est défini par sa fonction d'appartenance. L'appartenance d'un élément e à un ensemble S revient alors à appliquer S à e, le schéma de compréhension $\{x \mid F\}$ revient à abstraire la variable x dans la formule F, etc. Nous obtenons les définitions :

– de l'appartenance	$x \in S$	$\hat{=}$	$(S)\ x$
– du schéma de compréhension	$\{x \mid F\}$	$\hat{=}$	$\lambda x\ F$
– de l'inclusion	$A \subseteq B$	$\hat{=}$	$\forall x\ (x \in A \Rightarrow x \in B)$
– des parties d'un ensemble	$\mathcal{P}(S)$	$\hat{=}$	$\{X \mid X \subseteq S\}$
– l'ensemble vide	$\emptyset$	$\hat{=}$	$\{x \mid \bot\}$
– la réunion	$A \cup B$	$\hat{=}$	$\{x \mid A\ x \vee B\ x\}$
– l'intersection	$A \cap B$	$\hat{=}$	$\{x \mid A\ x \wedge B\ x\}$
– la différence	$A \setminus B$	$\hat{=}$	$\{x \mid A\ x \wedge \neg B\ x\}$

Un ensemble est défini par une fonction polymorphe. Cela revient à dire qu'il n'y a pas « un » ensemble vide $\emptyset$ mais autant d'ensembles vides $\emptyset_\alpha$ que de sortes α. Si A est un type de donnée pour la sorte α alors l'ensemble $\{x \mid F\}$ des éléments de sorte α qui vérifient la propriété F est équivalent à l'ensemble $\{x \in A \mid F\}$.

Le produit cartésien Nous choisissons de définir le produit cartésien dans une logique d'ordre supérieur par un type de donnée. Nous enrichissons le langage avec :
- la sorte $\alpha * \beta$ quelles que soient les sortes α et β ;
- la constante infixe , de sorte $\alpha \to (\beta \to \alpha * \beta)$;
- le produit cartésien $A \times B$ est l'expression[6] $\{x \mid \forall X(\forall a \in A\ \forall b \in B\ (a,b) \in X \Rightarrow x \in X)\}$ qui est l'ensemble des couples composés d'un élément d'un type de donnée A et d'un élément d'un type de donnée B.

Nous ajoutons l'axiome qui affirme que le couple est une fonction injective :

$$\frac{}{\Vdash \forall x,y,z,v\ ((x,y) = (z,v) \Rightarrow x = z \wedge y = v)}$$

Les règles d'introduction et d'élimination qui suivent sont dérivées des règles de base de PhoX :

$$\frac{\Gamma \Vdash A\ x \qquad \Gamma \Vdash B\ y}{\Gamma \Vdash A \times B\ (x,y)}\times_i \qquad \frac{\Gamma \Vdash A \times B\ (x,y)}{\Gamma \Vdash A\ x}\times_e^1 \qquad \frac{\Gamma \Vdash A \times B\ (x,y)}{\Gamma \Vdash B\ y}\times_e^2$$

[6]La formule $\forall x \in A\ F$ est une abréviation pour $\forall x\ (A\ x \Rightarrow F)$ et la formule $\forall x,y\ F$ désigne $\forall x \forall y\ F$.
De même, la formule $\exists x \in A\ F$ est une abréviation pour $\exists x\ (A\ x \wedge F)$ et la formule $\exists x,y\ F$ désigne $\exists x \exists y\ F$.

Les relations et les fonctions Les points de vue entre le premier ordre et l'ordre supérieur concernant les relations et les fonctions sont diamétralement opposés. La théorie des ensembles du premier ordre considère, en effet, les fonctions comme la classe de relations dont le graphe associe à toutes les premières projections qu'une seule seconde projection. En revanche, une fonction d'ordre supérieur est une expression de sorte $\alpha \to \beta$ et une relation d'ordre supérieur est une fonction de sorte $\alpha \to \texttt{prop}$. L'application d'une fonction à un élément et la construction d'une fonction par abstraction ne sont pas de opérations primitives d'un langage du premier ordre contrairement au langage d'ordre supérieur. Malgré tout, nous avons, à l'instar de J. Bowen et M. Gordon dans [Bowen et al.95], choisi de représenter les fonctions et les relations à la manière de la théorie des ensembles et de (re)définir explicitement un opérateur d'application d'une fonction à un argument. La raison est qu'il nous est nécessaire de pouvoir exprimer le concept de fonction partielle tandis que PhoX ne permet d'exprimer que des fonctions totales.

– les relations	$A \leftrightarrow B$	$\hat{=}$	$\{R \mid R \subseteq \mathcal{P}(A \times B)\}$
– le domaine d'une relation	$\text{dom } R$	$\hat{=}$	$\{x \mid \exists y\ (x,y) \in R\}$
– l'image d'une relation	$\text{ran } R$	$\hat{=}$	$\{y \mid \exists x\ (x,y) \in R\}$
– les fonctions partielles	$A \nrightarrow B$	$\hat{=}$	$\{f \mid f \in (A \leftrightarrow B) \wedge \text{dom}(f) \subseteq A\}$
– les fonctions totales	$A \to B$	$\hat{=}$	$\{f \mid f \in (A \nrightarrow B) \wedge \text{dom}(f) = A\}$
– l'application	$f \frown x$	$\hat{=}$	$\varepsilon\ \{y \mid (x,y) \in f\}$

L'application $f \frown x$ renvoie un élément de choix de l'ensemble des secondes projections des couples (x, y) de f. Si f est une fonction, cet ensemble est un singleton ; par conséquent, ce « choix » est unique.

Les nombres naturels fournissent une excellente illustration de construction d'un type de donnée. Nous enrichissons le langage et la théorie de PhoX par :

- la sorte de base `nat` qui représente l'ensemble des nombres naturels ;
- la constante 0 de sorte `nat` qui désigne l'entier zéro ;
- la constante *succ* de sorte $\texttt{nat} \to \texttt{nat}$ qui désigne la fonction $n \mapsto n + 1$;
- le type de donnée des entiers naturels qui est l'expression

$$N \mathrel{\hat{=}} \{x \mid \forall X\ (\forall y \in X\ (succ\ y) \in X \Rightarrow (0 \in X \Rightarrow x \in X))\}$$

qui exprime qu'un nombre naturel est obtenu par itération de la fonction *succ* à partir de 0.

Nous ajoutons les deux axiomes qui définissent la théorie de l'arithmétique d'ordre supérieur. Le premier qui exige que 0 ne soit le successeur d'aucun nombre. Le second impose que *succ* soit un fonction injective[7] :

$$\frac{}{\Vdash \forall x \in N\ (succ\ x \neq 0)} \qquad \frac{}{\Vdash \forall x, y \in N\ (succ\ x = succ\ y \Rightarrow x = y)}$$

Les règles qui suivent sont alors dérivées de règles de base de PhoX. Les deux premières règles correspondent aux théorèmes de totalité pour les constructeurs des nombres naturels et la dernière est la principe de récurrence sur les entiers :

$$\frac{}{\Gamma \Vdash 0 \in N} \qquad \frac{\Gamma \Vdash x \in N}{\Gamma \Vdash succ\ x \in N} \qquad \frac{\Gamma \Vdash 0 \in X \qquad \Gamma, y \in N, y \in X \Vdash succ\ y \in X}{\Gamma \Vdash \forall x \in N\ x \in X}$$

Cette arithmétique d'ordre supérieur contient l'arithmétique du premier ordre de Peano. Elle permet, en effet, de définir l'addition et la multiplication comme des abréviations d'expressions d'ordre supérieur de telle sorte que les axiomes de la théorie de Peano sont des conséquences logiques de cette théorie.

[7] Un tel enrichissment du langage et de la théorie constitue une définition inductive des entiers naturels comme l'explique C. Raffalli dans [Raffalli et al., chapitre 7].

Les suites finies sont aussi un exemple classique de construction de types de donnée. Nous ajoutons au langage et à la théorie de PhoX :

- la fonction de sorte à un argument $\mathtt{list}[\alpha]$ qui représente les suites finies d'éléments de sorte α ;
- la constante $[\,]$ de sorte $\mathtt{list}[\alpha]$ qui désigne la liste vide ;
- la constante infixe $::$ de sorte $\alpha \to (\mathtt{list}[\alpha] \to \mathtt{list}[\alpha])$ qui désigne l'insertion d'un élément de sorte α en tête d'une liste ;
- le type de donnée des listes de type A qui est l'expression

$$L_A \mathrel{\hat{=}} \{x \mid \forall X(\forall y \in A\ \forall l \in X\ (y :: l) \in X \Rightarrow ([\,] \in X \Rightarrow x \in X))\}$$

  qui affirme qu'une liste est le plus petit ensemble qui contient la liste vide et qui est clos par l'insertion d'un élément en tête de liste.

Nous ajoutons les axiomes qui posent que le symbole $[\,]$ est la liste vide :

$$\frac{}{\Vdash \forall A\ \forall x \in A\ \forall l \in L_A\ (x :: l \neq [\,])}$$

et que l'insertion d'un élément de type A en tête d'une liste est une opération injective :

$$\frac{}{\Vdash \forall A\ \forall x, x' \in A\ \forall l, l' \in L_A\ (x :: l = x' :: l' \Rightarrow x = x' \wedge l = l')}$$

Nous obtenons alors les règles d'introduction et d'élimination pour les listes qui sont dérivées des règles de bases de PhoX. Les deux premières sont des théorèmes de totalité pour les constructeurs, la troisième correspond au principe d'induction structurelle sur les listes :

$$\frac{}{\Gamma \Vdash [\,] \in L_A} \qquad \frac{\Gamma \Vdash x \in A \qquad \Gamma \Vdash l \in L_A}{\Gamma \Vdash x :: l \in L_A} \qquad \frac{\Gamma \Vdash [\,] \in X \qquad \Gamma, A\ x, L_A\ l, X\ l \Vdash x :: l \in X}{\Gamma \Vdash \forall l \in L_A\ l \in X}$$

3.3. La logique du premier ordre de B

Nous passons maintenant à la présentation de la théorie de B qui constitue le domaine de notre plongement. Cette section concerne la logique de B qui est exposée dans [Abrial96, chapitre 1]. Elle se base sur ces trois connecteurs logiques, la négation, la conjonction et l'implication. Ce sont les trois symboles primitifs de la logique de Floyd-Hoare dédiée à la preuve de programme et dont la logique de B s'inspire. À ceux-ci s'ajoutent le quantificateur universel, l'égalité et la paire ordonnée. Les formules du calcul des prédicats avec égalité et paire ordonnée sont obtenues par la grammaire :

$$t \ ::= \ x \mid (t, t)$$

$$F \ ::= \ t = t \mid \neg F \mid F \wedge F \mid F \Rightarrow F \mid \forall x F$$

Nous pouvons définir ce qu'est la disjonction, l'équivalence et la quantificateur existentiel comme des abréviations de formules du calcul des prédicats de manière analogue à ce qui se fait dans PhoX :

$$\begin{array}{rcl} F \vee G & \hat{=} & \neg F \Rightarrow G \\ F \Leftrightarrow G & \hat{=} & (F \Rightarrow G) \wedge (G \Rightarrow F) \\ \exists x\ F & \hat{=} & \neg \forall x\ \neg F \end{array}$$

Le système d'inférence de B est un système de déduction naturelle intuitionniste qui comporte des traits d'un calcul des séquents. La règle BR 4 du B-Book est, en effet, la règle de coupure. Ce n'est pas un système minimal de règles parce que les règles BR 4 et R 4 peuvent être dérivées des autres règles. Les théorèmes de la logique classique utilisent la règle R 5 parce qu'elle contient implicitement l'élimination d'une double négation. Les preuves du calcul des prédicats en B sont obtenues par les règles qui suivent :

$$\frac{}{\Gamma, F \vdash F}\ \text{BR 1} \qquad \frac{\Gamma \vdash F \qquad \Gamma, F \vdash G}{\Gamma \vdash G}\ \text{BR 4} \qquad \frac{\Gamma \vdash F \qquad \Gamma \vdash F \Rightarrow G}{\Gamma \vdash G}\ \text{MP}$$

$$\frac{\Gamma \vdash F \qquad \Gamma \vdash G}{\Gamma \vdash F \wedge G}\ \text{R 1} \qquad \frac{\Gamma \vdash F \wedge G}{\Gamma \vdash F}\ \text{R 2} \qquad \frac{\Gamma \vdash F \wedge G}{\Gamma \vdash F}\ \text{R 2'}$$

$$\frac{\Gamma, F \vdash G}{\Gamma \vdash F \Rightarrow G}\ \text{R 3} \qquad \frac{\Gamma \vdash F \Rightarrow G}{\Gamma, F \vdash G}\ \text{R 4}$$

$$\frac{\Gamma, \neg G \vdash \neg F \qquad \Gamma, \neg G \vdash F}{\Gamma \vdash G}\ \text{R 5} \qquad \frac{\Gamma, G \vdash \neg F \qquad \Gamma, G \vdash F}{\Gamma \vdash \neg G}\ \text{R 6}$$

$$\frac{\Gamma \vdash F}{\Gamma \vdash \forall x F}\ \text{R 7 *} \qquad \frac{\Gamma \vdash \forall x F}{\Gamma \vdash [x \leftarrow t]F}\ \text{R 8} \qquad \frac{\Gamma \vdash t = u \qquad \Gamma \vdash [x \leftarrow t]F}{\Gamma \vdash [x \leftarrow u]F}\ \text{R 9} \qquad \frac{}{\vdash t = t}\ \text{R 10}$$

La condition * impose que la variable x ne soit libre dans aucune formule de Γ.

Il y a des règles d'introduction et d'élimination pour chaque symbole sauf pour la paire ordonnée. C'est un symbole de fonction et non de relation. Elle ne définit donc pas une formule mais un terme. Or, les règles de séquents ne concernent que des formules. Nous définissons la traduction $F^\dagger$ (resp. $t^\dagger$) de la formule F (resp. du terme t) :

$$\begin{array}{rcl@{\qquad\qquad}rcl} (\neg F)^\dagger & \equiv & \neg F^\dagger & (\forall x\ F)^\dagger & \equiv & \forall x\ F^\dagger \\ (F \wedge G)^\dagger & \equiv & F^\dagger \wedge G^\dagger & (u = v)^\dagger & \equiv & u^\dagger = v^\dagger \\ (F \Rightarrow G)^\dagger & \equiv & F^\dagger \Rightarrow G^\dagger & (u, v)^\dagger & \equiv & (u^\dagger, v^\dagger) \end{array}$$

Il faut remarquer que nous avons attribué le même symbole aux constantes d'ordre supérieur et aux fonctions, relations, connecteurs et quantificateurs du premier ordre. Ne les confondons pas : les expressions à gauche des équations sont du premier ordre et celles à droite sont d'ordre supérieur.

3.4. La théorie des ensembles en B

Le langage de la théorie des ensembles de B est plus riche que celui de la théorie des ensembles de Zermelo. Ce dernier ne contient, en effet, qu'un unique symbole pour la relation d'appartenance. Le langage de B contient en plus des symboles pour le produit cartésien, l'ensemble des parties et le schéma de compréhension. Ces trois symboles sont des « constructeurs » de la catégorie des ensembles pour reprendre l'expression de J.-R. Abrial dans [Abrial03, section 2.1]. Les formules de la théorie des ensembles de B sont obtenues par la grammaire :

$$t \ ::= \ x \mid (t,t) \mid s$$

$$s \ ::= \ s \times s \mid \mathcal{P}(s) \mid \{x \mid F\}$$

$$F \ ::= \ t \in s \mid t = t \mid \neg F \mid F \wedge F \mid F \Rightarrow F \mid \forall x F$$

Les axiomes de la théorie des ensembles de B sont :

$$\frac{}{\vdash (x,y) \in S \times T \Leftrightarrow (x \in S \wedge y \in T)}\ \text{SET 1}$$

$$\frac{}{\vdash S \in \mathcal{P}(T) \Leftrightarrow \forall x(x \in S \Rightarrow x \in T)}\ \text{SET 2} \qquad \frac{}{\vdash t \in \{x \mid x \in S \wedge F\} \Leftrightarrow (t \in S \wedge [x \leftarrow t]F)}\ \text{SET 3}$$

$$\frac{}{\vdash \forall x(x \in S \Leftrightarrow x \in T) \Rightarrow S = T}\ \text{SET 4}$$

Nous prolongeons la traduction aux formules et aux termes de la théorie des ensembles de B par :

$$\begin{array}{rcl@{\qquad\qquad}rcl} (e \in S)^\dagger & \equiv & e^\dagger \in S^\dagger & \mathcal{P}(S)^\dagger & \equiv & \mathcal{P}(S^\dagger) \\ (S \times U)^\dagger & \equiv & S^\dagger \times U^\dagger & \{x \mid F\}^\dagger & \equiv & \{x \mid F^\dagger\} \end{array}$$

Axiome du choix et axiome de l'infini À proprement parler, ce n'est pas le langage des ensembles de B décrit dans [Abrial96, chapitre 2] mais c'est celui de $B^{\#}$ dans [Abrial03, section 2.1]. Le langage de B contient deux autres symboles, l'un pour la fonction de choix et l'axiome du choix, l'autre pour l'ensemble infini et l'axiome de l'infini. Ces symboles et ces axiomes sont absents des fondements de $B^{\#}$. Nous pouvons cependant étendre PhoX à cette théorie. D'une part, il nous suffit d'utiliser la constante ϵ pour la fonction de « choix » et d'ajouter la règle qui correspond à une version « forte » de l'axiome du choix :

$$\frac{\Gamma \Vdash \exists x\ F\ x}{\Gamma \Vdash F(\epsilon\ (\lambda y\ F\ y))}$$

D'autre part, il nous suffit de définir formellement ce qu'est un ensemble fini :

$$\text{Finite } E \mathrel{\hat{=}} \forall X\ (\forall Y \in X \forall x\ \{x\} \cup Y \in X \Rightarrow (\emptyset \in X \Rightarrow E \in X))$$

et d'affirmer qu'il existe un ensemble qui n'est pas fini. C'est la raison pour laquelle nous ne présentons que les axiomes qui concernent le produit cartésien SET 1, l'ensemble des parties SET 2 , le schéma de compréhension SET 3 auxquels s'ajoute l'axiome d'extensionnalité SET 4.

3.5. Les points fixes avec B

Nous abordons la construction des objets mathématiques en B. Nous nous limitons, comme pour les extensions de PhoX, aux cas des entiers naturels et des suites finies. La technique des points fixes dans ce cadre n'alourdit pas la théorie des ensembles mais elle oblige à manipuler des expressions complexes, ce qui constitue une « asymétrie » par rapport à la construction avec des types de donnée.

Dans ces conditions, comment pouvons-nous démontrer que notre traduction constitue toujours un plongement ? Une première solution consiste à vérifier, au cas par cas, que les théorèmes du premier ordre sont aussi des théorèmes d'ordre supérieur, comme c'est le cas avec les exemples concernant les entiers naturels et les suites finies. Dans le cas des entiers naturels, J.-R. Abrial démontre que les axiomes de la théorie de l'arithmétique du premier ordre de Peano deviennent de simples théorèmes dans le cadre de la théorie des ensembles de B [Abrial96, section 3.5.2]. Comme nous l'avons déja remarqué, la théorie d'ordre supérieur concernant les entiers naturels contient aussi les axiomes de Peano. La seconde consiste à montrer que la définition par point fixe est équivalente à la définition du type de donnée correspondant modulo la traduction.

Nous présentons brièvement la construction d'un ensemble par point fixe à la manière de ce qui est fait dans [Abrial96, chapitre 3]. Pour construire un ensemble non vide S, il suffit de trouver une opération croissante[8] *genset* : l'ensemble S est l'intersection des parties stables[9] par *genset*[10]. Pour définir l'opération *genset*, il suffit de se donner un élément c et une (ou plusieurs) fonction(s) injective(s) f (à un argument pour simplifier) : *genset* est la fonction qui associe l'ensemble $\{c\} \cup \{y \mid \exists x \in X\ y = f(x)\}$ à l'ensemble X. La traduction d'un point fixe est alors l'expression :

$$\begin{array}{rcl} S^{\dagger} & \hat{=} & (\cap\{X \mid genset\ (X) \subset X\})^{\dagger} \\ & \equiv & \lambda x\ \forall X\ ((\forall y\ (y = c^{\dagger} \vee \exists y \in X\ y = f^{\dagger}(z)) \Rightarrow Xy) \Rightarrow X\ x) \end{array}$$

En revanche, définir l'ensemble $\mathcal{S}$ par un type de donnée d'ordre supérieur consiste à introduire une sorte `set` et les symboles c et f (respectivement de sorte `set` et `set` $\rightarrow$ `set`) dans le langage. L'ensemble $\mathcal{S}$ est alors l'expression : $\lambda x\ \forall X\ (\forall y \in X\ (f\ y) \in X \Rightarrow (X\ c \Rightarrow Xx))$.

[8] Une opération ϕ est croissante, ou monotone, si $\phi(X) \subseteq \phi(Y)$ quels que soient les ensembles X et Y tels que $X \subseteq Y$.
[9] Un ensemble X est stable par une opération ϕ si $\phi(X) \subseteq X$.
[10] Il est alors le plus petit point fixe de *genset* d'après le théorème de Tarski, cf. [David et al.01, théorème 6.3.20].

Nous avons démontré dans [Rocheteau et al.03] que la traduction du point fixe est identique à la définition du type de donnée, c'est-à-dire que $S^\dagger = \mathcal{S}$, à condition que nous n'expansions pas les traductions de c et de f et que nous supposions que $c^\dagger = c$ et que $f^\dagger = f$.

Nous en arrivons au théorème principal de cette étude :

Théorème. *La traduction* $()^\dagger$ *constitue un plongement des preuves de B dans celles de PhoX.*

La démonstration de ce théorème se situe dans le rapport [Rocheteau et al.03]. L'idée consiste à associer chaque règle de B à la même » règle dans PhoX afin de prolonger la traduction $()^\dagger$ sur les séquents en posant que $(\Gamma \vdash F)^\dagger \equiv \Gamma^\dagger \Vdash F^\dagger$.

4. Comparaison avec d'autres travaux

Nous présentons cinq travaux similaires au nôtre en en restituant, à chaque fois, les objectifs, les démarches et les résultats. Nous dégageons les points de convergence ou de divergence entre notre travail et le leur. Ces points peuvent être, d'une part, le type de plongement effectué et, d'autre part, le domaine de B pris en compte par ce plongement.

Nous avons défini un plongement *superficiel* (shallow embedding) qui est généralement opposé à un plongement *profond* (deep embedding). Un plongement profond de B dans PhoX consiste à définir la syntaxe et la sémantique du formalisme B dans PhoX. Notre interprétation du formalisme B reste, quant à elle, extérieur à PhoX. Il s'agit d'une sémantique dénotationnelle.

Une autre différence concerne le domaine de B pris en compte par le plongement. Nous distinguons trois niveaux dans la méthode B :

- les fondements mathématiques [Abrial96, chapitres 1-3],
- les substitutions généralisées [Abrial96, chapitres 4-6]
- la théorie des machines abstraites [Abrial96, chapitres 4-8].

Nous nous sommes limités, au cours de cette étude, à la théorie mathématique qui fonde la méthode B. La raison est que la plate-forme Brillant dispose d'un outil de génération des obligations de preuves. Son existence nous évite de constituer des librairies spécifiques aux substitutions généralisées, à la théorie des machines abstraites et des raffinements et nous permet de traiter tous modules B.

A shallow embedding of Z in HOL [Bowen et al.95] Z est un langage de spécifications formelles dont la méthode B s'est inspiré. Comme la méthode B, Z se base sur un langage du premier ordre et la théorie des ensembles. C'est pourquoi le plongement de Z dans HOL – un assistant à la preuve d'ordre supérieur – nous concerne puisque Z et B partagent les mêmes fondements et que notre domaine de recherche se limite aux fondements. La motivation de J. Bowen et de M. Gordon autour de Z et de HOL est proche de la nôtre à l'égard de B et de PhoX : ils suggèrent comment HOL fournit un 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 [Chartier98] L'objectif de P. Chartier est de déterminer un prédicat d'ordre supérieur dans Isabelle/HOL[11] qui définit formellement la génération d'obligations de preuve d'une machine B et de démontrer sa validité. Le domaine de la méthode B qui est traité dans [Chartier98] 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 dû réaliser un plongement profond du formalisme B dans Isabelle/HOL.

Formalisation de la méthode B en Coq et PVS [Bodeveix et al.00] 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

[11] Isabelle est un méta-environnement de preuves. Son couplage avec HOL en fait un assistant à la preuve d'ordre supérieur.)

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 et al.02] 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 et al.95] 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 (type checking). Notons que ce programme (*btyper*) s'appuit sur un outil issu de la plate-forme Brillant (le *bparser*) et qu'en retour le *btyper* a été intégré à la plate-forme.

Validation des règles de base de l'Atelier B [Berkani et al.03] 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.

Synthèse Les travaux qui ont pour ambition de développer un assistant de preuve efficace pour le formalisme B ont opté pour un plongement superficiel. Ce logiciel doit être efficace parce qu'il peut être amené à traiter des milliers d'obligations de preuve pour un seul module B ! Ceux qui ont effectué le choix contraire, à savoir un plongement profond, ont pour objectif de valider des éléments de la méthode B dans un environnement qui est lui-même valide. Nous récapitulons dans ce tableau les différents points de convergence et de divergence :

	[Bowen et al.95]	[Chartier98]	[Bodeveix et al.00]	[Bodeveix et al.02]	[Berkani et al.03]	B/PhoX
type de plongement						
plongement profond		⋆	⋆		⋆	
plongement superficiel	⋆			⋆		⋆
domaine de B						
fondements mathématiques	⋆	⋆		⋆	⋆	⋆
substitutions généralisées		⋆	⋆			
machines abstraites		⋆				
objectifs						
validation de processus de la méthode B		?	⋆		⋆	
support informatique pour la méthode B	?			⋆		⋆

FIG. 4 – Synthèse des différents travaux

5. Étude de cas : la modèle B de « la chaudière »

Dans [Abrial03, section 4.1], J.-R. Abrial présente les trois « chaînes » d'outils principaux nécessaires au traitement d'un module complet de B ou de $B^{\#}$:

Type de chaîne	Fonction des outils
La chaîne supérieure (*the upper chain*)	analyses lexicale et syntaxique, synthèse des types
La chaîne intermédiaire (*the middle chain*)	génération des obligations de preuves
La chaîne de preuve (*the lower chain*)	logiciel de preuves interactives et automatiques

À cela, nous ajoutons, une « chaîne externe » destinée à générer automatiquement le code source dans différents langages de programmation « en dehors » de B. B/PhoX concerne la chaîne de preuve.

5.1. Architecture de la plate-forme Brillant

Nous présentons la plate-forme Brillant en fonction du classement ci-dessus.

la chaîne supérieure Le *btyper* (vérification de types pour les modules B) est un programme qui accepte en entrée un projet écrit en langage B et qui le transforme en un projet écrit en B-xml. Il a été écrit par Mamoun Filali et il correspond à la partie « type synthesis in B » dans [Bodeveix et al.02]. Il prolonge le *bparser* (analyses lexicale et syntaxique pour les modules B) qui appartient à la plate-forme Brillant.

la chaîne intermédiaire Le *bgop* (générateur d'obligations de preuves pour les modules B), s'appuie sur les définitions de [Abrial96]. Il génère des obligations de preuves d'un module B. Il transforme une obligation de preuves de la forme $H \Rightarrow (C_1 \wedge \ldots \wedge C_n)$ en n obligations de preuves de la forme $H \Rightarrow C_i$.

la chaîne de preuve Le programme *bphox* (prouveur interactif et automatique PhoX pour B) accepte en entrée des obligations de preuves générées par le *bgop*. Il traduit ces obligations de preuves de B vers B/PhoX et applique PhoX à ces obligations de preuves.

la chaîne externe Le *bunfold* (algorithme de dépliage des composants B) prend en entrée plusieurs composants B et renvoie une seule implémentation B. Il résulte directement des travaux de Dorian Petit dans [Petit et al.03]. Du code source est alors généré à partir du composant déplié. Il s'agit, à l'heure actuelle, de programmes en Ada/Spark ou en Eiffel.

5.2. Résultats de B/PhoX

Nous donnons les résultats les plus récents de l'application de B/PhoX aux obligations de preuves d'un module B complet à savoir l'étude de la chaudière (ou du boiler) qui est un cas d'école. Ces résultats concernent, d'une part, le taux de succès des preuves et, d'autre part, le type des échecs rencontrés. Ces résultats sont comparables aux résultats obtenus avec la version 3.5 du « prouveur » l'Atelier B en preuve automatique comme le remarque G. Mariano dans [Mariano97, section 5.3.3.3].

Nous scindons l'ensemble des preuves générées en deux catégories :

- les succès, c'est-à-dire les obligations de preuves que PhoX réussit à prouver
- et les échecs, c'est-à-dire toutes les autres.

Nous les étudions composants par composants afin d'identifier les composants « problèmatiques ».

Composants B	nombre de preuves	nombre d'échecs	taux de succès
Acq 1	480	21	95%
Acq 2	19	18	5%
...	...	...	...
Service Y 1	4	2	50%
Service Y 2	2	0	100%
Boiler	1422	368	74%

Parmi la catégorie des échecs, nous distinguons quatre causes :

Erreur de typage L'algorithme d'inférence de types dans PhoX échoue. Lorsqu'un un tel problème surgit, il provient d'une mauvaise traduction des formules du premier ordre de B dans les expressions d'ordre supérieur de PhoX. Mais nous n'avons pas ce type d'erreur actuellement.

Échec de la tactique L'algorithme de démonstration automatique échoue. Ou bien, l'obligation de preuve n'est pas un théorème (le module B sous-jacent doit être revu), ou bien, sa preuve est trop « complexe » pour PhoX.

Temps dépassé En effet, l'application de PhoX à une obligation de preuve est limitée à 3 minutes La démonstration nécessite donc plus de 3 minutes pour qu'elle se solde par un succès ou par un échec. Cependant, nous devons être vigilant à ne pas créer de cycle[12] lors de l'appel des règles par PhoX sans quoi son exécution ne s'arrêterait pas s'il n'était pas limité dans le temps.

Autres erreurs Sous la classification « autres erreurs » sont regroupés les problèmes bloquants pour PhoX dont l'erreur est en général l'utilisation d'identificateurs inconnus de PhoX. Dans le cas de la chaudière, les erreurs proviennent d'un manque de spécification des obligations de preuve dans le cadre de la modularité B.

Type d'erreur	Nombre d'erreurs	Proportion
Erreur de typage	0	0 %
Échec de la tactique	0	0 %
Temps dépassé	331	90 %
Autres erreurs	37	10 %

Conclusion

Étant donné que notre traduction constitue un plongement des preuves de B dans celles de PhoX, aucun obstacle théorique ne s'oppose au bon fonctionnement de l'outil informatique. Cela nous a permis de valider l'extensibilité, l'adaptabilité et la flexibilité de PhoX dans un contexte de preuves automatiques. Alors qu'il a été créé pour une utilisation interactive et bien que nous l'utilisons uniquement en mode automatique, nous obtenons sur le module B de la chaudière des résultats équivalents aux logiciels de démonstration automatique exclusivement dédié au formalisme B. Ces résultats sont d'autant plus prometteurs que, jusque là, il a fallu 7 hommes/mois pour constituer les librairies de B/PhoX qui comptent 1500 lignes de code et 3000 lignes de documentation.

Plusieurs perspectives s'offrent maintenant à nous. La première concerne un traitement de la modularité en B similaire à ce qu'étudie D. Petit dans [Petit et al.02] afin d'éliminer les « autres erreurs ». Une perspective de nature quantitative consiste à étendre le plongement à l'ensemble de la méthode B comme P. Chartier dans [Chartier98]. Une perspective de nature qualitative est d'utiliser B/PhoX pour valider les extensions expérimentales autour de B comme par exemple l'introduction du calcul des durées à la suite des travaux de S. Colin dans [Colin01] ou d'une logique de la partialité, suite à la thèse de L. Burdy [Burdy00].

Remerciements

Nous tenons à remercier Stéphane Leitao-Louro pour la mise en œuvre du support informatique de B/PhoX et de l'évalutation quantitative de la « chaîne de preuve » de la plate-forme Brillant qui se trouve dans [Leitao louro03]. Nous remercions également Frédéric Ruyer pour avoir conçu la première version de B/PhoX et Christophe Raffalli pour tous ses conseils sur PhoX.

[12]Par exemple, les règles R 3 et R 4 de la logique des prédicats de B, si elles appartenaient aux règles pour l'implication de PhoX, constitueraient un cycle dès lors qu'une formule de la forme $F \Rightarrow G$ se présenterait !

Références

[Abrial96] Abrial (Jean-Raymond). – *The B-Book. Assigning Programs to Meanings.* – Cambridge University Press, 1996.

[Abrial03] Abrial (Jean-Raymond). – $B^{\#}$: Toward a synthesis between Z and B. *ZB'2003 - Formal Specification and Development in Z and B*, pp. 168 – 177. – 2003.

[Berkani et al.03] Berkani (Karim), Dubois (Catherine), Faivre (Alain) et Falampin (Jérôme). – Validation des règles de base de l'Atelier B. *ZB'2002 – Formal Specification and Development in Z and B*, pp. 121–136. – 2003.

[Bodeveix et al.00] Bodeveix (Jean-Paul), Filali (Mamoun) et Munoz (César A.). – Formalisation de la méthode B en COQ et PVS. *AFADL'2000*, pp. 96–110. – 2000.

[Bodeveix et al.02] Bodeveix (Jean-Paul) et Filali (Mamoun). – Type synthesis in B and the translation of B to PVS. *ZB'2002 – Formal Specification and Development in Z and B*, pp. 350–369. – 2002.

[Bowen et al.95] Bowen (J.P.) et Gordon (M.J.C.). – A shallow embedding of Z in HOL. *Information and software technology*, pp. 269–276. – 1995.

[Burdy00] Burdy (Lilian). – *Traitement des expressions dépourvues de sens de la théorie des ensembles : Application à la méthode B.* – Thèse de doctorat, CEDRIC-CNAM, 2000.

[Chartier98] Chartier (Pierre). – Formalisation of B in Isabelle/HOL. *B'98 : The 2nd International B Conference*, pp. 66–82. – 1998.

[Colin01] Colin (Samuel). – *Méthode B et temps réel : étude de l'intégration du calcul des durées.* – Mémoire de DEA, Université Denis Diderot de Paris VII, 2001.

[Coq03] Coq (development team). – *The Coq Proof Assistant Reference Manual – Version V7.4*, février 2003.

[David et al.01] David (René), Nour (Karim) et Raffalli (Christophe). – *Introduction à la logique.* – Masson, 2001.

[Gordon et al.93] Gordon (M.J.C.) et Melham (T.F.) (édité par). – *Introduction to HOL : A theorem proving environment for high order logic.* – Cambridge University Press, 1993.

[Leitao louro03] Leitao-Louro (Stéphane). – *Évaluation du couplage B/PhoX.* – Mémoire de maîtrise, Université de Valenciennes et du Hainaut-Cambrésis, 2003.

[Mariano97] Mariano (Georges). – *Évaluation de logiciels critiques développés par la méthode B : une approche quantitative.* – Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis, décembre 1997.

[Nipkow et al.02] Nipkow (Tobias), Paulson (Lawrence C.) et Wenzel (Markus). – *Isabelle/HOL — A Proof Assistant for Higher-Order Logic.* – Springer, 2002, *LNCS*, volume 2283.

[Petit et al.02] Petit (Dorian), Mariano (Geaorge) et Poirriez (Vincent). – Vers un système de module à la Harper-Lillibridge-Leroy pour les spécifications formelles B. *JFLA : Journées Francophones des Langages Applicatifs*, pp. 85 – 100. – janvier 2002.

[Petit et al.03] Petit (Dorian), Mariano (Georges) et Poirriez (Vincent). – Génération de composants à partir de spécifications B. *Actes de AFADL : Approches Formelles dans l'Assistance au Développement de Logiciels*, pp. 103 – 119. – janvier 2003.

[Raffalli] Raffalli (Christophe). – *User's manual of the PhoX library.* version 0.83.

[Raffalli et al.] Raffalli (Christophe) et Rozière (Paul). – *The PhoX Proof checker Documentation.* version 0.83.

[Rocheteau et al.03] Rocheteau (Jérôme), Mariano (Georges) et Colin (Samuel). – *B/Phox : assistant à la preuve en logique d'ordre supérieur pour B.* – Rapport technique, INRETS-ESTAS, 2003. À paraître.

[Shankar et al.93] Shankar (N.), Owre (S.) et Rushby (J.M.). – *PVS Tutorial*, février 1993.