CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS | le cnam

École Doctorale Informatique, Télécommunication et Électronique

Cédric

THÈSE DE DOCTORAT

présentée par : **Mélanie JACQUEL**

soutenue le : **23 avril 2013**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline / Spécialité : **Informatique**

Automatisation des preuves pour la vérification des règles de l'Atelier B

THÈSE DIRIGÉE PAR

M. BERKANI Karim — *Ingénieur, SIEMENS IC-MOL*
M. DELAHAYE David — *Maître de conférences, CNAM*
Mme DUBOIS Catherine — *Professeur, ENSIIE*

RAPPORTEURS

M. KIRCHNER Claude — *Directeur de recherche, Inria*
M. MERZ Stephan — *Directeur de recherche, LORIA*

EXAMINATEURS

M. BOULANGER Jean-Louis — *Ingénieur, CERTIFER*
M. DOLIGEZ Damien — *Chargé de recherche, Inria*
Mme POTET Marie-Laure — *Professeur, ENSIMAG*

Remerciements

Je tiens tout d'abord à remercier chaleureusement mes encadrants Karim Berkani, David Delahaye et Catherine Dubois qui par leur disponibilité, leurs remarques, leurs idées m'ont poussé dans mes derniers retranchements et ont permis que ce travail soit mené à bien. Je remercie aussi tout particulièrement ma supérieure de Siemens, Catherine Larvor, qui est à l'initiative de cette thèse et l'a toujours soutenue et défendue avec ferveur. Sa confiance et son ouverture m'ont permis d'être libre dans mes travaux, ce qui est très appréciable.

Je remercie Claude Kirchner et Stephan Merz d'avoir accepté de rapporter cette thèse. Merci aussi à Jean-Louis Boulanger, Damien Doligez et Marie-Laure Potet de m'avoir fait l'honneur d'examiner ce travail et de faire partie de mon jury.

Je remercie l'équipe CPR et Deducteam qui m'ont accueillie chaleureusement dans leurs bureaux. Je remercie en particulier Gilles Dowek, Olivier Hermant et Guillaume Burel qui par leur idées ont fortement influencé cette thèse. Je remercie l'équipe de SafeRiver de m'avoir accueillie alors que ma thèse n'était pas finalisée.

J'ai la chance d'avoir été soutenue pendant ses trois années par l'ensemble de l'équipe PTS de Siemens. Je n'oublierai pas la bonne ambiance qui y régnait, les nombreux gâteaux engloutis, les débats parfois animés et souvent drôles, les trolls de certains, ... Je ne peux garder qu'un bon souvenir de mon passage à Siemens grâce à vous. Je tiens aussi à remercier Nicolas, Aurélie, Jean-Pascal, Nadia, Caroline et Karen pour nos discussions allant de la cuisine aux voyages en passant par des questions plus sérieuses sur la vie en générale. Je remercie Diane, que je côtoie depuis un bout de temps maintenant et qui m'a supportée durant toutes ces années en tant que binôme d'abord puis en tant que collègue. Merci de m'avoir soutenue pendant les moments difficiles, de m'avoir écoutée et conseillée toujours avec franchise. Merci à toute la bande de Jussieu (qui s'est bien élargie depuis) pour leurs invitations, les afterworks, les fêtes, les billards. Merci à tous mes amis qui, *via* leurs petites attentions, ont égayé ces trois années.

Merci aussi à toute ma famille pour leur soutien inébranlable. Merci à mes parents qui m'ont toujours laissé libre de mes choix tout en me soutenant. Merci à mon frère Benoît et à ma sœur Stéphanie (et leur conjoint) pour les moments que l'on a passé ensemble permettant de relâcher la pression, et à leurs enfants avec qui jouer est un véritable plaisir. Merci à ma famille lorraine qui a suivi avec intérêt l'avancée de ma thèse. Une pensée particulière pour les deux femmes Guignard qui hélas sont parties avant de me voir docteur.

Enfin un grand merci à mon *malala* Toky qui a subi pendant ces trois années mes sautes d'humeur en conservant un calme olympien. Merci pour son soutien incomparable et pour ses nombreuses aides techniques qui m'ont été d'un grand secours. Merci d'avoir été là chaque jour à mes côtés.

Résumé

Cette thèse porte sur la vérification des règles ajoutées de l'Atelier B en utilisant une plate-forme appelée BCARe qui repose sur un plongement de la théorie sous-jacente à la méthode B (théorie de B) dans l'assistant à la preuve Coq. En particulier, nous proposons trois approches pour prouver la validité d'une règle, ce qui revient à prouver une formule exprimée dans la théorie de B. Ces trois approches ont été évaluées sur les règles de la base de règles de Siemens IC-MOL. La première approche dite autarcique est développée avec le langage de tactiques de Coq $\mathcal{L}_{\text{tac}}$. Elle repose sur une première étape qui consiste à déplier tous les opérateurs ensemblistes pour obtenir une formule de la logique du premier ordre. Puis nous appliquons une procédure de décision qui met en œuvre une heuristique naïve en ce qui concerne les instanciations. La deuxième approche, dite sceptique, appelle le prouveur automatique de théorèmes Zenon après avoir effectué l'étape de normalisation précédente. Nous vérifions ensuite les preuves trouvées par Zenon dans le plongement profond de B en Coq. La troisième approche évite l'étape de normalisation précédente grâce à une extension de Zenon utilisant des règles d'inférence spécifiques à la théorie de B. Ces règles sont obtenues grâce à la technique de superdéduction. Cette dernière approche est généralisée en une extension de Zenon à toute théorie grâce à un calcul dynamique des règles de superdéduction. Ce nouvel outil, appelé Super Zenon, peut par exemple prouver des problèmes issus de la bibliothèque de problèmes TPTP.

Mots clés : Vérification, Déduction automatique, Superdéduction, Méthode B, Règles de l'Atelier B, Coq, Zenon

Abstract

The purpose of this thesis is the verification of Atelier B added rules using the framework named BCARe which relies on a deep embedding of the B theory within the logic of the Coq proof assistant. We propose especially three approaches in order to prove the validity of a rule, which amounts to prove a formula expressed in the B theory. These three approaches have been assessed on the rules coming from the rule database maintained by Siemens IC-MOL. To do so, the first approach, so-called autarkic approach, is developed thanks to the Coq tactic language, $\mathcal{L}_{\text{tac}}$. It rests upon a first step which consists in unfolding the set operators so as to obtain a first order formula. A decision procedure which implements an heuristic is applied afterwards to deal with instantiation. We propose a second approach, so-called skeptic approach, which uses the automated first order theorem prover Zenon, after the previous normalization step has been applied. Then we verify the Zenon proofs in the deep embedding of B in Coq. A third approach consists in using an extension of Zenon to the B method thanks to the superdeduction. Superdeduction allows us to add the axioms of the B theory by means of deduction rules in the proof mechanism of Zenon. This last approach is generalized in an extension of Zenon to every theory thanks to a dynamic calculus of the superdeduction rules. This new tool, named Super Zenon, is able to prove problems coming from the problem library TPTP, for example.

Keywords : Verification, Automated Theorem Proving, Superdeduction, B method, Atelier B rules, Coq, Zenon

Table des matières

Liste des tableaux

Table des figures

Introduction

De nos jours, l'informatique est partout dans notre quotidien. Nos téléphones portables et nos voitures sont dotés de logiciels dont le but est de faciliter certaines tâches. De la même manière, les métros automatiques et les centrales électriques sont désormais gérés à l'aide de logiciels, permettant d'assurer plus de disponibilité et de fiabilité. Les conséquences d'une défaillance sur notre téléphone ne sont évidemment pas les mêmes que dans un métro automatique. Par exemple, une défaillance dans un métro ne doit jamais engendrer de collision entre deux rames pour respecter la sécurité des voyageurs. Afin de répondre à ces exigences, des solutions sont proposées dès la phase de spécification des logiciels. Dans ce cadre, comment garantir que ces solutions sont correctement mises en œuvre et qu'ainsi la sécurité est réellement assurée ?

Les méthodes formelles répondent à cette problématique puisqu'elles regroupent un ensemble de méthodes et d'outils permettant d'analyser et de raisonner formellement sur un logiciel. Ces méthodes comprennent un langage formel et, un mécanisme de raisonnement formel ou des techniques de vérification comme dans le model-checking par exemple. Le langage formel est utilisé pour écrire les spécifications du logiciel dans un langage non ambigu, avec une syntaxe et une sémantique précises, exprimées à l'aide de la logique ou de théories logiques comme la théorie des ensembles, théorie des types, ... Le mécanisme de raisonnement formel permet de prouver des formules à partir d'autres formules ou d'hypothèses. Toutes ces méthodes ont pour objectif de développer le logiciel tout en gardant l'assurance de sa correction vis-à-vis des spécifications.

La méthode B[1] est une méthode formelle qui permet de spécifier, concevoir et implanter un logiciel. Des propriétés peuvent être démontrées sur la spécification à l'aide d'un système de preuve. De plus, toutes les étapes de développement doivent être prouvées

correctes vis-à-vis des spécifications formelles et des propriétés qu'elles doivent respecter. Ainsi, le logiciel obtenu respectera les propriétés définies dans les spécifications. Le formalisme sous-jacent à la méthode B combine le langage des prédicats et une théorie des ensembles typée. Dans la suite, nous appellerons théorie B les axiomes et règles d'inférence qui définissent la théorie ensembliste de B. Cette méthode a été utilisée avec succès à de nombreuses reprises par le passé. En particulier, elle a permis à MATRA Transport International (racheté depuis par Siemens) de réaliser la première ligne de métro automatique de Paris (avec le projet Meteor pour la ligne 14) en 1998. Plus récemment, Siemens a aussi automatisé la ligne de métro de New York (Canarsie line) ainsi que le métro de la ligne 1 de Paris, là aussi en utilisant la méthode B.

Plus généralement, Siemens se positionne sur le marché des transports automatisés de personnes avec le VAL (Véhicule Automatique Léger), les automatismes d'aide à la conduite et le guidage optique. Les solutions clés en main, comme le système VAL (un métro entièrement automatique sans conducteur), sont déclinées pour les navettes d'aéroport (Roissy CDG), ou en transport urbain, comme la ligne de métro en service à Rennes. Siemens propose également des automatismes d'aide à la conduite, comme Trainguard MT CBTC : solution de contrôle et commande de trains à forte capacité utilisant un système radio, qui est en service sur la ligne 14 parisienne. Pour finir, Siemens propose le dispositif Optiguide de guidage optique pour bus et trolleys, en service à Rouen et à l'étranger.

Les applications sécuritaires des automatismes pour les métros proposés par Siemens IC-MOL ont été développées en utilisant l'Atelier B qui est un outil développé par ClearSy. Cet outil accompagne le cycle de développement sous-jacent à la méthode B, c'est-à-dire de la spécification d'un logiciel à la génération de son code exécutable. Les propriétés à vérifier (les obligations de preuve) qui assurent la correction du développement sont produites automatiquement par l'outil. Il fournit des outils de preuve automatiques et un outil de preuve interactif à utiliser lorsque ces derniers échouent. Le prouveur interactif permet également à l'utilisateur de rajouter des axiomes, appelés règles ajoutées, si besoin pour avancer dans sa preuve. Ces règles doivent être prouvées correctes pour que le processus entier de preuve soit correct. Mais elles ne sont pas toutes vérifiables par l'Atelier B. Lors du développement du métro automatique de la ligne 14 de Paris, 27 800 obligations de preuve

ont été démontrées à l'aide du prouveur interactif de l'Atelier B, nécessitant l'ajout de 1 400 règles. Parmi elles, 900 ont été prouvées par un prouveur élémentaire de l'Atelier B. Les autres ont été prouvées manuellement (sans assistance d'un outil) par des experts. Un ensemble d'outils appelé BCARe, est développé par Siemens IC-MOL depuis 2008 pour faciliter le travail des experts et éviter les erreurs humaines. Le but de cette thèse est de continuer le développement de BCARe et en particulier d'automatiser la preuve de la correction de ces règles ajoutées.

Les principales contributions de cette thèse concernent essentiellement la preuve automatique de règles dans le cadre de l'outil BCARe. En particulier, notre approche repose sur l'utilisation d'outils de déduction externes. Ainsi, nous proposons d'utiliser conjointement l'outil d'aide à la preuve Coq [60] (plus précisément un encodage de B en Coq) et le prouveur du premier ordre Zenon [17] (développé à Inria par D. Doligez) pour prouver des formules issues de la théorie sous-jacente à la méthode B, en particulier celles issues des règles ajoutées. Trois approches sont développées et comparées dans cette thèse :

1. une approche autarcique dite « approche $\mathcal{L}_{\text{tac}}$ ». Nous proposons ici une tactique écrite dans le langage $\mathcal{L}_{\text{tac}}$ [32] (le langage de tactique de Coq) permettant de prouver automatiquement certaines formules du formalisme sous-jacent à B. Nous nous reposons ici uniquement sur notre encodage profond de B en Coq.
2. une approche externe qui utilise Zenon, dite « approche Zenon ». Cette approche consiste à normaliser une formule B de manière à la transformer en une formule du premier ordre ne contenant plus que le symbole ensembliste « $\in$ », tous les opérateurs ensemblistes ayant été dépliés selon leur définition et les axiomes de la théorie B. Cette formule normalisée est ensuite envoyée à Zenon qui considère alors le symbole « $\in$ » comme un symbole non interprété. Zenon fournit un script de preuve Coq qui est transformé par notre outil en une preuve B, c'est-à-dire une preuve utilisant les règles d'inférence de la théorie de B.
3. une approche externe appelée « approche Zenon étendu ». Cette approche étend Zenon avec des règles dédiées issues de la théorie des ensembles de B. Ces règles spécifiques sont obtenues en traitant les définitions des opérateurs ensemblistes comme des règles de réécriture et en mettant en œuvre les techniques de superdéduction [20].

L'approche ici ne requiert plus de normaliser la formule B qui est traitée telle quelle par Zenon étendu.

Une contribution complémentaire de cette thèse concerne la généralisation de la dernière approche à toute théorie et permet donc d'étendre dynamiquement Zenon avec des règles de déduction spécifiques issues de la théorie utilisée. L'approche généralisée, comme l'approche 3, met en œuvre les techniques de superdéduction.

Le chapitre 1 introduit la méthode B puis présente un état de l'art sur les méthodes de preuve B et sur les méthodes de preuve en général, notamment celles traitant de la théorie des ensembles ou qui intègrent des prouveurs interactifs et des prouveurs automatiques.

Le chapitre 2 présente l'outil BCARe développé chez Siemens IC-MOL pour la vérification des règles ajoutées de l'Atelier B. Tout d'abord la notion de règles et leur processus de vérification sont définis. Pour qu'une règle soit considérée comme correcte, il faut vérifier qu'elle n'occasionnera pas de problème de capture de variable lors de son application, qu'elle est bien typée et bien définie, et enfin qu'elle est valide. BCARe implante ce processus et pour prouver les obligations de preuve produites par chacune de ces étapes, il offre un environnement de preuve B qui s'appuie sur Coq appelé BCoq.

Le chapitre 3 propose une automatisation des obligations de preuves issues des étapes de vérification au sein de BCoq grâce à une approche autarcique (c'est-à-dire interne à BCoq) développé au moyen de l'utilisation du langage de tactique de Coq.

Les chapitres suivants se concentrent sur la preuve de la validité d'une règle, et donc sur l'automatisation des preuves dans la théorie de B de manière générale.

Le chapitre 4 présente l'outil de déduction automatique de théorèmes Zenon et décrit une méthode de preuve pour la théorie B utilisant Zenon, qui requiert une étape préliminaire de normalisation. Celle-ci consiste à déplier tous les opérateurs ensemblistes selon leur définition pour obtenir une formule du premier ordre. De plus, pour être sûr que la preuve trouvée par Zenon est correcte vis-à-vis de la théorie de B, elle est transformée en preuve BCoq afin d'être rejouée dans Coq, qui joue alors le rôle de vérificateur de preuve. Ce travail est publié dans les articles [42, 44].

Le chapitre 5 présente une extension du prouveur automatique Zenon à la théorie de B

développée en utilisant la superdéduction. Cette technique permet de créer des nouvelles règles de déduction pour chaque opérateur B. Ces règles ont l'avantage de raccourcir certaines étapes de preuve et donc de permettre de diminuer le temps de recherche de preuve. Ce travail est publié dans les articles [43, 33].

Le chapitre 6 présente une généralisation de l'extension de Zenon à la méthode B utilisant la superdéduction, à toute théorie. Ainsi à partir d'une théorie axiomatique, le nouvel outil (appelé Super Zenon) calcule dynamiquement les nouvelles règles de déduction et les intègre au noyau de règles de base de Zenon pour essayer de trouver une preuve.

Le chapitre 7 résume tous les résultats expérimentaux obtenus par les différentes approches proposées. Puis nous présentons une interface graphique permettant de lancer le processus de vérification implanté par BCARe, sur un ensemble de règles.

Chapitre 1

État de l'art

Cette thèse porte sur l'automatisation des preuves dans la théorie de B. Après avoir succinctement rappelé les bases de la méthode B dans la section 1.1, nous présentons les différents travaux faits sur la preuve dans la théorie de B et notamment ceux portant sur la preuve automatique qui interagissent ou non avec des assistants à la preuve (voir section 1.2).

1.1 Méthode B

La méthode B [1] est une méthode formelle qui permet de spécifier, concevoir et implanter des logiciels. L'étape initiale consiste à définir un modèle abstrait, composé d'un état, un invariant (représentant les propriétés abstraites que le modèle doit respecter) et des opérations qui modifient cet état. Le modèle est ensuite raffiné en plusieurs étapes jusqu'à l'obtention d'un modèle concret à partir duquel du code exécutable peut être généré. Une étape de raffinement consiste à se rapprocher d'une solution algorithmique (supprimer le non-déterminisme, utiliser des structures de données proches de celles que l'on trouve dans les langages de programmation ...) tout en respectant les propriétés précédemment définies. Vérifier la cohérence d'un modèle consiste à vérifier que l'invariant est préservé par les opérations. De plus, lorsqu'un modèle résulte d'un raffinement, il faut également vérifier que ce modèle préserve les propriétés établies dans le modèle précédent. Pour vérifier la cohérence de chaque modèle et des raffinements, des obligations de preuve sont générées. Celles-ci doivent être démontrées pour assurer la correction du logiciel développé.

La figure 1.1 schématise le développement de logiciel avec la méthode B.

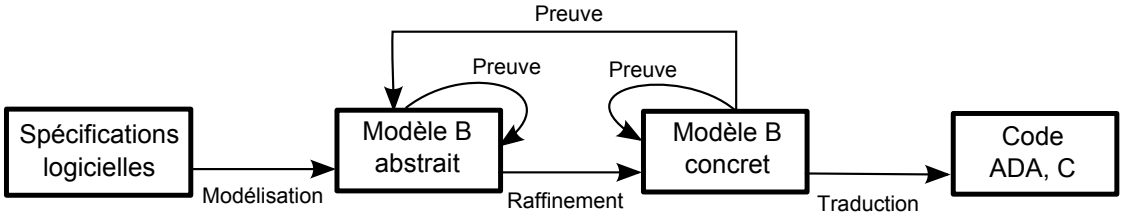

FIGURE 1.1 – Développement logiciel avec la méthode B

Chez Siemens IC-MOL, l'outil de développement relatif à la méthode B utilisé est l'Atelier B. Il couvre toutes les étapes de développement d'un logiciel, à savoir la spécification (grâce à un modèle abstrait), les étapes de raffinement pour obtenir un modèle concret ainsi que la génération de code. Toutes les obligations de preuve pour vérifier la correction sont générées automatiquement. L'Atelier B offre un prouveur automatique (voir sous-section 1.2.1) ainsi qu'un environnement de preuve interactif utilisé dans le cas où le prouveur automatique échoue.

Dans cette partie, nous présentons uniquement les notions théoriques relatives à la méthode B, qui sont nécessaires pour la compréhension des chapitres suivants. Des informations complémentaires peuvent être trouvées dans [1].

Le formalisme sous-jacent à la méthode B combine le calcul des prédicats (sur une logique classique) et une théorie ensembliste typée. La syntaxe des formules B est définie inductivement ainsi :

$$
\begin{array}{lcl}
V & := & I \mid V \mapsto V \\
E & := & V \mid [V := E]E \mid E \mapsto E \mid S \\
S & := & S \times S \mid \mathbb{P}(S) \mid \{V|P\} \mid \mathsf{BIG} \mid I \\
P & := & P \wedge P \mid P \Rightarrow P \mid \neg P \mid \forall V.P \mid [V := E]P \mid E = E \mid E \in S \mid I
\end{array}
$$

où V est l'ensemble représentant les variables, I les identificateurs, E les expressions, S les ensembles et P les prédicats.

B utilise les notations ensemblistes usuelles, ainsi $\mathbb{P}(S)$ désigne l'ensemble des parties de S. On peut remarquer que les substitutions sont explicites dans le langage et que ce ne sont pas des opérations méta. La substitution $[V := E]P$ représente le prédicat P dans

lequel les occurrences libres de la variable V sont substituées par l'expression E. Enfin BIG est un ensemble infini.

Les autres connecteurs logiques (tels que $\Leftrightarrow$, $\vee$ et $\exists$) sont définis à partir de ceux donnés ci-dessus.

Dans la suite, nous appellerons les notations introduites par la syntaxe et les connecteurs dérivés des constructions B.

1.1.1 Le système d'inférence de B

Le système d'inférence de B est proche de la déduction naturelle et traite de la logique du premier ordre avec égalité. Le jugement est de la forme $H \vdash P$ où H est l'ensemble des hypothèses.

Nous présentons ici quelques règles, avec les noms donnés dans le B-Book, qui seront utilisées dans les chapitres suivants (où H est l'ensemble des hypothèses) :

$$\frac{H \vdash P \qquad H \vdash Q}{H \vdash P \wedge Q}\ \text{REGLE 1} \qquad \frac{H \vdash P \wedge Q}{H \vdash P}\ \text{REGLE 2}$$

$$\frac{H \vdash P \wedge Q}{H \vdash Q}\ \text{REGLE 2'} \qquad \frac{H, P \vdash Q}{H \vdash P \Rightarrow Q}\ \text{REGLE 3}$$

$$\frac{x \backslash E \qquad H \vdash P}{H \vdash \forall x.P}\ \text{REGLE 7} \qquad \frac{H \vdash E = F \qquad H \vdash [x := E]P}{H \vdash [x := F]P}\ \text{REGLE 9}$$

Toutes les règles d'inférence sont présentées dans l'annexe A.1.

1.1.2 Théorie des ensembles de B

Le langage de spécification de la méthode B est une théorie des ensembles proche de la théorie des ensembles de Zermelo-Fraenkel. Cette théorie est typée afin d'éviter le paradoxe de Russell. Six schémas d'axiomes définissent les opérateurs et prédicats de base dont le produit cartésien, l'ensemble des parties, l'ensemble en compréhension et l'égalité ensembliste :

Soient deux ensembles s et t, deux expressions E et F, un prédicat P, et une variable x.

SET 1 : $(E \mapsto F) \in (s \times t) \Leftrightarrow (E \in s \wedge F \in t)$.

SET 2 : $s \in \mathbb{P}(t) \Leftrightarrow \forall x.(x \in s \Rightarrow x \in t)$, si $x \backslash (s,t)$.

SET 3 : $E \in \{x | x \in s \wedge P\} \Leftrightarrow (E \in s \wedge [x := E]P)$, si $x \backslash s$.

SET 4 : $\forall x.(x \in s \Leftrightarrow x \in t) \Rightarrow s = t$ si $x \backslash (s,t)$.

SET 5 : $\exists x \in s \Rightarrow \mathsf{choice}(s) \in s$

SET 6 : $infinite(\mathsf{BIG})$

où BIG est un ensemble constant, et $x \backslash s$ signifie que x est non libre dans s, c'est-à-dire que soit x est sous la portée d'un lieur dans s, soit x n'apparaît pas dans s. Nous appelons lieur les constructions suivantes : $\{V|P\}$,$[V := E]$, $\forall V.P$ et $\exists V.P$.

Les quatres premiers axiomes définissent respectivement le produit cartésien, l'ensemble des parties, l'ensemble en compréhension et l'égalité ensembliste. Le cinquième axiome signifie que si un ensemble a un membre alors l'opérateur choice appliqué à cet ensemble dénote un certain membre de cet ensemble. Enfin, le sixième axiome spécifie que l'ensemble constant BIG est un ensemble infini.

Le quatrième axiome peut être considéré comme une équivalence étant donnée que l'implication dans l'autre sens peut être démontrée grâce à la REGLE 9, qui est la règle de remplacement par les égaux. Dans la suite nous considérerons uniquement la version de cet axiome avec l'équivalence que nous appelons SET 4'. Les quatre premiers axiomes (dont SET 4') permettent de transformer une formule ensembliste en une formule du premier ordre (en utilisant les équivalences de gauche à droite pour les trois premiers et de droite à gauche pour le quatrième).

Les opérateurs ($\cup$, $\cap$, etc.) sont définis à partir des opérateurs de base précédents. Par exemple, l'union est définie à l'aide d'un ensemble en compréhension :

Définition 1 (Union)
Soient a une variable et u, s, t des ensembles.
Si $s \subseteq u$ et $t \subseteq u$ alors
$s \cup t \triangleq \{a | a \in u \wedge (a \in s \vee a \in t)\}$

où le symbole « $\triangleq$ » signifie que $s \cup t$ est défini par $\{a|a \in u \wedge (a \in s \vee a \in t)\}$.

Cette définition est conditionnelle car on ne peut l'appliquer que si les hypothèses de

typage $s \subseteq u$ et $t \subseteq u$ sont respectées.

Les définitions des opérateurs traités dans cette thèse sont présentées dans l'annexe B.

1.1.3 Le typage en B

Comme nous l'avons dit précédemment, la méthode B est basée sur une théorie des ensembles typée afin d'éliminer des formules incohérentes, comme le paradoxe de Russell par exemple. Ainsi, la formule $\exists x.x \in x$ est mal typée, elle est donc rejetée.

En B, les ensembles servent à contraindre les données puisqu'ils peuvent représenter des types. Par exemple, $\mathbb{N}$ est un type et des variables d'ensemble arbitraire sont aussi des types. Si T_1 et T_2 sont des types alors le produit cartésien de ces types $T_1 \times T_2$ est le type des paires $t_1 \mapsto t_2$ où t_1 est de type T_1 et t_2 est de type T_2. Si T est un type alors les sous-ensembles de T sont de type $\mathbb{P}(T)$. Ainsi, si e_1 et e_2 sont de type T, alors $\{e_1, e_2\}$ est de type $\mathbb{P}(T)$. La méthode B impose de vérifier le typage d'une formule avant d'essayer de la prouver. Pour cela, chaque variable doit être contrainte par un ensemble que l'on appelle son type support, et tous les ensembles mis en jeu doivent s'organiser de manière cohérente c'est-à-dire avoir des relations d'inclusion ou avoir des types différents.

Concrètement, le typage en B permet de vérifier qu'un prédicat P est bien typé relativement à un ensemble d'hypothèses de typage H avec le jugement $H \vdash_\tau \mathsf{check}(P)$. Les règles de ce système de types utilise un autre ensemble de règles dont le jugement est de la forme $H \vdash_\tau T_1 \equiv T_2$ où H est un ensemble d'hypothèses de typage, « $\equiv$ » signifie que le type T_1 est équivalent au type T_2, et T_1 et T_2 sont deux éléments de la catégorie syntaxique définie ci-dessous :

$$Type \quad := \quad \mathsf{type}(E) \mid \mathsf{super}(S) \mid Type \times Type \mid \mathbb{P}(Type) \mid I$$

où $type(E)$ désigne le type d'une expression et $super(S)$ le type d'un ensemble.

Dans une formule contenant des expressions ensemblistes, le type de chaque expression est soit le produit cartésien de deux $Type$, soit l'ensemble des parties d'un $Type$, soit un identificateur I.

Par exemple, si nous avons dans une formule une expression E et deux ensembles s et t tels que $E \in s$ et $s \subseteq t$, le type support intervenant dans cette formule (mais il peut y en

avoir plusieurs) est t car c'est le plus grand sur-ensemble contenant les autres expressions de la formule, à savoir E et s. Les types supports doivent être identifiés par given dans les hypothèses de typage. Pour vérifier que P est bien typé, il suffit d'appliquer ensuite les règles de typage du B-Book. Certaines de ces règles sont présentées ci-dessous. Une partie des règles de typage sont dans l'annexe A.2.

$$\frac{H \vdash_\tau \mathsf{check}(P) \qquad H \vdash_\tau \mathsf{check}(Q)}{H \vdash_\tau \mathsf{check}(P \wedge Q)}\ \text{T1} \qquad \frac{H \vdash_\tau \mathsf{check}(P) \qquad H \vdash_\tau \mathsf{check}(Q)}{H \vdash_\tau \mathsf{check}(P \Rightarrow Q)}\ \text{T2}$$

$$\frac{H \vdash_\tau \mathsf{check}(P)}{H \vdash_\tau \mathsf{check}(\neg P)}\ \text{T3} \qquad \frac{x \setminus s \qquad x \setminus H \qquad H, x \in s \vdash_\tau \mathsf{check}(P)}{H \vdash_\tau \mathsf{check}(\forall x.(x \in s \Rightarrow P))}\ \text{T4}$$

Nous pouvons remarquer que contrairement au système d'inférence qui ne traite que de la logique du premier ordre, le système de types contient des règles traitant des opérateurs ensemblistes. Néanmoins, il y a dans le B-Book de nombreuses propriétés dérivées des définitions d'opérateurs qui peuvent être considérées comme des règles dérivées.

Exemple 1 (Vérification du typage)

Nous allons vérifier que la formule suivante est bien typée :

$$\forall(a \mapsto b).((a \mapsto b) \in \mathbb{P}(s) \times \mathbb{P}(s) \Rightarrow \{x | x \in s \wedge (x \in a \wedge x \in b)\} \subseteq s)$$

avec les règles de typage de B. Le type support de cette formule est s, donc l'ensemble des hypothèses de typage initial est $\mathsf{given}(s)$*. L'arbre de vérification du typage est le suivant, dans lequel* Γ_b *désigne* $\mathsf{given}(s), a \in \mathbb{P}(s), b \in \mathbb{P}(s)$ *:*

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_b, x \in s \vdash_\tau s \equiv s}\;\mathit{T21}}{\Gamma_b, x \in s \vdash_\tau \mathsf{super}(s) \equiv s}\;\mathit{T17}}{\Gamma_b, x \in s \vdash_\tau \mathsf{type}(x) \equiv s}\;\mathit{T9}}{\Gamma_b, x \in s \vdash_\tau \mathsf{type}(x) \equiv \mathsf{super}(s)}\;\mathit{T17'}}{\Gamma_b, x \in s \vdash_\tau \mathbb{P}(\mathsf{type}(x)) \equiv \mathbb{P}(\mathsf{super}(s))}\;\mathit{T19}}{\Gamma_b, x \in s \vdash_\tau \mathbb{P}(\mathsf{type}(x)) \equiv \mathsf{super}(\mathbb{P}(s))}\;\mathit{T15'}}{\Gamma_b, x \in s \vdash_\tau \mathsf{type}(x) \equiv \mathsf{super}(a)}\;\mathit{T13'}}{\Gamma_b, x \in s \vdash_\tau \mathsf{check}(x \in a)}\;\mathit{T8}
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_b, x \in s \vdash_\tau s \equiv s}\;\mathit{T21}}{\Gamma_b, x \in s \vdash_\tau \mathsf{super}(s) \equiv s}\;\mathit{T17}}{\Gamma_b, x \in s \vdash_\tau \mathsf{type}(x) \equiv s}\;\mathit{T9}}{\Gamma_b, x \in s \vdash_\tau \mathsf{type}(x) \equiv \mathsf{super}(s)}\;\mathit{T17'}}{\Gamma_b, x \in s \vdash_\tau \mathbb{P}(\mathsf{type}(x)) \equiv \mathbb{P}(\mathsf{super}(s))}\;\mathit{T19}}{\Gamma_b, x \in s \vdash_\tau \mathbb{P}(\mathsf{type}(x)) \equiv \mathsf{super}(\mathbb{P}(s))}\;\mathit{T15'}}{\Gamma_b, x \in s \vdash_\tau \mathsf{type}(x) \equiv \mathsf{super}(b)}\;\mathit{T13'}}{\Gamma_b, x \in s \vdash_\tau \mathsf{check}(x \in b)}\;\mathit{T8}
$$

$$
\dfrac{\dfrac{\Gamma_b, x \in s \vdash_\tau \mathsf{check}(x \in a) \qquad \Gamma_b, x \in s \vdash_\tau \mathsf{check}(x \in b)}{\Gamma_b, x \in s \vdash_\tau \mathsf{check}(x \in a \wedge x \in b)}\;\mathit{T1}}{\Gamma_b \vdash_\tau \mathsf{check}(\forall x.(x \in s \Rightarrow (x \in a \wedge x \in b)))}\;\mathit{T4}
$$

$$
\dfrac{\dfrac{}{\Gamma_b \vdash_\tau s \equiv s}\;\mathit{T21}}{\Gamma_b \vdash_\tau \mathsf{super}(s) \equiv s}\;\mathit{T17}
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma_b \vdash_\tau \mathsf{check}(\forall x.(x \in s \Rightarrow (x \in a \wedge x \in b))) \qquad \Gamma_b \vdash_\tau \mathsf{super}(s) \equiv s}{\Gamma_b \vdash_\tau \mathsf{super}(\{x | x \in s \wedge (x \in a \wedge x \in b)\}) \equiv s}\;\mathit{T16}}{\Gamma_b \vdash_\tau \mathsf{super}(\{x | x \in s \wedge (x \in a \wedge x \in b)\}) \equiv \mathsf{super}(s)}\;\mathit{T17'}}{\mathsf{given}(s), a \in \mathbb{P}(s), b \in \mathbb{P}(s) \vdash_\tau \mathsf{check}(\{x | x \in s \wedge (x \in a \wedge x \in b)\} \subseteq s)}\;\mathit{T8'}}{\mathsf{given}(s), a \in \mathbb{P}(s) \vdash_\tau \mathsf{check}(\forall b.(b \in \mathbb{P}(s) \Rightarrow \{x | x \in s \wedge (x \in a \wedge x \in b)\} \subseteq s))}\;\mathit{T4}}{\mathsf{given}(s) \vdash_\tau \mathsf{check}(\forall a.(a \in \mathbb{P}(s) \Rightarrow \forall b.(b \in \mathbb{P}(s) \Rightarrow \{x | x \in s \wedge (x \in a \wedge x \in b)\} \subseteq s)))}\;\mathit{T4}}{\mathsf{given}(s) \vdash_\tau \mathsf{check}(\forall (a \mapsto b).((a \mapsto b) \in \mathbb{P}(s) \times \mathbb{P}(s) \Rightarrow \{x | x \in s \wedge (x \in a \wedge x \in b)\} \subseteq s))}\;\mathit{T5}
$$

1.2 Preuve de formules ensemblistes

Pour faire des preuves sur des formules B, il existe déjà des prouveurs automatiques dédiés (voir sous-section 1.2.1). Lorsque ces outils ne permettent pas d'effectuer automatiquement toutes les preuves B, certains outils basés sur des assistants à la preuve permettent d'effectuer des preuves B dans un cadre formel (voir sous-section 1.2.2). De manière plus large, il existe de nombreux prouveurs automatiques qui traitent de la logique du premier ordre ou de théories en général (voir sous-section 1.2.3). Ces outils peuvent être combinés à des assistants à la preuve afin d'allier le pouvoir d'expression des seconds à la recherche de preuve automatique faite par les premiers (voir sous-section 1.2.4).

1.2.1 État de l'art sur les prouveurs

Les méthodes formelles comme la méthode B s'appuient sur la preuve formelle pour assurer la correction du développement d'un logiciel. Pour cela des obligations de preuve sont produite et doivent être vérifiées. Celles-ci peuvent être démontrées automatiquement par le prouveur de l'Atelier B, par exemple.

Dans [3], les auteurs présentent les principes de ce prouveur, appelée prouveur de prédicats (pp), et intégré à l'Atelier B comme une tactique. Ce prouveur est également décrit dans [25] qui en présente une implantation dans le langage ELAN, dont le paradigme est la réécriture. La tactique pp traite des formules de la logique du premier ordre avec égalité et permet aussi de normaliser les formules ensemblistes en formule de la logique du premier ordre. Plus précisément pp met en œuvre une procédure de décision pour le calcul propositionnel proche de la méthode des tableaux sémantiques. Cette procédure décompose les formules et cherche une contradiction dans les hypothèses. Pour la logique du premier ordre, il s'agit d'une procédure de semi-décision. Une première étape consiste à décomposer les formules en formant des propositions atomiques ou des prédicats quantifiés universellement. Si aucune contradiction dans les hypothèses n'a été trouvée lors de cette étape, alors des instanciations pour les hypothèses universellement quantifiées sont déduites de l'analyse des autres hypothèses. La première étape est ensuite à nouveau appliquée et ainsi de suite. S'il y a des formules ensemblistes, une autre procédure normalise ces expressions afin

d'obtenir une formule en logique du premier ordre contenant uniquement le constructeur « $\in$ » qui ne sera pas interprété. Cela est possible puisque les constructions de la théorie des ensembles du B-Book est définie à partir de la logique du premier ordre. Ces procédures ont des limitations : les hypothèses inutiles ralentissent la recherche de preuve, les égalités ne sont pas toujours exploitées au maximum, la reconnaissance sur le fait que certaines formules nécessitent une stratégie non-triviale (telle que la preuve par cas) est limitée, ...

Afin de vérifier des propriétés sur des spécifications B, nous pouvons aussi mentionner le model-checker ProB [48]. Il traite aussi des spécifications issues de Z et TLA^+.

1.2.2 Plongements de la méthode B

Pour manipuler la théorie de la méthode B de manière formelle, il est possible de « plonger » cette théorie dans un assistant à la preuve c'est-à-dire d'encoder la théorie dans l'assistant à la preuve. Il existe deux types de plongement : le plongement superficiel et le plongement profond [19]. Dans le plongement superficiel, des constructions logiques du langage hôte peuvent être utilisées ainsi que les tactiques automatiques des assistants. Alors que dans un plongement profond, toute la logique est redéfinie afin de séparer les deux théories et les tactiques existantes ne peuvent pas être utilisées.

Il existe des plongements de B dans HOL([23]), dans PVS([14]) mais aussi dans Coq [60]. Dans cette thèse nous présentons le plongement profond BCoq qui suit fidèlement la théorie du B-Book et permet donc de prouver des formules B. Les définitions inductives de Coq nous permettent de suivre fidèlement les jugements de type et de preuve présentés dans le B-Book. Cela assure une bonne traçabilité entre le B-Book et son implantation en Coq. BiCoq est aussi un plongement profond proposé dans [45] dans lequel les indices de De Bruijn sont utilisés pour représenter les variables.

Des plongements superficiels de B dans Coq ont aussi été développés. Par exemple, BiCoax [29] est un plongement superficiel dont l'objectif est de disposer d'un outil de preuve pour B sur la base d'un assistant à la preuve ainsi que de valider et tracer les preuves théorèmes présentés dans le B-Book.

1.2.3 Prouveurs automatiques

En élargissant à la preuve automatique en général, de nombreux outils ont été développés ces dernières années. Ces outils peuvent être vus par les utilisateurs comme une boite noire à laquelle on donne en entrée une formule et qui renvoie une trace plus ou moins exploitable de la preuve (ou un contre-exemple) trouvée par l'outil.

Parmi les prouveurs automatiques, on peut différencier les solveurs SMT (Satisfiability Modulo Theories) comme Alt-Ergo [13], CVC3 [8] et Z3 [30] qui utilisent des solveurs SAT (pour les problèmes de satisfaisabilité booléenne), des prouveurs du premier ordre comme Zenon [17] et Vampire [55], dans la mesure où les premiers vérifient la satisfaisabilité d'une formule par rapport à une théorie donnée alors que les seconds recherchent une preuve directement de la formule.

Pour chaque famille de prouveurs un format d'entrée commun existe :TPTP et SMT-LIB. Concernant la trace, il n'y a actuellement pas de format de sortie standard, elle peut être :

- une information sur le fait que le prouveur a trouvé ou non une preuve (OK/KO) ;
- un contre-exemple ;
- des éléments de preuve ;
- des preuves dans un format propre à l'outil ;
- des preuves lisibles par un humain ;
- des preuves vérifiables automatiquement par un assistant à la preuve.

Dans le cadre industriel, les traces des prouveurs doivent être au moins lisibles par un expert et dans l'idéal vérifiable automatiquement par un assistant à la preuve. Le prouveur Zenon est un bon candidat puisqu'il produit des preuves vérifiables automatiquement dans Coq [17] et Isabelle [51].

Il existe des prouveurs automatiques dédiés à la théorie des ensembles. Par exemple, le système Theorema [63] est un prouveur dédié à la théorie des ensembles de Zermelo-Fraenkel. Il a des règles spécifiques permettant de traiter les constructions ensemblistes et de faciliter la recherche de preuve dans certains cas.

Muscadet [53] est un prouveur automatique de théorèmes de la logique du premier ordre

avec égalité basé sur la déduction naturelle. Bien qu'il traite toutes les théories, il se révèle approprié pour les problèmes issus de la théorie des ensembles (plus que pour les problèmes des autres catégories). Certaines de ses heuristiques permettent d'améliorer la recherche de preuve. Par exemple, à partir d'une théorie axiomatique, les axiomes sont transformés en méta-règles. Pour cela les connecteurs logiques sont décomposés. Ces règles s'appliquent sur les hypothèses et permettent d'en déduire de nouvelles. Toutes les règles ne sont pas utilisées : seules les règles contenant des opérateurs contenus dans la formule à prouver sont sélectionnées car elles sont potentiellement applicables. Enfin, les règles traitant des quantificateurs ont une faible priorité par rapport aux autres règles.

1.2.4 Interaction entre les assistants à la preuve et les prouveurs automatiques

Nous avons mentionné d'une part les plongements qui reposent sur des assistants à la preuve. D'autre part, nous avons vu qu'il y a de nombreux prouveurs automatiques qui fournissent des traces plus ou moins exploitables. Nous nous intéressons maintenant aux différentes approches qui permettent d'améliorer l'automatisation des preuves des assistants à la preuve, dont certaines font appel à des prouveurs automatiques. Nous suivons pour cela la classification donnée par Barendregt dans [7].

La première approche, appelée approche autarcique, consiste à automatiser la preuve directement dans l'assistant à la preuve sans aucun outil externe. L'approche crédule consiste à déléguer la preuve (ou une partie) à un prouveur externe sans vérifier la correction du prouveur. Le prouveur externe est alors vu comme un oracle. Enfin, l'idée de l'approche sceptique est d'utiliser un prouveur externe mais en vérifiant sa correction c'est-à-dire par exemple, la trace de ses preuves.

Un exemple de l'approche autarcique est l'approche LCF [38] qui consiste à utiliser des langages de tactiques (comme $\mathcal{L}_{\text{tac}}$ [32] pour Coq). Ils permettent d'augmenter l'automatisation d'un assistant à la preuve en développant des schémas de preuve automatiques.

Concernant l'approche crédule, un exemple est l'utilisation des prouveurs (dont des solveurs SMT) dans la multi-plateforme Why3 [50] ou encore l'utilisation des solveurs SMT dans Rodin. Cette dernière plate-forme est un outil de développement de Event-B [2] qui est

une extension de la méthode B. La plate-forme Rodin [28] permet de développer des modèles Event-B et de les prouver. Lorsque les prouveurs intégrés n'arrivent pas à prouver toutes les obligations de preuve, un mode de preuve interactif est proposé à l'utilisateur. Des solveurs SMT peuvent être appelés depuis l'interface de Rodin [31]. Pour cela, une traduction des formules Event-B vers le langage d'entrée des solveurs SMT est proposé. Actuellement, les preuves trouvées par ces solveurs ne sont pas reconstruites en preuves Event-B afin d'être rejouées dans Rodin. Néanmoins, certains solveurs SMT peuvent produire des traces permettant de vérifier les preuves. De la même manière, Merz et Vanzetto [51] proposent d'utiliser des solveurs SMT afin de prouver automatiquement des obligations de preuve exprimées en TLA^+. TLA^+ est un langage de spécification formelle basé sur la théorie des ensembles et sur TLA (Temporal Logic of Actions). Les solveurs SMT testés sont CVC3, Yices et Z3. Les auteurs envisagent à terme de reconstruire les preuves obtenues par les solveurs SMT afin de les vérifier dans un plongement de TLA^+ dans Isabelle.

Enfin, il existe de nombreuses implantations d'approches sceptiques, dans lesquelles les traces des prouveurs automatiques sont systématiquement vérifiées. Par exemple, dans [37], une approche sceptique est proposée entre le solveur SMT haRVey et l'assistant à la preuve Isabelle. Ainsi la formule Isabelle est niée puis traduite en format SMT-LIB afin d'être transmise à haRVey. Ce prouveur produit une trace avec les clauses de conflit générant les contradictions avec la négation du but, ainsi que des indications sur comment les contradiction peuvent se démontrer. Étant données ces clauses, leur preuve et la formule à prouver, la preuve finale est reconstruite. Toujours dans Isabelle, la tactique Sledgehammer [15] permet d'appeler plusieurs prouveurs automatiques sur des problèmes issus de Isabelle\HOL. Un outil reconstruit la preuve obtenue à partir d'un sous ensemble des hypothèses. Concernant Coq et les solveurs SMT, les auteurs de [5] proposent d'utiliser des solveurs SMT pour augmenter l'automatisation des preuves de Coq. Après avoir traduit les problèmes Coq dans les langages d'entrée des solveurs SMT, les traces fournies par les solveurs sont traduites dans un format de certificat. Les certificats sont ensuite vérifiés grâce à Coq dans lequel les structures de données ont été optimisées pour pouvoir gérer un grand nombre de clauses. Dans cette thèse, nous développons une approche sceptique entre un plongement profond de B en Coq et le prouveur Zenon. Ainsi, les preuves générées par Zenon sont reconstruites

en preuves BCoq afin d'être vérifiées automatiquement.

Chapitre 2

BCARe : un outil pour la vérification des règles ajoutées de l'Atelier B

Dans le but de remplacer les preuves à la main qui peuvent être sources d'erreur, Siemens IC-MOL développe depuis quelques années un outil appelé BCARe consacré à la vérification des règles ajoutées. Ce chapitre expose le processus de vérification d'une règle et sa mise en œuvre dans BCARe.

Après avoir défini la notion de règle ajoutée de l'Atelier B dans la section 2.1, nous motiverons le développement d'un outil tel que BCARe dans la section 2.2. La vérification d'une règle ajoutée est composée de quatre étapes qui concernent respectivement la capture de variable, la bonne définition, le typage et la validité de la règle (voir la section 2.3). Pour trois de ces étapes une obligation de preuve est générée dans un plongement profond de B en Coq (voir la section 2.4). Une traduction des règles dans cet environnement est donc nécessaire avant d'effectuer ces étapes de vérification, elle sera exposée dans la section 2.5. Enfin chacune des étapes sera détaillée ainsi que la génération de l'obligation de preuve associée dans les sections 2.6, 2.7, 2.8 et 2.9.

2.1 Les règles ajoutées de l'Atelier B

L'Atelier B [27] est un outil, développé par ClearSy, qui implante la méthode B. Il permet notamment de spécifier un modèle, de le raffiner, de générer les obligations de preuve (et de les démontrer) puis de générer du code exécutable conforme au modèle. Les

ingénieurs de Siemens IC-MOL l'utilisent pour développer les applications sécuritaires des métros automatiques.

2.1.1 Preuves dans l'Atelier B

L'Atelier B permet de spécifier un modèle dont la correction est assurée par plusieurs mécanismes. Le premier est la vérification du typage qui est complètement automatique (car, comme nous l'avons vu, la méthode B repose sur une théorie des ensembles typée). Si cette étape se déroule avec succès, alors les obligations de preuve sont générées afin d'assurer la cohérence des machines et la correction du raffinement.

Ces obligations de preuve peuvent être démontrées automatiquement avec la tactique pp de l'assistant à la preuve de l'Atelier B (voir plus de précisions sur cette tactique dans la sous-section 1.2.1 du chapitre 1). Lorsque cette tactique échoue, un mode de preuve interactive peut être utilisé. Dans ce cas, l'utilisateur applique d'autres tactiques sur le but et les hypothèses. Une tactique est une liste ordonnée de théories (une théorie est un ensemble de règles), qui détermine le parcours d'une base de règles pour identifier la ou les règles qui pourront être appliquées.

Ces règles permettent de compléter le prouveur de l'Atelier B. On les appelle les règles ajoutées. Certaines sont fournies avec l'Atelier B et sont donc appelées par le prouveur. Les utilisateurs peuvent en ajouter des nouvelles via le patchprover ou dans des fichiers pmm. Le patchprover peut contenir des règles utilisateurs mais aussi des tactiques les utilisant. Les règles ajoutées pourront être appelées par la tactique pp ou interactivement.

Cette notion de règle concerne aussi la plate-forme Rodin sous forme d'une extension de la théorie de Event-B [49]. L'expressivité de ces règles est à l'heure actuelle moins importante que celle des règles de l'Atelier B.

2.1.2 Règles

Il y a deux types de règles dans l'Atelier B : les règles de déduction et les règles de réécriture. Les règles de déduction sont de la forme $A_1 \wedge \ldots \wedge A_n \Rightarrow B$, où les A_i, gardes ou prédicats, sont les antécédents et le prédicat B est le conséquent. Les gardes constituent les conditions d'application de la règle. Une règle de déduction peut être déductive (« for-

ward ») si elle s'applique sur une hypothèse ou inductive (« backward ») si elle s'applique sur le but. Elle est définie comme déductive ou inductive lors de sa création. L'utilisateur doit contrôler à chaque application le mode d'utilisation de la règle. Dans la pratique, chaque application de la règle est vérifiée pour déterminer son mode d'utilisation. Une règle de réécriture est de la forme $A_1 \wedge \ldots \wedge A_n \Rightarrow B == C$ où les A_i, gardes ou prédicats, sont les antécédents et B et C sont soit des expressions soit des prédicats. Le symbole binaire « == » désigne un remplacement syntaxique : si une instance de B apparaît dans le but, alors elle est remplacée par l'instance correspondante de C (si les gardes de la règle sont vérifiées par les hypothèses).

La syntaxe des règles est définie ci-dessous :

$$
\begin{array}{lcl}
V & := & I \mid V \mapsto V \\
E & := & V \mid [V := E]E \mid E \mapsto E \mid S \\
S & := & S \times S \mid \mathbb{P}(S) \mid \{V|P\} \mid \mathsf{BIG} \mid I \\
P & := & P \wedge P \mid P \Rightarrow P \mid \neg P \mid \forall V.P \mid [V := E]P \mid E = E \mid E \in S \mid I \\
G & := & \mathsf{binhyp}(P) \mid \mathsf{blvar}(I) \mid I \setminus P \mid I \setminus E \\
A & := & P \mid G \mid A \wedge A \\
C & := & P \mid E == E \mid P == P \\
R & := & C \mid A \Rightarrow C
\end{array}
$$

où V représente une variable, I un identificateur, E une expression, S un ensemble, P un prédicat, G une garde, A un antécédent, C un conséquent et R une règle.

Des variables libres peuvent apparaître dans E, S, et P ; ces variables sont considérées comme des métavariables et seront instanciées lors de l'application de la règle. Les autres connecteurs logiques (tels que $\Leftrightarrow$, $\vee$ et $\exists$) et les opérateurs ensemblistes ($\cup$, $\cap$, etc.), sont définis à partir de ceux donnés ci-dessus. Ce sont ceux définis dans le langage B.

Quoique l'Atelier B et BCARe offrent de plus nombreuses possibilités, nous restreignons cette présentation aux gardes de non-liberté de variables ($I \setminus P$ et $I \setminus E$) et aux gardes $\mathsf{binhyp}(P)$ et $\mathsf{blvar}(I)$. $I \setminus P$ (respectivement $I \setminus E$) signifie que I doit être non libre dans P (respectivement dans E). $\mathsf{binhyp}(P)$ vérifie la présence de P dans le contexte de la preuve et $\mathsf{blvar}(I)$ instancie I avec les variables liées du but au point de réécriture.

Lors de l'application d'une règle de réécriture, l'Atelier B effectue un remplacement syntaxique sans aucune vérification. L'application d'une telle règle peut donc introduire des

captures de variable par conséquent des incohérences dans un but sans que l'Atelier B ne prévienne l'utilisateur. Néanmoins, la garde blvar si elle est correctement utilisée permet de faire la vérification nécessaire. Dans ce cas, l'Atelier B permet d'effectuer des réécritures sous les quantificateurs. Ce n'est pas possible avec tous les assistants à la preuve, notamment avec Coq [60] qui interdit ce type de réécriture.

Les deux exemples ci-dessous illustrent les deux sortes de règles.

Exemple 2 (Règle de déduction)

ForAllX.3 : $(a \setminus A) \wedge A = \emptyset \Rightarrow \forall a.a \notin A$

Si la règle est définie comme inductive, alors elle peut être appliquée sur un but de la forme $\forall b.b \notin B$ *si* $b \setminus B$. *Cela remplace le but à prouver par* $B = \emptyset$.

Exemple 3 (Règle de réécriture)

SimplifyRelDorXY.2 :

$\mathsf{binhyp}(f \in u \nrightarrow v) \wedge \mathsf{binhyp}(a \in \mathsf{dom}(f)) \wedge$

$\mathsf{blvar}(Q) \wedge (Q \setminus (f \in u \nrightarrow v)) \wedge (Q \setminus (a \in \mathsf{dom}(f))) \Rightarrow \{a\} \lhd f == \{(a \mapsto f(a))\}$

Cette règle signifie que si f *est une fonction partielle et que* a *appartient à son domaine de définition alors* f *restreinte au domaine* $\{a\}$ *peut être remplacée par le singleton* $\{(a \mapsto f(a))\}$.

Cette règle peut être appliquée si le but contient un terme de la forme $\{b\} \lhd g$ *et si* $g \in s \nrightarrow t$ *et* $b \in \mathsf{dom}(g)$ *sont en hypothèse. Dans ce cas, l'occurrence de* $\{b\} \lhd g$ *la plus à droite sera remplacée syntaxiquement par* $\{(b \mapsto g(b))\}$.

2.1.3 Application d'une règle dans l'Atelier B

Nous formalisons dans cette partie l'application d'une règle à un séquent. Nous nous appuierons sur la présentation informelle de l'application d'une règle faite dans le manuel [26] de ClearSy. Dans la suite, nous utilisons les notations suivantes :

Notation 1

Nous noterons :

- $\sigma(L)$ *le résultat de l'application de la substitution* σ *sur* L *où* L *est une expression ou un prédicat.*

- $P[L/L']$ *le résultat du remplacement de* L *par* L' *dans le prédicat* P *où* L *et* L' *sont des expressions ou des prédicats. S'il y a plusieurs occurrences de* L *alors seule celle correspondant au sous-arbre le plus à droite dans l'arbre de syntaxe abstraite sera remplacée.*
- BRV *l'ensemble des variables liées au point de réécriture.*
- A_i *des prédicats et* G_i *des gardes.*

Une règle peut être appliquée sur un séquent (constitué d'un ensemble d'hypothèses et d'un but) dans l'Atelier B uniquement si toutes ses gardes sont vérifiées par le séquent. Selon la nature de la règle appliquée (règle de réécriture, déductive ou inductive), l'application aura un effet sur le but ou sur les hypothèses. Ainsi, une règle inductive s'applique sur le but et peut générer de nouveaux sous-buts. Une règle déductive s'applique sur les hypothèses pour en générer une nouvelle. Enfin, une règle de réécriture remplace syntaxiquement une expression (ou un prédicat) du but par une autre. Ces règles s'appliquent s'il existe une substitution σ qui unifie le conséquent de la règle avec un sous-terme du but (si c'est une règle de réécriture ou inductive) ou une hypothèse (si elle est déductive). Cette substitution peut être complétée par l'application de chaque garde, si ces dernières nécessitent des unifications.
L'application d'une règle $G_1 \wedge \ldots \wedge G_m \wedge A_1 \wedge \ldots \wedge A_n \Rightarrow C$ sur un séquent $Hyp \vdash But$ est définie par les fonctions App_{RD} pour les règles de déduction, App_{RR} pour les règles de réécriture et App_{G_s} et App_G pour les gardes, définies ci-dessous.

Étant donnés une règle, un séquent et la nature de la règle de déduction (Inductif ou Deductif), App_{RD} retourne un ensemble de séquents.

$$App_{RD}(A_1 \wedge \ldots \wedge A_n \Rightarrow C, Hyp \vdash But, \mathsf{Inductif}) =$$
$$\begin{cases} \{Hyp \vdash \sigma_0(A_1), \ldots, Hyp \vdash \sigma_0(A_n)\} \text{ si } \exists \sigma_0 \text{ tel que } \sigma_0(C) = But \\ \{Hyp \vdash But\} \text{ sinon} \end{cases}$$

$$App_{RD}(G_1 \wedge \ldots \wedge G_m \wedge A_1 \wedge \ldots \wedge A_n \Rightarrow C, Hyp \vdash But, \mathsf{Inductif}) =$$
$$\begin{cases} \{Hyp \vdash \sigma_m(A_1), \ldots, Hyp \vdash \sigma_m(A_n)\} \\ \qquad \text{si } \exists \sigma_0 \text{ tel que } \sigma_0(C) = But \\ \qquad \text{et } App_{G_s}(\{G_1, \ldots, G_m\}, Hyp, \sigma_0, \emptyset) = \top \\ \{Hyp \vdash But\} \text{ sinon} \end{cases}$$

$$App_{RD}(G_1 \wedge \ldots \wedge G_m \wedge A_1 \wedge \ldots \wedge A_n \Rightarrow C, Hyp \vdash But, \mathsf{Deductif}) =$$
$$\begin{cases} \{Hyp \wedge \{\sigma_{m+n}(C)\} \vdash But\} \\ \qquad \text{si } \exists \sigma_1.\sigma_1(A_1) \in Hyp \\ \qquad \text{et } \forall i.2 \leq i \leq n,\ \exists \sigma'.\exists \sigma_i.\sigma_i = \sigma_{i-1}\sigma' \text{ et } \sigma_i(A_i) \in Hyp \\ \qquad \text{et } App_{G_s}(\{G_1, \ldots, G_m\}, Hyp, \sigma_n, \emptyset) = \top \\ \{Hyp \vdash But\} \text{ sinon} \end{cases}$$

Etant donné une règle de réécriture et un séquent, App_{RR} retourne une liste de séquents.

$$App_{RR}(A_1 \wedge \ldots \wedge A_n \Rightarrow E == F, Hyp \vdash But) =$$
$$\begin{cases} \{Hyp \vdash \sigma_0(A_1), \ldots, Hyp \vdash \sigma_0(A_n), Hyp \vdash But[\sigma_0(E)/\sigma_0(F)]\} \\ \qquad \text{si } \exists \sigma_0 \text{ tel que } \sigma_0(E) \text{ soit un sous terme de } But \\ \{Hyp \vdash But\} \text{ sinon} \end{cases}$$
$$App_{RR}(G_1 \wedge \ldots \wedge G_m \wedge A_1 \wedge \ldots \wedge A_n \Rightarrow E == F, Hyp \vdash But) =$$
$$\begin{cases} \{Hyp \vdash \sigma_m(A_1), \ldots, Hyp \vdash \sigma_m(A_n), Hyp \vdash But[\sigma_m(E)/\sigma_m(F)]\} \\ \qquad \text{si } \exists \sigma_0 \text{ tel que } \sigma_0(E) \text{ soit un sous terme de } But \\ \qquad \text{et } App_{G_s}(\{G_1, \ldots, G_m\}, Hyp, \sigma_0, \emptyset) = \top \\ \{Hyp \vdash But\} \text{ sinon} \end{cases}$$

Pour simplifier, nous ne considérons dans ce chapitre que les gardes binhyp, blvar et $I \backslash P$ mais nous traitons aussi d'autres gardes dans BCARe comme bpattern, bfresh ... pour lesquelles le calcul de la substitution σ est moins trivial. Pour la garde binhyp(h), le moteur de preuve de l'Atelier B cherche une instance de h dans les hypothèses. S'il y en a une, la garde est un succès et une substitution est appliquée au reste de la règle. Sinon, l'application de la règle est refusée. La garde blvar(Q) permet de contrôler si une capture de variable peut avoir lieu lors de l'application d'une règle de réécriture. Elle est toujours associée à une garde $Q \backslash P$. Lors de la vérification de ces deux gardes par l'Atelier B, Q est instancié par l'ensemble des variables liées au point de réécriture (nous noterons cet ensemble BRV). Le point de réécriture correspond à l'occurrence de l'expression (ou du prédicat) à réécrire qui est le sous-arbre le plus à droite de l'arbre de syntaxe abstraite du but. Les deux gardes sont évaluées avec succès si chaque variable de BRV est non libre dans $\sigma(P)$, c'est-à-dire si BRV est non libre dans le résultat de la substitution globale trouvée par l'Atelier B sur P. Dans le cas où $I \backslash P$ n'est pas associée à une garde blvar, elle sera un succès si l'instanciation $\sigma(I)$ est non libre dans $\sigma(P)$.

La fonction App_{G_s} formalise la vérification de toutes les gardes lors de l'application d'une règle : étant donnés un ensemble de gardes, les hypothèses (donc une liste de pré-

dicats), une substitution et un ensemble de variables, elle retourne $\top$, si toutes les gardes sont vérifiées.

$$App_{G_s}(\{\}, Hyp, \sigma, Q) = \top$$

$$App_{G_s}(\{G_1, \ldots, G_g\}, \sigma, Q) = \begin{cases} App_{G_s}(\{G_2, \ldots, G_g\}, \sigma'\sigma, Q') \\ \qquad \text{si } App_G(G_1, Hyp, \sigma, Q) = (\top, \sigma', Q') \\ \bot \text{ sinon} \end{cases}$$

La fonction App_G formalise la vérification des gardes lors de l'application d'une règle : étant donnés une garde, les hypothèses (donc une liste de prédicats), une substitution et un ensemble de variables, elle retourne $\top$ ainsi que l'ensemble des variables liées au point de réécriture, si la garde est vérifiée.

$$App_G(\mathsf{binhyp}(h), Hyp, \sigma, Q) = \begin{cases} (\top, \sigma\sigma', Q) \text{ si } \exists \sigma' \text{ tel que } \sigma\sigma'(h) \in Hyp \\ (\bot, \sigma, Q) \text{ sinon} \end{cases}$$

$$App_G(\mathsf{blvar}(I), Hyp, \sigma, Q) = (\top, \sigma, \mathsf{BRV})$$

$$App_G(I \backslash P, Hyp, \sigma, Q) = \begin{cases} (\top, \sigma, Q) \text{ si } \mathsf{blvar}(I) \in Hyp \text{ et } \forall x.x \in Q, x \backslash \sigma(P) \\ \quad \text{ou si } \mathsf{blvar}(I) \notin Hyp \text{ et } \sigma(I) \backslash \sigma(P) \\ (\bot, \sigma, Q) \text{ sinon} \end{cases}$$

Exemple 4 (Application d'une règle inductive)

Soit la règle : $a \subseteq t \wedge b \in u \Rightarrow a \times \{b\} \in t \nrightarrow u$

Nous allons simuler l'application de cette règle inductive sur le séquent

$a \subseteq c \wedge b \in d \vdash a \times \{b\} \in c \nrightarrow d$ *avec la fonction* App_{RD}.

$App_{RD}(a \subseteq t \wedge b \in u \Rightarrow a \times \{b\} \in t \nrightarrow u, a \subseteq c \wedge b \in d \vdash a \times \{b\} \in c \nrightarrow d, \mathsf{Inductif}) =$

$\{a \subseteq c \wedge b \in d \vdash \sigma(a \subseteq t), a \subseteq c \wedge b \in d \vdash \sigma(b \in u)\}$

avec $\sigma = [t \leftarrow c, u \leftarrow d]$ *car* $\sigma(a \times \{b\} \in t \nrightarrow u) = a \times \{b\} \in c \nrightarrow d$

Donc les nouveaux buts à prouver, après l'application de la règle, sont

$a \subseteq c \wedge b \in d \vdash a \subseteq c$ *et* $a \subseteq c \wedge b \in d \vdash b \in d$.

Exemple 5 (Application d'une règle déductive)

Soit la règle $(a \setminus A) \wedge \forall a.a \notin A \Rightarrow A = \emptyset$.

Nous allons simuler l'application de cette règle déductive sur le séquent $\forall x.x \notin E \vdash E = \emptyset$ *avec la fonction* App_{RD}.

$App_{RD}((a \setminus A) \wedge \forall a.a \notin A \Rightarrow A = \emptyset, \forall x.x \notin E \vdash E = \emptyset, \mathsf{Deductif}) =$

$\{\forall x.x \notin E \wedge \sigma(A = \emptyset) \vdash E = \emptyset\}$

avec $\sigma = [a \leftarrow x, A \leftarrow E]$

car $\sigma(\forall a.a \notin A) = \forall x.x \notin E$ *est dans les hypothèses*

et $App_G(a \setminus A, E = \emptyset, \sigma, \emptyset) = (\top, \sigma, \emptyset)$ *car* $\mathsf{blvar}(a)$ *n'est pas dans les hypothèses et*

$\sigma(a) \setminus \sigma(A) = x \setminus E$ *est vrai.*

Donc le nouveau but à prouver, après l'application de la règle, est

$\forall x.x \notin E \wedge\ E = \emptyset \vdash E = \emptyset$.

Exemple 6 (Application d'une règle de réécriture)

Soit la règle SimplifyRelDorXY.2 :

$\mathsf{binhyp}(f \in u \nrightarrow v) \wedge \mathsf{binhyp}(a \in \mathsf{dom}(f)) \wedge$

$\mathsf{blvar}(Q) \wedge (Q \setminus (f \in u \nrightarrow v)) \wedge (Q \setminus (a \in \mathsf{dom}(f))) \Rightarrow \{a\} \lhd f == \{(a \mapsto f(a))\}$

Nous allons simuler l'application de cette règle de réécriture sur le séquent

$f \in s \nrightarrow t \wedge x \in \mathsf{dom}(f) \vdash \{x\} \lhd f = \{(x \mapsto f(x))\}$ *avec la fonction* App_{RR}.

$App_{RR}(SimplifyRelDorXY.2, f \in s \nrightarrow t \wedge x \in \mathsf{dom}(f) \vdash \{x\} \lhd f = \{(x \mapsto f(x))\}) =$

$\{f \in s \nrightarrow t \wedge x \in \mathsf{dom}(f) \vdash \sigma_1(\{(a \mapsto f(a))\}) = \{(x \mapsto f(x))\}\}$

avec $\sigma_1 = [a \leftarrow x, f \leftarrow f, u \leftarrow s, v \leftarrow t]$ *qui est la substitution finale calculée lors de l'application.*

La règle a pu s'appliquer car le but contient bien une instance du membre gauche de la règle : $\{x\} \lhd f$ *est une instance de* $\{a\} \lhd f$. *On en déduit une première substitution* $\sigma_0 = [a \leftarrow x, f \leftarrow f]$.

De plus toutes les gardes sont vraies :

- $\mathsf{binhyp}(f \in u \nrightarrow v)$

  Soit H *le contexte d'hypothèses* $f \in s \nrightarrow t, x \in \mathsf{dom}(f)$.

  $App_G(\mathsf{binhyp}(f \in u \nrightarrow v), H, \sigma_0, \emptyset) = (\top, \sigma_1, \emptyset)$ *avec* $\sigma_1 = \sigma_0\sigma'$ *et* $\sigma' = [u \leftarrow s, v \leftarrow t]$ *car* $\sigma_1(f \in u \nrightarrow v) = f \in s \nrightarrow t$ *est dans* H.
- $\mathsf{binhyp}(a \in \mathsf{dom}(f))$

  $App_G(\mathsf{binhyp}(a \in \mathsf{dom}(f)), H, \sigma_1, \emptyset) = (\top, \sigma_1, \emptyset)$ *car* $\sigma_1(a \in \mathsf{dom}(f)) = x \in \mathsf{dom}(f)$ *est dans* H. *La substitution* σ_1 *n'est pas complétée ici puisqu'elle traite toutes les variables libres de la règle.*
- $\mathsf{blvar}(Q)$

  $App_G(\mathsf{blvar}(Q), H, \sigma_1, \emptyset) = (\top, \sigma_1, \emptyset)$ *car étant donné qu'il n'y a pas de variable liée au point de réécriture,* BRV *est vide*
- $Q \setminus (f \in u \nrightarrow v)$

  $App_G(Q \setminus (f \in u \nrightarrow v), H, \sigma, \emptyset) = (\top, \emptyset)$ *car* $\forall x.x \in \emptyset, x \setminus \sigma(f \in u \nrightarrow v)$.

Ici, il n'y a pas de variable liée ($\mathsf{BRV} = \emptyset$*).*

– $Q \setminus (a \in \mathsf{dom}(f))$

$App_G(Q \setminus (a \in \mathsf{dom}(f)), H, \sigma_1, \emptyset) = (\top, \sigma_1, \emptyset)$ *car* $\forall y.y \in \emptyset, y \setminus x \in \mathsf{dom}(f)$.

Donc le nouveau but à prouver, après l'application de la règle, est

$f \in s \nrightarrow t \wedge x \in \mathsf{dom}(f) \vdash \{(x \mapsto f(x))\} = \{(x \mapsto f(x))\}$.

2.2 BCARe : les motivations

A chaque nouveau projet développé avec la méthode B par Siemens IC-MOL, de nouvelles règles sont ajoutées à la base de règles interne (qui comprend aussi des règles venant de l'Atelier B). Plusieurs vérifications sont nécessaires lorsqu'une nouvelle règle est ajoutée. Voici des exemples de règles qui mettent toutes en valeur un certain type de vérification qui est nécessaire pour déterminer si une règle est correcte.

1. Soit la règle $\mathsf{binhyp}(x = a) \Rightarrow x == a$. Cette règle semble vraie dans la mesure où $x = a \Rightarrow x = a$ est trivial. Pourtant si on l'applique sur le but faux $n = 0 \vdash \forall n.n \in \mathbb{N} \Rightarrow n = 0$, $\mathsf{binhyp}(x = a)$ est vérifiée grâce à l'hypothèse $n = 0$, et l'occurrence de n la plus à droite du but est remplacée par 0. On obtient le but $n = 0 \vdash \forall n.n \in \mathbb{N} \Rightarrow 0 = 0$ qui peut être démontré. L'incohérence est introduite à cause de la capture de la variable n qui est liée au point de réécriture par le $\forall$. Il est donc nécessaire de vérifier pour toute règle de réécriture qu'elle ne pourra pas introduire de capture de variable.
2. Soit la règle $\{x|x \in s \cup \{b\} \wedge b \notin d\} == \{x|x \in s \wedge b \notin d\} \cup (\{b\} \cap \{x|b \notin d\})$ qui permet de décomposer un certain schéma d'ensemble en compréhension. Cette règle semble correcte selon la définition de l'union. Néanmoins, elle pose problème au niveau du typage puisque l'ensemble en compréhension $\{x|b \notin d\}$ n'a de sens que si x est typé et contraint. Une correction possible de cet ensemble en compréhension est $\{x|x \in s \wedge b \notin d\}$. La vérification du typage de cette règle permet de détecter qu'elle contient un ensemble mal formé. De manière générale, cette étape est nécessaire pour interdire les règles mal typées, ces règles pouvant introduire par exemple le paradoxe de Russell.
3. Soit la règle $\mathsf{binhyp}(f \in s \nrightarrow t) \Rightarrow x = y == f(x) = f(y)$ où $f \in s \nrightarrow t$ signifie que f est une fonction partielle et injective. Cette règle très simple ne pose, a priori,

aucun problème. Pourtant dans la théorie de B, une application de fonction $f(x)$ est bien définie si et seulement si f est une fonction et si son argument x appartient à son domaine de définition. Ici, on ne sait rien sur x, par conséquent cette règle peut introduire une expression mal définie. Sans cette vérification, une règle peut introduire une expression mal définie telle que $\frac{1}{0}$ ou $\mathsf{min}(\emptyset)$. C'est pourquoi il est nécessaire de vérifier la bonne définition des règles.

4. Soit la règle $\neg(a \vee b \wedge c) \Rightarrow (\neg a \wedge \neg(b \wedge c))$ qui semble distribuer le $\neg$ sur le $\wedge$. Pourtant, cette règle est fausse puisque $\neg(a \vee b \wedge c)$ doit se lire, selon les priorités des opérateurs de l'Atelier B, $\neg((a \vee b) \wedge c)$. On doit donc vérifier ce que l'on appellera la validité de la règle c'est-à-dire que la règle peut être dérivée des règles du B-Book. Toute règle non valide peut permettre de prouver un but faux et donc être la cause, in fine, d'un dysfonctionnement.

Actuellement, l'outil automatique CHAINE VALIDATION [58] est utilisé pour vérifier ces règles. Cet outil ne fait pas les vérifications sur la capture de variable et sur la bonne définition (cf. règles 1 et 3). Concernant le typage, il fait une vérification mais le problème de la règle 2 n'est pas détecté car l'outil accepte les ensembles en compréhension sans information de typage. De plus, lorsque l'outil trouve une preuve il ne donne pas de justification. Par conséquent, lorsqu'une règle n'est pas prouvée par cet outil, la règle doit être vérifiée à la main.

L'objectif principal de l'environnement BCARe, développé par Karim Berkani et Eric Le Lay depuis 2008, est de compléter les outils existants et de fournir une aide mécanisée à la vérification formelle des règles de preuve. Par exemple, dans la règle $a \setminus A \wedge A = \emptyset \Rightarrow \forall a.a \notin A$, la condition $a \setminus A$ doit être vérifiée avant que la règle soit appliquée. BCARe permet de vérifier que cette condition est nécessaire, alors que cela est impossible avec les autres outils disponibles. De plus, grâce à son plongement de B dans Coq, BCARe offre un environnement formel de preuve B interactif et automatique. A terme, cela permettrait de remplacer totalement les preuves manuelles qui peuvent être sources d'erreur.

Dans le cadre de cette thèse, les trois premières étapes de vérification ont été consolidées. La contribution principale est l'automatisation de la vérification de la validité de la règle (voir les chapitres 3, 4 et 5).

2.3 Vérification d'une règle

La vérification d'une règle est composée de quatre étapes, qui ont émergées de la pratique (voir la figure 2.1). Certaines d'entre elles sont aussi présentées dans [49] dans le cadre de Event-B.

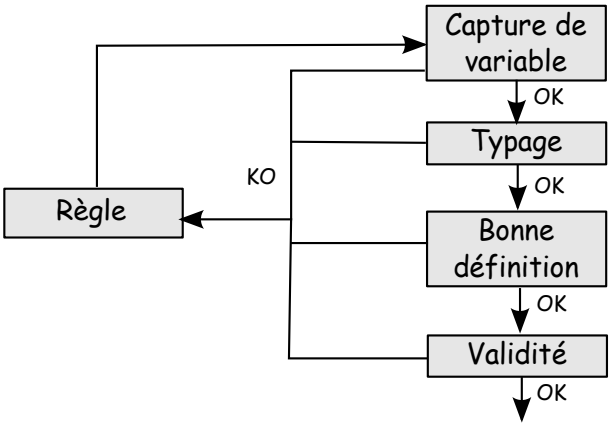

FIGURE 2.1 – Processus de vérification d'une règle

1. La première étape concerne uniquement les règles de réécriture et consiste à vérifier que ces règles sont correctement protégées contre la capture de variable. Cela est nécessaire puisque l'Atelier B ne vérifie pas le contexte d'application lorsqu'une règle de réécriture est appliquée : il l'applique syntaxiquement. Par conséquent, lorsqu'une règle de réécriture est appliquée sur des variables liées, une capture de variable peut apparaître et introduire des incohérences.
2. L'objectif de la deuxième étape est de vérifier que la règle est bien typée, ce qui revient à vérifier que les expressions de la règle sont bien typées selon les règles de typage du B-Book (voir [1]). Pour cela, tous les types doivent être explicités en hypothèse. Cependant, une règle peut contenir des métavariables dont le type peut être implicite. Une étape préliminaire est donc nécessaire pour inférer le type de toutes les métavariables. Ainsi la règle résultante est enrichie avec les contraintes de type trouvées et son type peut être vérifié. A partir de cette étape, on considérera uniquement la règle enrichie puisque les informations de typage peuvent être nécessaires pour faire les preuves de bonne définition et de validité.

3. La troisième étape consiste à vérifier que la règle est bien définie. Dans [4], les auteurs montrent que les définitions conditionnelles peuvent conduire à des expressions mal définies, comme par exemple la division par zéro ou l'application d'une fonction à un argument en dehors de son domaine de définition. Certaines de ces conditions peuvent être vérifiées en amont grâce au typage. Lorsqu'elles sont plus complexes, il est possible de les extraire de la formule afin de vérifier qu'elles sont respectées. Un opérateur syntaxique à appliquer sur la règle, qui agit comme un filtre, est proposé et extrait les obligations de preuve de bonne définition pour l'application de fonction.

4. La dernière étape consiste à vérifier qu'une règle est valide c'est-à-dire qu'elle peut être dérivée en utilisant le système d'inférence logique de B (voir [1]). Un deuxième filtre, défini dans [4], peut être appliqué sur la règle afin de supprimer les applications de fonction lorsqu'elles ne peuvent pas introduire d'expressions mal définies. Cela permet de simplifier la règle avant de la démontrer.

Les étapes de la vérification d'une règle avec BCARe suivent les quatre étapes précédentes. Si la règle est une règle de réécriture alors un composant de BCARe vérifie que les gardes protègent correctement les variables libres (voir la section 2.6). Un autre composant infère ensuite les types des expressions de la règle grâce à un algorithme d'inférence de type qui a été conçu en suivant les règles de typage de B (voir [1]). Puis il génère le lemme de typage correspondant (voir la section 2.7) à la règle enrichie avec les types inférés par l'algorithme. Ce lemme de typage doit être vérifié avec les règles du système de types de B données dans le B-Book [1]. Nous avons étendu ce système pour traiter les métavariables de prédicat (qui peuvent apparaître dans les règles) et pour avoir un traitement plus général de l'ensemble vide (voir la sous-section 2.7.1). Un composant de BCARe génère le lemme de bonne définition (voir la section 2.8) qui permet de vérifier que les arguments des applications de fonction appartiennent bien au domaine de définition de la fonction. Enfin, le lemme de la validité de la règle (voir la section 2.9) est généré. Nous appellerons ces trois lemmes, les lemmes de vérification. Ces lemmes sont écrits dans la syntaxe de Coq, ou plus exactement sont générés dans le cadre d'un plongement profond de la théorie des ensembles de B dans Coq (voir la section 2.4), appelé BCoq.

2.4 Le plongement BCoq

BCoq est un plongement profond de la théorie des ensembles de B dans Coq [60] dans lequel les trois lemmes de vérification sont générés.

Les quatre catégories syntaxiques *Variable*, *Expression*, *Set* et *Predicate* présentées dans la section 1.1 du chapitre 1 sont définies inductivement dans Coq (voir [9]) par *Variable_B*, *Expression*, *Set_B* et *Predicate* :

```
Variable I : Set.

Inductive Variable_B : Set :=
        Identifier : I → Variable_B
        | Pair_var : Variable_B → Variable_B → Variable_B.

Inductive Expression : Set :=
        Var : Variable_B → Expression
        | Subst_expr : Variable_B → Expression → Expression → Expression
        | Pair_expr : Expression → Expression → Expression
        | Choice : Set_B → Expression
        | Ens : Set_B → Expression
with Set_B : Set :=
        Cart_prod : Set_B → Set_B → Set_B
        | Pow : Set_B → Set_B
        | Comp : Variable_B → Predicate → Set_B
        | BIG : Set_B
with Predicate : Set :=
        Conj : Predicate → Predicate → Predicate
        | Impl : Predicate → Predicate → Predicate
        | Not : Predicate → Predicate
        | Gen : Variable_B → Predicate → Predicate
        | Subst_pred : Variable_B → Expression → Predicate → Predicate
        | Eq : Expression → Expression → Predicate
        | In : Expression → Set_B → Predicate.
```

Dans cette syntaxe, l'opérateur choice apparaît, alors qu'il n'y était pas dans la syntaxe des règles. Cela s'explique par le fait que cet opérateur est utile uniquement pour définir l'application de fonction et par conséquent, il n'existe pas dans l'Atelier B.

Dans la suite nous utiliserons la syntaxe abstraite (avec la notation « · ») définie ci-dessous plutôt que la syntaxe concrète Coq donnée ci-dessus.

$$
\begin{array}{lcl}
\dot{V} & := & \dot{I} \mid \dot{V} \mapsto \dot{V} \\
\dot{E} & := & \dot{V} \mid [\dot{V} :\doteq \dot{E}]\dot{E} \mid \dot{E} \mapsto \dot{E} \mid \dot{\mathsf{choice}}\,(\dot{S}) \mid \dot{S} \\
\dot{S} & := & \dot{S} \times \dot{S} \mid \dot{\mathbb{P}}\,(\dot{S}) \mid \{\dot{V} \mid \dot{P}\} \mid \dot{\mathsf{BIG}} \mid \dot{I} \\
\dot{P} & := & \dot{P} \mathbin{\dot{\wedge}} \dot{P} \mid \dot{P} \mathbin{\dot{\Rightarrow}} \dot{P} \mid \dot{\neg}\, \dot{P} \mid \dot{\forall}\, \dot{V}.\dot{P} \mid [\dot{V} :\doteq \dot{E}]\dot{P} \mid \dot{E} \doteq \dot{E} \mid \dot{E} \mathbin{\dot{\in}} \dot{S} \mid \dot{I}
\end{array}
$$

où $\dot{I}$, $\dot{V}$, $\dot{E}$, $\dot{S}$, et $\dot{P}$ représentent respectivement les versions réifiées (c'est-à-dire enco-

dées dans Coq) des ensembles I, V, E, S et P, définis dans la sous-section 2.1.2.

Par rapport à la syntaxe concrète présentée ci-dessus, des métavariables c'est-à-dire des variables libres ($\dot{I}$), apparaissent dans la catégorie syntaxique des ensembles et des prédicats. En réalité, contrairement à la syntaxe concrète qui permet d'exprimer des formules B, cette syntaxe abstraite nous permet d'exprimer des formules B encodé en Coq. Ces métavariables ne sont donc pas exprimables en B mais uniquement en BCoq puisqu'elles seront liées par un quantificateur universel Coq.

Le système de preuve est défini inductivement par la relation Coq $infer$. Chaque règle d'inférence est un constructeur de ce prédicat. La notation Coq $infer\ H\ P$ est donc la traduction du séquent $H \vdash P$. Par exemple, les règles $RULE\_n$ (avec $n \in \mathbb{N}$) ci-dessous correspondent aux règles exposées dans l'annexe A.1. Le type Hyp représente le type du contexte des hypothèses, à savoir $list\ \ \dot{P}$.

Dans la suite nous noterons $\forall$, le forall Coq.

```
Inductive infer : Hyp → Ṗ → Prop :=
                  |RULE_1 : ∀ h : Hyp, ∀ P Q : Ṗ,
                                (infer h P) →
                                (infer h Q) → (infer h (P ∧̇ Q))
                  |RULE_2 : ∀ Q P : Ṗ, ∀ h : Hyp,
                                (infer h (P ∧̇ Q)) →
                                (infer h P)
                  ...
```

Concernant le typage, les deux catégories syntaxiques $Type$ et $Type\_predicate$ (voir section 1.1 du chapitre 1) sont aussi définies inductivement dans Coq de la manière suivante :

```
Inductive Type_B : Set :=
                        type : Ė → Type_B
                        |super : Ṡ → Type_B
                        |cart_prod_type : Type_B → Type_B → Type_B
                        |pow_Type : Type_B → Type_B
                        |idt : Ṡ → Type_B.

Inductive Type_Predicate : Set :=
                        check : Ṗ → Type_Predicate
                        |eq_type : Type_B → Type_B → Type_Predicate.
```

Comme le système de preuve, celui correspondant au typage est défini inductivement par la relation Coq $infer\_type$. Chaque règle d'inférence est un constructeur du prédicat.

La notation Coq $infer\_type\ H\ (check\ \ P)$ est donc la traduction du séquent $H \vdash_\tau \mathsf{check}(P)$ où $\vdash_\tau$ représente le jugement de type de B. Par exemple, les règles T_n ci-dessous correspondent aux règles exposées dans l'annexe A.2 (où $n \in \mathbb{N}$).

```
Inductive infer_type : Hyp → Type_Predicate → Prop :=
          | T_1 : ∀ h : Hyp, ∀ P Q : Ṗ,
                  (infer_type h (check P)) →
                  (infer_type h (check Q)) →
                  (infer_type h (check (P ∧̇ Q)))
          | T_2 : ∀ h : Hyp, ∀ P Q : Ṗ,
                  (infer_type h (check P)) →
                  (infer_type h (check Q)) →
                  (infer_type h (check (P ⇒̇ Q)))
```

Dans la suite, nous noterons les jugements de preuve et de typage $infer$ et $infer\_type$ respectivement « $\dot{\vdash}$ » et « $\dot{\vdash}_\tau$ ».

Les autres connecteurs logiques dérivés comme $\neg$, $\Leftrightarrow$ et $\exists$ sont définis par des définitions Coq.

```
Definition Disj (P Q : Predicate) : Predicate :=
                                    (Impl (Not P) Q).

Definition Equiv (P Q : Predicate) : Predicate :=
                                    (Conj (Impl P Q)(Impl Q P)).

Definition Exist (x : Variable_B) (P : Predicate) : Predicate :=
                                    (Not (Gen x (Not P) )).
```

La théorie des ensembles de B étant axiomatique, nous utilisons les axiomes équationnels de Coq pour définir les opérateurs dérivés de B. Voici par exemple comment est définie l'union dans BCoq.

```
Axiom def_Union_Coq : ∀ u s t : Ṡ, ∀ a : V̇, ∀ h : Hyp,
  (a ∖̇ u) →(a ∖̇ s) →(a ∖̇ t) →
  (infer h (s ⊆̇ u)) →(infer h (t ⊆̇ u)) →
  (s ∪̇ t) = ({a ∣ a ∈̇ u ∧̇ (a ∈̇ s ∨̇ a ∈̇ t)}).
```

Actuellement, seul un sous-ensemble de B est traité dans BCoq : la logique propositionnelle et la logique du premier ordre, les opérateurs ensemblistes de base, les relations, les fonctions, les unions et intersections généralisées et quantifiées.

2.5 La réification des règles

Le processus de réification permet de passer de la syntaxe d'une règle ajoutée de l'Atelier B définie dans la sous-section 2.1.2 à une syntaxe d'une formule B définie dans la section 2.4. Dans la syntaxe abstraite précédente, les règles n'apparaissent pas puisqu'elles sont réifiées (c'est-à-dire traduites en BCoq) en prédicats par ce processus. Cette traduction n'est pas immédiate car les règles contiennent des métavariables et des gardes qui requièrent un traitement spécifique pour être traduites en formules Coq.

Notation 2

Le processus de réification utilisent les ensembles et fonctions suivantes :

- $\mathcal{M}_E$, $\mathcal{M}_S$ *et* $\mathcal{M}_P$, *les ensembles respectivement des métavariables d'expression, d'ensembles et de prédicats*
- $\mathcal{M}$ *l'ensemble des métavariables tel que* $\mathcal{M} = \mathcal{M}_E \cup \mathcal{M}_S \cup \mathcal{M}_P$
- $\mathcal{F}$ *l'ensemble des couples associant une variable liée par* $\{\dot{V} \mid \dot{P}\}$, $[\dot{V} :\dot{=} \dot{E}]\dot{E}$, $[\dot{V} :\dot{=} \dot{E}]\dot{P}$ *ou* $\dot{\forall}\, \dot{V}.\dot{P}$ *à une variable fraîche.* $\mathcal{F}$ *peut être considéré comme une fonction partielle.*
- $\mathcal{B}$ *l'ensemble des variables fraîches liées tel que* $\mathcal{B} = \mathsf{ran}(\mathcal{F})$
- $\mathcal{N}$ *l'ensemble des hypothèses de non-liberté de la règle à conserver dans la formule réifiée*
- $\mathcal{R}$ *l'ensemble des variables liées par la garde* blvar
- $[\![.]\!]_R$ *la fonction de réification d'une règle*
- $[\![.]\!]_A^{\mathcal{M},\mathcal{F},\mathcal{N},\mathcal{R}}$ *la fonction de réification d'un antécédent*
- $[\![.]\!]_X^{\mathcal{M},\mathcal{F}}$ *où* $X \in \{V, E, S, P, C\}$, *les fonctions de réification pour les catégories syntaxiques des variables, des expressions, des ensembles, des prédicats et des conséquents.*

Notation 3

On notera pour simplifier γ *pour* $\mathcal{M}, \mathcal{F}$ *et* γ_A *pour* $\mathcal{M}, \mathcal{F}, \mathcal{N}, \mathcal{R}$. *Ainsi, par exemple, les trois notations suivantes sont équivalentes :*
$[\![.]\!]_A^{\gamma_A}$, $[\![.]\!]_A^{\gamma,\mathcal{N},\mathcal{R}}$ *et* $[\![.]\!]_A^{\mathcal{M},\mathcal{F},\mathcal{N},\mathcal{R}}$.

Notation 4

Soit $\forall_{x \in U} x : T$ *la notation pour* $\forall x_1, \ldots, x_n : T$ *où* $U = \{x_1, \ldots, x_n\}$, *qui signifie que*

$x_1, \ldots, x_n$ sont de type T.

Notation 5

On notera $\forall x \in \mathcal{M}, \mathcal{B}, \mathcal{N}.T$ la formule suivante :

$$\underbrace{\forall_{x \in \mathcal{M}_E} x : \dot{E}.\forall_{x \in \mathcal{M}_S} x : \dot{S}.\forall_{x \in \mathcal{M}_P} x : \dot{P}.\forall_{x \in \mathcal{B}} x : \dot{V}.N_1 \rightarrow \ \ldots \ \rightarrow N_n \rightarrow}_{\mathsf{Coq}} \underbrace{T}_{\dot{P}}$$

où $\mathcal{N} = \{N_1, \ldots, N_n\}$, N_i est un antécédent de non-liberté de variable réifié et T est un terme (de type $\dot{P}$). Dans cette notation, il y a tout d'abord les $\forall$ Coq permettant de lier les métavariables BCoq (d'expressions $\dot{E}$, d'ensembles $\dot{S}$ et de prédicats $\dot{P}$) ainsi que les variables qui sont liées dans la formule BCoq ($\dot{V}$). Ensuite, nous retrouvons dans les hypothèses Coq les hypothèses de non-liberté de variables BCoq (N_i). Enfin, il y a le terme BCoq T qui est un prédicat de type $\dot{P}$.

Les fonctions précédentes de réification calcule non seulement un terme (une variable, un ensemble, une expression ou un prédicat selon la nature de l'entrée) mais peut aussi mettre à jour certains de ses arguments. La fonction $[\![.]\!]_R$ réifie une règle en utilisant les fonctions $[\![.]\!]_A$ pour les antécédents et $[\![.]\!]_C$ pour le conséquent. La réification commence par le conséquent afin de calculer l'ensemble $\mathcal{F}$ avant de réifier les antécédents. Cela est nécessaire pour calculer les variables liées qui peuvent intervenir dans les antécédents (par exemple dans une hypothèse de non liberté de variable). Puis, la réification introduit tous les $\forall$ superficiels (c'est-à-dire les $\forall$ Coq) qui lient les métavariables et les variables liées de la règle. Lorsqu'il n'y a pas d'antécédent alors les ensembles $\mathcal{N}$ et $\mathcal{R}$ sont vides.

$$[\![A_1 \Rightarrow C_1]\!]_R = \forall x \in \mathcal{M}', \mathcal{B}, \mathcal{N}.A' \dot{\Rightarrow} C',$$
$$\text{avec} \begin{cases} [\![C_1]\!]_C^{\emptyset,\emptyset} = (C', \mathcal{M}, \mathcal{F}), \\ [\![A_1]\!]_A^{\mathcal{M},\mathcal{F},\emptyset,\emptyset} = (A', \mathcal{M}', \mathcal{F}', \mathcal{N}, \mathcal{R}) \text{ et} \\ \mathcal{B} = \mathsf{ran}(\mathcal{F}') \end{cases}$$
$$[\![C_1]\!]_R = \forall x \in \mathcal{M}, \mathcal{B}.C', \text{ avec } [\![C_1]\!]_C^{\emptyset,\emptyset} = (C', \mathcal{M}, \mathcal{F}) \text{ et } \mathcal{B} = \mathsf{ran}(\mathcal{F})$$

La fonction $[\![.]\!]_C$ permet de transformer le symbole « == » d'une règle de réécriture par une égalité ou une équivalence selon que celle-ci concerne des expressions ou des prédicats. Lorsque la règle est une règle de déduction, le conséquent est réifié directement en prédicat.

$[\![P_1]\!]^\gamma_C = [\![P_1]\!]^\gamma_P$

$[\![E_1 == E_2]\!]^\gamma_C = [\![E_1 == E_2]\!]^\gamma_P$

$$[\![P_1 == P_2]\!]^\gamma_C = (P_1' \Leftrightarrow P_2', \gamma_2) \text{ avec } \begin{cases} [\![P_1]\!]^\gamma_P = (P_1', \gamma_1) \text{ et} \\ [\![P_2]\!]^{\gamma_1}_P = (P_2', \gamma_2)(P_2', \gamma_2) \end{cases}$$

La fonction $[\![.]\!]_A$ traite des antécédents. Les antécédents sur la non liberté des variables, $I\backslash E$ et $I\backslash P$, sont collectés dans l'ensemble $\mathcal{N}$ si les variables concernées n'apparaissent pas dans une garde $\mathsf{blvar}(I)$, c'est-à-dire si I n'est pas dans l'ensemble $\mathcal{R}$. Dans ce cas, ils sont réifiés en métaprédicats (c'est-à-dire en prédicats Coq) puisqu'ils pourront être utilisés lors des preuves. Si I est dans $\mathcal{R}$ alors l'hypothèse de non-liberté de variable est réifiée dans le prédicat $\top$. Il en est de même pour $\mathsf{blvar}(I)$. $\mathsf{blvar}(I)$ et $I\backslash E$, lorsqu'ils sont associés, sont utiles uniquement pour la vérification concernant la capture de variable, vérification qui est faite avant que la règle ne soit réifiée. $\mathsf{binhyp}(P)$ est réifié en un prédicat P' qui est simplement la réification de P. Une fois que tous les antécédents sont réifiés, une simplification est faite afin de supprimer tous les potentiels $\top$. Dans la définition ci-dessous, nous supposons que les antécédents $\mathsf{blvar}(I)$ sont placés avant les antécédents $I\backslash E$ ou $I\backslash P$ associés.

$$[\![P_1]\!]^{\gamma_0,\mathcal{N},\mathcal{R}}_A = (P_1', \gamma_1, \mathcal{N}, \mathcal{R}) \text{ avec } [\![P_1]\!]^{\gamma_0}_P = (P_1', \gamma_1)$$

$$[\![I_1 \setminus P_1]\!]^{\gamma_0,\mathcal{N},\mathcal{R}}_A = \begin{cases} (\top, \gamma_0, \mathcal{N}, \mathcal{R}) \text{ si } I_1 \in \mathcal{R} \\ (\top, \gamma_2, \mathcal{N} \cup \{I' \mathbin{\dot\setminus} P'\}, \mathcal{R}) \text{ sinon} \\ \quad \text{avec} \quad [\![I_1]\!]^{\gamma_0}_V = (I', \gamma_1) \text{ et} \\ \qquad\quad [\![P_1]\!]^{\gamma_1}_P = (P', \gamma_2) \end{cases}$$

$$[\![I_1 \setminus P_1]\!]^{\gamma_0,\mathcal{N},\mathcal{R}}_A = \begin{cases} (\top, \gamma_0, \mathcal{N}, \mathcal{R}) \text{ si } I_1 \in \mathcal{R} \\ (\top, \gamma_2, \mathcal{N} \cup \{I' \mathbin{\dot\setminus} E'\}, \mathcal{R}) \text{ sinon} \\ \quad \text{avec} \quad [\![I_1]\!]^{\gamma_0}_V = (I', \gamma_1) \text{ et} \\ \qquad\quad [\![E_1]\!]^{\gamma_1}_E = (E', \gamma_2) \end{cases}$$

$$[\![\mathsf{binhyp}(P_1)]\!]^{\gamma_0,\mathcal{N},\mathcal{R}}_A = (P_1', \gamma_1, \mathcal{N}, \mathcal{R}) \text{ avec } [\![P_1]\!]^{\gamma_0}_P = (P_1', \gamma_1)$$

$$[\![\mathsf{blvar}(I_1)]\!]^{\gamma,\mathcal{N},\mathcal{R}}_A = (\top, \gamma, \mathcal{N}, \mathcal{R} \cup \{I_1\})$$

$$[\![A_1 \wedge A_2]\!]^{\gamma_{A_0}}_A = (A_1' \mathbin{\dot\wedge} A_2', \gamma_{A_2}) \text{ avec } \begin{cases} [\![A_1]\!]^{\gamma_{A_0}}_A = (A_1', \gamma_{A_1}) \text{ et} \\ [\![A_2]\!]^{\gamma_{A_1}}_A = (A_2', \gamma_{A_2}) \end{cases}$$

$[\![.]\!]_P$ réifie les prédicats ; les connecteurs logiques, les quantificateurs universels, les égalités, l'appartenance et les substituions sont traduits par leur correspondants BCoq. Les

variables des lieurs sont réifiées en une variable fraîche par une fonction dédiée de réification $[\![.]\!]_b$ (voir sa définition plus loin) qui permet d'ajouter les variables liées dans $\mathcal{F}$. Les identificateurs sont ajoutés à l'ensemble des métavariables de prédicats $\mathcal{M}_P$.

$$[\![P_1 \wedge P_2]\!]_P^{\gamma_0} = (P_1' \dot{\wedge} P_2', \gamma_2) \text{ avec } \begin{cases} [\![P_1]\!]_P^{\gamma_0} = (P_1', \gamma_1) \text{ et} \\ [\![P_2]\!]_P^{\gamma_1} = (P_2', \gamma_2) \end{cases}$$

$$[\![P_1 \Rightarrow P_2]\!]_P^{\gamma_0} = (P_1' \dot{\Rightarrow} P_2', \gamma_2) \text{ avec } \begin{cases} [\![P_1]\!]_P^{\gamma_0} = (P_1', \gamma_1) \text{ et} \\ [\![P_2]\!]_P^{\gamma_1} = (P_2', \gamma_2) \end{cases}$$

$$[\![\neg P_1]\!]_P^{\gamma_0} = (\dot{\neg}\, P_1', \gamma_1) \text{ avec } [\![P_1]\!]_P^{\gamma_0} = (P_1', \gamma_1)$$

$$[\![\forall V_1.P_1]\!]_P^{\gamma_0} = (\dot{\forall}\, V_2.P_1', \gamma_2) \text{ avec } \begin{cases} [\![V_1]\!]_b^{\gamma_0} = (V_2, \gamma_1) \text{ et} \\ [\![P_1]\!]_P^{\gamma_1} = (P_1', \gamma_2) \end{cases}$$

$$[\![[V_1 := E_1]P_1]\!]_P^{\gamma_0} = ([V_2 \doteq E_1']P_1', \gamma_3) \text{ avec } \begin{cases} [\![V_1]\!]_b^{\gamma_0} = (V_2, \gamma_1), \\ [\![E_1]\!]_E^{\gamma_1} = (E_1', \gamma_2) \text{ et} \\ [\![P_1]\!]_P^{\gamma_2} = (P_1', \gamma_3) \end{cases}$$

$$[\![E_1 = E_2]\!]_P^{\gamma_0} = (E_1' \doteq E_2', \gamma_2) \text{ avec } \begin{cases} [\![E_1]\!]_E^{\gamma_0} = (E_1', \gamma_1) \text{ et} \\ [\![E_2]\!]_E^{\gamma_1} = (E_2', \gamma_2) \end{cases}$$

$$[\![E_1 \in S_1]\!]_P^{\gamma_0} = (E_1' \dot{\in} S_1', \gamma_2) \text{ avec } \begin{cases} [\![E_1]\!]_E^{\gamma_0} = (E_1', \gamma_1) \text{ et} \\ [\![S_1]\!]_S^{\gamma_1} = (S_1', \gamma_2) \end{cases}$$

$$[\![I_1]\!]_P^{\mathcal{M},\mathcal{F}} = (I_1, \mathcal{M}_E \cup \mathcal{M}_S \cup (\mathcal{M}_P \cup \{I_1\}), \mathcal{F})$$

La fonction $[\![.]\!]_S$ réifie les ensembles. S'il s'agit d'un identificateur alors il est ajouté dans l'ensemble des métavariables d'ensemble $\mathcal{M}_S$. Pour les ensembles en compréhension, la variable liée est ajoutée dans $\mathcal{F}$ par la fonction de réification $[\![.]\!]_b$.

$$[\![S_1 \times S_2]\!]_S^{\gamma_0} = (S_1' \dot{\times} S_2', \gamma_2) \text{ avec } \begin{cases} [\![S_1]\!]_S^{\gamma_0} = (S_1', \gamma_1) \text{ et} \\ [\![S_2]\!]_S^{\gamma_1} = (S_2', \gamma_2) \end{cases}$$

$$[\![\mathbb{P}(S_1)]\!]_S^{\gamma_0} = (\dot{\mathbb{P}}\,(S_1'), \gamma_1) \text{ avec } [\![S_1]\!]_S^{\gamma_0} = (S_1', \gamma_1)$$

$$[\![\{V_1|P_1\}]\!]_S^{\gamma_0} = (\{V_2 \mid P_1'\}, \gamma_2) \text{ avec } \begin{cases} [\![V_1]\!]_b^{\gamma_0} = (V_2, \gamma_1) \text{ et} \\ [\![P_1]\!]_P^{\gamma_1} = (P_1', \gamma_2) \end{cases}$$

$$[\![I_1]\!]_S^{\mathcal{M},\mathcal{F}} = (I_1, \mathcal{M}_E \cup (\mathcal{M}_S \cup \{I_1\}) \cup \mathcal{M}_P, \mathcal{F})$$

La fonction $[\![.]\!]_E$ réifie les expressions. Les métavariables d'expression sont traitées par la fonction de réification des variables.

$[\![V_1]\!]^\gamma_E = [\![V_1]\!]^\gamma_V$

$[\![[V_1 := E_1]E_2]\!]^{\gamma_0}_E = ([V_2 :\doteq E'_1]E'_2, \gamma_3)$ avec $\begin{cases} [\![V_1]\!]^{\gamma_0}_b = (V_2, \gamma_1), \\ [\![E_1]\!]^{\gamma_1}_E = (E'_1, \gamma_2) \text{ et} \\ [\![E_2]\!]^{\gamma_2}_E = (E'_2, \gamma_3) \end{cases}$

$[\![E_1 \mapsto E_2]\!]^{\gamma_0}_E = (E'_1 \dot{\mapsto} E'_2, \gamma_2)$ avec $\begin{cases} [\![E_1]\!]^{\gamma_0}_E = (E'_1, \gamma_1) \text{ et} \\ [\![E_2]\!]^{\gamma_1}_E = (E'_2, \gamma_2) \end{cases}$

$[\![S_1]\!]^\gamma_E = [\![S_1]\!]^\gamma_S$

Les fonctions $[\![.]\!]_b$ et $[\![.]\!]_V$ réifient respectivement les variables des lieurs (les occurrences soulignées dans $\{\underline{I_1}|P\}$,$[\underline{I_1} := E]E$, $[\underline{I_1} := E]P$ ou $\forall \underline{I_1}.P$) et les variables. La première fonction génère des variables fraîches et met à jour l'ensemble $\mathcal{F}$ (avec une nouvelle association entre la variable du lieur et son correspondant dans la formule réifiée). La seconde permet, dans le cas d'une variable liée, de récupérer la variable fraîche qui lui est associée dans $\mathcal{F}$. Si la variable est libre, alors c'est une métavariable d'expression venant de $[\![V_1]\!]^\gamma_E$.

$[\![I_1]\!]^{\mathcal{M},\mathcal{F}}_b = \begin{cases} (\mathcal{F}(I_1), \mathcal{M}, \mathcal{F}) \text{ si } I_1 \in \mathsf{dom}(\mathcal{F}) \\ (I_2, \mathcal{M}, \mathcal{F} \cup \{(I_1, I_2)\}), \text{ sinon où } I_2 \notin \mathsf{dom}(\mathcal{F}) \cup \mathsf{ran}(\mathcal{F}) \end{cases}$

$[\![V_1 \mapsto V_2]\!]^{\gamma_0}_b = (V'_1 \dot{\mapsto} V'_2, \gamma_2)$ avec $\begin{cases} [\![V_1]\!]^{\gamma_0}_b = (V'_1, \gamma_1) \text{ et} \\ [\![V_2]\!]^{\gamma_1}_b = (V'_2, \gamma_2) \end{cases}$

$[\![I_1]\!]^{\mathcal{M},\mathcal{F}}_V = \begin{cases} (\mathcal{F}(I_1), \mathcal{M}, \mathcal{F}), \text{ si } I_1 \in \mathsf{dom}(\mathcal{F}) \\ (I_1, (\mathcal{M}_E \cup \{I_1\}) \cup \mathcal{M}_S \cup \mathcal{M}_P, \mathcal{F}), \text{ sinon} \end{cases}$

$[\![V_1 \mapsto V_2]\!]^{\gamma_0}_V = (V'_1 \mapsto V'_2, \gamma_2)$ avec $\begin{cases} [\![V_1]\!]^{\gamma_0}_V = (V'_1, \gamma_1) \text{ et} \\ [\![V_2]\!]^{\gamma_1}_V = (V'_2, \gamma_2) \end{cases}$

Nous illustrons ci-dessous le processus de réification sur les deux règles précédentes.

Exemple 7 (Réification de 4649)
Soit $a \subseteq t \wedge b \in u \Rightarrow a \times \{b\} \in t \nrightarrow u$ la règle à réifier. La règle est de la forme $A_1 \Rightarrow C_1$ donc le conséquent est d'abord réifié.

1. *Réification du conséquent*
   $[\![a \times \{b\} \in t \nrightarrow u]\!]^{\emptyset,\emptyset}_C = [\![a \times \{b\} \in t \nrightarrow u]\!]^{\emptyset,\emptyset}_P$
   On a $[\![a \times \{b\}]\!]^{\emptyset,\emptyset}_E = (a \dot{\times} \dot{\{}\ b\ \dot{\}}, \{b\}_{\mathcal{M}_E} \cup \{a\}_{\mathcal{M}_S}, \emptyset)$
   et $[\![t \nrightarrow u]\!]^{\{b\}_{\mathcal{M}_E} \cup \{a\}_{\mathcal{M}_S}, \emptyset}_S = (t \dot{\nrightarrow} u, \{b\}_{\mathcal{M}_E} \cup \{a, t, u\}_{\mathcal{M}_S}, \emptyset)$
   Donc $[\![a \times \{b\} \in t \nrightarrow u]\!]^{\emptyset,\emptyset}_P = (a \dot{\times} \dot{\{}\ b\ \dot{\}} \dot{\in} t \dot{\nrightarrow} u, \{b\}_{\mathcal{M}_E} \cup \{a, t, u\}_{\mathcal{M}_S}, \emptyset)$.

2. *Réification des antécédents*

   Nous allons maintenant réifier les antécédents avec les ensembles obtenus c'est-à-dire nous allons calculer $[\![a \subseteq t \wedge b \in u]\!]_A^{\{b\}_{\mathcal{M}_E} \cup \{a,t,u\}_{\mathcal{M}_S},\emptyset,\emptyset,\emptyset}$

   (a) *Réification du premier antécédent :*
   $[\![a \subseteq t]\!]_A^{\{b\}_{\mathcal{M}_E} \cup \{a,t,u\}_{\mathcal{M}_S},\emptyset,\emptyset,\emptyset} = (a \dot{\subseteq}\ t, \{b\}_{\mathcal{M}_E} \cup \{a,t,u\}_{\mathcal{M}_S}, \emptyset, \emptyset, \emptyset)$

   (b) *Réification du deuxième antécédent :*
   $[\![b \in u]\!]_A^{\{b\}_{\mathcal{M}_E} \cup \{a,t,u\}_{\mathcal{M}_S},\emptyset,\emptyset,\emptyset} = (b \dot{\in} u, \{b\}_{\mathcal{M}_E} \cup \{a,t,u\}_{\mathcal{M}_S}, \emptyset)$

   La réification des antécédents est donc :
   $[\![a \subseteq t \wedge b \in u]\!]_A^{\{b\}_{\mathcal{M}_E} \cup \{a,t,u\}_{\mathcal{M}_S},\emptyset,\emptyset,\emptyset} =$
   $(a \dot{\subseteq}\ t \dot{\wedge} b \dot{\in} u, \{b\}_{\mathcal{M}_E} \cup \{a,t,u\}_{\mathcal{M}_S}\}, \emptyset, \emptyset, \emptyset)$

La réification de la règle est donc :
$[\![a \subseteq t \wedge b \in u \Rightarrow a \times \{b\} \in t \nrightarrow u]\!]_R \emptyset, \emptyset, \emptyset =$
$\forall x \in \{b\}_{\mathcal{M}_E} \cup \{a,t,u\}_{\mathcal{M}_S}, \emptyset, \emptyset . a \dot{\subseteq}\ t \dot{\wedge} b \dot{\in} u \dot{\Rightarrow} a \dot{\times} \dot{\{}\ b\ \dot{\}} \dot{\in} t \dot{\nrightarrow} u$
Soit $\forall b : \dot{E}, \forall a\ t\ u : \dot{S}, a \dot{\subseteq}\ t \dot{\wedge} b \dot{\in} u \dot{\Rightarrow} a \dot{\times} \dot{\{}\ b\ \dot{\}} \dot{\in} t \dot{\nrightarrow} u.$

Exemple 8 (Réification de SimplifyRelDorXY.2)

Soit

$\mathsf{binhyp}(f \in u \nrightarrow v) \wedge \mathsf{binhyp}(a \in \mathsf{dom}(f)) \wedge$
$\mathsf{blvar}(Q) \wedge (Q \setminus (f \in u \nrightarrow v)) \wedge (Q \setminus (a \in \mathsf{dom}(f))) \Rightarrow \{a\} \lhd f == \{(a \mapsto f(a))\}$

la règle à réifier. La règle est de la forme $A_1 \Rightarrow C_1$ *donc le conséquent est d'abord réifié.*

1. *Réification du conséquent*
   $[\![\{a\} \lhd f == \{(a \mapsto f(a))\}]\!]_C^{\emptyset,\emptyset} = [\![\{a\} \lhd f = \{(a \mapsto f(a))\}]\!]_P^{\emptyset,\emptyset}$
   On a $[\![\{a\} \lhd f = \{(a \mapsto f(a))\}]\!]_P^{\emptyset,\emptyset} = (\{a\} \dot{\lhd} f \doteq \{(a \dot{\mapsto} f(a))\}, \{a\}_{\mathcal{M}_E} \cup \{f\}_{\mathcal{M}_S}, \emptyset)$

2. *Réification des antécédents*

   (a) *Antécédent n° 1 :* $\mathsf{binhyp}(f \in u \nrightarrow v)$
   $[\![f \in u \nrightarrow v]\!]_P^{\{a\}_{\mathcal{M}_E} \cup \{f\}_{\mathcal{M}_S},\emptyset} = (f \dot{\in} u \dot{\nrightarrow} v, \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset)$
   Donc $[\![\mathsf{binhyp}(f \in u \nrightarrow v)]\!]_A^{\{a\}_{\mathcal{M}_E} \cup \{f\}_{\mathcal{M}_S},\emptyset,\emptyset,\emptyset} =$
   $(f \dot{\in} u \dot{\nrightarrow} v, \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \emptyset)$

(b) *Antécédent n° 2 :* $\mathsf{binhyp}(a \in \mathsf{dom}(f))$

$[\![a \in \mathsf{dom}(f)]\!]_P^{\{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset} = (a \dot{\in} \dot{\mathsf{dom}}\,(f), \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset)$

Donc $[\![\mathsf{binhyp}(a \in \mathsf{dom}(f))]\!]_A^{\{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \emptyset} =$

$(a \dot{\in} \dot{\mathsf{dom}}\,(f), \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \emptyset)$

(c) *Antécédent n° 3 :* $\mathsf{blvar}(Q)$

$[\![\mathsf{blvar}(Q)]\!]_A^{\{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \emptyset} = (\top, \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \{Q\})$

(d) *Antécédent n° 4 :* $Q \setminus (f \in u \nrightarrow v)$

$[\![Q \setminus (f \in u \nrightarrow v)]\!]_A^{\{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \{Q\}} = (\top, \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \{Q\})$

car Q appartient à $\mathcal{R}$ qui est égal à $\{Q\}$.

(e) *Antécédent n° 5 :* $Q \setminus (a \in \mathsf{dom}(f))$

$[\![Q \setminus (a \in \mathsf{dom}(f))]\!]_A^{\{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \{Q\}} = (\top, \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \{Q\})$

car Q appartient à $\mathcal{R}$ qui est égal à $\{Q\}$.

(f) *Résultat pour les antécédents :*

$[\![\mathsf{binhyp}(f \in u \nrightarrow v) \wedge \mathsf{binhyp}(a \in \mathsf{dom}(f)) \wedge$

$\mathsf{blvar}(Q) \wedge (Q \setminus (f \in u \nrightarrow v)) \wedge (Q \setminus (a \in \mathsf{dom}(f)))]\!]_A^{\{a\}_{\mathcal{M}_E} \cup \{f\}_{\mathcal{M}_S}, \emptyset, \emptyset, \emptyset} =$

$(f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f) \dot{\wedge} \top \dot{\wedge} \top \dot{\wedge} \top, \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \{Q\}) =$

$(f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f), \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset, \{Q\})$

Le résultat de la réification de la règle est donc :

$\forall x \in \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset.$

$f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f) \dot{\Rightarrow} \{a\} \dot{\lhd} f \doteq \{(a \dot{\mapsto} f(a))\}$ *c'est-à-dire*

$\forall a :\dot{E} .\forall f\ u\ v :\dot{S} .f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f) \dot{\Rightarrow} \{a\} \dot{\lhd} f \doteq \{(a \dot{\mapsto} f(a))\}$

2.6 Vérification de la non capture de variable

Comme nous l'avons vu dans la section 2.3, l'Atelier B ne vérifie pas le contexte d'application lorsqu'une règle de réécriture est appliquée : il l'applique de manière purement syntaxique. Par conséquent, lorsqu'une règle de réécriture est appliquée sous des quantificateurs et met en jeu des variables liées, des captures de variable peuvent apparaître et conduire à des incohérences (voir l'exemple de règle n° 1 page 45).

Pour éviter les captures de variable, une première solution est d'interdire la réécriture sous les quantificateurs lorsque des variables liées sont mises en jeu. Autrement dit, toutes les variables liées du but doivent être non libres dans le conséquent de la règle. Plus formellement, étant donnée une règle de réécriture de la forme $A \Rightarrow C$, où A la conjonction des antécédents (dont les gardes), et C le conséquent de la forme $E == F$ où E et F sont deux expressions, cela correspond au critère syntaxique suivant :

$$\mathsf{BRV} \setminus C \tag{2.1}$$

où BRV représente l'ensemble des variables liées au point de réécriture (voir notation 1 page 40).

L'équivalent de ce critère en syntaxe de l'Atelier B est $\mathsf{blvar}(Q) \wedge Q \setminus C$. En appliquant ce critère, la précédente règle de réécriture $\mathsf{binhyp}(x = a) \Rightarrow x == a$ est rejetée puisque les variables x et a ne sont pas protégées. Pour être correcte du point de vue de la non capture de variable, la règle doit être de la forme $\mathsf{binhyp}(x = a) \wedge \mathsf{blvar}(Q) \wedge Q \setminus \{x, a\} \Rightarrow x == a$.

Cependant, ce critère est un peu trop restrictif puisqu'il rejette aussi des règles correctes comme $s \cap t == t \cap s$ qui ne poseront pas de problème de capture de variable.

Pour accepter ce type de règle, une seconde solution consiste à autoriser la réécriture sous les quantificateurs seulement si les variables liées impliquées n'apparaissent pas dans les antécédents. Ce nouveau critère syntaxique peut être formulé ainsi :

$$\mathsf{BRV} \setminus A \tag{2.2}$$

L'équivalent de ce critère en syntaxe de l'Atelier B est $\mathsf{blvar}(Q) \wedge Q \setminus A$. Avec ce critère, $\mathsf{binhyp}(x = a) \wedge \mathsf{blvar}(Q) \wedge Q \setminus \{x, a\} \Rightarrow x == a$ et $s \cap t == t \cap s$ sont toutes deux acceptées. Les critères (2.1) et (2.2) sont complémentaires, ils ont été mis au point dans [47].

Nous voulons remplacer ces deux critères par un seul critère moins contraignant que chacun d'entre eux.

Si les variables liées du but n'apparaissent que dans E et/ou F alors il n'y a pas de problème de capture puisque BRV sera inchangé par l'application de la règle. De même si

elles n'apparaissent que dans A, BRV sera également inchangé. Le cas où il peut y avoir un problème est donc lorsqu'une variable liée du but apparaît à la fois dans les antécédents (A) et dans le conséquent (E ou F). Cela peut être formulé ainsi par ce critère :

$$\mathsf{BRV} \setminus (FV(A) \cap FV(C)) \tag{2.3}$$

où FV calcule l'ensemble des variables libres. Avec ce nouveau critère, la règle corrigée et $s \cap t == t \cap s$ sont toutes les deux acceptées.

Ce critère est moins fort que les deux autres car on a :

$$\mathsf{BRV} \setminus (FV(A) \cap FV(C)) \Rightarrow \mathsf{BRV} \setminus C \vee \mathsf{BRV} \setminus A$$

Remarque Dans le cas où le critère n'est pas vérifié, il est facile de proposer une correction à l'utilisateur tout simplement en calculant l'ensemble $FV(A) \cap FV(C)$ et en ajoutant les gardes $\mathsf{blvar}(Q) \wedge (Q \setminus (FV(A) \cap FV(C)))$.

Si nous reprenons les exemples de règles introduits en section 2.3, leur vérification est effectuée de la manière suivante :

Exemple 9 (Vérification de 4649)

$a \subseteq t \wedge b \in u \Rightarrow a \times \{b\} \in t \pfun u$

Cette règle est une règle de déduction donc elle n'est pas concernée par cette vérification.

Exemple 10 (Vérification de SimplifyRelDorXY.2)

$\mathsf{binhyp}(f \in u \pfun v) \wedge \mathsf{binhyp}(a \in \mathsf{dom}(f)) \wedge$

$\mathsf{blvar}(Q) \wedge (Q \setminus (f \in u \pfun v)) \wedge (Q \setminus (a \in \mathsf{dom}(f))) \Rightarrow \{a\} \domres f == \{(a \mapsto f(a))\}$

Cette règle est une règle de réécriture, donc nous devons vérifier que les variables sont correctement protégées. Comme $FV(C) = \{a, f\}$ *et* $FV(A) = \{a, f, u, v\}$ *donc le critère (2.3) à vérifier est* $\mathsf{BRV} \setminus (\{a, f\})$. *Il est vérifié ici grâce à* $\mathsf{blvar}(Q) \wedge (Q \setminus (a \in \mathsf{dom}(f)))$

2.7 Vérification du typage

Comme nous l'avons vu dans la section 2.3, il est nécessaire de vérifier qu'une règle est bien typée afin d'éviter d'introduire des incohérences comme par exemple le paradoxe de Russell. Pour cela, un système de types est introduit dans le B-Book.

Pour vérifier qu'une formule est bien typée avec les règles du B-Book, il faut que tous les types des variables soient explicités en hypothèse. Cependant, les règles peuvent contenir des métavariables dont le type est implicite. Par exemple, dans la règle $a \cup b = b \cup a$, les types de a et b ne sont pas connus et le système de types du B-Book ne permet pas d'inférer les types de ces métavariables. Il faut donc inférer préalablement un type pour toutes les métavariables de la règle (voir 2.7.2). Puis le lemme de vérification du typage, enrichi des types des métavariables, est généré (voir 2.7.3) pour être ensuite vérifié avec les règles de typage du B-Book.

Le typage de B est introduit avec la théorie des ensembles dans le B-Book. Il permet donc de typer toutes les expressions et ensembles B. Néanmoins, il ne traite pas des métavariables de prédicat alors que les règles de l'Atelier B peuvent en contenir. Nous avons donc rajouté des règles de typage permettant de traiter ces cas (voir 2.7.1.1). Une autre limitation du typage de B concerne le typage de l'ensemble vide qui n'est pas complet. Par exemple, on ne peut pas vérifier le typage de la propriété du B-Book $a \subseteq s \vdash \emptyset - a = \emptyset$ avec ces règles. Pour régler ce problème, nous avons ajouté d'autres règles permettant de compléter le typage de l'ensemble vide (voir 2.7.1.2).

2.7.1 Extension du système de types de B

2.7.1.1 Typage des métavariables de prédicat

Les règles du système de types de B (voir la sous-section 1.1.3 du chapitre 1) ne traitent pas du cas des métavariables de prédicats puisque ces métavariables ne sont pas définies dans le B-Book. Or, elles peuvent apparaître dans les règles ajoutées de l'Atelier B. Pour vérifier aussi le typage de ces règles dans BCARe, nous avons ajouté la règle suivante, dans laquelle $\mathsf{given_p}$ permet d'identifier une métavariable de prédicat et E est l'ensemble des hypothèses de typage :

$$\frac{\mathsf{given_p}(P) \text{ apparaît dans } E}{E \vdash_\tau \mathsf{check}(P)} \text{ T0}$$

Lors de la génération du lemme de vérification, il faut donc qu'un $\mathsf{given_p}$ soit ajouté en hypothèse pour chaque métavariable de prédicat présente dans la formule à vérifier.

Exemple 11

Soit la formule $P \wedge Q \Rightarrow Q \wedge P$. P et Q sont des métavariables de prédicat, le contexte de typage est donc $E = \{\mathsf{given_p}(P), \mathsf{given_p}(Q)\}$. Voici l'arbre qui permet de vérifier le typage :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{E \vdash_\tau \mathsf{check}(P)}\,T0 \quad \dfrac{}{E \vdash_\tau \mathsf{check}(Q)}\,T0}{E \vdash_\tau \mathsf{check}(P \wedge Q)}\,T1 \quad \dfrac{\dfrac{}{E \vdash_\tau \mathsf{check}(Q)}\,T0 \quad \dfrac{}{E \vdash_\tau \mathsf{check}(P)}\,T0}{E \vdash_\tau \mathsf{check}(Q \wedge P)}\,T1}{E \vdash_\tau \mathsf{check}(P \wedge Q \Rightarrow Q \wedge P)}\,T2$$

2.7.1.2 Typage de l'ensemble vide

Le système de types de B est incomplet en ce qui concerne le typage de l'ensemble vide. En effet, il ne permet pas de vérifier le typage de la formule $a \subseteq s \vdash \emptyset - a = \emptyset$.

Essayons de vérifier le typage de cette propriété $a \subseteq s \vdash \emptyset - a = \emptyset$ avec les règles exposées dans l'annexe A.2. s étant le plus grand ensemble contenant a, s est le type support de a. Il faut donc rajouter dans les hypothèses de typage $\mathsf{given}(s)$. Soit E le contexte contenant les hypothèses $a \subseteq s$ et $\mathsf{given}(s)$, voici le début de l'arbre correspondant à la vérification du type :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathsf{super}(\emptyset)} \quad \dfrac{\dots}{E \vdash_\tau \mathsf{super}(a) \equiv \mathsf{super}(\emptyset)}}{E \vdash_\tau \mathsf{super}(\emptyset - a) \equiv \mathsf{super}(\emptyset)}\,\mathrm{T24}}{E \vdash_\tau \mathbb{P}(\mathsf{super}(\emptyset - a)) \equiv \mathbb{P}(\mathsf{super}(\emptyset))}\,\mathrm{T19}}{E \vdash_\tau \mathbb{P}(\mathsf{super}(\emptyset - a)) \equiv type(\emptyset)}\,\mathrm{T12}}{E \vdash_\tau type(\emptyset - a) \equiv type(\emptyset)}\,\mathrm{T12}}{E \vdash_\tau check(\emptyset - a = \emptyset)}\,\mathrm{T7}$$

La branche gauche de l'arbre ne peut se démontrer avec les règles du B-Book puisqu'aucune règle n'est applicable.

Essayons maintenant de vérifier le type de la formule

$$x \subseteq s, y \subseteq t, z \subseteq u, f \in \mathbb{P}(s \times \mathbb{P}(t \times u)) \vdash f = \{(x, \{(y, z)\})\} \Rightarrow \mathsf{dom}(f(x)) = \{y\}$$

s, t et u sont des types support donc il faut rajouter dans les hypothèses de typage $\mathsf{given}(s)$, $\mathsf{given}(t)$ et $\mathsf{given}(u)$. Soit E le contexte contenant les hypothèses $f \in \mathbb{P}(s \times \mathbb{P}(t \times u))$, $x \subseteq s$, $y \subseteq t$, $z \subseteq u$ et les given précédents. Voici l'arbre correspondant à la vérification du type de $\mathsf{dom}(f(x)) = \{y\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{E \vdash_\tau \mathsf{super}(s) \equiv \mathsf{super}(s)} \quad \dfrac{E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathbb{P}(\mathsf{super}(t \times u))}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathsf{super}(\mathbb{P}(t \times u))}\,\text{T15}}{E \vdash_\tau \mathsf{super}(s) \times \mathsf{super}(\emptyset) \equiv \mathsf{super}(s) \times \mathsf{super}(\mathbb{P}(t \times u))}\,\text{T20}}{E \vdash_\tau \mathsf{super}(s) \times \mathsf{super}(\emptyset) \equiv \mathsf{super}(s \times \mathbb{P}(t \times u))}\,\text{T14}}{E \vdash_\tau \mathbb{P}(\mathsf{super}(s) \times \mathsf{super}(\emptyset)) \equiv \mathbb{P}(\mathsf{super}(s \times \mathbb{P}(t \times u)))}\,\text{T19}}{E \vdash_\tau \mathbb{P}(\mathsf{super}(s) \times \mathsf{super}(\emptyset)) \equiv \mathsf{super}(\mathbb{P}(s \times \mathbb{P}(t \times u)))}\,\text{T15}}{E \vdash_\tau \mathsf{super}(s) \times \emptyset \equiv \mathsf{super}(f)}\,\text{T13}}{E \vdash_\tau \mathsf{super}(s) \equiv \mathsf{super}(\mathsf{dom}(f))}\,\text{T29}}{E \vdash_\tau type(x) \equiv \mathsf{super}(\mathsf{dom}(f))}\,\text{T9}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{E \vdash_\tau \mathsf{super}(f(x)) \equiv \mathsf{super}(\{y\}) \times \mathsf{super}(\emptyset)}{E \vdash_\tau \mathsf{super}(\mathsf{dom}(f(x))) \equiv \mathsf{super}(\{y\})}\,\text{T14}}{E \vdash_\tau \mathbb{P}(\mathsf{super}(\mathsf{dom}(f(x)))) \equiv \mathbb{P}(\mathsf{super}(\{y\}))}\,\text{T19}}{E \vdash_\tau \mathbb{P}(\mathsf{super}(\mathsf{dom}(f(x)))) \equiv type(\{y\})}\,\text{T12}}{E \vdash_\tau type(\mathsf{dom}(f(x)) \equiv type(\{y\})}\,\text{T12}}{E \vdash_\tau check(\mathsf{dom}(f(x)) = \{y\})}\,\text{T7}$$

Il n'y aucune règle qui peut s'appliquer sur $E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathbb{P}(\mathsf{super}(t \times u))$

Pour résoudre ces problèmes, nous avons ajouté les règles suivantes au système de règles existant :

$$\dfrac{}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathsf{super}(\emptyset)}\,\text{T27}_1$$

$$\dfrac{E \vdash_\tau \mathsf{super}(\emptyset) \equiv U \qquad E \vdash_\tau \mathsf{super}(\emptyset) \equiv V}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv (U \times V)}\,\text{T27}_2$$

$$\dfrac{E \vdash_\tau \mathsf{super}(\emptyset) = U}{E \vdash_\tau \mathsf{super}(\emptyset) = \mathbb{P}(U)}\,\text{T27}_3$$

La règle T27_1 permet de terminer la branche gauche de l'arbre de preuve du séquent $E \vdash_\tau check(\emptyset - a = \emptyset)$.

Les règles T27_2 et T27_3 permettent de terminer le deuxième arbre ainsi :

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{given}(t) \text{ est dans } E \quad \dfrac{\mathsf{given}(t) \text{ est dans } E}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv t}\,\text{T27}}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathsf{super}(t)}\,\text{T17} \quad \dfrac{\mathsf{given}(u) \text{ est dans } E \quad \dfrac{\mathsf{given}(u) \text{ est dans } E}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv u}\,\text{T27}}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathsf{super}(u)}\,\text{T17}}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathsf{super}(t) \times \mathsf{super}(u)}\,\text{T27}_2}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathsf{super}(t \times u)}\,\text{T14}}{E \vdash_\tau \mathsf{super}(\emptyset) \equiv \mathbb{P}(\mathsf{super}(t \times u))}\,\text{T27}_3$$

La preuve que ces nouvelles règles n'introduisent pas d'incohérence dans le typage de B n'a pas été faite. Néanmoins, nous n'avons pas trouvé de contre-exemple lors de nos expérimentations. Grâce à ces nouvelles règles, le typage de l'ensemble vide est maintenant

complet puisque tous les constructions de types de B tels $\times$ et $\mathbb{P}$ peuvent être décomposées si besoin.

2.7.2 Inférence de type

Tout le travail sur l'inférence de type est détaillé dans le rapport [47]. La nouveauté, par rapport au travail d'Eric Le Lay, est essentiellement au niveau de la présentation. Pour inférer un type aux métavariables, nous définissons un autre système de types, décrit dans les figures 2.2 et 2.3, où tous les t_i sont des variables de type fraîches, le symbole « $=$ » est utilisé ici pour exprimer les contraintes de type et Γ est le contexte de typage. A partir de ce système de types, un algorithme d'inférence à la Hindley-Milner (avec contraintes) peut être construit. Les différentes règles correspondent au cœur du langage décrit dans la section 2.4 et traitent des règles réifiées (voir figure 2.2). Il y a aussi des règles dédiées aux autres connecteurs logiques et aux opérateurs ensemblistes, mais elles ne sont pas présentées ici dans leur totalité (voir figure 2.3). Ce système d'inférence permet de trouver les types des variables d'expression (liées ou non) et des métavariables d'ensemble et d'expression mais pas des métavariables de prédicat puisque leur type n'est pas pris en compte dans les règles de typage B. Cela est possible, si comme cela est fait dans le B-Book pour le typage, les métavariables d'ensemble et les variables d'expression ne sont pas différenciées et sont plongées dans les variables. Dans la suite, le mot variable représentera soit une variable d'expression (liée ou non) soit une métavariable d'ensemble. Pour les métavariables de prédicat, leur typage sera traité lors de la génération du lemme de typage par les $\mathsf{given_p}$ (voir 2.7.1.1 et 2.7.3).

L'algorithme affecte tout d'abord une variable de type unique à chaque variable libre et toutes ces associations entre variable et variable de type sont conservées dans un contexte de typage noté Γ. L'arbre de type peut alors être construit grâce aux règles du système d'inférence, où des variables de type sont introduites pour chaque type à déterminer. Des contraintes entre variables de type peuvent apparaître durant cette étape. Une fois que l'arbre est complet, l'algorithme essaie de résoudre les contraintes et s'il y arrive, les variables de type de Γ sont mises à jour avec leur instanciation.

Règles pour V

$$\frac{}{x : s, \Gamma \vdash x : t, \{s = t\}}\ var \qquad \frac{\Gamma \vdash x : t_1, q_1 \quad \Gamma \vdash y : t_2, q_2}{\Gamma \vdash x \mapsto y : t, q_1 \cup q_2 \cup \{t = t_1 \dot{\times} t_2\}}\ \dot{\mapsto}_V$$

Règles pour E

$$\frac{\Gamma, x : t_1 \vdash E : t_2, q}{\Gamma \vdash x :\doteq E : t, q \cup \{t = S_\tau, t_1 = t_2\}}\ :\doteq \qquad \frac{\Gamma \vdash x :\doteq E : t_1, q_1 \quad \Gamma \vdash F : t_2, q_2}{\Gamma \vdash [x :\doteq E]F : t, q_1 \cup q_2 \cup \{t_2 = t\}}\ \mathsf{subst}_E$$

$$\frac{\Gamma \vdash E : t_1, q_1 \quad \Gamma \vdash F : t_2, q_2}{\Gamma \vdash E \mapsto F : t, q_1 \cup q_2 \cup \{t = t_1 \dot{\times} t_2\}}\ \dot{\mapsto}_E \qquad \frac{\Gamma \vdash s : t_1, q}{\Gamma \vdash \dot{\mathsf{choice}}\ (s) : t, q \cup \{t_1 = \dot{\mathbb{P}}\ (t)\}}\ \dot{\mathsf{choice}}$$

où S_τ est le type des substitutions.

Règles pour S

$$\frac{\Gamma \vdash S : t_1, q_1 \quad \Gamma \vdash T : t_2, q_2}{\Gamma \vdash S \dot{\times} T : t, q_1 \cup q_2 \cup \{t = \dot{\mathbb{P}}\ (t_3 \dot{\times} t_4), t_1 = \dot{\mathbb{P}}\ (t_3), t_2 = \dot{\mathbb{P}}\ (t_4)\}}\ \dot{\times}$$

$$\frac{\Gamma \vdash E : t_1, q}{\Gamma \vdash \dot{\mathbb{P}}\ (E) : t, q \cup \{t = \dot{\mathbb{P}}\ (\dot{\mathbb{P}}\ (t_2)), t_1 = \dot{\mathbb{P}}\ (t_2)\}}\ \dot{\mathbb{P}}$$

$$\frac{\Gamma, x : t_1 \vdash P : P_\tau, q}{\Gamma \vdash \{x \mid P\} : t, q \cup \{t = \dot{\mathbb{P}}\ (t_1)\}}\ \{\dot{|}\} \qquad \frac{}{\Gamma \vdash \dot{\mathsf{BIG}} : t, \{t = \dot{\mathbb{P}}\ (\dot{\mathsf{BIG}})\}}\ \dot{\mathsf{BIG}}$$

Règles pour P

$$\frac{\Gamma \vdash P : P_\tau, q_1 \quad \Gamma \vdash Q : P_\tau, q_2}{\Gamma \vdash P \dot{\wedge} Q : P_\tau, q_1 \cup q_2}\ \dot{\wedge} \qquad \frac{\Gamma \vdash P : P_\tau, q_1 \quad \Gamma \vdash Q : P_\tau, q_2}{\Gamma \vdash P \dot{\Rightarrow} Q : P_\tau, q_1 \cup q_2}\ \dot{\Rightarrow}$$

$$\frac{\Gamma \vdash P : P_\tau, q}{\Gamma \vdash\ \dot{\neg}\ P : P_\tau, q}\ \dot{\neg} \qquad \frac{\Gamma, x : t \vdash P : P_\tau, q}{\Gamma \vdash\ \dot{\forall}\ x.P : P_\tau, q}\ \dot{\forall}$$

$$\frac{\Gamma \vdash x :\doteq E : t, q_1 \quad \Gamma \vdash P : P_\tau, q_2}{\Gamma \vdash [x :\doteq E]P : P_\tau, q_1 \cup q_2}\ \mathsf{subst}_P \qquad \frac{\Gamma \vdash E : t_1, q_1 \quad \Gamma \vdash F : t_2, q_2}{\Gamma \vdash E \doteq F : P_\tau, q_1 \cup q_2 \cup \{t_1 = t_2\}}\ \doteq$$

$$\frac{\Gamma \vdash E : t_1, q_1 \quad \Gamma \vdash S : t_2, q_2}{\Gamma \vdash E \dot{\in} S : P_\tau, q_1 \cup q_2 \cup \{t_2 = \dot{\mathbb{P}}\ (t_1)\}}\ \dot{\in}$$

où P_τ est le type des prédicats.

FIGURE 2.2 – Règles d'inférence de type

Règles pour les constructions dérivées

$$\frac{\Gamma \vdash S : t_2, q_1 \quad \Gamma \vdash T : t_3, q_2}{\Gamma \vdash S \mathbin{\dot\cup} T : t, q_1 \cup q_2 \cup \{t = \dot{\mathbb{P}}(t_1), t = t_2, t = t_3\}} \; \dot\cup$$

$$\frac{\Gamma \vdash r : t_1, q}{\Gamma \vdash \mathsf{dom}\,(r) : t, q \cup \{t = \dot{\mathbb{P}}(t_2), t_1 = \dot{\mathbb{P}}(t_2 \mathbin{\dot\times} t_3)\}} \; \mathsf{dom}$$

$$\frac{\Gamma \vdash S : t_1, q_1 \quad \Gamma \vdash T : t_2, q_2}{\Gamma \vdash S \mathbin{\dot\to} T : t, q_1 \cup q_2 \cup \{t = \dot{\mathbb{P}}(\dot{\mathbb{P}}(t_3 \mathbin{\dot\times} t_4)), t_1 = \dot{\mathbb{P}}(t_3), t_1 = \dot{\mathbb{P}}(t_4)\}} \; \dot\to$$

$$\ldots$$

où $\dot\to$ est le symbole de la fonction partielle dans B*Coq*.

FIGURE 2.3 – Règles d'inférence de type (suite)

Exemple 12 (Inférence de type de $a \mathbin{\dot\cup} b = b \mathbin{\dot\cup} a$)

Deux métavariables apparaissent dans la formule, l'algorithme affecte donc une variable de type à chacune des métavariables. Le contexte de type Γ *résultant est* $\{(a, \alpha), (b, \alpha')\}$. *L'algorithme peut alors construire l'arbre de typage suivant :*

$$\dfrac{\dfrac{\overline{\Gamma \vdash a : t_3, \{\alpha = t_3\}} \quad \overline{\Gamma \vdash b : t_4, \{\alpha' = t_4\}}}{\Gamma \vdash a \mathbin{\dot\cup} b : t_1, q_1} \quad \dfrac{\overline{\Gamma \vdash b : t_5, \{\alpha' = t_5\}} \quad \overline{\Gamma \vdash a : t_6, \{\alpha = t_6\}}}{\Gamma \vdash b \mathbin{\dot\cup} a : t_2, q_2}}{\Gamma \vdash a \mathbin{\dot\cup} b \doteq b \mathbin{\dot\cup} a : P_\tau, q_P}$$

où $q_1 = \{\alpha = t_3, \alpha' = t_4, t_1 = \dot{\mathbb{P}}(t_7), t_1 = t_3, t_1 = t_4\}$,
$q_2 = \{\alpha' = t_5, \alpha = t_6, t_2 = \dot{\mathbb{P}}(t_8), t_2 = t_5, t_2 = t_6\}$,
$q_P = \{t_1 = t_2, \alpha = t_3, \alpha' = t_4, t_1 = \dot{\mathbb{P}}(t_7), t_1 = t_3, t_1 = t_4, \alpha' = t_5, \alpha = t_6, t_2 = \dot{\mathbb{P}}(t_8),$
$t_2 = t_5, t_2 = t_6\}$.

L'arbre étant clos, l'algorithme cherche une solution aux contraintes et trouve que $\alpha = \dot{\mathbb{P}}(t_7)$ *et* $\alpha' = \dot{\mathbb{P}}(t_7)$. Γ *est finalement mis à jour avec* $(a, \dot{\mathbb{P}}(t_7))$ *et* $(b, \dot{\mathbb{P}}(t_7))$.

La correction de cette inférence de type n'a pas été démontrée. Cependant, l'inférence trouvée par l'algorithme est vérifiée lors de la démonstration du lemme de vérification de typage puisque celui-ci doit être démontré avec les règles de typage du B-Book.

2.7.3 Génération du lemme de vérification

Nous allons maintenant montrer comment générer le lemme de vérification associé au typage. Avant cela, il est nécessaire d'ajouter de nouvelles hypothèses de non liberté de variable signifiant que les variables liées sont non libres dans l'ensemble des variables de type (qui ont été générées par l'inférence de type, comme par exemple t_7 dans l'exemple 12).

Cela est réalisé par l'opérateur suivant :

$$\mathcal{S}_{U,V,\mathcal{N}}(T) = N_1 \to \ldots \to N_n \to T$$

où T est un terme réifié, et pour tout $u \in U$, pour tout $v \in V$ tel que $u \neq v$ et $u \mathbin{\dot{\setminus}} v \notin \mathcal{N}$ (c'est-à-dire $u \mathbin{\dot{\setminus}} v$ n'est pas déjà une hypothèse de non liberté de variable conservée par l'étape de réification), alors il existe $i \in 1 \ldots n$ tel que $N_i = u \mathbin{\dot{\setminus}} v$.

Pour une règle réifiée de la forme $\forall x.x \in \mathcal{M}_E \cup \mathcal{M}_S \cup \mathcal{M}_P, \mathcal{B}, \mathcal{N}.P$ et à partir du contexte de typage résultant Γ, le lemme de typage généré prend la forme suivante :

$$\forall x.x \in \mathcal{M}'_S \cup \mathcal{M}_P, \mathcal{B} \cup \mathcal{M}_S \cup \mathcal{M}_E, \mathcal{N}.\mathcal{S}_{\mathcal{B},\mathcal{B}\cup\mathcal{M}_\Gamma,\mathcal{N}}(G, G_p, H \mathrel{\dot{\vdash}_\tau} \dot{\mathsf{check}}\ (P))$$

où :

- $\mathcal{M}'_S = \mathcal{M}_\Gamma$ où $\mathcal{M}_\Gamma$ est l'ensemble des variables de type de Γ
- G est la réification des types de Γ tel que pour toute variable $t \in \mathcal{M}_\Gamma$, $\dot{\mathsf{given}}\ (t) \in G$ (dans lequel, pour rappel, $\dot{\mathsf{given}}\ (t)$ signifie que t est le type support de lui-même)
- G_p est la réification de l'ensemble des $\dot{\mathsf{given}}_p$ tel que pour tout $p \in \mathcal{M}_P$, $\dot{\mathsf{given}}_p\ (p) \in G_p$ (cf. la section 2.7.1.1)
- H est la réification du contexte de typage Γ tel que pour tout $(v, t) \in \Gamma$ et $v \notin \mathcal{B}$, $v' \in t' \in H$ où $(v' \in t', \_, \_) = [\![v \in t]\!]_P^{\emptyset,\emptyset}$.

Exemple 13 (Vérification du typage de la règle 4649)
Soit $\forall b :\dot{E}, \forall a\ t\ u :\dot{S}, a \mathbin{\dot{\subseteq}} t \mathbin{\dot{\wedge}} b \mathbin{\dot{\in}} u \mathbin{\dot{\Rightarrow}} a \mathbin{\dot{\times}} \{ b \} \mathbin{\dot{\in}} t \mathbin{\dot{\mapsto}} u$ *le résultat de la réification de la règle 4649. On a*

- $\Gamma = \{(a, \dot{\mathbb{P}}\ (t_1)), (b, t_2), (t, \dot{\mathbb{P}}\ (t_1)), (u, \dot{\mathbb{P}}\ (t_2))\}$
- $\mathcal{M}'_S = \mathcal{M}_\Gamma = \{t_1, t_2\}$
- $G = \{\ \dot{\mathsf{given}}\ (t_1), \dot{\mathsf{given}}\ (t_2)\}$

– $\mathcal{M}_P$ *est vide donc* G_p *l'est aussi*

– $H = \{a \subseteq t_1, b \dot{\in} t_2, t \subseteq t_1, u \subseteq t_2\}$

Le lemme généré pour le typage de la règle 4649 est donc :

$\forall x.x \in \{t_1, t_2\}_{\mathcal{M}_S}, \{b, a, t, u\}, \emptyset.$
$\mathcal{S}_{\emptyset,\{t_1,t_2\},\emptyset}(\dot{\mathsf{given}}\ (t_1), \dot{\mathsf{given}}\ (t_2), a \subseteq t_1, b \dot{\in} t_2, t \subseteq t_1, u \subseteq t_2 \dot{\vdash}_\tau$
$\dot{\mathsf{check}}\ (a \dot{\subseteq}\ t \dot{\wedge} b \dot{\in} u \dot{\Rightarrow} a \dot{\times} \{\!\!\dot{\{}\ b\ \dot{\}}\!\!\} \dot{\in} t \dot{\mapsto} u))$

En syntaxe Coq cela donne :

```
Lemma type_4649 :
  ∀ t1 t2 : Ṡ, ∀ b a t u : V̇,
   given (t1), given (t2) a ⊆ t1, b ∈̇ t2, t ⊆ t1, u ⊆ t2 ⊢̇τ check (a ⊆̇ t ∧̇ b ∈̇ u ⇒̇ a ×̇ {̇ b }̇∈̇ t ↦̇ u).
```

Les résultats de l'inférence de type se retrouvent en hypothèse sous forme d'inclusions.

Exemple 14 (Vérification du typage de SimplifyRelDorXY.2)

Soit $\forall x \in \{a\}_{\mathcal{M}_E} \cup \{f, u, v\}_{\mathcal{M}_S}, \emptyset, \emptyset.f \dot{\in} u \dot{\mapsto} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\ (f) \dot{\Rightarrow} \{a\} \dot{\lhd} f \dot{=} \{(a \mapsto f(a))\}$ *le résultat de la réification de SimplifyRelDorXY.2. On a :*

– $\Gamma = \{(u, \dot{\mathbb{P}}\ (t_2)), (v, \dot{\mathbb{P}}\ (t_1)), (f, \dot{\mathbb{P}}\ (t_2 \dot{\times} t_1)), (a, t_2)\}$

– $\mathcal{M}'_S = \mathcal{M}_\Gamma = \{t_1, t_2\}$

– $G = \{\ \dot{\mathsf{given}}\ (t_1), \dot{\mathsf{given}}\ (t_2)\}$

– $\mathcal{M}_P$ *est vide donc* G_p *l'est aussi*

– $H = \{u \dot{\subseteq} (t_2), v \dot{\subseteq} (t_1), f \dot{\in}\ \dot{\mathbb{P}}\ (t_2 \dot{\times} t_1), a \dot{\in} t_2\}$

Le lemme généré pour le typage de SimplifyRelDorXY.2 est donc :

$\forall x.x \in \{t_1, t_2\}_{\mathcal{M}_S}, \{a, f, u, v\}, \emptyset.\mathcal{S}_{\emptyset,\{t_1,t_2\},\emptyset}$
$(\dot{\mathsf{given}}\ (t_1), \dot{\mathsf{given}}\ (t_2), u \dot{\subseteq} (t_2), v \dot{\subseteq} (t_1), f \dot{\in}\ \dot{\mathbb{P}}\ (t_2 \dot{\times} t_1), a \dot{\in} t_2 \dot{\vdash}_\tau$
$\dot{\mathsf{check}}\ (f \dot{\in} u \dot{\mapsto} v \dot{\wedge} a \dot{\in}\ \dot{\mathsf{dom}}\ (f) \dot{\Rightarrow} \{a\} \dot{\lhd} f \dot{=} \{(a \mapsto f(a))\}))$

En syntaxe Coq cela donne :

```
Lemma type_SimplifyRelDorXY_2 :
  ∀ t1 t2 : Ṡ, ∀ a f u v : V̇,
  given (t1), given (t2), u ∈̇ ℙ̇ (t2), v ∈̇ ℙ̇ (t1), f ∈̇ ℙ̇ (t2 ×̇ t1),
   a ∈̇ t2 ⊢̇τ check (f ∈̇ u ↦̇ v ∧̇ a ∈̇ dom(f) ⇒̇ {a} ◁̇ f ≐̇ {a ↦ f(a)}).
```

2.8 Vérification de la bonne définition

Dans l'approche exposée dans [21], une sémantique à trois valeurs (vrai, faux et mal défini) du langage B est proposée afin de traiter de la bonne définition en B. L'auteur traite de chaque étape de développement de la méthode B et donc aussi des règles ajoutées de l'Atelier B. De manière générale, une règle est considérée comme bien définie si, étant donné un but bien défini, l'application de la règle ne peut pas introduire une expression mal définie. Ainsi, une règle en mode backward est bien définie si la bonne définition des antécédents peut être déduite de la bonne définition du conséquent. Une règle en mode forward est bien définie si la bonne définition de la nouvelle hypothèse peut être déduite de l'hypothèse source. Une règle de réécriture est bien définie si la bonne définition de la nouvelle formule peut être déduite de la bonne définition de l'ancienne. Par exemple la règle, $1/0 == 1/0$ est considérée comme bien définie car si le membre gauche $(1/0)$ est bien défini alors on peut en déduire que le second l'est aussi. Concrètement, le but étant supposé être bien défini, on ne pourra jamais appliquer cette règle.

Par rapport à cette approche, nous avons choisi de ne pas supposer la bonne définition du but et donc d'être plus restrictif. Cela permet, tout d'abord d'éliminer dès cette étape des règles non intéressantes (comme $1/0 == 1/0$) et ensuite, avec notre approche les règles peuvent être appliquées de manière déductive ou inductive sans nouvelle vérification de la bonne définition (voir 2.1.2). Enfin cela permet d'être indépendant de l'outil de développement (de l'Atelier B) puisque les règles doivent être bien définies quel que soit leur contexte d'application.

Comme nous l'avons vu dans la section 2.3, il est montré dans [4] que les définitions conditionnelles peuvent conduire à des expressions mal définies, comme par exemple la division par zéro, le minimum d'un ensemble vide ou encore l'application d'une fonction à un argument en dehors de son domaine de définition. Le respect des conditions de ces définitions peut être vérifié en amont grâce au typage si les conditions sont des inclusions. Lorsqu'elles sont plus complexes, il est possible de les extraire de la formule afin de vérifier qu'elles sont respectées. Un opérateur syntaxique à appliquer sur la règle (noté $\mathcal{L}$), qui agit comme un filtre, est proposé et génère les obligations de preuve de bonne définition pour

l'application de fonction. Ce filtre s'applique d'abord aux sous-termes les plus profonds de la formule, et si l'un de ces sous-termes est une application de fonction $f(x)$ alors il faudra prouver que f est bien une fonction et que son argument x appartient bien au domaine de définition de f. $\mathcal{L}$ est défini dans [4] par deux ensembles de règles : les règles de décomposition et les règle atomiques. Les règles de décomposition sont les suivantes :

$$\mathcal{L}(P \wedge Q) = \mathcal{L}(P) \wedge (P \Rightarrow \mathcal{L}(Q))$$
$$\mathcal{L}(\neg P) = \mathcal{L}(P)$$
$$\mathcal{L}(P \Rightarrow Q) = \mathcal{L}(P) \wedge (P \Rightarrow \mathcal{L}(Q))$$
$$\mathcal{L}(\forall x.P) = \forall x.\mathcal{L}(P)$$

Les règles atomiques traitent des applications de fonction dans les prédicats atomiques, c'est-à-dire tous les prédicats non considérés ci-dessus. Les occurrences de $f(E)$ peuvent être extraits grâce à la règle appelée « règle à un point » qui permet de remplacer de manière équivalente $A_{f(E)}$, qui contient $f(E)$, par $\forall y.(y = f(E) \Rightarrow A_y)$, où y est supposée être une variable fraîche et A_y est la formule précédente dans laquelle toutes les occurrences de $f(E)$ ont été remplacées par y.

$$\begin{array}{lcl} \mathcal{L}(A) & = & \mathsf{true} \\ \mathcal{L}(A_{f(E)}) & = & \mathsf{fnc}(f) \wedge E \in \mathsf{dom}(f) \wedge \forall y.(y = f(E) \Rightarrow \mathcal{L}(A_y)) \\ \mathcal{L}(A_{\{x|x \in S \wedge P\}}) & = & \forall x.(x \in S \Rightarrow \mathcal{L}(P)) \wedge \forall y.(y = \{x|x \in S \wedge P\} \Rightarrow \mathcal{L}(A_y)) \end{array}$$

où $\mathsf{fnc}(f)$ est équivalent à $\forall (x \mapsto y \mapsto z).((x \mapsto y) \in f \wedge (x \mapsto z) \in f \Rightarrow y = z)$, y est une variable fraîche.

Les différents cas de A atomique sont :

1. A ne contient pas d'occurrence de terme de la forme $f(E)$ ou $\{x|x \in S \wedge P\}$
2. $A_{f(E)}$ contient des occurrences de terme de la forme $f(E)$ mais f et E ne contiennent pas d'occurrence de terme de la forme $g(F)$ ou $\{x|x \in S \wedge P\}$
3. $A_{\{x|x \in S \wedge P\}}$ contient des occurrences de terme de la forme $\{x|x \in S \wedge P\}$ mais S ne contient pas d'occurrence de la forme $f(E)$ ou $\{x|x \in T \wedge Q\}$

Ces conditions impliquent un ordre sur l'application de $\mathcal{L}$.

Néanmoins, dans certains cas, cet ordre n'est pas déterministe. Considérons le prédicat

$\{x|x \in \{f(E)\} \wedge g(x) = F\} = C$. Aucune règle de décomposition ne peut être appliquée et pour les règles atomiques, seule la deuxième est applicable. Mais dans ce prédicat il y a deux applications de fonction à traiter, or la deuxième règle laisse le choix sur l'application de fonction à traiter en premier. Si nous choisissons de traiter $g(x)$ d'abord, nous obtenons $\mathsf{fnc}(g) \wedge x \in \mathsf{dom}(g) \wedge \forall y.(y = g(x) \Rightarrow \mathcal{L}(\{x|x \in \{f(E)\} \wedge y = F\} = C))$ où x qui était lié par l'ensemble en compréhension est devenu libre.

Pour traiter aussi ces cas, nous avons défini notre filtre en se basant sur celui proposé par Abrial et Mussat, mais dans lequel l'ordre est imposé pour éviter le problème avec les lieurs. Nous obtenons un filtre plus fonctionnel. Ce nouveau filtre est défini pour des formules réifiées en BCoq par un ensemble de fonctions $\mathcal{L}_X$ où $X \in \{E, S, P\}$ selon que le filtre est appliqué à une expression, un ensemble ou un prédicat. De plus, il permet de calculer $\mathcal{B}_\mathcal{L}$, l'ensemble des variables liées générées par l'application du $\mathcal{L}_X$.

$\mathcal{L}_P$ génère les obligations de preuve de bonne définition pour les prédicats. Cette fonction permet de passer sous les lieurs $\forall V_1$ et $[V_1 := \dots]$. Au premier appel de $\mathcal{L}_P$, l'ensemble des variables liées est $\emptyset$. A partir d'un prédicat de type $\dot{P}$ et de l'ensemble des variables liées, $\mathcal{L}_P$ calcule le prédicat à vérifier de type $\dot{P}$ et le nouvel ensemble de variables liées.

$$
\begin{array}{lcl}
\mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(P_1 \mathbin{\dot{\wedge}} P_2) & = & (\mathcal{L}_{P1} \mathbin{\dot{\wedge}} (P_1 \mathbin{\dot{\Rightarrow}} \mathcal{L}_{P2}), \mathcal{B}_{\mathcal{L}2}) \\
 & & \text{avec} \left\{ \begin{array}{l} \mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(P_1) = (\mathcal{L}_{P1}, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \mathcal{L}_P^{\mathcal{B}_{\mathcal{L}1}}(P_2) = (\mathcal{L}_{P2}, \mathcal{B}_{\mathcal{L}2}) \end{array} \right. \\
\mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(P_1 \mathbin{\dot{\Rightarrow}} P_2) & = & (\mathcal{L}_{P1} \mathbin{\dot{\wedge}} (P_1 \mathbin{\dot{\Rightarrow}} \mathcal{L}_{P2}), \mathcal{B}_{\mathcal{L}2}) \\
 & & \text{avec} \left\{ \begin{array}{l} \mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(P_1) = (\mathcal{L}_{P1}, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \mathcal{L}_P^{\mathcal{B}_{\mathcal{L}1}}(P_2) = (\mathcal{L}_{P2}, \mathcal{B}_{\mathcal{L}2}) \end{array} \right. \\
\mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(\dot{\neg} P_1) & = & \mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(P_1) \\
\mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(\dot{\forall} V_1.P_1) & = & (\dot{\forall} V_1.\mathcal{L}_{P1}, \mathcal{B}_{\mathcal{L}1}) \text{ avec } \mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(P_1) = (\mathcal{L}_{P1}, \mathcal{B}_{\mathcal{L}1}) \\
\mathcal{L}_P^{\mathcal{B}_\mathcal{L}}([V_1 :\doteq E_1]P_1) & = & \dot{\forall} V_1.\mathcal{L}'_P \mathbin{\dot{\wedge}} (V_1 \doteq E_1 \mathbin{\dot{\Rightarrow}} \mathcal{L}_{P1}) \\
 & & \text{avec} \left\{ \begin{array}{l} \mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(V_1 \doteq E_1) = (\mathcal{L}'_P, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \mathcal{L}_P^{\mathcal{B}_{\mathcal{L}1}}(P_1) = (\mathcal{L}_{P1}, \mathcal{B}_{\mathcal{L}2}) \end{array} \right. \\
\mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(E_1 \doteq E_1) & = & (\mathcal{L}_{E1} \mathbin{\dot{\wedge}} \mathcal{L}_{E2}, \mathcal{B}_\mathcal{L}) \\
 & & \text{avec} \left\{ \begin{array}{l} \mathcal{L}_E^{\mathcal{B}_\mathcal{L}}(E_1) = (\mathcal{L}_{E1}, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \mathcal{L}_E^{\mathcal{B}_{\mathcal{L}1}}(E_2) = (\mathcal{L}_{E2}, \mathcal{B}_{\mathcal{L}2}) \end{array} \right. \\
\mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(E_1 \mathbin{\dot{\in}} S_1) & = & (\mathcal{L}_{E1} \mathbin{\dot{\wedge}} \mathcal{L}_{S1}, \mathcal{B}_\mathcal{L}) \\
 & & \text{avec} \left\{ \begin{array}{l} \mathcal{L}_E^{\mathcal{B}_\mathcal{L}}(E_1) = (\mathcal{L}_{E1}, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \mathcal{L}_S^{\mathcal{B}_{\mathcal{L}1}}(S_1) = (\mathcal{L}_{S1}, \mathcal{B}_{\mathcal{L}2}) \end{array} \right.
\end{array}
$$

$$\mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(I_1) \quad = \quad (\top, \mathcal{B}_\mathcal{L})$$

$\mathcal{L}_S$ génère les obligations de preuve de bonne définition pour les ensembles. Cette fonction permet de passer sous le lieur $\{V_1 | \dots\}$. Nous pouvons remarquer que $\mathcal{L}_S$ s'applique aussi sur les constructions dérivées notées $E_1\ op \dots op\ E_n$ où op représente un constructeur ensembliste pour lequel il n'y a pas de condition de bonne définition (comme l'union, le domaine, etc.) et $E_i, \dots, E_n$ représentent ses arguments (s'il y en a). Dans ce cas, il suffit de générer les obligations de preuve pour chaque argument E_i. A partir d'un ensemble de type $\dot{S}$ et de l'ensemble des variables liées, $\mathcal{L}_S$ calcule le prédicat à vérifier de type $\dot{P}$ et le nouvel ensemble de variables liées.

$$\begin{array}{lcl}
\mathcal{L}_S^{\mathcal{B}_\mathcal{L}}(S_1 \dot{\times} S_2) & = & (\mathcal{L}_{S1} \dot{\wedge} \mathcal{L}_{S2}, \mathcal{B}_{\mathcal{L}2}) \\
 & & \text{avec} \left\{ \begin{array}{l} \mathcal{L}_S^{\mathcal{B}_\mathcal{L}}(S_1) = (\mathcal{L}_{S1}, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \mathcal{L}_S^{\mathcal{B}_{\mathcal{L}1}}(S_2) = (\mathcal{L}_{S2}, \mathcal{B}_{\mathcal{L}2}) \end{array} \right. \\
\mathcal{L}_S^{\mathcal{B}_\mathcal{L}}(\dot{\mathbb{P}}\ (S_1)) & = & \mathcal{L}_S(S_1, \mathcal{B}_\mathcal{L}) \\
\mathcal{L}_S^{\mathcal{B}_\mathcal{L}}(\{V_1 | V_1 \dot{\in} S_1 \dot{\wedge} P_1\}) & = & (\dot{\forall}\ V_1 . \mathcal{L}'_P \dot{\wedge} (V_1 \dot{\in} S_1 \dot{\Rightarrow} \mathcal{L}_{P1}), \mathcal{B}_{\mathcal{L}2}) \\
 & & \text{avec} \left\{ \begin{array}{l} \mathcal{L}_P^{\mathcal{B}_\mathcal{L}}(V_1 \dot{\in} S_1) = (\mathcal{L}'_P, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \mathcal{L}_P^{\mathcal{B}_{\mathcal{L}1}}(P_1) = (\mathcal{L}_{P1}, \mathcal{B}_{\mathcal{L}2}) \end{array} \right. \\
\mathcal{L}_S^{\mathcal{B}_\mathcal{L}}(\dot{\mathsf{BIG}}) & = & (\top, \mathcal{B}_\mathcal{L}) \\
\mathcal{L}_S^{\mathcal{B}_\mathcal{L}}(E_1\ op \dots op\ E_n) & = & (\mathcal{L}_{E_1} \dot{\wedge} \dots \dot{\wedge} \mathcal{L}_{E_n}, \mathcal{B}_{\mathcal{L}n}) \\
 & & \text{avec} \left\{ \begin{array}{l} \mathcal{L}_E^{\mathcal{B}_\mathcal{L}}(E_1) = (\mathcal{L}_{E1}, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \dots \\ \mathcal{L}_E^{\mathcal{B}_{\mathcal{L}n-1}}(E_n) = (\mathcal{L}_{En}, \mathcal{B}_{\mathcal{L}n}) \end{array} \right. \\
\mathcal{L}_S^{\mathcal{B}_\mathcal{L}}(I_1) & = & (\top, \mathcal{B}_\mathcal{L})
\end{array}$$

$\mathcal{L}_E$ génère les obligations de preuve de bonne définition pour les expressions. Elle permet de passer sous le lieur $[V_1 :\doteq E_1]$. Concernant $\mathsf{choice}(s)$, nous pouvons remarquer qu'il est toujours bien défini. Or, on pourrait vouloir vérifier que s est non nul. Mais cette condition est déjà présente dans l'axiome SET 5 (voir section 1.1 du chapitre 1) donc, ce n'est pas une définition conditionnelle. C'est pourquoi le cas du $\mathsf{choice}(s)$ n'est pas traité ici.

Lorsqu'il y a une application de fonction $f(e_1)$, alors les conditions de bonne définition qui consistent à vérifier que f est une fonction partielle et que e_1 appartient au domaine de définition de f sont générées. Ici, « f est une fonction partielle » est traduit

par $\exists(s \dot{\mapsto} t).f \dot{\in} s \dot{\leftrightarrow} t$ car il n'existe pas de construction « fnc » dans le B-Book (alors que cette construction est utilisée dans la définition du $\mathcal{L}$ dans [4]). Par conséquent, deux nouvelles variables sont introduites, elles sont conservées dans l'ensemble $\mathcal{B}_{\mathcal{L}}$. Le cas de l'application de fonction couvre le cas de $\frac{1}{x}$ et du minimum d'un ensemble puisqu'en B, ce sont des applications de fonction. A partir d'une expression de type $\dot{E}$ et de l'ensemble des variables liées, $\mathcal{L}_S$ calcule le prédicat à vérifier de type $\dot{P}$ et le nouvel ensemble de variables liées.

$$
\begin{array}{lcl}
\mathcal{L}_E^{\mathcal{B}_{\mathcal{L}}}(V_1) & = & (\top, \mathcal{B}_{\mathcal{L}}) \\
\mathcal{L}_E^{\mathcal{B}_{\mathcal{L}}}([V_1 \dot{:=} E_1]E_2) & = & (\dot{\forall}\, V_1.\mathcal{L}'_P \,\dot{\wedge}\, (V_1 \dot{=} E_1 \dot{\Rightarrow} \mathcal{L}_{E2}) \\
 & & \text{avec} \begin{cases} \mathcal{L}_P^{\mathcal{B}_{\mathcal{L}}}(V_1 \dot{=} E_1) = (\mathcal{L}'_P, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \mathcal{L}_P^{\mathcal{B}_{\mathcal{L}1}}(E_2) = (\mathcal{L}_{E2}, \mathcal{B}_{\mathcal{L}2}) \end{cases} \\
\mathcal{L}_E^{\mathcal{B}_{\mathcal{L}}}(E_1 \dot{\mapsto} E_2) & = & (\mathcal{L}_{E1} \,\dot{\wedge}\, \mathcal{L}_{E2}, \mathcal{B}_{\mathcal{L}2}) \\
 & & \text{avec} \begin{cases} \mathcal{L}_E^{\mathcal{B}_{\mathcal{L}}}(E_1) = (\mathcal{L}_{E1}, \mathcal{B}_{\mathcal{L}1}) \text{ et} \\ \mathcal{L}_E^{\mathcal{B}_{\mathcal{L}1}}(E_2) = (\mathcal{L}_{E2}, \mathcal{B}_{\mathcal{L}2}) \end{cases} \\
\mathcal{L}_E^{\mathcal{B}_{\mathcal{L}}}(\dot{\text{choice}}\,(S_1)) & = & \mathcal{L}_S(S_1, \mathcal{B}_{\mathcal{L}}) \\
\mathcal{L}_E^{\mathcal{B}_{\mathcal{L}}}(f(e_1)) & = & (\exists(s \dot{\mapsto} t).f \dot{\in} s \dot{\leftrightarrow} t \,\dot{\wedge}\, e_1 \dot{\in}\ \dot{dom}\,(f) \,\dot{\wedge}\, \mathcal{L}'_S \,\dot{\wedge}\, \mathcal{L}_{E1}, \\
 & & \mathcal{B}_{\mathcal{L}2} \cup \{s, t\}) \\
 & & \text{avec} \begin{cases} \mathcal{L}_S^{\mathcal{B}_{\mathcal{L}}}(f) = (\mathcal{L}'_S, \mathcal{B}_{\mathcal{L}1}), \\ \mathcal{L}_E^{\mathcal{B}_{\mathcal{L}1}}(E_1) = (\mathcal{L}_{E1}, \mathcal{B}_{\mathcal{L}2}) \text{ et} \\ s \text{ et } t \text{ sont des variables fraîches} \end{cases} \\
\mathcal{L}_E^{\mathcal{B}_{\mathcal{L}}}(S_1) & = & \mathcal{L}_S(S_1, \mathcal{B}_{\mathcal{L}}) \\
\mathcal{L}_E^{\mathcal{B}_{\mathcal{L}}}(I_1) & = & (\top, \mathcal{B}_{\mathcal{L}})
\end{array}
$$

Une fois que le filtre a été appliqué à la règle réifiée de la forme $\forall x \in \mathcal{M}_E \cup \mathcal{M}_S \cup \mathcal{M}_P, \mathcal{B}, \mathcal{N}.P$, le lemme de vérification sur la bonne définition à prouver est généré comme suit :

$$\forall x \in \mathcal{M}_E \cup (\mathcal{M}_S \cup \mathcal{M}_\Gamma) \cup \mathcal{M}_P, \mathcal{B} \cup \mathcal{B}_{\mathcal{L}}, \mathcal{N}.\mathcal{S}_{\mathcal{B}, \mathcal{B} \cup \mathcal{M}_\Gamma, \mathcal{N}}(\mathcal{S}_{\mathcal{B}_{\mathcal{L}}, \mathcal{M}, \mathcal{N}}(H \dot{\vdash} P_{\mathcal{L}}))$$

où

- $\mathcal{L}_P^{\emptyset}(P) = (P_{\mathcal{L}}, \mathcal{B}_{\mathcal{L}})$
- H est la réification du contexte de typage Γ,
- $\mathcal{S}_{\mathcal{B}, \mathcal{B} \cup \mathcal{M}_\Gamma, \mathcal{N}}$ permet de générer les hypothèses de non-liberté de variable de toutes les variables de P entre elles et par rapport aux métavariables introduites par l'inférence de type,

– $\mathcal{S}_{\mathcal{B}_\mathcal{L},\mathcal{M},\mathcal{N}}$ celles des nouvelles variables introduites par $\mathcal{L}$ avec toutes les métavariables (les nouvelles métavariables comprenant $\mathcal{M}_\Gamma$).

Les hypothèses de typage sont conservées car elles peuvent être nécessaires pour la preuve du lemme.

Exemple 15 (Vérification de la bonne définition de la règle 4649)

Cette règle ne contient pas d'application de fonction. Il n'y a donc pas de lemme de bonne définition associé.

Exemple 16 (Vérification de la bonne définition de SimplifyRelDorXY.2)

Dans la règle SimplifyRelDorXY.2, il y a une application de fonction $f(a)$ dans les antécédents. Il faut donc prouver que sous les hypothèses de typage et les antécédents précédents, f est une fonction et a appartient bien au domaine de définition de f.

Calculons tout d'abord $\mathcal{L}^\emptyset_P(SimplifyRelDorXY.2) =$
$\mathcal{L}^\emptyset_P(f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f)) \dot{\Rightarrow} \{a\} \dot{\lhd} f \doteq \{(a \dot{\mapsto} f(a))\})$:
Pour cela nous devons d'abord calculer $\mathcal{L}^\emptyset_P(f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f))$.
On a $\mathcal{L}^\emptyset_P(f \dot{\in} u \dot{\nrightarrow} v) = (\top, \emptyset)$ *et* $\mathcal{L}^\emptyset_P(a \dot{\in} \dot{\mathsf{dom}}\,(f)) = (\top, \emptyset)$.
Donc $\mathcal{L}^\emptyset_P(f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f)) = (\top, \emptyset)$.
Maintenant, calculons $\mathcal{L}^\emptyset_P(\{a\} \dot{\lhd} f \doteq \{(a \dot{\mapsto} f(a))\})$.
On a $\mathcal{L}^\emptyset_E(\{a\} \dot{\lhd} f) = (\top, \emptyset)$ *et*
$\mathcal{L}^\emptyset_E(\{(a \dot{\mapsto} f(a))\}) = (\dot{\exists}\,(s \dot{\mapsto} t).(f \dot{\in} s \dot{\nrightarrow} t \dot{\wedge} a \in \dot{\mathsf{dom}}(f)), \{s,t\})$
car $\mathcal{L}^\emptyset_E(a) = (a, \emptyset)$ *et* $\mathcal{L}^\emptyset_E(f(a)) = (\dot{\exists}\,(s \dot{\mapsto} t).(f \dot{\in} s \dot{\nrightarrow} t \dot{\wedge} a \in \dot{\mathsf{dom}}(f)), \{s,t\})$.
Finalement $\mathcal{L}^\emptyset_P(SimplifyRelDorXY.2) =$
$(f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \in \dot{\mathsf{dom}}(f) \dot{\Rightarrow} (\dot{\exists}\,(s \dot{\mapsto} t).(f \dot{\in} s \dot{\nrightarrow} t \dot{\wedge} a \in \dot{\mathsf{dom}}(f)), \{s,t\}))$

Soit $\forall x \in \{a\}_{\mathcal{M}_E} \cup \{f,u,v\}_{\mathcal{M}_S}, \emptyset, \emptyset.f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f) \dot{\Rightarrow} \{a\} \dot{\lhd} f \doteq \{(a \dot{\mapsto} f(a))\}$ *le résultat de la réification de SimplifyRelDorXY.2.*

Pour générer le lemme de vérification de bonne définition, on a besoin de :

– $\mathcal{B}_\mathcal{L} = \{s,t\}$
– $\Gamma = \{(u, \dot{\mathbb{P}}\,(t_2)), (v, \dot{\mathbb{P}}\,(t_1)), (f, \dot{\mathbb{P}}\,(t_2 \dot{\times} t_1)), (a, t_2)\}$
– $\mathcal{M}_\Gamma = \{t_1, t_2\}$
– $H = \{u \dot{\in} \dot{\mathbb{P}}\,(t_2),\ v \dot{\in} \dot{\mathbb{P}}\,(t_1),\ f \dot{\in} \dot{\mathbb{P}}\,(t_2 \dot{\times} t_1),\ a \dot{\in} t_2\}$

Le lemme de vérification de bonne définition est donc :

$$\forall x \in \{a\}_{\mathcal{M}_E} \cup (\{f,u,v\} \cup \{t_1,t_2\})_{\mathcal{M}_S}, \{s,t\}, \emptyset.\mathcal{S}_{\emptyset,\{t_1,t_2\},\emptyset}(\mathcal{S}_{\{s,t\},\{s,t,a,f,u,v,t_1,t_2\},\emptyset}$$
$$(u \dot{\in} \dot{\mathbb{P}}(t_2),\ v \dot{\in} \dot{\mathbb{P}}(t_1),\ f \dot{\in} \dot{\mathbb{P}}(t_2 \dot{\times} t_1),\ a \dot{\in} t_2 \ \dot{\vdash}$$
$$(f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}(f)) \dot{\Rightarrow} \dot{\exists}(s \dot{\mapsto} t).(f \dot{\in} s \dot{\nrightarrow} t \dot{\wedge} a \in \dot{\mathsf{dom}}(f))))$$

ce qui correspond au lemme Coq suivant :

Lemma bdef_SimplifyRelDorXY_2 :
$\forall\ f\ t_1\ t_2\ u\ v : \dot{S},\ \forall\ a : \dot{E},\ \forall\ s\ t : \dot{V},$
$s \setminus (t,f,u,v,a,t_1,t_2) \to t \setminus (s,f,u,v,a,t_1,t_2) \to$
$u \dot{\in} \dot{\mathbb{P}}(t_2),\ v \dot{\in} \dot{\mathbb{P}}(t_1),\ f \dot{\in} \dot{\mathbb{P}}(t_2 \dot{\times} t_1),\ a \dot{\in} t_2 \ \dot{\vdash}$
$f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}(f) \dot{\Rightarrow} \dot{\exists}(s \dot{\mapsto} t).(f \dot{\in} s \dot{\nrightarrow} t \dot{\wedge} a \in \dot{\mathsf{dom}}(f))\,.$

2.9 Vérification de la validité de la règle

Une fois que l'obligation de preuve relative à la bonne définition est générée grâce au premier filtre, il est possible d'appliquer un autre filtre à la règle qui élimine syntaxiquement les opérateurs définis conditionnellement. Ce deuxième filtre, aussi introduit dans [4], est noté $\mathcal{E}$. Il permet de supprimer sans condition les applications de fonction d'une formule lorsque sa bonne définition a été démontrée et donc de simplifier la formule à démontrer. L'idée est de remplacer toutes les applications de fonction $f(E)$ par une variable y qui est définie tel que $(E,y) \in f$. Par rapport au filtre $\mathcal{L}$ introduit précédemment, il peut être démontré que $\mathcal{L}(P) \dot{\Rightarrow} (P \dot{\Leftrightarrow} \mathcal{E}(P))$.

Nous avons redéfini le filtre $\mathcal{E}$ comme nous avons redéfini précédemment le filtre $\mathcal{L}$. L'équivalence n'est pas démontrée pour ces nouvelles définitions mais la démonstration est semblable à celle présentée dans [4]. $\mathcal{E}$ est défini par un ensemble de fonctions $\mathcal{E}_X$ où $X \in \{E,S,P\}$ et permet de calculer l'ensemble des variables liées introduites par l'application de $\mathcal{E}$.

La fonction $\mathcal{E}_P$ permet d'appeler le filtre sur les sous-termes des prédicats et de récupérer les variables liées introduites par cette application. Dans le cas $f(E_1) \doteq E_2$, $\mathcal{E}_P$ remplace cette expression par $(E_1 \dot{\mapsto} E_2) \dot{\in} f$ pour éviter d'insérer un nouveau quantificateur $\forall$ (qui serait inséré par l'appel de $\mathcal{E}_E(f(E_1))$. A partir d'un prédicat $\dot{P}$, $\mathcal{E}_P$ retourne le prédicat simplifié et l'ensemble des variables liées introduites. Pour faciliter les calculs, nous avons introduit deux autres fonctions annexes :

– $\mathcal{Q}$ qui permet à partir d'un ensemble de prédicats et d'un prédicat de lier tous les prédicats par des implications, définie ainsi :

$\mathcal{Q}(\{Q_1, \ldots, Q_n\}, P) = Q_1 \dot{\Rightarrow} \ \ldots \ \dot{\Rightarrow} Q_n \dot{\Rightarrow} P$

– $\mathcal{B}_\mathcal{E}$ qui permet à partir d'un ensemble de prédicats de la forme $\forall q.P$ de retourner l'ensemble des variables liées q (c'est-à-dire les variables liées introduites par l'application de $\mathcal{E}$), définie ainsi :

$\mathcal{B}_\mathcal{E}(\{\forall q_1.(E_1, q_1) \in f_1, \ldots, \forall q_n.(E_n, q_n) \in f_n\}) = \{q_1, \ldots, q_n\}$.

$$
\begin{array}{lcl}
\mathcal{E}_P(P_1 \dot{\wedge} P_2) & = & (\mathcal{E}_{P1} \dot{\wedge} \mathcal{E}_{P2}, \mathcal{B}_{\mathcal{E}1} \cup \mathcal{B}_{\mathcal{E}2}) \\
 & & \text{où} \left\{ \begin{array}{l} \mathcal{E}_P(P_1) = (\mathcal{E}_{P1}, \mathcal{B}_{\mathcal{E}1}) \text{ et} \\ \mathcal{E}_P(P_2) = (\mathcal{E}_{P2}, \mathcal{B}_{\mathcal{E}2}) \end{array} \right. \\
\mathcal{E}_P(P_1 \dot{\Rightarrow} P_2) & = & (\mathcal{E}_{P1} \dot{\Rightarrow} \mathcal{E}_{P2}, \mathcal{B}_{\mathcal{E}1} \cup \mathcal{B}_{\mathcal{E}2}) \\
 & & \text{où} \left\{ \begin{array}{l} \mathcal{E}_P(P_1) = (\mathcal{E}_{P1}, \mathcal{B}_{\mathcal{E}1}) \text{ et} \\ \mathcal{E}_P(P_2) = (\mathcal{E}_{P2}, \mathcal{B}_{\mathcal{E}2}) \end{array} \right. \\
\mathcal{E}_P(\dot{\neg} P_1) & = & (\dot{\neg} \mathcal{E}_{P1}, \mathcal{B}_{\mathcal{E}1}) \text{ où } \mathcal{E}_P(P_1) = (\mathcal{E}_{P1}, \mathcal{B}_{\mathcal{E}1}) \\
\mathcal{E}_P(\dot{\forall} V_1.P_1) & = & (\dot{\forall} V_1.\mathcal{E}_{P1}, \mathcal{B}_{\mathcal{E}1}) \text{ où } \mathcal{E}_P(P_1) = (\mathcal{E}_{P1}, \mathcal{B}_{\mathcal{E}1}) \\
\mathcal{E}_P([V_1 :\dot{=} E_1]P_1) & = & (\mathcal{Q}(Q_1 \cup Q_2, [V_1 :\dot{=} E_1']P_1'), \mathcal{B}_\mathcal{E}(Q_1 \cup Q_2)) \\
 & & \text{où} \left\{ \begin{array}{l} \mathcal{E}_E(E_1) = (E_1', Q_1), \\ \mathcal{E}_P(P_1) = (P_1', Q_2) \end{array} \right. \\
\mathcal{E}_P(f(E_1) \dot{=} E_2) & = & (\mathcal{Q}(Q_f, (E_1' \dot{\mapsto} E_2') \dot{\in} f'), \mathcal{B}_\mathcal{E}(Q_f)) \\
 & & \text{où} \left\{ \begin{array}{l} \mathcal{E}_E(E_1) = (E_1', Q_1), \\ \mathcal{E}_E(E_2) = (E_2', Q_2) \\ \mathcal{E}_S(f) = (f', Q_3) \text{ et} \\ Q_f = Q_1 \cup Q_2 \cup Q_3 \end{array} \right. \\
\mathcal{E}_P(E_1 \dot{=} E_2) & = & (\mathcal{Q}(Q_1 \cup Q_2, E_1' \dot{=} E_2'), \mathcal{B}_\mathcal{E}(Q_1 \cup Q_2)) \\
 & & \text{où} \left\{ \begin{array}{l} \mathcal{E}_E(E_1) = (E_1', Q_1) \text{ et} \\ \mathcal{E}_E(E_2) = (E_2', Q_2) \end{array} \right. \\
\mathcal{E}_P(E_1 \dot{\in} S_1) & = & (\mathcal{Q}(Q_1 \cup Q_2, E_1' \dot{\in} S_1'), \mathcal{B}_\mathcal{E}(Q_1 \cup Q_2)) \\
 & & \text{où} \left\{ \begin{array}{l} \mathcal{E}_E(E_1) = (E_1', Q_1) \text{ et} \\ \mathcal{E}_S(S_1) = (S_1', Q_2) \end{array} \right. \\
\mathcal{E}_P(I_1) & = & (I_1, \emptyset)
\end{array}
$$

La fonction $\mathcal{E}_S$ permet d'appeler le filtre sur les sous-termes des ensembles et retourne les ensembles simplifiés ainsi que les prédicats introduits par l'application du filtre.

$$
\begin{array}{lcl}
\mathcal{E}_S(S_1 \dot{\times} S_2) & = & (S_1' \dot{\times} S_2', Q_1 \cup Q_2) \text{ où } \left\{ \begin{array}{l} \mathcal{E}_S(S_1) = (S_1', Q_1) \text{ et} \\ \mathcal{E}_S(S_2) = (S_2', Q_2) \end{array} \right. \\
\mathcal{E}_S(\dot{\mathbb{P}}\,(S_1)) & = & (\dot{\mathbb{P}}\,(S_1'), Q_1) \text{ où } \mathcal{E}_S(S_1) = (S_1', Q_1) \\
\mathcal{E}_S(\{V_1 | V_1 \dot{\in} S_1 \dot{\wedge} P_1\}) & = & (\{V_1 | V_1 \dot{\in} S_1' \dot{\wedge} P_1'\}, Q_1 \cup Q_2) \\
 & & \text{où} \left\{ \begin{array}{l} \mathcal{E}_S(S_1) = (S_1', Q_1) \text{ et} \\ \mathcal{E}_P(P_1) = (P_1', Q_2) \end{array} \right.
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{E}_S(\dot{\mathsf{BIG}}) & = & (\mathsf{BIG}, \emptyset) \\
\mathcal{E}_S(I_1) & = & (I_1, \emptyset) \\
\mathcal{E}_S(e_1\ op\ \ldots op\ e_n)) & = & (e'_1\ op\ \ldots op\ e'_n, Q_1 \cup \ldots \cup Q_n) \\
 & & \text{où } \left\{ \begin{array}{l} \mathcal{E}_E(e_1) = (e'_1, Q_1), \\ \ldots \\ \mathcal{E}_E(e_n) = (e'_n, Q_n) \end{array} \right. \\
\mathcal{E}_S(I_1) & = & (I_1, \emptyset)
\end{array}
$$

La fonction $\mathcal{E}_E$ permet d'appeler le filtre sur les sous-termes des expressions et retourne les expressions simplifiées ainsi que les prédicats introduits par l'application du filtre. Lorsqu'il y a une application de fonction $f(E_1)$, alors elle est supprimée puis remplacée par y avec comme condition que $\forall y.(E_1, y) \in f$.

$$
\begin{array}{lcl}
\mathcal{E}_E(V_1) & = & (V_1, \emptyset) \\
\mathcal{E}_E([V_1 :\doteq E_1]E_2) & = & ([V_1 :\doteq E'_1]E'_2, Q_1 \cup Q_2) \text{ où } \left\{ \begin{array}{l} \mathcal{E}_E(E_1) = (E'_1, Q_1) \text{ et} \\ \mathcal{E}_E(E_2) = (E'_2, Q_2) \end{array} \right. \\
\mathcal{E}_E(E_1 \mapsto E_2) & = & (E'_1 \mapsto E'_2, Q_1 \cup Q_2) \text{ où } \left\{ \begin{array}{l} \mathcal{E}_E(E_1) = (E'_1, Q_1) \text{ et} \\ \mathcal{E}_E(E_2) = (E'_2, Q_2) \end{array} \right. \\
\mathcal{E}_E(\dot{\mathsf{choice}}\ (S_1)) & = & (\dot{\mathsf{choice}}\ (S'_1), Q_1) \text{ où } \mathcal{E}_S(S_1) = (S'_1, Q_1) \\
\mathcal{E}_E(f(E_1)) & = & (y, \{\forall y.(E'_1, y) \dot{\in} f'\} \cup Q_1 \cup Q_2) \text{ où } \left\{ \begin{array}{l} \mathcal{E}_E(E_1) = (E'_1, Q_1) \text{ et} \\ \mathcal{E}_S(f) = (f', Q_2) \end{array} \right. \\
\mathcal{E}_E(S_1) & = & \mathcal{E}_S(S_1) \\
\mathcal{E}_E(I_1) & = & (I_1, \emptyset)
\end{array}
$$

où y est une variable fraîche ($y \notin \mathcal{B} \cup \mathcal{M}$).

Une fois que ce filtre a été appliqué à la règle réifiée de la forme $\forall x \in \mathcal{M}, \mathcal{B}, \mathcal{N}.P$, le lemme de vérification de la validité à prouver est généré de la manière suivante :

$$\forall x \in \mathcal{M}_E \cup (\mathcal{M}_S \cup \mathcal{M}_\Gamma) \cup \mathcal{M}_P, \mathcal{B}_\mathcal{E}, \mathcal{N}.\mathcal{S}_{\mathcal{B}, \mathcal{B} \cup \mathcal{M}_\Gamma, \mathcal{N}}(\mathcal{S}_{\mathcal{B}_\mathcal{E}, \mathcal{M}, \mathcal{N}}(H \dot{\vdash} P_\mathcal{E}))$$

où $\mathcal{E}_P(P, \emptyset) = (P_\mathcal{E}, \mathcal{B}_\mathcal{E})$, $\mathcal{B}_\mathcal{E}$ est l'ensemble des variables liées de $\mathcal{E}_P(P)$ et H est la réification du contexte de typage Γ. Les nouvelles hypothèses de non-liberté de variables sont les mêmes que celles pour le lemme de vérification de la bonne définition. Les hypothèses de typage sont conservées car elles peuvent être nécessaires pour la preuve du lemme.

Exemple 17 (Vérification de la règle 4649)

La règle 4649 ne contient pas d'application de fonction. Le lemme généré est donc la règle initiale enrichie avec les hypothèses de typage. Soit $\forall x \in \{b\}_{\mathcal{M}_E} \cup \{a, t, u, t_1, t_2\}_{\mathcal{M}_S}, \emptyset, \emptyset.a \dot{\subseteq}$

$t \dot{\wedge} b \dot{\in} u \dot{\Rightarrow} a \dot{\times}\{ b \}\dot{\in} t \dot{\mapsto} u$ *le résultat de la réification.*

On a

- $\mathcal{B}_{\mathcal{E}} = \emptyset$ *(comme l'application de* $\mathcal{E}$ *n'insère pas de nouvelle variable)*
- $\Gamma = \{(a, \mathbb{P}(t_1)), (b, t_2), (t, \mathbb{P}(t_1)), (u, \mathbb{P}(t_2))\}$
- $\mathcal{M}_\Gamma = \{t_1, t_2\}$
- $H = \{a \subseteq t_1, b \dot{\in} t_2, t \subseteq t_1, u \subseteq t_2\}$

Le lemme de vérification de la validité de la règle 4649 est donc :

$\forall x \in \{b\}_{\mathcal{M}_E} \cup \{a, t, u, t_1, t_2\}_{\mathcal{M}_S}, \emptyset, \emptyset.$

$\mathcal{S}_{\emptyset,\{t_1,t_2\},\emptyset}(\mathcal{S}_{\emptyset,\{b,a,t,u,t_1,t_2\},\emptyset}(a \subseteq t_1, b \dot{\in} t_2, t \subseteq t_1, u \subseteq t_2 \dot{\vdash} a \dot{\subseteq}\ t \dot{\wedge} b \dot{\in} u \dot{\Rightarrow} a \dot{\times}\{ b \}\dot{\in} t \dot{\mapsto} u))$

ce qui correspond au lemme Coq *suivant :*

```
Lemma regle_4649 :
  ∀ b : Ė, ∀ a t u t1 t2 : Ṡ,
  a ⊆ t1, b ∈̇ t2, t ⊆ t1, u ⊆ t2 ⊢̇ a ⊆̇ t ∧̇ b ∈̇ u ⇒̇ a ×̇{ b }∈̇ t ↦̇ u.
```

Exemple 18 (Vérification de SimplifyRelDorXY.2)

Calculons $\mathcal{E}_P(SimplifyRelDorXY.2)$

$\mathcal{E}_P(f \dot{\in} u \dot{\mapsto} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f) \dot{\Rightarrow} \{a\} \dot{\lhd} f \doteq \{(a \mapsto f(a))\})$

Pour cela nous devons d'abord calculer $\mathcal{E}_P(f \dot{\in} u \dot{\mapsto} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f))$.

On a $\mathcal{E}_P(f \dot{\in} u \dot{\mapsto} v) = (\mathcal{Q}(\emptyset, f \dot{\in} u \dot{\mapsto} v), \mathcal{B}_{\mathcal{E}}(\emptyset)) = (f \dot{\in} u \dot{\mapsto} v, \emptyset)$

et $\mathcal{E}_P(a \dot{\in} \dot{\mathsf{dom}}\,(f)) = (\mathcal{Q}(\emptyset, a \dot{\in} \dot{\mathsf{dom}}\,(f)), \mathcal{B}_{\mathcal{E}}(\emptyset)) = (a \dot{\in} \dot{\mathsf{dom}}\,(f), \emptyset).$

Donc $\mathcal{E}_P(f \dot{\in} u \dot{\mapsto} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f)) = (f \dot{\in} u \dot{\mapsto} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f), \emptyset).$

Maintenant, calculons $\mathcal{E}_P(\{a\} \dot{\lhd} f \doteq \{(a \mapsto f(a))\}).$

On a $\mathcal{E}_E(\{a\} \dot{\lhd} f) = \mathcal{E}_S(\{a\} \dot{\lhd} f) = (\{a\} \dot{\lhd} f, \emptyset)$

et $\mathcal{E}_E(\{(a \mapsto f(a))\}) = \mathcal{E}_S(\{(a \mapsto f(a))\}) = (a \dot{\mapsto} y, \{\dot{\forall}\, y.(a \dot{\mapsto} y) \dot{\in} f\})$

car $\mathcal{E}_E(a) = (a, \emptyset)$ *et* $\mathcal{E}_E(f(a)) = (y, \{\dot{\forall}\, y.(a \dot{\mapsto} y) \dot{\in} f\})$

Finalement $\mathcal{E}_P(\{a\} \dot{\lhd} f \doteq \{(a \mapsto f(a))\})$

$$= (\mathcal{Q}(\{\dot{\forall}\, y.(a \dot{\mapsto} y) \dot{\in} f\}, \{a\} \dot{\lhd} f \doteq \{(a \mapsto y)\}), \mathcal{B}_{\mathcal{E}}(\{\dot{\forall}\, y.(a \dot{\mapsto} y) \dot{\in} f\}))$$
$$= (\dot{\forall}\, y.(a \dot{\mapsto} y) \dot{\in} f \dot{\Rightarrow} \{a\} \dot{\lhd} f \doteq \{(a \mapsto y)\}, \{y\})$$

On voit que l'application de fonction $f(a)$ *a bien été supprimée.*

Soit $\forall x \in \{a\}_{\mathcal{M}_E} \cup \{f, u, v\}_{\mathcal{M}_S}, \emptyset, \emptyset. f \dot{\in} u \dot{\mapsto} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}\,(f) \dot{\Rightarrow} \{a\} \dot{\lhd} f \doteq \{(a \mapsto f(a))\}$ *le résultat de la réification de SimplifyRelDorXY.2.*

Pour générer le lemme de vérification de bonne définition, on a besoin de :

- $\mathcal{B}_{\mathcal{E}} = \{y\}$
- $\Gamma = \{(u, \dot{\mathbb{P}}\,(t_2)), (v, \dot{\mathbb{P}}\,(t_1)), (f, \dot{\mathbb{P}}\,(t_2 \dot{\times} t_1)), (a, t_2)\}$
- $\mathcal{M}_{\Gamma} = \{t_1, t_2\}$
- $H = \{u \dot{\in} \dot{\mathbb{P}}\,(t_2),\ v \dot{\in} \dot{\mathbb{P}}\,(t_1),\ f \dot{\in} \dot{\mathbb{P}}\,(t_2 \dot{\times} t_1),\ a \dot{\in} t_2\}$

Le lemme de vérification de la validité de la règle est donc :

$\forall x \in \{a\}_{\mathcal{M}_E} \cup \{f, u, v, t_1, t_2\}_{\mathcal{M}_S} \{y\}, \emptyset. \mathcal{S}_{\{y\},\{y,t_1,t_2\},\emptyset}(\mathcal{S}_{\{y\},\{a,f,u,v\},\emptyset}$
$(u \dot{\in} \dot{\mathbb{P}}\,(t_2),\ v \dot{\in} \dot{\mathbb{P}}\,(t_1),\ f \dot{\in} \dot{\mathbb{P}}\,(t_2 \dot{\times} t_1),\ a \dot{\in} t_2 \vdash$
$\dot{\forall}\, y.(a \dot{\mapsto} y) \dot{\in} f \dot{\Rightarrow} \{a\} \dot{\lhd} f \doteq \{(a \dot{\mapsto} y)\}))$

ce qui correspond au lemme Coq suivant :

```
Lemma regle_SimplifyRelDorXY_2 :
  ∀ f t1 t2 u v : Ṡ, ∀ a : Ė, ∀ y : V̇,
  y \̇ (f, u, v, t1, t2, a) → u ∈̇ ℙ̇ (t2), v ∈̇ ℙ̇ (t1), f ∈̇ ℙ̇ (t2 ×̇ t1), a ∈̇ t2 ⊢̇
  f ∈̇ u ⇸̇ v ∧̇ a ∈̇ doṁ(f) ⇒̇ ∀̇y.((a ↦̇ y) ∈̇ f ⇒̇ {a} ⩤̇ f ≐ {a ↦̇ y}).
```

Chapitre 3

Automatisation des preuves de vérification avec $\mathcal{L}_{\text{tac}}$

Comme nous l'avons vu dans le chapitre précédent, la vérification d'une règle ajoutée est composée de quatre étapes : vérification de la non capture de variable, du typage, de la bonne définition et enfin de la validité de la règle. La vérification de la non capture de variable nécessite uniquement la vérification d'un critère syntaxique. Les trois autres vérifications consistent à prouver un jugement (que l'on appelle lemme de vérification) avec les règles de typage ou les règles de preuve de B. Notre outil BCARe (voir le chapitre 2) offre un environnement de preuve B, appelé BCoq qui repose sur l'assistant à la preuve Coq [60]. Les trois lemmes de vérification doivent être démontrés dans BCoq. Leur preuve peut être effectuée de manière interactive, c'est-à-dire grâce à une suite d'application des règles du B-Book encodées dans BCoq. Mais, cela peut se révéler très fastidieux et répétitif. C'est pourquoi, nous avons automatisé leur preuve.

Dans ce chapitre, nous présentons une approche autarcique [7] (voir sous-section 1.2.4 du chapitre 1) puisque cette approche n'utilise pas d'outil externe à Coq. Cela est possible, notamment grâce au langage de tactiques $\mathcal{L}_{\text{tac}}$ [32] qui permet à l'utilisateur d'écrire ses propres schémas de preuve. Nous automatisons donc des preuves B dans le plongement BCoq grâce à $\mathcal{L}_{\text{tac}}$. Au premier abord, cette approche semble naturelle puisqu'elle construit le terme de preuve au fur et à mesure que la recherche de preuve avance. Dans la suite de la thèse, elle servira de référence pour comparer les différentes approches de preuve automatique (sceptiques cette fois-ci) qui sont développées dans les chapitres suivants.

Nous l'appellerons l'approche $\mathcal{L}_{\text{tac}}$.

Nous allons détailler chacune des tactiques dédiées à la preuve de chaque type de lemme de vérification, c'est-à-dire la tactique dédiée à la preuve du lemme de typage (voir la section 3.1), celle pour la preuve du lemme de bonne définition (voir la section 3.2) et celle pour la preuve du lemme de validité (voir la section 3.3).

Les tactiques de vérification du typage et de bonne définition étaient développées avant le début de cette thèse et je les ai peu modifiées depuis. Le développement de la tactique pour la preuve du lemme de validité a débuté lors de mon stage de master [41].

3.1 Vérification du typage

La preuve du lemme de vérification du typage consiste à vérifier que les expressions d'une formule sont bien formées, c'est-à-dire qu'il n'y a pas d'expression $x \in x$ par exemple. Pour cela, il suffit d'appliquer les règles de typage du B-Book (voir sous-section 1.1.3 du chapitre 1). Pour automatiser ces applications de règles avec $\mathcal{L}_{\text{tac}}$, nous avons suivi la procédure de décision proposée dans le B-Book qui consiste à essayer d'appliquer toutes les règles de typage dans l'ordre où elles sont introduites dans le B-Book. Ces règles étant syntaxiques, nous aurions pu appliquer uniquement la règle qui correspond syntaxiquement au but.

Le typage de B présenté dans le B-Book n'est pas suffisant pour traiter les cas de l'ensemble vide. De plus, comme il n'y a pas de métavariables dans la théorie de B, le typage de B ne traite pas des métavariables de prédicats, or elles peuvent apparaître dans les règles ajoutées. Nous avons donc complété le système de types (voir la sous-section 2.7.1). Notre tactique applique aussi ces règles. Les règles de typage étant syntaxiques, la taille de la preuve trouvée par notre tactique est proportionnelle au nombre de constructeurs et opérateurs B.

La correction de la tactique est immédiate, nous avons en effet suivi de très près les règles de typage du B-Book. Concernant la complétude, elle n'est pas montrée ici. Mais chaque opérateur et constructeur B ayant au moins une règle de typage associée (en considérant le système de types étendu), on peut considérer que la tactique est complète.

Reprenons les exemples de règles ajoutées du chapitre 2.

Exemple 19

Le lemme de vérification du typage généré par BCARe pour la règle 4649 est :

Lemma type_4649 :
$\forall\ t_1\ t_2\ :\ \dot{S}\,,\ \forall$ b a t u $:\ \dot{V}\,,$
$\dot{\mathsf{given}}\,(t_1), \dot{\mathsf{given}}\,(t_2)\ a \subseteq t_1, b \dot{\in} t_2, t \subseteq t_1, u \subseteq t_2\ \dot{\vdash}_\tau\ \dot{\mathsf{check}}\,(a \dot{\subseteq}\ t \dot{\wedge} b \dot{\in} u \dot{\Rightarrow} a \dot{\times} \dot{\{}\ b\ \dot{\}} \dot{\in} t \dot{\nrightarrow} u)\,.$

Notre tactique de vérification de typage prouve ce lemme.

Exemple 20

Le lemme de vérification du typage généré par BCARe pour la règle SimplifyRelDorXY.2 est :

Lemma type_SimplifyRelDorXY_2 :
$\forall\ t_1\ t_2\ :\ \dot{S}\,,\ \forall\ a\ f\ u\ v\ :\ \dot{V}\,,$
$\dot{\mathsf{given}}\,(t_1),\ \dot{\mathsf{given}}\,(t_2),\ u \dot{\in}\ \dot{\mathbb{P}}\,(t_2),\ v \dot{\in}\ \dot{\mathbb{P}}\,(t_1),\ f \dot{\in}\ \dot{\mathbb{P}}\,(t_2 \dot{\times} t_1),$
$a \dot{\in} t_2\ \dot{\vdash}_\tau\ \dot{\mathsf{check}}\,(f \dot{\in} u \dot{\nrightarrow} v \dot{\wedge} a \dot{\in} \dot{\mathsf{dom}}(f) \dot{\Rightarrow} \{a\} \dot{\lhd} f \doteq \{a \mapsto f(a)\}\,)\,.$

Notre tactique de vérification de typage prouve ce lemme.

3.2 Vérification de la bonne définition

Le lemme de vérification de bonne définition se prouve avec le système de preuve du B-Book. Cette preuve est encore peu automatisée dans BCARe. De plus, nous n'utilisons pas la notion de fonction (l'opérateur fnc défini dans [4], voir la section 2.8 du chapitre 2) car elle est inexistante dans le B-Book. Par conséquent, au lieu de vérifier lorsqu'il y a une application de fonction $f(a)$ que f est une fonction $\mathsf{fnc}(f)$, nous vérifions qu'« il existe s et t tels que f est une fonction partielle de s vers t » ce qui complexifie la preuve.

Il y a néanmoins une tactique dédiée à la preuve de ce lemme qui a été développée avant le début de cette thèse. Elle consiste à simplifier au maximum les propositions du lemme puis à chercher une instanciation aux variables s et t. Cette tactique permet de démontrer les lemmes de vérification de bonne définition assez simples ou de seulement simplifier les plus complexes.

Exemple 21

Le lemme de vérification de la bonne définition généré par BCARe pour la règle Simplify-RelDorXY.2 est :

Lemma bdef_SimplifyRelDorXY_2 :
$\forall\ f\ t_1\ t_2\ u\ v\ :\ \dot{S},\ \forall\ a\ :\ \dot{E},\ \forall\ s\ t\ :\ \dot{V},$
$s \setminus (t, f, u, v, a, t_1, t_2) \to t \setminus (s, f, u, v, a, t_1, t_2) \to$
$u \mathrel{\dot\in} \dot{\mathbb{P}}\,(t_2),\ v \mathrel{\dot\in} \dot{\mathbb{P}}\,(t_1),\ f \mathrel{\dot\in} \dot{\mathbb{P}}\,(t_2 \mathbin{\dot\times} t_1),\ a \mathrel{\dot\in} t_2\ \dot\vdash$
$f \mathrel{\dot\in} u \mathrel{\dot\nrightarrow} v \mathbin{\dot\wedge} a \mathrel{\dot\in} \dot{\mathsf{dom}}(f) \Rightarrow \dot\exists (s \mapsto t).(f \mathrel{\dot\in} s \mathrel{\dot\nrightarrow} t \mathbin{\dot\wedge} a \in \dot{\mathsf{dom}}(f))\,.$

Notre tactique traitant de la bonne définition prouve ce lemme car les instanciations des variables s et t peuvent être déduites de la proposition $f \in u \nrightarrow v$. Ainsi, s peut être instancié par u et t par v. Ensuite, la preuve revient à démontrer $P \Rightarrow P$, ce qui est trivial.

Exemple 22

La tactique permet certaines fois de simplifier le but à défaut de le démontrer totalement. Par exemple si on doit montrer que :

$u \mathrel{\dot\in} \dot{\mathbb{P}}\,(t_2),\ v \mathrel{\dot\in} \dot{\mathbb{P}}\,(t_1),\ f \mathrel{\dot\in} \dot{\mathbb{P}}\,(t_2 \mathbin{\dot\times} t_1),\ a \mathrel{\dot\in} t_2\ \dot\vdash$
$f \mathrel{\dot\in} u \mathrel{\dot\nrightarrow} v \mathbin{\dot\wedge} a \mathrel{\dot\in} \dot{\mathsf{ran}}(f^{-1}) \Rightarrow \dot\exists (s \mapsto t).(f \mathrel{\dot\in} s \mathrel{\dot\nrightarrow} t \mathbin{\dot\wedge} a \in \dot{\mathsf{dom}}(f))$

La tactique simplifiera ce but ainsi :

$u \mathrel{\dot\in} \dot{\mathbb{P}}\,(t_2),\ v \mathrel{\dot\in} \dot{\mathbb{P}}\,(t_1),\ f \mathrel{\dot\in} \dot{\mathbb{P}}\,(t_2 \mathbin{\dot\times} t_1),\ a \mathrel{\dot\in} t_2,\ f \mathrel{\dot\in} u \mathrel{\dot\nrightarrow} v, a \mathrel{\dot\in} \dot{\mathsf{ran}}(f^{-1})\ \dot\vdash$
$a \in \dot{\mathsf{dom}}(f)$

Alors qu'en réalité on a $\mathsf{ran}(f^{-1}) = \mathsf{dom}(f)$.

3.3 Vérification de la validité de la règle

Le lemme de la validité de la règle se prouve aussi avec le système de preuve du B-Book. Automatiser cette preuve revient à développer une heuristique de preuve automatique pour la théorie des ensembles de B. Dans cette section, nous utiliserons uniquement le langage de tactiques de Coq, appelé $\mathcal{L}_{\text{tac}}$. Nous verrons dans les chapitres suivants comment les prouveurs automatiques de théorèmes peuvent être utilisés dans ce contexte.

Cette approche est détaillée dans [41], néanmoins quelques optimisations non présentes dans [41] ont été apportées et sont présentées ici.

Les étapes de l'algorithme que nous avons développé sont les suivantes :

1. Normaliser : remplacer toutes les constructions ensemblistes par leur définition afin d'obtenir une formule du premier ordre. Ainsi, il faut déplier les égalités, les inclusions,

tous les opérateurs ensemblistes (comme par exemple l'union, l'intersection, ...), les ensembles en compréhension et les produits cartésiens. Cela est possible car toutes ces constructions sont définies par des axiomes et les opérateurs par des définitions syntaxiques. Par exemple, l'union est définie par un ensemble en compréhension (voir la définition 2 page 88) et cet ensemble en compréhension pourra être supprimé grâce à l'axiome SET3.

2. Mettre en forme prénexe : nous mettons la formule en forme prénexe en remontant en priorité les quantificateurs universels par rapport aux quantificateurs existentiels. Cette étape est en réalité une préparation à l'étape suivante.

3. Élimination des quantificateurs : pour supprimer le $\forall$ il suffit d'appliquer la règle REGLE 7 de B, mais pour le $\exists$ il faut trouver le terme tel que la propriété est vraie, c'est-à-dire le témoin. Pour cela, nous avons implanté une heuristique qui récolte toutes les variables libres du but et nous instancions la variable existentielle avec une de ces variables. Si, *a posteriori*, nous n'arrivons pas à prouver la formule alors nous testons une autre variable, autrement dit nous faisons un « backtrack ». Cette heuristique n'est évidemment pas efficace. Mais nous verrons qu'elle permet de prouver un grand nombre de règles.

4. Prouver la formule restante : la formule à cette étape est une formule du premier ordre sans quantificateur où seul l'opérateur ensembliste « $\in$ » subsiste. Il suffit donc d'appliquer les règles propositionnelles du B-Book pour essayer de prouver la formule. Une procédure de preuve est proposée dans le B-Book pour la logique du premier ordre sans quantificateur. Elle consiste à d'abord vérifier s'il y a une contradiction dans les hypothèses (P et $\neg P$) ou si le but est en hypothèse. Si ce n'est pas le cas, alors il faut appliquer un ensemble de règles qui simplifient la partie droite du séquent au maximum avant d'introduire les propositions en hypothèse. Ensuite on revient au début de la procédure et ainsi de suite. Si plus aucune règle n'est applicable alors la preuve échoue.

Concernant la dernière étape, deux stratégies de preuve propositionnelle qui utilisent les règles du B-Book sous forme de règles de réécriture (implantées dans ELAN) sont présentées et comparées dans [25]. Ces deux stratégies sont en réalité des variantes de celle du B-Book :

on retrouve les 3 étapes (recherche dans les hypothèses, simplification du but, introduction dans les hypothèses) ordonnées différemment.

Pour chaque étape, nous avons développé une tactique dédiée. Nous ne détaillerons pas ici le code $\mathcal{L}_{\text{tac}}$ de ces tactiques, mais seulement leurs spécificités.

3.3.1 Normalisation

Les opérateurs ensemblistes de la méthode B sont définis avec les constructions ensemblistes, définis par les axiomes suivants de la théorie des ensembles :

SET 1 : $(E \mapsto F) \in (s \times t) \Leftrightarrow (E \in s \wedge F \in t)$.

SET 2 : $s \in \mathbb{P}(t) \Leftrightarrow \forall x.(x \in s \Rightarrow x \in t)$, si $x \backslash (s, t)$.

SET 3 : $E \in \{x | x \in s \wedge P\} \Leftrightarrow (E \in s \wedge [x := E]P)$, si $x \backslash s$.

SET 4 : $\forall x.(x \in s \Leftrightarrow x \in t) \Rightarrow s = t$ si $x \backslash (s, t)$.

A partir d'une formule ensembliste, il est toujours possible d'obtenir une formule du premier ordre en dépliant les opérateurs ensemblistes puis les constructeurs ensemblistes (comme l'ensemble en compréhension, le produit cartésien ...). C'est ce qu'on appelle la normalisation.

Pour normaliser des formules ensemblistes, voici la procédure que nous avons implantée (chaque étape est en réalité une tactique) :

- Suppression des égalités ensemblistes (avec SET4) et des inclusions (avec SET2)
- Remplacement des opérateurs par leur définition
- Suppression des ensembles en compréhension (avec SET3) et des produits cartésiens (avec SET1)

3.3.1.1 Suppression des égalités ensemblistes et des inclusions

Cette étape n'est pas triviale dans le sens où il ne suffit pas d'appliquer SET4 ou SET2 pour supprimer les égalités et les inclusions. La subtilité réside dans le fait qu'il faut des informations de typage sur les expressions ensemblistes mises en jeu dans l'égalité ou l'inclusion. En effet, si on a la proposition $E = F$ où E et F sont des ensembles, lorsque

l'on applique SET4, on obtient $\forall x.(x \in E \Leftrightarrow x \in F)$ où x est une variable fraîche. Si E est $s \times t$ la formule obtenue est $\forall x.(x \in s \times t \Leftrightarrow x \in F)$. Dans ce cas, nous ne pouvons pas supprimer le produit cartésien présent dans $x \in s \times t$ en appliquant directement SET1, puisque SET1 s'applique sur des paires d'expressions $(E_1 \mapsto E_2)$.

Il y a deux solutions pour contourner ce problème. La première consiste à calculer le type ou au moins la dimension de E et de F afin de proposer le bon nombre de variables lors de l'application de SET4. Soit s un ensemble. La dimension d'une expression de type s est 1, de type $s \times t$ est 2, de type $(s \times t) \times u$ est 3 ... Par exemple, si E est $s \times t$ et si on suppose que s et t sont de dimension 1 alors $s \times t$ est de dimension 2. Dans ce cas, lors de l'application de SET4, nous allons donc générer deux variables x et y afin d'obtenir $\forall(x \mapsto y).((x \mapsto y) \in s \times t \Leftrightarrow (x \mapsto y) \in E)$. On peut alors appliquer directement SET1 sur cette formule pour déplier le produit cartésien ainsi : $\forall(x \mapsto y).((x \in s \wedge y \in t) \Leftrightarrow (x \mapsto y) \in E)$.

La deuxième solution consiste à utiliser des propriétés permettant d'insérer la paire de variables. Par exemple, pour les produits cartésiens la propriété

$$x \in s \times t \Leftrightarrow \exists(y \mapsto z).x = (y \mapsto z) \wedge (y \mapsto z) \in s \times t)$$

permet d'insérer la paire de variables après l'application de SET4. Cette propriété se prouve avec les règles du B-Book. Donc, si on obtient $\forall x.(x \in s \times t \Leftrightarrow x \in E)$ après l'application de SET4, alors en appliquant la propriété précédente, on obtient

$$\forall x.(\exists(y \mapsto z).x = (y \mapsto z) \wedge (y \mapsto z) \in s \times t) \Leftrightarrow x \in E$$

formule sur laquelle SET1 peut être appliquée.

Pour la tactique $\mathcal{L}_{\text{tac}}$, nous avons choisi la première solution afin de ne pas compliquer la formule avec de nouveaux quantificateurs existentiels. Nous avons donc développé une tactique permettant de calculer la dimension des expressions mis en jeu dans l'égalité (ou dans l'inclusion, étant donné que le problème est le même, voir SET2). Selon le résultat de cette tactique, nous déplions l'égalité avec une, deux ou trois variables.

3.3.1.2 Remplacement des opérateurs par leur définition

Les opérateurs ensemblistes sont définis dans la théorie de B en utilisant les constructeurs ensemblistes (tel que l'ensemble en compréhension par exemple) et les opérateurs

définis auparavant. Par exemple, l'union est définie ainsi dans le B-Book.

Définition 2 (Union)

Soient a une variable et u, s, t des ensembles. Si $s \subseteq u$ et $t \subseteq u$ alors
$s \cup t \triangleq \{a | a \in u \wedge (a \in s \vee a \in t)\}$

Cette définition est dite conditionnelle puisqu'elle présente des conditions, ici des conditions de typage $s \subseteq u$ et $t \subseteq u$.

Dans BCoq, ces définitions sont des axiomes utilisant l'égalité Coq.

```
Axiom def_Union_Coq : ∀ u s t : Ṡ, ∀ a : V̇, ∀ h : Hyp,
  (a \̇ u) → (a \̇ s) → (a \̇ t) →
  (infer h (s ⊆̇ u)) → (infer h (t ⊆̇ u)) →
  (s ∪̇ t) = ({a | a ∈̇ u ∧̇ (a ∈̇ s ∨̇ a ∈̇ t)}).
```

où $a \dot{\setminus} u$ signifie que a est non libre dans u.

Pour supprimer ces opérateurs, il suffit de remplacer tous les opérateurs ensemblistes par leur définition en utilisant les tactiques de remplacement fournies par Coq.

Exemple 23 (Suppression de l'union)

Soit le but $s \subseteq u, t \subseteq u \vdash s \cup t = t \cup s$. $s \cup t$ peut être remplacé par $\{a | a \in u \wedge (a \in s \vee a \in t)\}$.
Nous obtenons les buts suivants :

$s \subseteq u, t \subseteq u \vdash s \subseteq u$

$s \subseteq u, t \subseteq u \vdash t \subseteq u$

$s \subseteq u, t \subseteq u \vdash \{a | a \in u \wedge (a \in s \vee a \in t)\} = t \cup s$

Notons que a est une variable fraîche. Notre outil génère donc les hypothèses que a est non libre par rapport à tous les ensembles et variables déjà introduits, à savoir s, t et u dans cet exemple. Cela se note $a \dot{\setminus} u$, $a \dot{\setminus} t$ et $a \dot{\setminus} u$ en BCoq. Ces hypothèses sont nécessaires notamment pour appliquer l'axiome Coq définissant l'union.
Les deux premiers buts vérifient les hypothèses de typage, ils sont triviaux.
Ensuite, de la même manière, $t \cup s$ peut être déplié. Le but restant est le suivant :
$s \subseteq u, t \subseteq u \vdash \{a | a \in u \wedge (a \in s \vee a \in t)\} = \{a | a \in u \wedge (a \in t \vee a \in s)\}$
On peut remarquer qu'ici, on utilise deux fois la variable a. Cela ne pose pas de problème puisque la variable que l'on utilise doit être non libre dans u, s et t (voir les conditions de la définition de l'union ci-dessus), ce qui est le cas pour la variable a.

Pour les opérateurs définis à partir d'autres opérateurs, il existe une seconde définition (prouvée correcte) dans laquelle tous les opérateurs ont été dépliés. Nous appellerons dans la suite la première définition la définition syntaxique, et la seconde définition le lemme de définition. Nous utilisons les secondes pour ne pas avoir à appeler la tactique de normalisation plus d'une fois sur la formule. Par exemple, l'opérateur ran est défini dans le B-Book de la manière suivante :

Définition 3 (Définition de ran)
Soit p tel que $p \in u \leftrightarrow v$, $\mathsf{ran}(p) \triangleq \mathsf{dom}(p^{-1})$.

On voit que ran est défini à partir du domaine et de la relation inverse donc la définition n'est pas normalisée. C'est pourquoi il existe une seconde définition de ran (qui peut être déduite de la première définition donc elle peut être considérée comme un théorème) :

Théorème 1 (Deuxième définition de ran)
Soit p tel que $p \in u \leftrightarrow v$, $\mathsf{ran}(p) = \{b | b \in v \wedge \exists a.(a \in u \wedge (a \mapsto b) \in p)\}$.

Nous avons encodé cette définition par un lemme Coq (que nous avons démontré) et avec une égalité BCoq.

Dans le cas où on utilise les lemmes de définition plutôt que la définition syntaxique, nous ne pouvons pas utiliser le processus de remplacement de Coq. Nous utilisons donc la REGLE 9 du B-Book qui est la règle des remplacements par des égaux :

$$\frac{H \vdash E = F \qquad H \vdash [x := E]P}{H \vdash [x := F]P}\ \mathsf{REGLE\ 9}$$

Exemple 24 (Remplacement de ran)
Soit le but $s \neq \emptyset \vdash \mathsf{ran}(s \times t) = t$. *Si nous remplaçons* $\mathsf{ran}(s \times t)$ *par sa définition (voir définition 3), nous obtenons les buts suivants :*
$s \neq \emptyset \vdash (s \times t) \in (s \leftrightarrow t)$
$s \neq \emptyset \vdash \mathsf{dom}((s \times t)^{-1}) = t$.
Nous voyons qu'il faut maintenant déplier la relation inverse ainsi que le domaine.
Pour être plus efficace, nous allons utiliser la deuxième définition de ran (voir théorème 1) pour déplier cette expression. Mais ce théorème est une égalité B, nous devons donc utiliser la REGLE 9. Par l'application de la REGLE 9, nous identifions dans le but (grâce à une

variable fraîche x) l'expression à remplacer :

$s \neq \emptyset \vdash [x := \mathsf{ran}(s \times t)]x = t$

Puis la règle produit les buts suivants :

$s \neq \emptyset \vdash \mathsf{ran}(s \times t) = \{b | b \in t \wedge \exists a.(a \in s \wedge (a \mapsto b) \in (s \times t))\}$

$s \neq \emptyset \vdash [x := \{b | b \in t \wedge \exists a.(a \in s \wedge (a \mapsto b) \in (s \times t))\}]x = t.$

Le premier but peut être démontré en appliquant le théorème 1 et le deuxième est le résultat du remplacement c'est-à-dire :

$s \neq \emptyset \vdash \{b | b \in t \wedge \exists a.(a \in s \wedge (a \mapsto b) \in (s \times t))\} = t.$

3.3.1.3 Suppression des ensembles en compréhension et des produits cartésiens

Une fois que tous les opérateurs ensemblistes sont dépliés, il reste les ensembles en compréhension et les produits cartésiens. Pour les supprimer, nous appliquons les axiomes SET1 (pour les produits cartésiens) et SET3 (pour les ensembles en compréhension) en utilisant leur équivalence de gauche à droite. Nous devons donc remplacer un prédicat contenant par exemple un ensemble en compréhension par un autre prédicat équivalent dans lequel l'ensemble en compréhension a été supprimé. Pour cela, nous utilisons un schéma de preuve généralisé qui effectue un remplacement de prédicats qui sera reproduit par une tactique $\mathcal{L}_{\text{tac}}$ dédiée.

Nous pouvons remarquer que pour les remplacements de prédicats, Éric Jaeger et Catherine Dubois étendent dans [45] la syntaxe de B afin de simplifier ce remplacement (qu'ils appellent substitution de prédicats).

Notation 6

Soit P, Pa, Pb des prédicats. $P[Pa/Pb]$ désigne P dans lequel Pa a été remplacé par Pb.

L'objectif du schéma de preuve que nous proposons, est de remplacer un prédicat P_1 par un prédicat P_2 dans P tout en s'assurant que P_1 et P_2 sont équivalents. La règle Modus-Ponens permet d'effectuer ce remplacement. P_2 est au départ laissé inconnu (?) et la formule $P[P_1 \backslash ?] \Rightarrow P$ est simplifiée au maximum en supprimant toutes les expressions qui apparaissent à la fois dans P et dans $P[P_1 \backslash ?]$ (c'est-à-dire toutes les expressions de P

sauf P_1 ou P_2). L'arbre correspondant est le suivant :

$$\mathsf{MP}\,\dfrac{H \vdash P[P_1 \backslash ?] \qquad \dfrac{\dfrac{}{H \vdash P_1 \Leftrightarrow ?}}{\vdots}}{H \vdash P}$$

L'inconnue est résolue lors de l'application d'une règle R.

$$\mathsf{MP}\,\dfrac{H \vdash P[P_1 \backslash P_2] \qquad \dfrac{\dfrac{\mathrm{R}\,\dfrac{}{H \vdash P_1 \Leftrightarrow P_2}}{\vdots}}{H \vdash P[P_1 \backslash P_2] \Rightarrow P}}{H \vdash P}$$

Dans les cas de l'ensemble en compréhension et du produit cartésien, la règle R est respectivement SET3 et SET1.

Exemple 25 (Suppression de l'ensemble en compréhension)
Soit le but $H \vdash (A \wedge x \in \{y | y \in s \wedge y \in a\}) \Leftrightarrow Q$. *Pour supprimer l'ensemble en compréhension, nous appliquons le schéma de preuve présenté précédemment.*

$$\mathit{MP}\,\dfrac{H \vdash ? \qquad \dfrac{\dfrac{H \vdash ?_2 \Rightarrow x \in \{y | y \in s \wedge y \in a\}}{H \vdash ?_1 \Rightarrow (A \wedge x \in \{y | y \in s \wedge y \in a\})}}{H \vdash ? \Rightarrow ((A \wedge x \in \{y | y \in s \wedge y \in a\}) \Leftrightarrow Q)}}{H \vdash (A \wedge x \in \{y | y \in s \wedge y \in a\}) \Leftrightarrow Q}$$

L'application de SET3 permet de trouver toutes les inconnues :

$$\mathit{MP}\,\dfrac{H \vdash P' \qquad \dfrac{\dfrac{\mathit{SET3}_{\Leftarrow}\,\dfrac{}{H \vdash (x \in s \wedge x \in a) \Rightarrow x \in \{y | y \in s \wedge y \in a\}}}{H \vdash (A \wedge x \in s \wedge x \in a) \Rightarrow (A \wedge x \in \{y | y \in s \wedge y \in a\})}}{H \vdash ((A \wedge x \in s \wedge x \in a) \Leftrightarrow Q) \Rightarrow (A \wedge x \in \{y | y \in s \wedge y \in a\}) \Leftrightarrow Q}}{H \vdash (A \wedge x \in \{y | y \in s \wedge y \in a\}) \Leftrightarrow Q}$$

où P' *désigne le prédicat* $(A \wedge x \in s \wedge x \in a) \Leftrightarrow Q$ *dans lequel l'ensemble en compréhension a été supprimé, et* $\mathit{SET3}_{\Leftarrow}$ *est une version de SET3 dans laquelle l'équivalence a été remplacée par une implication.*

A la fin de cette étape, nous obtenons une formule normalisée c'est-à-dire une formule de la logique du premier ordre, ne contenant que « $\in$ » comme opérateur ensembliste.

3.3.2 Mise en forme prénexe

Avant d'essayer de prouver la formule du premier ordre obtenue, nous la mettons en forme prénexe. La procédure suivie est la procédure usuelle :

- introduction de variables fraîches si une variable est quantifiée plusieurs fois
- suppression de tous les $\Rightarrow$ et $\Leftrightarrow$
- propagation des $\neg$ en dessous des quantificateurs (cela est possible car la logique sous-jacente à la théorie de B est une logique classique)
- remontée en priorité des $\forall$ puis des $\exists$.

3.3.3 Élimination des quantificateurs

À ce stade, la formule à montrer est par construction en forme prénexe. Pour enlever les quantificateurs il faut, soit appliquer la REGLE 7 pour les $\forall$ soit trouver un témoin pour les $\exists$ et appliquer la règle DR 13, puis traiter les quantificateurs suivants. Quand tous les quantificateurs sont supprimés, la formule peut être considérée comme une formule propositionnelle.

La difficulté est de trouver les « bons » témoins pour instancier les variables quantifiées par un $\exists$. Pour cela, nous appliquons une heuristique simple. Tout d'abord, nous listons toutes les variables potentielles pour chaque variable à instancier, c'est-à-dire toutes les variables qui ne sont pas quantifiées par un $\exists$. Concrètement, ce sont soit des constantes soit des variables quantifiées par un $\forall$. Ensuite, nous testons grâce à un algorithme de « backtracking » toutes les possibilités d'instanciation. Ainsi, nous instancions les variables existentielles par des variables potentielles. Lorsque tous les quantificateurs sont supprimés, nous essayons de prouver la formule obtenue. Si cela échoue alors nous testons une nouvelle variable potentielle pour la dernière variable existentielle que nous avons instanciée. Si toutes les variables potentielles ont déjà été testées alors nous revenons en arrière pour tester une nouvelle combinaison de variables potentielles.

Exemple 26

Soit $\forall x.\forall y.\exists z.\neg P(x) \vee \neg Q(y) \vee Q(z)$ *la formule à prouver.*

Après deux applications de REGLE 7, si les conditions de non liberté de variable sont vérifiées, nous obtenons $\exists z.\neg P(x) \vee \neg Q(y) \vee Q(z)$. *Les variables potentielles pour* z *sont* x

et y. Les deux cas à tester sont :
$[z := x]\neg P(x) \vee \neg Q(y) \vee Q(z)$ *et* $[z := y]\neg P(x) \vee \neg Q(y) \vee Q(z)$. *Notre algorithme essaie de prouver le premier cas et échoue. Il revient donc sur la dernière instanciation pour tester le deuxième cas. La formule obtenue se prouve, l'algorithme termine avec succès.*

3.3.4 Exemple complet de normalisation

Soit la propriété $a \subseteq s, b \subseteq s \vdash a \cup b = b \cup a$.

Les expressions autour de l'égalité sont de type s. Par conséquent, nous pouvons supprimer l'égalité grâce à SET4 en insérant une nouvelle variable fraîche x (x est supposé être frais par rapport à a, b et s) :
$a \subseteq s, b \subseteq s \vdash \forall x.x \in (a \cup b) \Leftrightarrow x \in (b \cup a)$
Nous supprimons directement le $\forall$ généré par cette étape lorsque c'est possible, ce qui est le cas ici puisque x est frais. Le résultat de l'application de la REGLE 7 est :
$a \subseteq s, b \subseteq s \vdash x \in (a \cup b) \Leftrightarrow x \in (b \cup a)$
L'opérateur $\cup$ est remplacé par sa définition :
$a \subseteq s, b \subseteq s \vdash x \in \{z|z \in s \wedge (z \in a \vee z \in b)\} \Leftrightarrow x \in \{z|z \in s \wedge (z \in b \vee z \in a)\}$

Les ensembles en compréhension sont supprimés ainsi :
$a \subseteq s, b \subseteq s \vdash x \in s \wedge (x \in a \vee x \in b) \Leftrightarrow x \in s \wedge (x \in b \vee x \in a)$.

Il reste une formule qui peut être considérée comme une formule propositionnelle, une tactique dédiée la démontre.

3.3.5 Limitations

Tout d'abord, une première limitation de notre approche est qu'elle n'utilise jamais la règle de contraction. Cette règle est exprimée par les propriétés d'idempotence (de $\vee$ et $\wedge$ dans le B-Book) : $P \vee P \Leftrightarrow P$ et $P \wedge P \Leftrightarrow P$. Par conséquent, notre approche ne peut pas démontrer, par exemple, $\exists x.P(x) \Rightarrow P(a) \wedge P(b)$. Nous ne voulons pas ajouter l'application de cette règle afin d'obtenir un algorithme qui termine. En effet, si on l'ajoutait, on ne saurait pas combien de fois il faudrait l'appliquer.

De plus, l'instanciation est trop particulière. Elle ne fait pas d'unification et les termes

autres que les variables ne sont pas pris en compte. Par exemple, la formule $P(1) \Rightarrow \exists x.P(x)$ ne pourra pas être démontrée par cet algorithme puisque x doit être remplacée par le terme 1 et nous n'utilisons que les variables potentielles pour x, or ici il n'y en a pas.

Enfin, notre heuristique pour trouver les témoins des variables quantifiées par un $\exists$ est parfois inefficace. Lorsque ces quantificateurs sont nombreux, le temps de preuve devient beaucoup trop important.

La solution à ces limitations est d'utiliser un prouveur automatique de théorèmes qui gérera lui même la règle de contraction et les quantificateurs. Il suffit de lui donner la formule à démontrer et de récupérer sa preuve, lorsqu'elle existe. Dans les prochains chapitres, nous détaillons deux approches sceptiques utilisant le prouveur automatique de théorèmes Zenon.

3.4 Résultats expérimentaux

Nous avons lancé notre outil BCARe sur les 5077 règles inductives de la base de règles de Siemens IC-MOL avec un ordinateur Intel Pentium D Xeon 3.60GHz/32Go, avec des limitations pour les preuves de 30 minutes et de 10 Go en consommation mémoire. Il existe plusieurs versions de cette base, celle étudiée ici n'est pas la dernière mais peut contenir des règles fausses.

Dans un premier temps, un composant de BCARe réifie les règles : sur l'ensemble de 5077 règles, 3017 règles sont traduites en lemmes BCoq à vérifier. Parmi les 2060 règles restantes, 1 règle est rejetée à cause de gardes qui ne respectent pas la syntaxe de l'Atelier B. 2059 règles ne sont pas réifiées car elles contiennent au moins une construction non traitée par BCARe (pour 1996 d'entre elles) ou si elles sont traitées par BCARe, elles sont utilisées dans des cas particuliers (pour 63 d'entre elles).

La règle qui ne respecte pas la syntaxe des gardes est une règle de la base de règles de l'Atelier B appelée « s1.21 » qui est la suivante :

$$(\mathsf{band}(\mathsf{binhyp}(\neg(x \in \mathsf{dom}(f)), \mathsf{bsearch}(\{x\}, A \cup B, C)))) \wedge$$
$$(\mathsf{blvar}(Q)) \wedge (Q \backslash (x, f)) \Rightarrow f[A \cup B] == f[C]$$

où band est une garde qui rassemble deux gardes par une conjonction (comme le $\wedge$), et bsearch est une garde qui permet de rechercher un motif dans une formule et de renommer la formule dans lequel le motif a été supprimé. Ici, il y a un problème de parenthèses (voir les parenthèses soulignées) : band doit avoir deux gardes en argument selon sa définition, or ici elle n'a comme argument que $\mathsf{binhyp}(\neg(x \in \mathsf{dom}(f)), \mathsf{bsearch}(\{x\}, A \cup B, C))$. Cela est détecté par notre composant de BCARe qui par conséquent échoue dans la réification de cette règle. Concrètement, cette règle ne peut pas être appliquée dans une preuve de l'Atelier B, mais nous la considérons comme fausse.

Pour 1996 autres règles, les constructions non traitées par BCARe sont par exemple des opérateurs arithmétiques (min), des opérateurs traitant des listes (size), des gardes (bvrb). Concernant les constructions traitées par BCARe, elles doivent être présentées sous certaines formes. Par exemple, BCARe traite de la construction λ sous la contrainte qu'elle soit de la forme $\lambda x.(x \in s|E)$, comme cela est spécifié dans le B-Book. Mais des règles ajoutées contiennent des λ mal formées, comme par exemple $\lambda x.(x \in s \vee x \in t|E)$. Les règles mettant en jeu des constructions « mal formées » comme celle-ci sont rejetées par BCARe. On parlera alors de cas particulier, cela concerne 63 règles de la base de règles.

Ensuite, un composant de BCARe essaie d'inférer un type sur les règles réifiées. Pour 2972 règles des 3017 règles réifiées, l'inférence de type s'est bien déroulée. Parmi les 45 restantes, 31 contiennent une expression non connue de BCARe (« blankrule », seules des règles de l'Atelier B sont concernées), 6 règles sont fausses et 8 ont un problème de type dans les gardes sur la non-liberté de variable ou bfresh. Par exemple, dans la règle suivante « SimplifyRelFonXY.15 » (issue de la base de règles de l'Atelier B) :

$$(\mathsf{binhyp}(a \in \mathsf{dom}(s))) \wedge (\mathsf{blvar}(Q)) \wedge (Q \backslash (a \in \mathsf{dom}(s) \wedge r)) \Rightarrow (r \cup s)(a) == s(a)$$

$a \in \mathsf{dom}(s) \wedge r$ et $(r \cup s)(a) == s(a)$ ne permettent pas de déduire un type pour r puisque dans le premier cas r serait un prédicat et dans le second cas, il serait un ensemble. Notre composant rejette donc ce type de règle. Il semblerait que l'Atelier B, quant à lui, autorise ce genre de règle.

La tactique qui vérifie le typage réussit à prouver le lemme de typage pour 2409 règles

parmi les 2972. Pour 487 règles, elle échoue car ces règles contiennent de l'arithmétique, qui n'est pas encore géré par BCARe. Enfin 76 règles contiennent des cas particuliers non traités actuellement par BCARe. Tous les résultats sont résumés dans les tableaux 3.1 et 3.2.

	Réification	Inférence de type	Vérification de type
OK	3017	2972	2409
KO	2060	45	563

TABLE 3.1 – Résultats pour la réification et le typage

Raison de l'échec	Nombre de règles	Réification	Inférence de type	Typage
Règles fausses	7	1	6	0
Non traitée	2514	1996	31	487
Cas particuliers	139	63	0	76
Typage	8	0	8	0

TABLE 3.2 – Analyse des échecs

Concernant la bonne définition, parmi les règles que nous avons prouvé bien typées (2409 règles), notre outil montre que 2031 règles sont bien définies et échoue pour 378 règles. Nous vérifions ici la bonne définition sans considérer la propagation [21] c'est-à-dire étant donné un but bien défini est-ce que la règle peut introduire une expression mal définie.

Un développement dans BCARe est en cours pour pouvoir traiter aussi cette bonne définition. Nous l'avons testé sur les 378 règles sur lesquelles notre vérification avait échoué, et il démontre avec succès la bonne définition avec propagation pour 143 d'entre elles. Il reste de nombreuses règles pour lesquelles on ne sait pas si elles sont bien définies (avec ou sans propagation) car les lemmes de bonne définition correspondant peuvent contenir des applications de fonction qui ne sont pas gérées par nos tactiques. Nous pouvons remarquer que pour les règles qui sont démontrées bien définies uniquement en considérant la bonne définition avec propagation, elles ne pourront pas être démontrées valides car il manquera les hypothèses sur les relations des règles qui doivent être des fonctions.

La dernière vérification à effectuer sur les règles concerne la validité. Ces essais présentés ici ont été fait sans présélection des règles selon le sous-ensemble traité parce que même si un opérateur n'est pas traité par la tactique $\mathcal{L}_{\text{tac}}$, si la preuve revient à faire du propositionnel,

la tactique $\mathcal{L}_{\text{tac}}$ ne sera pas gênée par les opérateurs qu'elle ne connaît pas. Sur 2031 règles bien définies (et bien typées), notre tactique $\mathcal{L}_{\text{tac}}$ montre la validité de 856 règles. L'analyse des règles restantes est complexe ici puisqu'il faut rejouer les preuves pour savoir où est le problème. Cela montre des problèmes d'incomplétude (ou possiblement des règles fausses) ainsi que des problèmes de temps. Les résultats pour la bonne définition et la validité sont résumés dans les tableaux 3.3 et 3.4.

	Bonne définition	Validité
OK	2031	856
KO	378	1175

TABLE 3.3 – Résultats pour la bonne définition et la validité

Raison de l'échec	Nombre de règles	Bonne définition	Validité
Mémoire	35	11	14
Temps	214	74	140
Autre	1314	293	1021

TABLE 3.4 – Analyse des échecs

Chapitre 4

Automatisation de la preuve B avec Zenon

La tactique $\mathcal{L}_{\text{tac}}$ qui permet de prouver le lemme de validité de la règle souffre de limitations concernant l'incomplétude de l'algorithme de preuve. Ce genre de problème n'aurait pas eu lieu si nous avions utilisé un prouveur automatique de théorème plutôt que de (re)développer un algorithme de preuve.

Dans ce chapitre, nous nous interrogeons sur l'intégration d'un prouveur automatique de théorème dans notre processus de preuve (voir la section 4.1), puis nous décrivons notre implantation concernant l'intégration de Zenon dans notre processus. Zenon [17] est un prouveur automatique de théorème de la logique du premier ordre dont la recherche de preuve est basée sur la méthode des tableaux. Nous présentons tout d'abord ce prouveur (voir la section 4.2), puis nous détaillons chaque étape de la preuve du lemme de validité (voir la section 4.3), ainsi que leur implantation dans la sous-section 4.3.5.

4.1 Motivations

Notre objectif peut être généralisé ainsi : étant donné une formule φ_B, nous voulons trouver automatiquement une preuve P_B telle que $P_B : \varphi_B$ (lire P_B est une preuve de φ_B). Si on utilise notre plongement profond BCoq, soit φ_{BCoq} la version réifiée de φ_B, alors si on trouve une preuve P_{BCoq} telle que $P_{\mathsf{BCoq}} : \varphi_{\mathsf{BCoq}}$, on sait qu'il existe une preuve P_B telle que $P_B : \varphi_B$ (en supposant qu'il n'y a pas d'erreur dans le plongement).

Notre approche $\mathcal{L}_{\text{tac}}$ traite ce problème grâce au langage de tactique $\mathcal{L}_{\text{tac}}$. On peut dire que c'est une approche autarcique [7] dans le sens où l'automatisation de la recherche de preuve est faite directement au sein de Coq. Mais cette technique n'est pas efficace en pratique à cause de l'heuristique d'instanciation que nous avons développée et est incomplète.

Nous voulons donc utiliser un prouveur automatique de théorème qui se chargera de trouver une preuve mais avec une approche sceptique (voir la sous-section 1.2.4 du chapitre 1). Ainsi, nous voulons utiliser un prouveur externe mais en vérifiant la correction de la preuve qu'il trouve en utilisant par exemple les traces qu'il fournit. Cela répond exactement à notre problème.

Pour développer une telle approche se pose la question de comment vérifier la correction des preuves trouvées par le prouveur externe. Pour ce faire, l'idée consiste à exploiter les traces fournies par le prouveur. Les prouveurs de théorèmes n'ont aujourd'hui pas de norme commune sur leur trace. Il nous faut donc un prouveur qui génère des preuves assez compréhensibles pour qu'elles soient reconstruites facilement. Le prouveur automatique Zenon [17] propose des traces de preuve à différents niveaux d'abstraction et des preuves vérifiables dans Coq ou dans Isabelle (dans le format Isar pour ce dernier). Zenon est donc un candidat approprié pour notre expérimentation.

Une autre difficulté dans cette approche est de trouver la meilleure traduction entre le langage de départ (ici B ou BCoq) et le langage d'entrée du prouveur. Une première tentative a consisté à donner en entrée à Zenon le lemme BCoq en ajoutant en axiomes, les règles B réifiées qui permettraient à Zenon de traiter des constructions BCoq. Cela ne s'est pas révélé concluant, puisque Zenon ne trouvait des preuves que de formules très simples. Nous avons testé d'autres prouveurs comme E [56] et SPASS [62], pour lesquels les résultats étaient semblables. Nous avons donc déréifié toutes les règles données en entrée ainsi que le lemme c'est-à-dire que tous les connecteurs logiques utilisés sont ceux de Zenon alors que les constructions ensemblistes sont considérées comme des fonctions non interprétées par Zenon. Néanmoins, le résultat avec les prouveurs testés fut le même. C'est pourquoi, nous avons supprimé de la formule donnée en entrée à Zenon, toutes les constructions ensemblistes de B afin d'obtenir de la logique du premier ordre. En effet, la théorie de B est une théorie ensembliste reposant sur la logique du premier ordre et Zenon est basé sur

la logique du premier ordre avec égalité. Le lien évident entre les deux est la logique du premier ordre. De plus, nous avons vu dans la sous-section 3.3.1 que les formules B peuvent être normalisées en utilisant les définitions et axiomes de la théorie B afin d'obtenir de pures formules du premier ordre (sans opérateurs ensemblistes).

Le choix de Zenon a été conforté par la suite par une expérimentation dans laquelle nous avons comparé Zenon à d'autres prouveurs tels que E, iProver [46] et Vampire [55]. Les résultats sont publiés dans [44] et montrent que bien que Zenon ne soit pas le plus rapide, il démontre autant de formules du premier ordre (issues après transformation des règles de la base de règles de Siemens IC-MOL) que les autres prouveurs.

Par conséquent, nous avons développé une approche (illustrée dans la figure 4.1) qui suit les étapes suivantes :

1. La formule initiale est une formule B qui est réifiée grâce au processus de réification (cf. la section 2.5 du chapitre 2) en une formule BCoq ;
2. Une tactique $\mathcal{L}_{\text{tac}}$ normalise dans BCoq la formule afin d'obtenir une pure formule du premier ordre ;
3. La formule obtenue est interprétée dans la syntaxe d'entrée de Zenon et permet d'appeler le prouveur ;
4. Si Zenon trouve une preuve alors il génère une preuve Coq ;
5. Cette preuve Coq est (re)réifiée en une preuve BCoq. Coq peut alors vérifier que la preuve obtenue est correcte.

4.2 L'outil de déduction automatique Zenon

Zenon [17] est un prouveur automatique de théorème de la logique du premier ordre avec égalité. Sa recherche de preuve est basée sur la méthode des tableaux [39, 12, 57] qui fonctionne par réfutation et qui consiste à montrer que la négation d'une formule est insatisfiable. L'algorithme commence par nier la formule à prouver, puis la formule est décomposée par des règles (chaque règle traite d'un opérateur) ce qui peut générer des branches. La formule est prouvée lorsque chaque branche est fermée, c'est-à-dire qu'elle contient une contradiction.

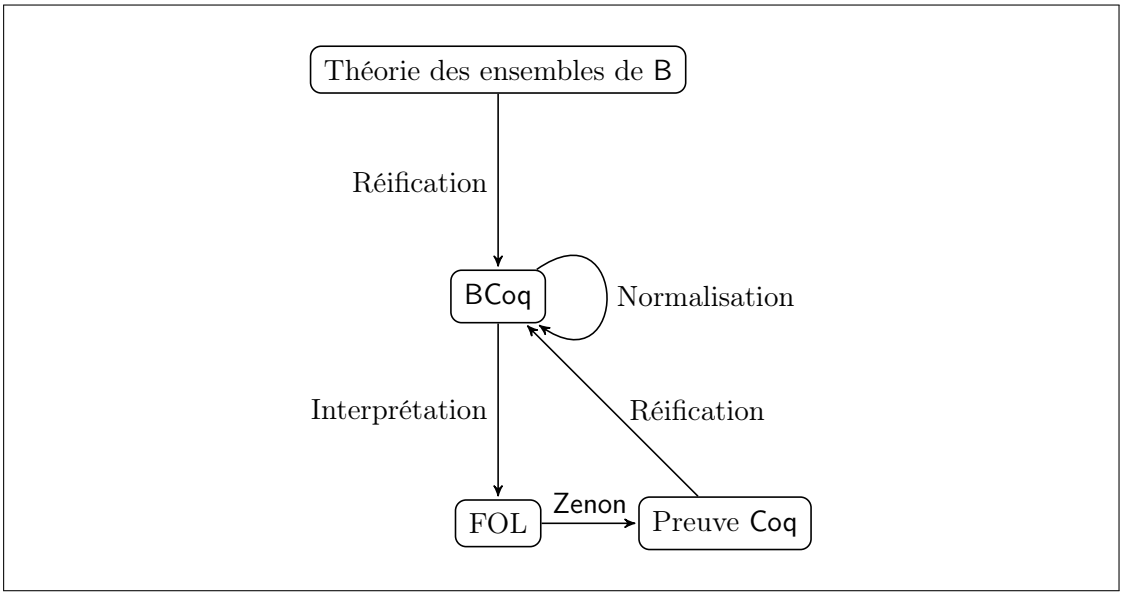

FIGURE 4.1 – Processus de preuve B utilisant Zenon

Les règles de recherche de preuve de Zenon sont décrites dans la figure 4.2. Dans ces règles, ϵ est l'opérateur de Hilbert : $\epsilon(x).P(x)$ représente un certain x qui satisfait $P(x)$, et est considéré comme un terme. Les majuscules désignent des métavariables qui sont usuellement appelées variables libres dans la littérature de la méthode des tableaux, elles ne sont pas utilisées comme variables puisqu'elles ne sont jamais substituées. R_r, R_s, R_t et R_{ts} sont des relations respectivement réflexive, symétrique, transitive, et transitive-symétrique. Les règles correspondantes s'appliquent aussi à l'égalité. Les règles δ sont traitées avec des ϵ-termes au lieu d'utiliser la skolémisation.

Les règles de recherche de preuve sont appliquées avec la méthode des tableaux usuelle : à partir de la négation du but, les règles sont appliquées de haut en bas pour construire un arbre. Lorsque toutes les branches sont closes (c'est-à-dire se terminent par une application d'une règle de clôture), alors l'arbre est clos et cet arbre représente une preuve du but. Cet algorithme est appliqué de manière séquentielle sur les branches : une branche doit être fermée avant de commencer à traiter une autre branche. De plus, la recherche de preuve est faite de manière non destructive puisque le traitement d'une branche ne changera jamais les formules d'une autre branche. Ainsi, si une instanciation est faite dans une branche, cela n'influera pas sur les autres branches. Néanmoins, un système de « pruning » permet

Règles de clôture et de coupure

$$\frac{\bot}{\odot}\ \odot_{\bot} \qquad \frac{\neg\top}{\odot}\ \odot_{\neg\top} \qquad \frac{}{P \mid \neg P}\ \text{cut}$$

$$\frac{\neg R_r(t,t)}{\odot}\ \odot_r \qquad \frac{P \qquad \neg P}{\odot}\ \odot \qquad \frac{R_s(a,b) \qquad \neg R_s(b,a)}{\odot}\ \odot_s$$

Règles analytiques

$$\frac{\neg\neg P}{P}\ \alpha_{\neg\neg} \qquad \frac{P \Leftrightarrow Q}{\neg P, \neg Q \mid P, Q}\ \beta_{\Leftrightarrow} \qquad \frac{\neg(P \Leftrightarrow Q)}{\neg P, Q \mid P, \neg Q}\ \beta_{\neg\Leftrightarrow}$$

$$\frac{P \wedge Q}{P, Q}\ \alpha_{\wedge} \qquad \frac{\neg(P \vee Q)}{\neg P, \neg Q}\ \alpha_{\neg\vee} \qquad \frac{\neg(P \Rightarrow Q)}{P, \neg Q}\ \alpha_{\neg\Rightarrow}$$

$$\frac{P \vee Q}{P \mid Q}\ \beta_{\vee} \qquad \frac{\neg(P \wedge Q)}{\neg P \mid \neg Q}\ \beta_{\neg\wedge} \qquad \frac{P \Rightarrow Q}{\neg P \mid Q}\ \beta_{\Rightarrow}$$

$$\frac{\exists x\ P(x)}{P(\epsilon(x).P(x))}\ \delta_{\exists} \qquad \frac{\neg\forall x\ P(x)}{\neg P(\epsilon(x).\neg P(x))}\ \delta_{\neg\forall}$$

Règles γ

$$\frac{\forall x\ P(x)}{P(X)}\ \gamma_{\forall M} \qquad \frac{\neg\exists x\ P(x)}{\neg P(X)}\ \gamma_{\neg\exists M}$$

$$\frac{\forall x\ P(x)}{P(t)}\ \gamma_{\forall \text{inst}} \qquad \frac{\neg\exists x\ P(x)}{\neg P(t)}\ \gamma_{\neg\exists \text{inst}}$$

Règles relationnelles

$$\frac{P(t_1,...,t_n) \qquad \neg P(s_1,..,s_n)}{t_1 \neq s_1 \mid ... \mid t_n \neq s_n}\ \text{pred} \qquad \frac{f(t_1,...,t_n) \neq f(s_1,...,s_n)}{t_1 \neq s_1 \mid ... \mid t_n \neq s_n}\ \text{fun}$$

$$\frac{R_s(s,t) \qquad \neg R_s(u,v)}{t \neq u \mid s \neq v}\ \text{sym} \qquad \frac{\neg R_r(s,t)}{s \neq t}\ \neg_{\text{refl}}$$

$$\frac{R_t(s,t) \qquad \neg R_t(u,v)}{u \neq s, \neg R_t(u,s) \mid t \neq v, \neg R_t(t,v)}\ \text{trans}$$

$$\frac{R_{ts}(s,t) \qquad \neg R_{ts}(u,v)}{v \neq s, \neg R_{ts}(v,s) \mid t \neq u, \neg R_{ts}(t,u)}\ \text{transsym}$$

$$\frac{s = t \qquad \neg R_t(u,v)}{u \neq s, \neg R_t(u,s) \mid \neg R_t(u,s), \neg R_t(t,v) \mid t \neq v, \neg R_t(t,v)}\ \text{transeq}$$

$$\frac{s = t \qquad \neg R_{ts}(u,v)}{v \neq s, \neg R_{ts}(v,s) \mid \neg R_{ts}(v,s), \neg R_{ts}(t,u) \mid t \neq u, \neg R_{ts}(t,u)}\ \text{transeqsym}$$

FIGURE 4.2 – Règles de Zenon (partie 1)

de simplifier la recherche de preuve si un sous-arbre, avec par exemple une instanciation, permet de clore une autre branche (voir l'exemple ci-dessous).

Les règles de Zenon sont divisées en cinq parties dans le but d'optimiser la recherche de preuve. Ces sont les ensembles usuels de règles traitant des formules α, β, δ, ou γ et de clôture ($\odot$) étendus avec les règles spécifiques de Zenon. Les cinq ensembles de règles ainsi que leurs éléments sont listés ci-dessous :

α	$\alpha_{\neg\vee}, \alpha_{\wedge}, \alpha_{\neg\Rightarrow}, \alpha_{\neg\neg}, \neg_{\text{refl}}$
β	$\beta_{\vee}, \beta_{\neg\wedge}, \beta_{\Rightarrow}, \beta_{\Leftrightarrow}, \beta_{\neg\Leftrightarrow}$, pred, fun, sym, trans$*$
δ	$\delta_{\exists}, \delta_{\neg\forall}$
γ	$\gamma_{\forall M}, \gamma_{\neg\exists M}, \gamma_{\forall\text{inst}}, \gamma_{\neg\exists\text{inst}}$
$\odot$	$\odot_{\top}, \odot_{\bot}, \odot, \odot_r, \odot_s$

où « trans$*$ » désigne toutes les règles transitives.

L'exemple suivant (présenté dans [17]) illustre la recherche de preuve de Zenon ainsi que le principe du « pruning ».

Exemple 27

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall x, P(x) \vee Q(x) \quad \neg P(a) \quad \neg Q(a)}{P(X) \vee Q(X)}\,\gamma_{\forall M}}{P(X) \quad Q(X)}\,\beta_{\vee}}{P(a) \vee Q(a)}\,\gamma_{\forall inst}}{P(a) \quad Q(a)}\,\beta_{\vee}}{\odot \qquad \odot}\,\odot$$

À partir de la formule $\forall x, P(x) \vee Q(x)$, une métavariable X est introduite par la règle $\gamma_{\forall M}$. Puis la présence de $P(X)$ et de $\neg P(a)$ dans la même branche induit l'instanciation de $\forall x, P(x) \vee Q(x)$ par a. Cet arbre n'est pas complet puisqu'il a une branche ouverte (sous $Q(X)$). Cette branche, n'a pas à être explorée car elle peut être supprimée (ainsi que certains nœuds) pour obtenir l'arbre de preuve clos suivant :

$$\dfrac{\dfrac{\dfrac{\forall x, P(x) \vee Q(x) \quad \neg P(a) \quad \neg Q(a)}{P(a) \vee Q(a)}\,\gamma_{\forall inst}}{P(a) \quad Q(a)}\,\beta_{\vee}}{\odot \qquad \odot}\,\odot$$

4.3 Processus de preuve avec Zenon

4.3.1 Prénormalisation

Comme dit précédemment, nous voulons utiliser Zenon afin de prouver une formule ensembliste B. La première étape de notre approche est de normaliser la formule B (plus précisément la formule réifiée BCoq) pour obtenir une pure formule du premier ordre. Ainsi, nous supprimons tous les opérateurs ensemblistes afin d'obtenir une formule du premier ordre contenant uniquement « $\in$ » comme opérateur ensembliste. Pour effectuer cette étape, nous appelons une tactique $\mathcal{L}_{\text{tac}}$ qui reprend l'algorithme et les tactiques exposés dans la sous-section 3.3.1 du chapitre 3. La seule différence est que cette fois-ci, nous appliquons la normalisation sur les hypothèses de typage aussi puisque nous donnerons ces hypothèses en entrée à Zenon. Dans l'approche $\mathcal{L}_{\text{tac}}$, il n'est pas nécessaire de normaliser ces hypothèses puisque les tactiques de recherche de preuve après la normalisation peuvent utiliser les hypothèses de typage même si elles sont sous la forme d'inclusion (donc non normalisées). De plus, lorsque l'on supprime ces hypothèses de typage, certaines formules ne sont plus démontrables.

Exemple 28

Soit la propriété

```
Lemma set_1_bcoq :
  forall (s b a : S),
a ⊆ s, b ⊆ s ⊢ a ∪ b = b ∪ a.
```

La formule pré-normalisée correspondante (dont les hypothèses ont aussi été normalisées) est :

```
Lemma set_1_bcoq :
  forall (x0 x1 x : V) (s b a : S),
  x \ (a, b, s) → x0 \ (a, b, s, x) → x1 \ (a, b, s, x, x0) →
  ∅ ⊢ (∀ x0.(x0 ∈ b ⇒ x0 ∈ s)) ⇒ (∀ x1.(x1 ∈ a ⇒ x1 ∈ s)) ⇒
      (x ∈ s ∧ (x ∈ a ∨ x ∈ b)) ⇔ (x ∈ s ∧ (x ∈ b ∨ x ∈ a)).
```

On peut voir que les hypothèses de typage ont été ajoutées dans le but (à l'aide de la REGLE 4) afin d'être normalisées et que les hypothèses suivantes de non-liberté de variable ont été générées par l'étape de normalisation : $x \setminus (a, b, s)$, $x_0 \setminus (a, b, s, x)$ *et*

$x_1 \dot{\setminus} (a, b, s, x, x_0)$.

4.3.2 Interprétation et preuve

Une fois normalisé, le lemme à prouver est une formule de la logique du premier ordre dans laquelle il n'y a pas d'autres opérateurs ensemblistes que « $\dot{\in}$ » (c'est-à-dire $\in$ dans sa forme réifiée). Pour faciliter la recherche de preuve, nous ne donnons pas la formule directement à Zenon puisqu'elle est toujours réifiée en BCoq après la normalisation. Au lieu de cela, nous interprétons partiellement la formule dans la logique de Zenon, dans laquelle le symbole « $\dot{\in}$ » est considéré comme un symbole de prédicat non interprété. Les hypothèses de non liberté de variables concernant les variables x, x_0 et x_1 ne sont pas interprétées puisque Zenon les traitent implicitement.

Concrètement, l'interprétation produit un nouveau lemme Coq (et non BCoq) qui est directement accepté par Zenon puisque Coq est l'un des formats d'entrée de Zenon.

Exemple 29 (Interprétation et preuve de Set 1)
Après la normalisation, la formule obtenue est la suivante :

$$\begin{array}{l}
\textbf{\textit{Lemma}}\ \text{set_1_bcoq}\ : \\
\quad \textbf{forall}\ (x_0\ x_1\ x\ :\ V)\ (s\ b\ a\ :\ S), \\
\quad x \dot{\setminus} (a,b,s) \to x_0 \dot{\setminus} (a,b,s,x) \to x_1 \dot{\setminus} (a,b,s,x,x_0) \to \\
\quad \dot{\emptyset} \dot{\vdash}\ (\dot{\forall}\, x_0.(x_0 \dot{\in} b \dot{\Rightarrow} x_0 \dot{\in} s)) \dot{\Rightarrow} (\dot{\forall}\, x_1.(x_1 \dot{\in} a \dot{\Rightarrow} x_1 \dot{\in} s)) \dot{\Rightarrow} \\
\qquad (x \dot{\in} s \dot{\wedge} (x \dot{\in} a \dot{\vee} x \dot{\in} b)) \dot{\Leftrightarrow} (x \dot{\in} s \dot{\wedge} (x \dot{\in} b \dot{\vee} x \dot{\in} a)).
\end{array}$$

Cette formule est interprétée en Coq comme suit :

$$\begin{array}{l}
\textbf{\textit{Lemma}}\ \text{set_1_coq}\ : \\
\quad (\textbf{forall}\ x_0\ :\ V,\ \text{In}\ x_0\ b \to \text{In}\ x_0\ s) \to \\
\quad (\textbf{forall}\ x_1 :\ V,\ \text{In}\ x_1\ a \to \text{In}\ x_1\ s) \to \\
\quad (\text{In}\ x\ s \wedge (\text{In}\ x\ a \vee \text{In}\ x\ b)) \leftrightarrow (\text{In}\ x\ s \wedge (\text{In}\ x\ b \vee \text{In}\ x\ a)).
\end{array}$$

où « In » est l'interprétation de « $\dot{\in}$ » et sera considéré comme un symbole de prédicat usuel non interprété par Zenon.

a, b et s (de type S) n'étant pas quantifiées, Zenon les considère comme des constantes. Cela permet d'améliorer l'efficacité de la recherche de preuve.

4.3.3 Preuve générée par Zenon

Une fois le lemme traduit en Coq, Zenon peut être appelé sur la formule interprétée. Si Zenon trouve une preuve, alors il peut générer un script de preuve dans des formats avec différents niveaux d'abstraction. Par exemple, il est capable de produire des preuves vérifiables par des outils de preuve interactive comme Coq ou Isabelle.

Le format qui nous intéresse ici est le format Coq puisque nous voulons vérifier la preuve de Zenon dans notre plongement de B dans Coq. Pour produire une preuve Coq, Zenon associe à chaque règle un lemme Coq qui est démontré une fois pour toutes (voir les détails dans [17]). Le script de preuve Coq, généré par Zenon, est une succession d'application de ces lemmes.

Par exemple, si l'on considère la règle $\dfrac{\neg(P \wedge Q)}{\neg P \mid \neg Q}\,\beta_{\neg\wedge}$ le lemme Coq associé est :

```
Lemma zenon_notand : forall P Q : Prop,
    ~ (P ∧ Q) → (~ P→ False) → (~ Q→ False) → False
```

Cet ordre des propositions permet d'appliquer partiellement le lemme avec *apply* (*notand* _ _ *H*) où *H* est une hypothèse de la forme ~(*P* ∧ *Q*) et les autres propositions sont déduites par Coq.

Exemple 30

Soit le lemme à démontrer :

```
Lemma set_1_coq :
  (forall x0 : V, In x0 b → In x0 s) →
  (forall x1: V, In x1 a → In x1 s) →
  (In x s ∧ (In x a ∨ In x b)) ↔ (In x s ∧ (In x b ∨ In x a)).
```

Zenon trouve une preuve dont voici un extrait :

```
Proof.
  apply NNPP. intro zenon_G.
  apply (zenon_notimply _ _ zenon_G).
  zenon_intro zenon_H2. zenon_intro zenon_H1. ...
Qed.
```

où NNPP *est une forme du tiers exclu,* zenon_intro *est une tactique qui introduit des hypothèses dans le contexte avec des noms frais si besoin, et* zenon_notimply *une règle analytique qui traite de la négation de l'implication (elle correspond à la règle* $\alpha_{\neg\Rightarrow}$ *de la figure 4.2).*

Ces deux lemmes ont les types suivants :

```
NNPP : forall P : Prop, ~~P → P.

zenon_notimply : forall P Q : Prop,
    ~ (P → Q) → (P → ~ Q → False) → False.
```

4.3.4 Reconstruction de la preuve

La dernière étape du processus de validation consiste à reconstruire une preuve B à partir de la preuve Coq générée par Zenon. Cette preuve sera en réalité une preuve réifiée dans le plongement BCoq que nous pourrons rejouer. Dans notre cas, la reconstruction est une traduction syntaxique qui repose sur la réification de chaque tactique et règle qui peuvent apparaître dans une preuve Coq produite par Zenon. Ainsi, à chaque lemme Coq *zenon_rule* correspond un lemme BCoq *bcoq_rule*. Par exemple le lemme BCoq correspondant au lemme Coq *zenon_notand* est le suivant :

```
Lemma bcoq_notand : forall (P Q : Ṗ) (h : list Ṗ),
  h ⊢̇ ¬̇ P ⇛ Faux → h ⊢̇ ¬̇ Q ⇛ Faux → h ⊢̇ ¬̇ (P ∧̇ Q) ⇛ Faux.
```

Cependant, certains autres aspects de la preuve de Zenon sont moins évidents à traduire. Cela provient essentiellement de la différence entre la logique de Zenon et le plongement de la théorie des ensembles de B. En effet, les règles de Zenon manipulent des formules en hypothèse or en BCoq, on ne peut pas le faire directement. Il faut insérer la formule dans la conclusion pour pouvoir la manipuler ensuite, comme cela est induit par les règles de preuve du B-Book. Ainsi, avant chaque application d'un lemme BCoq (correspondant à une règle Zenon), on insère l'hypothèse nécessaire dans la conclusion. Ensuite, on pourra utiliser les lemmes BCoq en mode backward (comme par exemple le lemme *bcoq_notand*).

Une autre difficulté réside dans l'absence de noms pour les hypothèses dans B, alors que la preuve générée par Zenon utilise les noms des hypothèses pour appliquer un lemme sur une hypothèse précisément. Pour résoudre ce problème, nous avons développé la tactique *select_R4* qui sélectionne une hypothèse par sa position dans la liste d'hypothèses, puis l'insère du côté droit du séquent. Le lemme utilisant cette hypothèse peut alors être directement appliqué. De plus, pour insérer la bonne hypothèse, nous calculons lors de la traduction de la preuve de Zenon, la liste des noms des hypothèses courante à chaque étape. Ce nom est introduit par la tactique *zenon_intro* dans la preuve Zenon. Ainsi, dans

la preuve BCoq, nous insérons une hypothèse de nom H en calculant son rang n dans notre liste puis en appelant *select_R4 n* (voir l'exemple ci-dessous).

Une fois la preuve traduite, nous la rejouons dans BCoq pour vérifier qu'elle est bien la preuve de la formule résultant de la normalisation. Par exemple, réifions la preuve de Zenon de Set 1.

Exemple 31 (Réification de la preuve de Set 1)

À partir de la preuve générée par Zenon, nous pouvons construire la preuve réifiée suivante :

```
Lemma set_1_bcoq :
  forall (x0 x1 x : V) (s b a : S),
  x \ (a,b,s) → x0 \ (a,b,s,x) → x1 \ (a,b,s,x,x0) →
  ∅ ⊢ (∀ x0.(x0 ∈ b ⇒ x0 ∈ s)) ⇒ (∀ x1.(x1 ∈ a ⇒ x1 ∈ s)) ⇒
      (x ∈ s ∧ (x ∈ a ∨ x ∈ b)) ⇔ (x ∈ s ∧ (x ∈ b ∨ x ∈ a)).
Proof.
  intros.
  apply bcoq_NNPP. apply RULE_3.
  select_R4 1. apply bcoq_notimply.
  apply RULE_3. apply RULE_3. ...
Qed.
```

On peut noter que des conditions de non-liberté apparaissent dans le type du lemme. Ce sont les conditions de non-liberté générées durant l'étape de normalisation (voir la sous-section 4.3.1).

La preuve réifiée débute en appliquant le tiers exclus qui, dans BCoq, correspond à :

```
Lemma bcoq_NNPP : forall (h : list Ṗ) (p : Ṗ),
  h ⊢ ¬̇ p ⇒ Fal̇se → h ⊢ p.
```

Le prédicat $\dot{\neg}\, p$ *est ensuite introduit en hypothèse grâce à la REGLE 3, qui correspond à l'introduction de l'implication en B. L'étape suivante consiste à appliquer la règle* zenon_notimply *dont le plongement est le suivant :*

```
Lemma bcoq_notimply : forall (h : list Ṗ) (p q : Ṗ),
  h ⊢ p ⇒ ¬̇ q ⇒ Fal̇se → h ⊢ ¬̇ (p ⇒ q) ⇒ Fal̇se.
```

Dans la preuve de Zenon, cette règle est appliquée sur l'hypothèse appelée zenon_G, *mais comme nous l'avons dit précédemment, les hypothèses n'ont pas de nom en B et donc dans le plongement non plus. Pour pallier ce problème, une liste d'association entre les hypothèses et les noms introduits par Zenon est conservée dans laquelle* zenon_G *est au rang 1.* select_R4 *1 permet d'insérer l'hypothèse correspondant à* zenon_G *à droite du*

séquent. bcoq_notimply *peut alors être appliqué.*

Ainsi, le script entier correspondant à la preuve produite par Zenon peut être réifié. Finalement, la preuve réifiée est vérifiée dans l'environnement BCoq, ce qui assure la validation de la preuve de Zenon et du lemme de vérification initial.

4.3.5 Implantation

Plusieurs tactiques ont été nécessaires pour effectuer le processus de preuve que nous avons mis en place avec Zenon. Tout d'abord, une tactique $\mathcal{L}_{\text{tac}}$ semblable à la tactique présentée dans la sous-section 3.3.1 permet de normaliser une formule BCoq. Ensuite, une tactique native OCaml interprète la formule BCoq, appelle Zenon puis (re)réifie la preuve générée par Zenon.

Il est possible, grâce à $\mathcal{L}_{\text{tac}}$, de développer des tactiques que l'on appelle native en OCaml. Cela permet d'élargir le champ d'action d'une tactique, pour par exemple d'interfacer Coq avec un outil externe. Dans notre cas, nous voulons manipuler les termes BCoq pour les interpréter en Coq (la formule Coq sera ensuite donnée en entrée à Zenon), puis appeler Zenon. La tactique native est donc une solution.

À partir d'une formule Coq normalisée notée φ^{BCoq}, la tactique OCaml (illustrée par la figure 4.3) effectue les étapes suivantes :

1. Interprétation de la formule BCoq du premier ordre φ^{BCoq} en une formule du premier ordre φ ;
2. Appel de Zenon sur φ, puis conservation de la preuve Coq $P\varphi$ générée par Zenon dans un fichier ;
3. Traduction (réification) syntaxique de la preuve Coq $P\varphi$ en une preuve BCoq $P\varphi^{\mathsf{BCoq}}$, puis génération d'un fichier contenant la preuve ;
4. Compilation de la preuve $P\varphi^{\mathsf{BCoq}}$ afin d'obtenir un fichier .vo ;
5. Importation de ce fichier, puis application de la preuve $P\varphi^{\mathsf{BCoq}}$.

Si la dernière étape se déroule avec succès alors nous avons une preuve BCoq de la formule φ^{BCoq}.

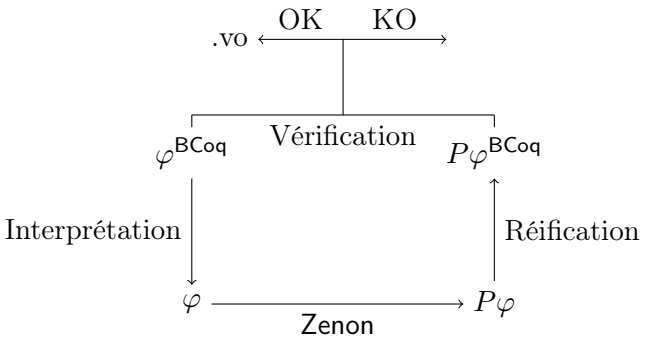

FIGURE 4.3 – Détails de la tactique OCaml

4.4 Résultats expérimentaux

Nous présentons dans cette section les résultats expérimentaux que nous avons obtenus. Nous considérons tout d'abord 200 propriétés du B-Book [1] (ou théorèmes dérivés). Certaines de nos règles ajoutées font partie de ces propriétés. Les résultats sont résumés dans le graphe de la figure 4.4, où un point représente le test d'une propriété et les axes des abscisses et des ordonnées correspondent respectivement aux temps mis par la tactique $\mathcal{L}_{\text{tac}}$ et par celle utilisant Zenon pour trouver une preuve (exprimés en secondes). Dans ce graphe, nous considérons uniquement les propriétés pour lesquelles le temps de preuve des deux approches est au maximum de 30s (ce qui correspond environ à 66% des propriétés testées). Nous pouvons voir que Zenon est plus rapide que la tactique $\mathcal{L}_{\text{tac}}$ pour la majorité des propriétés testées (pour 71% d'entre elles). De plus, sur les 200 propriétés pour lesquelles Zenon trouve une preuve, 15 propriétés ne sont pas démontrées avec la tactique $\mathcal{L}_{\text{tac}}$.

Nous avons aussi testé nos tactiques sur la base de règle de Siemens IC-MOL dans les mêmes conditions que le test pour la tactique $\mathcal{L}_{\text{tac}}$ à savoir sur un ordinateur Intel Pentium D Xeon 3.60GHz/32Go avec des limitations par processus de 30 minutes et de 10 Go en consommation mémoire. Sur les 2031 règles bien typées et bien définies de cette base (voir la section 3.4), la tactique $\mathcal{L}_{\text{tac}}$ en démontre 856 et notre nouvelle approche en montre 1382. Dans l'ensemble de règles testé, 238 règles ne peuvent pas être démontrées par l'approche utilisant Zenon car elles contiennent des opérateurs non traités. Ces règles

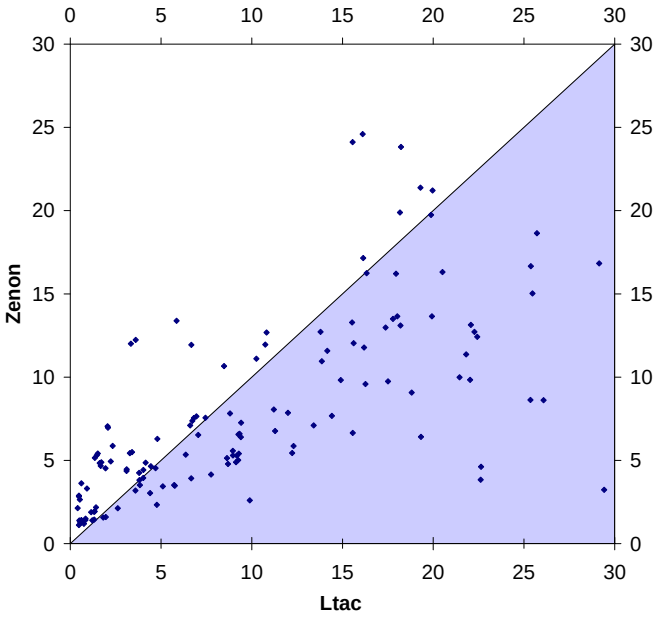

FIGURE 4.4 – Temps de preuves des propriétés du B-Book utilisant Zenon et $\mathcal{L}_{\text{tac}}$

pouvaient être prouvées par $\mathcal{L}_{\text{tac}}$ si la preuve était propositionnelle. Donc, notre approche démontre 1382 règles sur 1793 règles soit 77%. Sur ce même sous-ensemble, l'approche avec $\mathcal{L}_{\text{tac}}$ en démontre 843 soit 47%.

Les échecs de l'approche utilisant Zenon sont détaillés dans le tableau 4.1. Cette approche, contrairement à la précédente, permet de trouver facilement des règles fausses. Ainsi, lorsque l'appel de Zenon produit un contre-modèle, soit il y a eu un problème dans la normalisation, soit la règle est fausse. Cela a permis de découvrir 4 nouvelles règles fausses. Nous pouvons remarquer que là encore Coq est très consommateur en mémoire puisque pour 26 règles, le processus Coq dépasse les 10 Go (colonne « Mémoire »). Concernant les problèmes de temps, ils sont liés aux étapes de normalisation et de rejeu de la preuve BCoq, toutes deux très consommatrices en temps. Cela concerne 22 règles (colonne « Temps »). Le temps de l'appel de Zenon est quant à lui très faible : en moyenne, Zenon met 0,03 secondes pour trouver une preuve. Les problèmes techniques sont encore nombreux (rencontrés pour 358 règles) mais concrètement ce sont des détails techniques d'implantation et non des problèmes théoriques (colonne « Technique »). Enfin, pour une seule règle, Zenon ne trouve pas de preuve et ne génère pas de contre-modèle (colonne « Autre »).

	Règles fausses	Mémoire	Temps	Technique	Autre
Nombre de règles	4	26	22	358	1

TABLE 4.1 – Analyse des échecs

En utilisant Zenon, nous avons donc considérablement amélioré la recherche de preuve. Toutefois, cette approche est complexe à mettre en place techniquement, notamment au niveau de la normalisation et elle ne s'avère pas encore assez efficace en terme de temps.

Chapitre 5

Extension de Zenon à la théorie de B

L'approche qui permet de prouver le lemme de validité (de la règle à vérifier) avec Zenon est satisfaisante mais en pratique assez lente à cause de l'étape de normalisation qui peut se révéler longue. Nous avons donc étendu Zenon à la théorie de B, afin de supprimer totalement l'étape de normalisation. Cette approche est possible car Zenon est un prouveur automatique extensible (voir [17]). En effet, il offre la possibilité d'étendre le noyau de ses règles de recherche de preuve pour traiter des besoins spécifiques.

Il existe plusieurs techniques pour raisonner modulo une théorie dont la déduction modulo [36] et la superdéduction [20] présentées dans la section 5.1. Nous avons choisi d'utiliser la superdéduction car elle introduit uniquement de nouvelles règles de déduction et Zenon offre justement la possibilité d'étendre son noyau de règles. Nous avons donc adapté la superdéduction à la méthode des tableaux de Zenon (voir section 5.2), puis nous l'avons appliquée à la théorie de la méthode B (voir section 5.3) pour calculer de nouvelles règles de déduction traitant des opérateurs B. L'implantation de l'approche est présentée dans la section 5.4. Nous avons obtenu de bons résultats expérimentaux, notamment en améliorant considérablement le temps de recherche de preuve par rapport à l'approche avec Zenon sans extension (voir section 5.5). Ce travail a été publié dans [43].

5.1 Raisonner modulo une théorie

Plusieurs solutions permettent d'étendre un système déductif afin de raisonner modulo une théorie. Les travaux présentés dans [52], par exemple, permettent d'ajouter des axiomes

sans variable à un calcul des séquents, sous la forme de règles d'inférence non logique. Dans [24], les axiomes sont transformés en règles d'inférence équivalentes dans les calculs des séquents et des hyperséquents. Les règles d'élimination et d'introduction de Prawitz [54] (« folding/unfolding ») peuvent aussi permettre d'étendre un système déductif.

Ainsi, si l'on suit l'idée de Prawitz pour étendre Zenon à la théorie des ensembles, le principe est d'ajouter une règle à Zenon par nouveau constructeur. Par exemple, pour l'inclusion définie par $\forall a \forall b\ ((a \subseteq b) \Leftrightarrow (\forall x\ (x \in a \Rightarrow x \in b)))$, on peut ajouter la règle d'introduction $\dfrac{\Gamma \vdash \forall x\ (x \in a \Rightarrow x \in b)}{\Gamma \vdash a \subseteq b}$.

Néanmoins, nous voulons aller plus loin que les règles d'introduction de manière à diminuer le nombre d'étapes de preuve et par conséquent la taille des preuves produites. Il existe plusieurs techniques permettant d'étendre un système de déduction tout en réduisant le nombre d'étapes dont la déduction modulo [36] et la superdéduction [20] que nous présentons ci-dessous.

5.1.1 Déduction modulo

La déduction modulo [36] est une extension de la logique du premier ordre où les axiomes sont remplacés par des règles de réécritures. Par exemple, $P \Leftrightarrow (Q \Rightarrow R)$ est remplacé par la règle $P \rightarrow (Q \Rightarrow R)$. Ces règles définissent une relation d'équivalence et, dans une preuve, une proposition peut être remplacée par une autre proposition équivalente à n'importe quel moment. Cela peut être explicité dans l'arbre de preuve par les règles suivantes pour le calcul des séquents (où $\mathcal{R}$ est l'ensemble des règles de réécriture et $\equiv_{\mathcal{R}}$ est la relation d'équivalence induite par $\mathcal{R}$) :

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, B \vdash \Delta}\ A \equiv_{\mathcal{R}} B \qquad\qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash B, \Delta}\ A \equiv_{\mathcal{R}} B$$

La déduction modulo permet notamment de réduire l'espace de recherche de preuve car toute la partie calculatoire est effectuée par la réécriture.

Par exemple, considérons la définition de l'inclusion dans la théorie des ensembles $\forall a \forall b\ ((a \subseteq b) \Leftrightarrow (\forall x\ (x \in a \Rightarrow x \in b)))$. La preuve de $A \subseteq A$ dans le calcul des séquents est la suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ldots, x \in A \vdash A \subseteq A, x \in A}\,\mathsf{Ax}}{\ldots \vdash A \subseteq A, x \in A \Rightarrow x \in A}\,\Rightarrow\!\mathsf{R}}{\ldots \vdash A \subseteq A, \forall x\ (x \in A \Rightarrow x \in A)}\,\forall\mathsf{R} \quad \dfrac{}{\ldots, A \subseteq A \vdash A \subseteq A}\,\mathsf{Ax}}{\ldots, (\forall x\ (x \in A \Rightarrow x \in A)) \Rightarrow A \subseteq A \vdash A \subseteq A}\,\Rightarrow\!\mathsf{L}}{A \subseteq A \Leftrightarrow (\forall x\ (x \in A \Rightarrow x \in A)) \vdash A \subseteq A}\,\wedge\mathsf{L}}{\forall a \forall b\ ((a \subseteq b) \Leftrightarrow (\forall x\ (x \in a \Rightarrow x \in b))) \vdash A \subseteq A}\,\forall\mathsf{L} \times 2$$

En déduction modulo, l'axiome de l'inclusion peut être vu comme une règle de calcul et donc, il peut être remplacé par la règle de réécriture $a \subseteq b \rightarrow \forall x\ (x \in a \Rightarrow x \in b)$. La précédente preuve est alors transformée de la manière suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \in A \vdash x \in A}\,\mathsf{Ax}}{\vdash x \in A \Rightarrow x \in A}\,\Rightarrow\!\mathsf{R}}{\vdash \forall x\ (x \in A \Rightarrow x \in A)}\,\forall\mathsf{R}}{\vdash A \subseteq A}\; A \subseteq A \equiv_{\mathcal{R}} \forall x\ (x \in A \Rightarrow x \in A)$$

La preuve obtenue est bien plus courte que celle utilisant le calcul des séquents. La déduction modulo permet en effet de raccourcir considérablement la taille des preuves [22].

5.1.2 Superdéduction

Une deuxième technique appelée superdéduction [20] permet de prendre en compte des axiomes (ou une théorie) dans un système de déduction, en ajoutant de nouvelles règles de déduction appelées super-règles. C'est une amélioration de la déduction super-naturelle proposée dans [61].

La superdéduction est une variante de la déduction modulo puisqu'à partir d'un axiome de la forme d'une équivalence $\forall \bar{x}\ (P \Leftrightarrow \varphi)$ où P est une formule atomique, la superdéduction, tout comme la déduction modulo, remplace l'axiome par une règle de réécriture qui permet de transformer P en φ. Ensuite, cette règle de réécriture va permettre de calculer une nouvelle règle de déduction « simplifiée » dans le sens où la superdéduction permet de supprimer tous les connecteurs apparaissant dans φ, en une seule étape de déduction. Par exemple, pour l'inclusion ensembliste, la déduction modulo permet de remplacer l'axiome par une règle de réécriture qui transforme $a \subseteq b$ en $\forall x\ (x \in a \Rightarrow x \in b)$. La superdéduction permet en plus de supprimer le $\forall$ et le $\Rightarrow$. Cela correspond à une extension des règles d'introduction de Prawitz où toutes les règles d'introduction des connecteurs sont appliquées pour créer une nouvelle règle plus directe.

La nouvelle règle de l'inclusion se déduit de l'arbre suivant :

$$\dfrac{\dfrac{\dfrac{\Gamma, x \in a \vdash x \in b, \Delta}{\Gamma \vdash x \in a \Rightarrow x \in b, \Delta}\,\Rightarrow\!\mathsf{R}}{\Gamma \vdash \forall x\ (x \in a \Rightarrow x \in b), \Delta}\,\forall\mathsf{R},\ x \notin \Gamma, \Delta}{\Gamma \vdash a \subseteq b, \Delta}$$

La super-règle proposée est donc :

$$\dfrac{\Gamma, x \in a \vdash x \in b, \Delta}{\Gamma \vdash a \subseteq b, \Delta}\,\mathsf{IncR},\ x \notin \Gamma, \Delta$$

Ainsi, en utilisant la nouvelle règle, la preuve de $A \subseteq A$ devient :

$$\dfrac{\dfrac{}{x \in A \vdash x \in A}\,\mathsf{Ax}}{\vdash A \subseteq A}\,\mathsf{IncR}$$

Une super-règle gauche est associée (en appliquant la même technique) :

$$\dfrac{\Gamma \vdash t \in a, \Delta \qquad \Gamma, t \in b \vdash \Delta}{\Gamma, a \subseteq b \vdash \Delta}\,\mathsf{IncL}$$

La preuve de $A \subseteq A$ est plus courte que celle obtenue avec la déduction modulo. La règle IncR est très proche du raisonnement humain utilisé dans les preuves mathématiques. En effet, elle peut se lire : « Pour prouver que $a \subseteq b$, soit x tel que $x \in a$ alors il faut prouver que $x \in b$ ». On obtient donc avec la superdéduction des preuves proches de celles réalisées usuellement. De plus, le calcul des super-règles est systématique donc facilement automatisable. C'est pour ces raisons, et le fait que Zenon est un prouveur extensible, que nous avons choisi d'étendre Zenon à la méthode B avec la superdéduction.

5.2 La superdéduction pour la méthode des tableaux

La superdéduction est dérivée de la déduction super-naturelle introduite par Benjamin Wack dans [61]. Ensuite, dans [20], Paul Brauner et al. définissent la superdéduction pour le calcul des séquents. Cela est étendu à la superdéduction modulo par Clément Houtmann dans [40], qui adapte en particulier une méthode des tableaux définie pour la déduction modulo (TaMeD [16]) à la superdéduction modulo.

Pour raisonner modulo une théorie avec la superdéduction, il faut générer de nouvelles règles de déduction à partir de certains axiomes de la théorie. Les axiomes concernés par la superdéduction sont ceux de la forme $\forall \bar{x}\ (P \Leftrightarrow \varphi)$, où P est atomique. Cette forme spécifique d'axiome permet d'introduire une orientation de l'axiome allant de P vers φ qui correspond à la notion de règle de réécriture propositionnelle, introduite dans [20] et présentée ci-dessous :

Définition 4 (Règle de réécriture propositionnelle)
La notation $R : P \rightarrow \varphi$ représente l'axiome $\forall \bar{x}\ (P \Leftrightarrow \varphi)$, où R est le nom de la règle, P une proposition atomique, φ une proposition quelconque, et $\bar{x}$ les variables libres de P et φ.

P peut contenir des termes du premier ordre. Par exemple, $x \in a \cup b \rightarrow x \in a \vee x \in b$ est une règle de réécriture propositionnelle.

Notre objectif est de développer une procédure de recherche de preuve pour la théorie des ensembles de B en utilisant le prouveur automatique Zenon. Dans la suite, nous considérerons donc la méthode des tableaux implantée par Zenon comme méthode de preuve dans laquelle les règles de superdéduction seront produites à partir des règles de réécriture propositionnelles.

La recherche de preuve de Zenon est expliquée succinctement dans le chapitre précédent et détaillée dans [17]. Les règles de recherche de preuve sont exposées dans la figure 4.2 page 103. Nous allons maintenant décrire le calcul des super-règles pour Zenon à partir d'une règle de réécriture propositionnelle.

Définition 5 (Calcul des super-règles)
Soit $\mathcal{S}$ un ensemble de règles composé des règles de clôture, des règles analytiques et des règles $\gamma_{\forall M}$ et $\gamma_{\neg\exists M}$ issues des règles de recherche de preuve de Zenon. Étant donnée une règle de réécriture propositionnelle $R : P \rightarrow \varphi$, deux règles de superdéduction (une règle dite positive notée R et une règle dite négative notée $\neg R$) sont calculées de la manière suivante :

1. *Pour obtenir la règle positive R, initialiser la procédure avec la formule φ. Puis, appliquer les règles de $\mathcal{S}$ jusqu'à ce qu'il n'y ait plus de feuilles sur lesquelles elles*

peuvent être appliquées. Ensuite, collecter les prémisses et la conclusion, et remplacer φ par P pour obtenir la règle positive R.

2. *Pour obtenir la règle négative $\neg R$, initialiser la procédure avec la formule $\neg\varphi$. Puis, appliquer les règles de $\mathcal{S}$ jusqu'à ce qu'il n'y ait plus de feuilles sur lesquelles elles peuvent être appliquées. Ensuite, collecter les prémisses et la conclusion, et remplacer $\neg\varphi$ par $\neg P$ pour obtenir la règle négative $\neg R$.*

Si la règle R (resp. $\neg R$) contient des métavariables, des règles d'instanciation R_{inst} (resp. $\neg R_{\mathsf{inst}}$) sont ajoutées, où une ou plusieurs métavariables peuvent être instanciées. L'utilisation de ces règles sera expliquée dans la sous-section 5.3.4.

Illustrons le calcul des super-règles à partir d'une règle de réécriture propositionnelle correspondant à la définition de l'inclusion ensembliste.

Exemple 32 (Inclusion ensembliste)
À partir de la définition de l'inclusion ensembliste, nous introduisons la règle de réécriture propositionnelle $\mathsf{Inc} : a \subseteq b \rightarrow \forall x\ (x \in a \Rightarrow x \in b)$. *Les super-règles correspondantes* Inc *et* $\neg$Inc *sont générées de cette manière en utilisant les règles de Zenon :*

$$\dfrac{\dfrac{\forall x\ (x \in a \Rightarrow x \in b)}{X \in a \Rightarrow X \in b}\,\gamma_{\forall M}}{X \notin a \mid X \in b}\,\beta_{\Rightarrow} \qquad \dfrac{\dfrac{\neg\forall x\ (x \in a \Rightarrow x \in b)}{\neg(\epsilon_x \in a \Rightarrow \epsilon_x \in b)}\,\delta_{\neg\forall}}{\epsilon_x \in a, \epsilon_x \notin b}\,\alpha_{\neg\Rightarrow}$$

où $\epsilon_x = \epsilon(x).\neg(x \in a \Rightarrow x \in b)$.

Les super-règles sont alors les suivantes :

$$\dfrac{a \subseteq b}{X \notin a \mid X \in b}\,\mathsf{Inc} \qquad \dfrac{a \subseteq b}{t \notin a \mid t \in b}\,\mathsf{Inc}_{\mathsf{inst}} \qquad \dfrac{a \not\subseteq b}{\epsilon_x \in a, \epsilon_x \notin b}\,\neg\mathsf{Inc}$$

5.3 Application à la théorie de B

Le but de cette section est de construire un système de super-règles pour la théorie des ensembles de B. Pour cela, l'idée est d'appliquer l'algorithme de calcul des super-règles de la section 5.2 sur les axiomes de la théorie des ensembles de B.

5.3.1 Génération des super-règles

Dans cette sous-section, nous considérons seulement les axiomes et constructions de la théorie des ensembles de B définis dans la figure 5.1. Ces définitions diffèrent de celles présentées dans l'annexe B par l'absence total de typage. En effet, nous nous sommes abstraits du typage ici pour simplifier la recherche de preuve comme cela est proposé par Mussat et Abrial dans [4] qui modularisent typage et preuve (même si aucune démonstration n'a été faite sur ce sujet).

Nous calculons les règles de superdéduction correspondant à ces axiomes et constructions « à la main », en utilisant l'algorithme décrit dans la définition 5 de la section 5.2. Nous identifions plusieurs types de règle de réécriture propositionnelle. D'une part, les axiomes sont tous de la forme $P_i \Leftrightarrow Q_i$, et les règles de réécriture propositionnelle correspondantes sont de la forme $R_i : P_i \rightarrow Q_i$, où chaque axiome est orienté de la gauche vers la droite. D'autre part, les constructions sont exprimées par les définitions $E_i \triangleq F_i$, où E_i et F_i sont des expressions. Les règles de réécriture propositionnelle associées sont de la forme $R_i : x \in E_i \rightarrow x \in F_i$ (où x est non libre dans E_i et F_i) puisque E_i et F_i sont des expressions ensemblistes et non des propositions. Si E_i et F_i représentent des relations, les règles de réécriture propositionnelle sont de la forme $R_i : (x, y) \in E_i \rightarrow (x, y) \in F_i$. Les super-règles obtenues à partir de cet ensemble de règles de réécriture propositionnelle sont présentées dans les figures 5.2 et 5.3. Concernant les règles sur l'ensemble vide, il n'y a qu'une seule règle puisque la règle négative est inutile pour la recherche de preuve $x \notin \emptyset$ étant une tautologie. Les autres constructions qui s'appuient sur la fonction partielle (c'est-à-dire les fonctions totale, bijective, injective, ...) ne sont pas présentées ici car les règles de superdéduction associées sont aussi complexes que celle pour la fonction partielle. Comme elles peuvent être déduites respectivement des règles $\mathbb{P}$ et « ; », nous omettons les super-règles de $\subseteq$ (ainsi que de $\subset$), et de $\circ$ dans ces figures. De plus, pour simplifier, nous ne présentons pas les règles d'instanciation associées à chaque règle contenant une métavariable. En tout, pour les 28 axiomes présentés ici, il y a 55 super-règles calculées (dont 49 sont présentées ici) sans compter les règles d'instanciation associées. En règle générale, il y a 2 super-règles générées par axiome (une positive et une négative) sauf pour l'axiome de l'ensemble vide.

Axiomes

$$(x, y) \in a \times b \Leftrightarrow x \in a \wedge y \in b$$
$$a \in \mathbb{P}(b) \Leftrightarrow \forall x\ (x \in a \Rightarrow x \in b)$$
$$x \in \{\ y \mid P(y)\ \} \Leftrightarrow P(x)$$
$$a = b \Leftrightarrow \forall x\ (x \in a \Leftrightarrow x \in b)$$

Constructions dérivées

$$a \subseteq b \Leftrightarrow a \in \mathbb{P}(b)$$
$$a \subset b \Leftrightarrow a \subseteq b \wedge a \neq b$$
$$a \cup b \triangleq \{\ x \mid x \in a \vee x \in b\ \}$$
$$a \cap b \triangleq \{\ x \mid x \in a \wedge x \in b\ \}$$
$$a - b \triangleq \{\ x \mid x \in a \wedge x \notin b\ \}$$
$$\emptyset \triangleq \mathsf{BIG} - \mathsf{BIG}$$
$$\{\ e_1, \ldots, e_n\ \} \triangleq \{\ x \mid x = e_1\ \} \cup \ldots \cup \{\ x \mid x = e_n\ \}$$

Relations binaires : première série

$$a^{-1} \triangleq \{\ (y, x) \mid (x, y) \in a\ \}$$
$$\mathsf{dom}(a) \triangleq \{\ x \mid \exists y\ (x, y) \in a\ \}$$
$$\mathsf{ran}(a) \triangleq \mathsf{dom}(a^{-1})$$
$$a; b \triangleq \{\ (x, z) \mid \exists y\ ((x, y) \in a \wedge (y, z) \in b\ \}$$
$$a \circ b \triangleq b; a$$
$$\mathsf{id}(a) \triangleq \{\ (x, y) \mid (x, y) \in a \times a \wedge x = y\ \}$$
$$a \lhd b \triangleq \mathsf{id}(a); b$$
$$a \rhd b \triangleq a; \mathsf{id}(b)$$
$$a \lhd\!\!\!\cdot\ b \triangleq (\mathsf{dom}(b) - a) \lhd b$$
$$a \rhd\!\!\!\cdot\ b \triangleq a \rhd (\mathsf{ran}(a) - b)$$

Relations binaires : seconde série

$$a[b] \triangleq \mathsf{ran}(b \lhd a)$$
$$a \Leftarrow b \triangleq (\mathsf{dom}(b) \lhd\!\!\!\cdot\ a) \cup b$$
$$a \otimes b \triangleq \{\ (x, (y, z)) \mid (x, y) \in a \wedge (x, z) \in b\ \}$$
$$\mathsf{prj}_1(a, b) \triangleq (\mathsf{id}(a) \otimes (a \times b))^{-1}$$
$$\mathsf{prj}_2(a, b) \triangleq ((b \times a) \otimes \mathsf{id}(b))^{-1}$$
$$a \parallel b \triangleq \{\ (x, y), (z, t) \mid (x, z) \in a \wedge (y, t) \in b\ \}$$

Fonctions

$$a \nrightarrow b \triangleq \{\ t \mid t \in a \leftrightarrow b \wedge \forall (x, y, z)\ ((x, y) \in t \wedge (x, z) \in t \Rightarrow y = z)\ \}$$

FIGURE 5.1 – Axiomes et constructions de la théorie des ensembles de B

Lors du calcul de ces règles, nous sommes allés au delà du calcul des super-règles proposé dans la section 5.2, puisque étant donnée une règle de réécriture propositionnelle $R : P \rightarrow Q$, nous appliquons sur Q non seulement les règles considérées dans la définition 5, mais aussi toutes les nouvelles super-règles générées avant (sauf les règles pour l'égalité extensionnelle pour bénéficier des règles dédiées de Zenon pour gérer l'égalité) tant que cela est possible. Cela permet d'obtenir des super-règles qui remplacent un maximum d'étapes de déduction, c'est-à-dire qui déplient tous les opérateurs et introduisent tous les connecteurs logiques.

5.3.2 Traitement des relations

Comme nous l'avons mentionné précédemment, les super-règles pour les relations sont calculées à partir de règles de réécriture propositionnelle de la forme $R : (x, y) \in E \rightarrow (x, y) \in F$, où E et F sont des relations, et les conclusions de ces règles sont donc soit $(x, y) \in E$, soit $(x, y) \notin E$. Par conséquent, ces règles ne peuvent être appliquées sur les formules dans lesquelles les paires ne sont pas explicites telles que $x \in E$ ou $x \notin E$. Pour traiter ce problème, l'idée est d'ajouter une autre règle de réécriture propositionnelle pour le produit cartésien, c'est-à-dire la règle $\times^* : x \in a \times b \rightarrow \exists y \exists z\ (x = (y, z) \wedge (y, z) \in a \times b)$, à partir de laquelle les deux super-règles suivantes peuvent être générées :

$$\frac{x \in a \times b}{x = (\epsilon_y, \epsilon_z), \epsilon_y \in a, \epsilon_z \in b}\ \times^* \qquad \frac{x \notin a \times b}{x \neq (Y, Z) \mid Y \notin a \mid Z \notin b}\ \neg\times^*$$

avec pour la première règle, $\epsilon_y = \epsilon(y).(\exists z\ (x = (y, z) \wedge (y, z) \in a \times b))$, et $\epsilon_z = \epsilon(z).(x = (\epsilon_y, z) \wedge (\epsilon_y, z) \in a \times b))$.

Pour chaque opérateur p défini par une relation, nous avons aussi besoin d'ajouter une autre règle de réécriture propositionnelle de la forme $p^* : x \in p \rightarrow \exists y \exists z\ (x = (y, z) \wedge (y, z) \in p)$, et donc générer les super-règles correspondantes. Par exemple, pour la relation inverse nous obtenons les règles suivantes :

$$\frac{x \in a^{-1}}{x = (\epsilon_y, \epsilon_z), (\epsilon_z, \epsilon_y) \in a}\ a^{-1*} \qquad \frac{x \notin a^{-1}}{x \neq (Y, Z) \mid (Z, Y) \notin a}\ \neg a^{-1*}$$

avec pour la première règle, $\epsilon_y = \epsilon(y).(\exists z\ (x = (y, z) \wedge (y, z) \in a^{-1}))$, et $\epsilon_z = \epsilon(z).(x = (\epsilon_y, z) \wedge (\epsilon_y, z) \in a^{-1})$.

Règles pour les axiomes

$$\frac{(x,y) \in a \times b}{x \in a, y \in b}\ \times \qquad \frac{a \in \mathbb{P}(b)}{X \notin a \mid X \in b}\ \mathbb{P} \qquad \frac{x \in \{\ y \mid P(y)\ \}}{P(x)}\ \{|\}$$

$$\frac{(x,y) \notin a \times b}{x \notin a \mid y \notin b}\ \neg\times \qquad \frac{a \notin \mathbb{P}(b)}{\epsilon_x \in a, \epsilon_x \notin b}\ \neg\mathbb{P} \quad \text{avec } \epsilon_x = \epsilon(x).\neg(x \in a \Rightarrow x \in b) \qquad \frac{x \notin \{\ y \mid P(y)\ \}}{\neg P(x)}\ \neg\{|\}$$

$$\frac{a = b}{X \notin a, X \notin b \mid X \in a, X \in b}\ = \qquad \frac{a \neq b}{\epsilon_x \notin a, \epsilon_x \in b \mid \epsilon_x \in a, \epsilon_x \notin b}\ \neq \quad \text{avec } \epsilon_x = \epsilon(x).\neg(x \in a \Leftrightarrow x \in b)$$

Règles pour les constructions dérivées

$$\frac{x \in a \cup b}{x \in a \mid x \in b}\ \cup \qquad \frac{x \in a \cap b}{x \in a, x \in b}\ \cap \qquad \frac{x \in a - b}{x \in a, x \notin b}\ -$$

$$\frac{x \notin a \cup b}{x \notin a, x \notin b}\ \neg\cup \qquad \frac{x \notin a \cap b}{x \notin a \mid x \notin b}\ \neg\cap \qquad \frac{x \notin a - b}{x \notin a \mid x \in b}\ \neg-$$

$$\frac{x \in \{\ e_1, \ldots, e_n\ \}}{x = e_1 \mid \ldots \mid x = e_1}\ \{\} \qquad \frac{x \notin \{\ e_1, \ldots, e_n\ \}}{x \neq e_1, \ldots, x \neq e_n}\ \neg\{\} \qquad \frac{x \in \emptyset}{\odot}\ \emptyset$$

Règles pour les relations binaires : première série

$$\frac{(x,y) \in a^{-1}}{(y,x) \in a}\ a^{-1} \qquad \frac{x \in \mathsf{dom}(a)}{(x,\epsilon_y) \in a}\ \mathsf{dom} \quad \text{avec } \epsilon_y = \epsilon(y).((x,y) \in a) \qquad \frac{y \in \mathsf{ran}(a)}{(\epsilon_x, y) \in a}\ \mathsf{ran} \quad \text{avec } \epsilon_x = \epsilon(x).((x,y) \in a)$$

$$\frac{(x,y) \notin a^{-1}}{(y,x) \notin a}\ \neg a^{-1} \qquad \frac{x \notin \mathsf{dom}(a)}{(x,Y) \notin a}\ \neg\mathsf{dom} \qquad \frac{y \notin \mathsf{ran}(a)}{(X,y) \notin a}\ \neg\mathsf{ran}$$

$$\frac{(x,z) \in a;b}{(x,\epsilon_y) \in a,\ (\epsilon_y, z) \in b}\ ; \quad \text{avec } \epsilon_y = \epsilon(y).((x,y) \in a \wedge (y,z) \in b) \qquad \frac{(x,z) \notin a;b}{(x,Y) \notin a \mid (Y,z) \notin b}\ \neg;$$

$$\frac{(x,y) \in \mathsf{id}(a)}{x = y,\ x \in a,\ y \in a}\ id \qquad \frac{(x,y) \notin \mathsf{id}(a)}{x \neq y \mid x \notin a \mid y \notin a}\ \neg id$$

$$\frac{(x,y) \in a \lhd b}{(x,y) \in b,\ x \in a}\ \lhd \qquad \frac{(x,y) \notin a \lhd b}{(x,y) \notin b \mid x \notin a}\ \neg\lhd$$

FIGURE 5.2 – Super-règles pour la théorie des ensemble de B (Partie 1)

$$\frac{(x,y) \in a \lhd b}{(x,y) \in b,\ x \notin a}\ \lhd \qquad \frac{(x,y) \in a \rhd b}{(x,y) \in a,\ y \in b}\ \rhd \qquad \frac{(x,y) \in a \rhd\!\!> b}{(x,y) \in a,\ y \notin b}\ \rhd\!\!>$$

$$\frac{(x,y) \notin a \lhd b}{(x,y) \notin b \mid x \in a}\ \neg\lhd \qquad \frac{(x,y) \notin a \rhd b}{(x,y) \notin a \mid y \notin b}\ \neg\rhd \qquad \frac{(x,y) \notin a \rhd\!\!> b}{(x,y) \notin a \mid y \in b}\ \neg\rhd\!\!>$$

Règles pour les relations binaires : seconde série

$$\frac{y \in a[b]}{\epsilon_x \in b, (\epsilon_x, y) \in a}\ a[b] \quad \text{avec } \epsilon_x = \epsilon(x).(x \in b \wedge (x,y) \in a) \qquad \frac{(x,y) \in a \lhd\!\!\!- b}{(x,y) \in a,\ (x,Y) \notin b \mid (x,y) \in b}\ \lhd\!\!\!-$$

$$\frac{y \notin a[b]}{X \notin b \mid (X,y) \notin a}\ \neg a[b] \qquad \frac{(x,y) \notin a \lhd\!\!\!- b}{(x,y) \notin a,\ (x,y) \notin b \mid (x,\epsilon_y) \in b,\ (x,y) \notin b}\ \neg\lhd\!\!\!- \quad \text{avec } \epsilon_y = \epsilon(y).((x,y) \in b)$$

$$\frac{(x,(y,z)) \in a \otimes b}{(x,y) \in a,\ (x,z) \in b}\ \otimes \qquad \frac{((x,y),z) \in \mathsf{prj}_1(a,b)}{x \in a,\ y \in b,\ z \in a,\ z = x}\ \mathsf{prj}_1$$

$$\frac{(x,(y,z)) \in a \otimes b}{(x,y) \notin a \mid (x,z) \notin b}\ \neg\otimes \qquad \frac{((x,y),z) \notin \mathsf{prj}_1(a,b)}{x \notin a \mid y \notin b \mid z \notin a \mid z \neq x}\ \neg\mathsf{prj}_1$$

$$\frac{((x,y),z) \in \mathsf{prj}_2(a,b)}{x \in a,\ y \in b,\ z \in b,\ z = y}\ \mathsf{prj}_2 \qquad \frac{((x,y),(z,t)) \in a \parallel b}{(x,z) \in a,\ (y,t) \in b}\ \parallel$$

$$\frac{((x,y),z) \notin \mathsf{prj}_2(a,b)}{x \notin a \mid y \notin b \mid z \notin b \mid z \neq y}\ \neg\mathsf{prj}_2 \qquad \frac{((x,y),(z,t)) \notin a \parallel b}{(x,z) \notin a \mid (y,t) \notin b}\ \neg\parallel$$

Règles pour les fonctions

$$\frac{a \in b \nrightarrow c}{\begin{array}{c} T \notin a, (X,Y) \notin a \mid T \notin a, (X,Z) \notin a \mid T \notin a, Y = Z \mid \\ \Delta, (X,Y) \notin a \mid \Delta, (X,Z) \notin a \mid \Delta, Y = Z \\ \text{avec } \Delta \equiv T = (\epsilon_x, \epsilon_y), \epsilon_x \in b, \epsilon_y \in c \\ \text{où } \epsilon_x = \epsilon(x).\exists y\ (T = (x,y) \wedge (x,y) \in b \times c) \\ \epsilon_y = \epsilon(y).(T = (\epsilon_x, y) \wedge (\epsilon_x, y) \in b \times c) \end{array}}\ \nrightarrow$$

$$\frac{a \notin b \nrightarrow c}{\begin{array}{c} \epsilon'_x \in a, \epsilon'_x \neq (Y,Z) \mid \epsilon'_x \in a, Y \notin b \mid \epsilon'_x \in a, Z \notin c \mid \\ (\epsilon_x, \epsilon_y) \in a,\ (\epsilon_x, \epsilon_z) \in a,\ \epsilon_y \neq \epsilon_z \\ \text{avec } \epsilon'_x = \epsilon(x).\neg(x \in a \Rightarrow x \in b \times c) \\ \epsilon_x = \epsilon(x).\neg(\forall y \forall z\ ((x,y) \in a \wedge (x,z) \in a \Rightarrow y = z)) \\ \epsilon_y = \epsilon(y).\neg(\forall z\ ((\epsilon_x, y) \in a \wedge (\epsilon_x, z) \in a \Rightarrow y = z)) \\ \epsilon_z = \epsilon(z).\neg((\epsilon_x, \epsilon_y) \in a \wedge (\epsilon_x, z) \in a \Rightarrow \epsilon_y = z) \end{array}}\ \neg\nrightarrow$$

FIGURE 5.3 – Super-règles pour la théorie des ensemble de B (Partie 2)

Avec ces nouvelles règles, nous pouvons prouver par exemple que $p \in \mathbb{P}(a \times b) \Rightarrow x \in p \Rightarrow x \in (p^{-1})^{-1}$ de la manière suivante (nous débutons la preuve après l'application d'une série de $\alpha_{\neg\Rightarrow}$, et nous ne détaillons pas les définitions des ϵ-termes introduits) :

$$\dfrac{\dfrac{p \in \mathbb{P}(a \times b), x \in p, x \notin (p^{-1})^{-1}}{X \notin p}\,\mathbb{P} \qquad \dfrac{\dfrac{X \in a \times b}{X = (\epsilon_y, \epsilon_z), \epsilon_y \in a, \epsilon_z \in b}\,\times^*}{\dfrac{x \neq (Y,Z)}{\vdots}\,\neg a^{-1*} \qquad \dfrac{\dfrac{(Z,Y) \notin p^{-1}}{(Y,Z) \notin p}\,\neg a^{-1}}{\dfrac{x \neq (Y,Z) \qquad \dfrac{p \neq p}{\odot}\,\odot_r}{\vdots}\,\mathsf{pred}}}}{\odot}$$

où X, Y et Z sont respectivement instanciées par x, ϵ_y et ϵ_z.

Nous pouvons remarquer que sans les informations de typage $p \in \mathbb{P}(a \times b)$, ce n'est pas démontrable. L'arbre incomplet correspondant est le suivant :

$$\dfrac{\dfrac{x \in p, x \notin (p^{-1})^{-1}}{x \neq (Y,Z)}\,\neg a^{-1*} \qquad \dfrac{\dfrac{(Z,Y) \notin p^{-1}}{(Y,Z) \notin p}\,\neg a^{-1}}{x \neq (Y,Z) \qquad \dfrac{p \neq p}{\odot}\,\odot_r}\,\mathsf{pred}$$

5.3.3 Correction et complétude

Le théorème à démontrer pour la correction et complétude de notre extension de Zenon peut être énoncé ainsi :

Théorème 2 (Correction et complétude)
Soient Th_B le sous-ensemble de la théorie axiomatique de B présenté dans la figure 5.1 tel que $Th_B = \{Ax_1, \ldots, Ax_n\}$ et $\mathcal{R}_B$ regroupe les règles de réécriture propositionnelles associées. Toute preuve d'une formule $\Gamma \vdash_{+Th_B} \Delta$ construite à partir des règles de Zenon et des super-règles de son extension B peut être traduite en une preuve de la formule $\Gamma, Th_B \vdash \Delta$ utilisant uniquement des règles de Zenon (correction) et réciproquement (complétude).

Preuve du théorème 2 Pour la correction, il faut prouver que si Zenon trouve une preuve de Δ avec les super-règles alors il y a une preuve Zenon de Δ avec les axiomes de Th_B.

Pour cela, on peut remplacer chaque application d'une super-règle par un arbre de dérivation utilisant les axiomes qui correspond à l'arbre construit pour le calcul de la super-règle. La preuve s'effectue par induction sur les règles de réécriture.

- Considérons les super-règles calculées à partir de la règle de réécriture $R : P \to Q$ correspondant à un axiome de la forme $P \Leftrightarrow Q$.

  Soit les super-règles respectivement positive et négative :

  $$\frac{P}{\Gamma_1 \mid \ldots \mid \Gamma_n}\, R \qquad\qquad \frac{\neg P}{\Delta_1 \mid \ldots \mid \Delta_m}\, \neg R$$

  Ces super-règles peuvent être remplacées directement par les arbres de preuve utilisés pour leur calcul.

- Considérons les super-règles calculées à partir de la règle de réécriture $R : x \in e \to x \in f$ correspondant à un axiome de la forme $e \triangleq f$.

  Soient les super-règles respectivement positive et négative :

  $$\frac{x \in e}{\Gamma_1 \mid \ldots \mid \Gamma_n}\, R \qquad\qquad \frac{x \notin e}{\Delta_1 \mid \ldots \mid \Delta_m}\, \neg R$$

  Ces super-règles peuvent être remplacées directement par les arbres de preuve utilisés pour leur calcul.

- Considérons les super-règles calculées à partir de la règle de réécriture $R : (x, y) \in e \to (x, y) \in f$ correspondant à un axiome de la forme $e \triangleq f$ (quand e est une relation de type $s \times t$).

  Soient les super-règles respectivement positive et négative :

  $$\frac{(x, y) \in e}{\Gamma_1 \mid \ldots \mid \Gamma_n}\, R \qquad\qquad \frac{(x, y) \notin e}{\Delta_1 \mid \ldots \mid \Delta_m}\, \neg R$$

  Ces super-règles peuvent être remplacées directement par les arbres de preuve utilisés pour leur calcul.

- Considérons les super-règles calculées à partir de la règle de réécriture $R : x \in e \to \exists y \exists z (x = (y, z) \wedge (y, z) \in f$ correspondant un axiome de la forme $e \triangleq f$ (quand e est une relation de type $s \times t$).

  Soit la super-règle positive

$$\frac{x \in e}{x = (\epsilon_y, \epsilon_z), \Gamma_1 \mid \ldots \mid x = (\epsilon_y, \epsilon_z), \Gamma_n}\, R$$

où $\epsilon_y = \epsilon(y).(\exists z.x = (y, z) \wedge (y, z) \in f)$ et $\epsilon_z = \epsilon(z).(x = (\epsilon_y, z) \wedge (\epsilon_y, z) \in f)$.
Par construction de la super-règle, on sait qu'il existe la dérivation suivante utilisant uniquement les règles de Zenon :

$$\frac{\dfrac{\exists y \exists z(x = (y, z) \wedge (y, z) \in f)}{\vdots}}{x = (\epsilon_y, \epsilon_z), \Gamma_1 \mid \ldots \mid x = (\epsilon_y, \epsilon_z), \Gamma_n}$$

où $\epsilon_y = \epsilon(y).(\exists z.x = (y, z) \wedge (y, z) \in f)$ et $\epsilon_z = \epsilon(z).(x = (\epsilon_y, z) \wedge (\epsilon_y, z) \in f)$.
Pour construire le sous-arbre qui remplacera la super-règle, nous devons démontrer que $x \in e \Rightarrow \exists y \exists z(x = (y, z) \wedge (y, z) \in f)$ sachant que $e = f$. Soit fst et snd deux fonctions définies ainsi : $\mathsf{fst}(a, b) = a$ et $\mathsf{snd}(a, b) = b$. Des fonctions équivalentes existent dans la théorie de B. Pour démontrer notre formule, il suffit de prendre $\mathsf{fst}(x)$ comme témoin pour y et $\mathsf{snd}(x)$ pour z.
L'arbre de dérivation remplaçant la super-règle est alors le suivant :

$$\dfrac{\dfrac{\dfrac{\dfrac{e = f, x \in e}{x \in e \Rightarrow \exists y \exists z(x = (y, z) \wedge (y, z) \in f)}}{\dfrac{x \notin e}{\odot}} \beta_{\Rightarrow} \quad \dfrac{\dfrac{\exists y \exists z.(x = (y, z) \wedge (y, z) \in f)}{\vdots}}{x = (\epsilon_y, \epsilon_z), \Gamma_1 \mid \ldots \mid x = (\epsilon_y, \epsilon_z), \Gamma_n}}{\dfrac{\neg(x \in e \Rightarrow \exists y \exists z \ldots)}{\dfrac{\vdots}{\odot}}}\, \text{cut}$$

où $\epsilon_y = \epsilon(y).(\exists z.x = (y, z) \wedge (y, z) \in f)$, $\epsilon_z = \epsilon(z).(x = (\epsilon_y, z) \wedge (\epsilon_y, z) \in f)$, $e = f$ est un axiome de Th_B et le sous-arbre droit représente la preuve de $x \in e \Rightarrow \exists y \exists z(x = (y, z) \wedge (y, z) \in f)$ expliquée ci-dessus.
Soit la super-règle négative :

$$\frac{x \notin e}{x = (Y, Z) \mid \Delta_1 \mid \ldots \mid \Delta_m}\, R$$

L'arbre remplaçant la super-règle négative est le suivant :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{e = f, x \notin e}}{x \notin e \Rightarrow \neg(\exists y \exists z (x = (y,z) \wedge (y,z) \in f))}}{\dfrac{x \in e}{\odot}} \quad \dfrac{\dfrac{}{\neg(\exists y \exists z (x = (y,z) \wedge (y,z) \in f))}}{\vdots}}{x = (Y,Z) \mid \Delta_1 \mid \ldots \mid \Delta_m}\,\beta_{\Rightarrow} \quad \dfrac{\dfrac{}{\neg(x \notin e \Rightarrow \neg(\exists y \exists z \ldots))}}{\dfrac{\vdots}{\odot}}}{}\,\text{cut}$$

où $e = f$ est un axiome de Th_B et le sous-arbre droit représente la preuve de $x \notin e \Rightarrow \neg\exists y \exists z (x = (y,z) \wedge (y,z) \in f)$ dérivable de la preuve expliquée ci-dessus de $x \in e \Rightarrow \exists y \exists z (x = (y,z) \wedge (y,z) \in f)$.

Pour la complétude, il faut aussi considérer toutes les règles de réécriture que nous traitons, et nous allons montrer qu'elles n'introduisent pas d'incomplétude. Soit Π la preuve de Zenon de $Th_B \vdash \Delta$ avec les axiomes de B. Pour montrer la complétude (avec coupure), il faut montrer qu'il existe une preuve de Zenon étendu à B de $\vdash {}_{+Th_B}\Delta$. Pour cela, on peut récupérer directement la preuve Π à condition d'avoir réussi à prouver auparavant tous les axiomes de la théorie Th_B (on aura $Th_B \vdash {}_{+Th_B}\Delta$). Concrètement, il faut ajouter les axiomes grâce à des coupures, et donc démontrer indépendamment chaque axiome. Soient $Ax_1, \ldots, Ax_n$ les axiomes de la théorie Th_B. L'arbre de preuve (incomplet) de Δ est le suivant (comme nous sommes dans la méthode des tableaux, la formule Δ est niée) :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\Delta}}{Ax_1} \quad \neg Ax_1}\,\text{cut}}{Ax_2} \quad \neg Ax_2}\,\text{cut}}{\dfrac{\vdots}{\dfrac{Ax_n \quad \neg Ax_n}{\dfrac{\Pi}{\odot}}}}\,\text{cut}$$

Nous allons montrer par induction sur la forme des axiomes que pour chaque axiome de notre théorie, il existe une preuve de ces axiomes utilisant des super-règles.

– Considérons les axiomes de la forme $P \Leftrightarrow Q$. La règle de réécriture associée est $R : P \to Q$ et les super-règles sont :

$$\dfrac{P}{\Gamma_1 \mid \ldots \mid \Gamma_n}\,R \qquad \dfrac{\neg P}{\Delta_1 \mid \ldots \mid \Delta_m}\,\neg R$$

où $\Gamma_1, \ldots, \Gamma_n$ et $\Delta_1, \ldots, \Delta_m$ sont tels que l'on peut construire l'arbre suivant :

$$\dfrac{\dfrac{\dfrac{P, \neg P}{\Gamma_1 \mid \ldots \mid \Gamma_n}\, R}{\dfrac{\Delta_1 \mid \ldots \mid \Delta_m}{\odot \mid \ldots \mid \odot} \qquad \dfrac{\Delta_1 \mid \ldots \mid \Delta_m}{\odot \mid \ldots \mid \odot}}\, \neg R}$$

Dans la suite, nous appellerons cet arbre $\pi(P)$, paramétrable par P.

On sait par construction des super-règles que $\Gamma_1 \mid \ldots \mid \Gamma_n$ est une dérivation de Q et de même pour $\Delta_1 \mid \ldots \mid \Delta_m$ avec $\neg Q$.

Démontrons l'axiome $P \Leftrightarrow Q$ avec ces super-règles (celui-ci est nié à cause de la coupure qui l'a introduit dans l'arbre ci-dessus) :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg(P \Leftrightarrow Q)}{\dfrac{\neg P, Q}{\dfrac{\Delta_1 \mid \ldots \mid \Delta_m}{\neg R}}} \qquad \dfrac{\dfrac{P, \neg Q}{\vdots}}{\dfrac{\Delta_1 \mid \ldots \mid \Delta_m}{}}}{\beta_{\neg\Leftrightarrow}}}{\dfrac{\vdots}{\Gamma_1 \mid \ldots \mid \Gamma_n}} \qquad \dfrac{\vdots}{\Gamma_1 \mid \ldots \mid \Gamma_n}}{\dfrac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\odot \mid \ldots \mid \odot}} \qquad \dfrac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\odot \mid \ldots \mid \odot}}{\dfrac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\odot \mid \ldots \mid \odot}} \qquad \dfrac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\odot \mid \ldots \mid \odot}}{R}$$

– Considérons les axiomes de la forme $e = f$. La règle de réécriture associée est $R : x \in e \rightarrow x \in f$ et les super-règles sont :

$$\dfrac{x \in e}{\Gamma_1 \mid \ldots \mid \Gamma_n}\, \mathrm{R} \qquad\qquad \dfrac{x \notin e}{\Delta_1 \mid \ldots \mid \Delta_m}\, \neg R$$

telles que l'on peut construire l'arbre $\pi(x \in e)$.

On sait par construction des super-règles qu'il existe deux arbres tels que :

$$\dfrac{\dfrac{x \in f}{\vdots}}{\Gamma_1 \mid \ldots \mid \Gamma_n} \qquad\qquad \dfrac{\dfrac{x \notin f}{\vdots}}{\Delta_1 \mid \ldots \mid \Delta_m}$$

Démontrons l'axiome $e = f$ avec les super-règles R et $\neg R$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{e \neq f}{\dfrac{\epsilon_x \notin e, \epsilon_x \in f}{\dfrac{\Delta_1 \mid \ldots \mid \Delta_m}{\neg R}}} \qquad \dfrac{\dfrac{\epsilon_x \in e, \epsilon_x \notin f}{\vdots}}{\dfrac{\Delta_1 \mid \ldots \mid \Delta_m}{}}}{\neq}}{\dfrac{\vdots}{\Gamma_1 \mid \ldots \mid \Gamma_n}} \qquad \dfrac{\vdots}{\Gamma_1 \mid \ldots \mid \Gamma_n}}{\dfrac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\odot \mid \ldots \mid \odot}} \qquad \dfrac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\odot \mid \ldots \mid \odot}}{\dfrac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\odot \mid \ldots \mid \odot}} \qquad \dfrac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\odot \mid \ldots \mid \odot}}{R}$$

où $\epsilon_x = \epsilon(x).\neg(x \in e \Leftrightarrow x \in f)$.

□

Concernant la preuve de la complétude forte (sans coupure), elle est clairement intéressante ici puisque nous faisons de la déduction automatique. Pour en faire la preuve, on pourrait s'inspirer des travaux effectués dans [20] et de [18] mais nous n'avons pas fait la démonstration.

5.3.4 Traitement des métavariables

Comme les super-règles permettent de laisser implicites les étapes d'introduction des connecteurs dont celle du $\forall$, nous devons gérer l'instanciation des métavariables par des règles dédiées. Par exemple, pour la règle $\dfrac{x \notin \mathsf{dom}(a)}{(x,Y) \notin a}\ \neg\mathsf{dom}$ il y a une règle d'instanciation associée qui est $\dfrac{x \notin \mathsf{dom}(a)}{(x,t) \notin a}\ \neg\mathsf{dom}_{inst}$ dans laquelle la métavariable Y est instanciée par un terme t.

Cela semble moins évident lorsque l'on a dans une super-règle deux métavariables qui apparaissent, comme par exemple dans la règle $\dfrac{x \notin a^{-1}}{x \neq (Y,Z) \mid (Z,Y) \notin a}\ \neg a^{-1*}$. Le fait d'avoir deux métavariables différentes dans une règle n'est pas possible pour les règles de Zenon. En effet dans ces dernières, les quantificateurs sont gérés un par un, alors que l'application d'une seule super-règle peut supprimer plusieurs quantificateurs et donc produire plusieurs métavariables. Considérons le cas général : soit $\dfrac{P}{q(X,Y)}\ P_{\forall M}$ une super-règle utilisant deux métavariables X et Y. Nous allons voir que la règle $\dfrac{P}{q(t_X,t_Y)}\ P_{inst}$, où t_X et t_Y sont des termes, n'est pas suffisante pour la recherche de preuve de Zenon.

Avant de proposer notre solution, regardons ce que fait Zenon lorsqu'il doit gérer deux métavariables dans une formule. De manière générale, Zenon instancie une métavariable dès qu'il y a une contradiction potentielle. Par exemple, si les formules $P(X)$ et $\neg P(t)$ sont présentes dans la même branche alors Zenon instanciera la métavariable X par le terme t.

Soit une branche contenant les formules $\forall x.\forall y.q(x,y)$ et $\neg q(t_1,t_2)$, le processus d'instanciation des variables est effectuée ainsi dans Zenon :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall x.\forall y.q(x,y), \neg q(t_1,t_2)}{\forall y.q(X,y)}\,\gamma_{\forall M} \text{ sur } \forall x.\forall y.q(x,y)}{q(X,Y)}\,\gamma_{\forall M} \text{ sur } \forall y.q(X,y)}{q(X,t_2)}\,\gamma_{\forall \text{inst}} \text{ sur } \forall y.q(X,y)}{\forall y.q(t_1,y)}\,\gamma_{\forall \text{inst}} \text{ sur } \forall x.\forall y.q(x,y)}{q(t_1,Y')}\,\gamma_{\forall M} \text{ sur } \forall y.q(t_1,y)}{q(t_1,t_2)}\,\gamma_{\forall \text{inst}} \text{ sur } \forall y.q(t_1,y)}{\odot}$$

Nous pouvons remarquer que Zenon ne peut instancier la métavariable X après avoir instancié la métavariable Y, tout simplement car il n'y a pas de règle $\dfrac{P(X,t_Y)}{P(t_X,t_Y)}$. De plus, lors de la recherche de preuve, trois métavariables sont générées car la règle $\gamma_{\forall M}$ est appliquée trois fois : X est générée par l'application sur $\forall x.\forall y.q(x,y)$, Y est générée par l'application sur $\forall y.q(X,y)$ et Y' est générée par l'application sur $\forall y.q(t_1,y)$. Concrètement, chaque métavariable est liée à la formule qui l'a produite. Sans la génération de Y', Zenon ne pourrait pas instancier y tout en conservant l'information qu'une instanciation possible pour x est t_1. Le problème de l'instanciation de deux métavariables est donc de conserver la première instanciation de la première métavariable avant de pouvoir instancier la deuxième.

Nous venons de voir que Zenon ne peut pas instancier deux métavariables en même temps, alors qu'avec les super-règles que nous avons introduites, nous devons traiter ce cas. Pour cela, nous ajoutons des super-règles d'instanciation qui vont permettre de donner un ordre d'instanciation à Zenon. Sans ces nouvelles règles, Zenon n'arrive pas à gérer correctement les instanciations de deux métavariables. Les super-règles à ajouter sont les suivantes :

$$\dfrac{P}{q(t_X,Y)}\,P_{\forall \text{instX}} \qquad \dfrac{P}{q(X,t_Y)}\,P_{\forall \text{instY}}$$

Nous avons vu que pour instancier les deux métavariables, il faut conserver la première instanciation trouvée pour ensuite instancier la deuxième. Pour résoudre ce problème, nous lions une métavariable à l'ensemble des autres métavariables au lieu de la lier à une formule puisque chaque métavariable doit avoir accès à toutes les instanciations déjà faites pour reconstruire la formule. Notons X^M la métavariable X liée à M avec M un ensemble de la forme $\{m_1,\ldots,m_n\}$ où m_i est une métavariable X qui peut être associée à un terme t_X (cela est noté $X = t_X$). Si X n'a pas été instancié alors m est égal à X.

Les super-règles deviennent :

$$\frac{P}{q(X^{\{Y\}}, Y^{\{X\}})}\, P_{\forall M} \qquad \frac{P}{q(t_X, Y^{\{X=t_X\}})}\, P_{\forall \text{instX}}$$

$$\frac{P}{q(X^{\{Y=t_Y\}}, t_Y)}\, P_{\forall \text{instY}} \qquad \frac{P}{q(t_X, t_Y)}\, P_{\forall \text{inst}}$$

Lors de l'instanciation, si une métavariable n'a pas de terme dans sa liaison, alors c'est la première instanciation (l'une des super-règles $P_{\forall \text{instX}}$ et $P_{\forall \text{instY}}$ est appliquée). Si une métavariable est liée à un terme, alors ce terme est récupéré pour instancier l'autre métavariable, et l'instanciation de la métavariable courante est faite normalement (la super-règle $P_{\forall \text{inst}}$ est appliquée).

L'arbre précédent avec les nouvelles super-règles d'instanciation devient alors :

$$\cfrac{\cfrac{\cfrac{\cfrac{P, \neg q(t_1, t_2)}{q(X^{\{Y\}}, Y^{\{X\}})}\, P_{\forall M}}{q(X^{\{Y=t_2\}}, t_2)}\, P_{\forall \text{instY}}}{q(t_1, t_2)}\, P_{\forall \text{inst}}}{\odot}$$

Correction des règles introduites Nous devons vérifier que les règles que nous avons introduites n'influent pas sur l'instanciation des métavariables c'est-à-dire qu'avec et sans super-règles les instanciations sont les mêmes. De plus, nous devons montrer que Zenon peut toujours simplifier son arbre de recherche de preuve (ce qui est appelé « pruning »), comme cela est fait sans les super-règles.

Ainsi, lorsqu'un sous-arbre est fermé par Zenon, il examine ce sous-arbre pour déterminer les formules utiles. Une formule est dite utile dans un sous-arbre si elle apparaît dans les prémisses d'une application de règle dans ce sous-arbre. Puis, il remonte le sous-arbre en supprimant les applications de règles dont la conséquence n'est pas une formule utile.

Soit une branche contenant les formules $\forall x.\forall y.p(x,y) \vee q(x,y)$, $\neg p(t_1,t_2)$ et $\neg q(t_1,t_2)$, la recherche de preuve de Zenon est la suivante (sans les super-règles d'instanciation) :

$$\dfrac{\forall x.\forall y.p(x,y)\lor q(x,y),\neg p(t_1,t_2),\neg q(t_1,t_2)}{\dfrac{\forall y.p(X,y)\lor q(X,y)}{\dfrac{p(X,Y)\lor q(X,Y)}{\dfrac{p(X,Y)}{\dfrac{p(X,t_2)\lor q(X,t_2)}{\dfrac{p(X,t_2)}{\dfrac{\forall y.p(t_1,y)\lor q(t_1,y)}{\dfrac{p(t_1,Y')\lor q(t_1,Y')}{\dfrac{p(t_1,Y')}{\dfrac{p(t_1,t_2)\lor q(t_1,t_2)}{\dfrac{p(t_1,t_2)}{\odot}\odot\quad\dfrac{q(t_1,t_2)}{\odot}\odot}\beta_{\lor}}\gamma_{\forall\text{inst}}\quad q(t_1,Y')}\beta_{\lor}}\gamma_{\forall M}}\gamma_{\forall\text{inst}}\quad q(X,t_2)}\beta_{\lor}}\gamma_{\forall\text{inst}}\quad q(X,Y)}\beta_{\lor}}\gamma_{\forall M}}\gamma_{\forall M}$$

Comme cet arbre a un sous-arbre fermé, Zenon peut le simplifier. En effet, la formule $p(t_1,Y')$ n'est pas utile puisqu'aucune règle n'est appliquée dessus. Donc on peut supprimer le nœud $\beta_{\lor}$ qui la génère. Il en est de même avec la formule $p(t_1,Y')\lor q(t_1,Y')$ et le nœud $\gamma_{\forall M}$. En procédant de cette manière sur tout l'arbre, nous obtenons le nouvel arbre suivant :

$$\dfrac{\forall x.\forall y.p(x,y)\lor q(x,y),\neg p(t_1,t_2),\neg q(t_1,t_2)}{\dfrac{\forall y.p(t_1,y)\lor q(t_1,y)}{\dfrac{p(t_1,t_2)\lor q(t_1,t_2)}{\dfrac{p(t_1,t_2)}{\odot}\odot\quad\dfrac{q(t_1,t_2)}{\odot}\odot}\beta_{\lor}}\gamma_{\forall\text{inst}}}\gamma_{\forall\text{inst}}$$

Maintenant, effectuons la même preuve mais en utilisant des super-règles. Soient une branche contenant les formules $\neg p(t_1,t_2),\neg q(t_1,t_2)$ et φ tel que $\varphi\to\forall x.\forall y.p(x,y)\lor q(x,y)$, et les super-règles suivantes :

$$\dfrac{\varphi}{p(X^{\{Y\}},Y^{\{X\}})|q(X^{\{Y\}},Y^{\{X\}})}\varphi_{\forall M}\qquad\qquad\dfrac{\varphi}{p(t_X,t_Y)|q(t_X,t_Y)}\varphi_{\forall\text{inst}}$$

$$\dfrac{\varphi}{p(t_X,Y^{\{X=t_X\}})|q(t_X,Y^{\{X=t_X\}})}\varphi_{\forall\text{instX}}\qquad\dfrac{\varphi}{p(X^{\{Y=t_Y\}},t_Y)|q(X^{\{Y=t_Y\}},t_Y)}\varphi_{\forall\text{instY}}$$

La recherche de preuve de Zenon est alors la suivante :

$$\cfrac{\varphi, \neg p(t_1,t_2), \neg q(t_1,t_2)}{\cfrac{p(X,Y) \qquad\qquad q(X,Y)}{\cfrac{p(X^{\{Y=t_2\}},t_2) \qquad q(X^{\{Y=t_2\}},t_2)}{\cfrac{p(t_1,t_2)}{\odot}\odot \qquad \cfrac{q(t_1,t_2)}{\odot}\odot}\ \varphi_{\forall\text{inst}}}\ \varphi_{\forall\text{instY}}}\ \varphi_{\forall M}$$

En appliquant la même simplification que précédemment, nous obtenons l'arbre suivant (puisque $p(X,Y)$, $q(X,Y)$, $p(X^{\{Y=t_2\}},t_2)$ et $q(X^{\{Y=t_2\}},t_2)$ ne sont pas des formules utiles) :

$$\cfrac{\varphi, \neg p(t_1,t_2), \neg q(t_1,t_2)}{\cfrac{p(t_1,t_2)}{\odot}\odot \qquad \cfrac{q(t_1,t_2)}{\odot}\odot}\ \varphi_{\forall\text{inst}}$$

Même si ce n'est pas une preuve formelle, on voit bien que dans cet exemple, les instanciations de Zenon avec ou sans super-règles sont les mêmes, et que dans le cas où les super-règles sont appliquées, la simplification se fait sans problème.

Cas général Nous généralisons l'introduction des super-règles d'instanciation au cas où n métavariables sont introduites simultanément. Soit la règle de réécriture

$$R : P \to \forall x_1 \dots \forall x_n . q(x_1, \dots, x_n)$$

où q est une formule supposée ne contenir ni quantificateurs ni connecteurs pour simplifier la présentation. La formule du membre droit est par exemple une sous-formule de la règle de réécriture associée à l'axiome définissant la fonction partielle :
$f \in a \nrightarrow b \to f \in a \leftrightarrow b \wedge \forall (x,y,z)\ ((x,y) \in f \wedge (x,z) \in f \Rightarrow y = z)$.

La règle permettant de générer toutes les métavariables est la suivante :

$$\frac{P}{q(X_1^{\{X_1,\dots,X_n\}}, \dots, X_n^{\{X_1,\dots,X_n\}})}\ P_{\forall M}$$

Soit $X_i^{\{m_1,\dots,m_n\}}$ la métavariable à remplacer par le terme t_i, la règle permettant d'instancier la variable x_i par le terme t_i est la suivante :

$$\frac{P}{q(a_1, \dots, t_i, \dots, a_n)}\ P_{\forall\text{inst}X_i}$$

où pour tout j compris entre 1 et n et différent de i, si a_j est une métavariable X_j^M alors elle devient $X_j^{M \setminus X_i \cup \{X_i = t_i\}}$, ou a_j est un terme.

La règle permettant d'instancier toutes les métavariables est :

$$\frac{P}{q(t_1, \ldots, t_n)}\, P_{\forall \text{inst}}$$

5.4 Implantation

Concrètement, les extensions de Zenon sont des fichiers OCaml qui implantent de nouvelles règles et des fichiers Coq contenant les lemmes utilisés pour traduire les règles d'inférence introduites par l'extension. Les extensions sont invoquées au moyen d'options de la ligne de commande de Zenon. Après avoir calculé à la main les règles de superdéduction associées à la théorie de B (celles présentées dans les figures 5.2 et 5.3), nous avons implanté ces règles grâce à un fichier d'extension OCaml.

5.4.1 Validation avec Coq

Tout comme Zenon, Zenon étendu à B génère des preuves Coq qui peuvent être vérifiées automatiquement. Nous avons dû adapter la génération de la trace aux super-règles.

Étant donné que ces preuves utilisent des règles de superdéduction, nous avons dû étendre la bibliothèque Coq contenant toutes les règles de Zenon aux super-règles. Cette bibliothèque Coq contient pour chaque règle de Zenon le lemme Coq correspondant (voir [17] pour plus de précisions). Nous avons donc démontré un lemme Coq pour chaque super-règle associée à un axiome B. Pour cela, nous avons développé un plongement superficiel de B en Coq et donc indépendant du plongement BCoq. Les preuves Coq produites par notre extension de Zenon peuvent alors être vérifiées en utilisant ce plongement.

5.4.2 Validation avec BCoq

À terme, pour vérifier les règles ajoutées de l'Atelier B, nous voulons reconstruire la preuve obtenue dans BCoq afin de la rejouer dans cet environnement (comme cela est fait pour l'approche utilisant Zenon sans extension). Pour cela, il faut démontrer toutes les règles de l'extension B de Zenon dans BCoq, de la même manière que cela avait été fait

pour les règles Zenon dans la partie 4.3.4. Cette reconstruction de preuve est en cours d'implantation. Elle est nécessaire pour assurer la correction de notre approche vis-à-vis de la théorie de B.

Le processus obtenu avec cette approche est présenté dans la figure 5.4 dans lequel, l'interprétation consiste à traduire syntaxiquement une formule BCoq en une formule B dans la syntaxe de Zenon. Les flèches avec un trait correspondent aux étapes du processus déjà implantées, alors que les flèches avec des tirets représentent les étapes non implantées à l'heure actuelle.

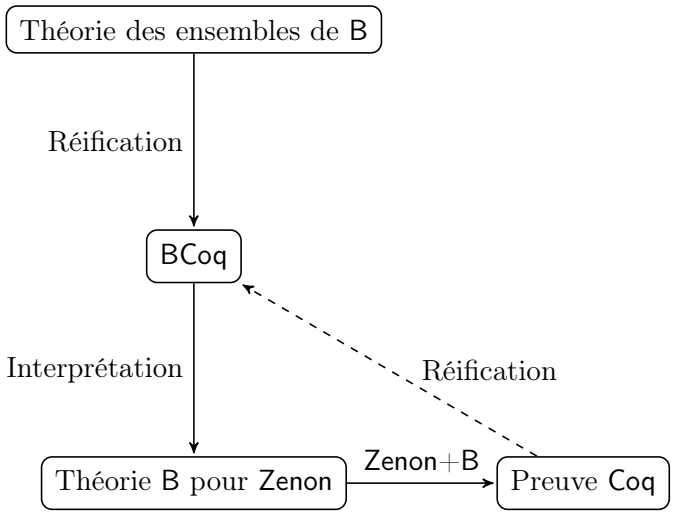

FIGURE 5.4 – Processus de preuve B utilisant Zenon étendu à B

5.5 Résultats expérimentaux

Nous présentons dans cette section les résultats expérimentaux que nous avons obtenus sur la base de règles Siemens IC-MOL avec l'extension de Zenon à la théorie de B. Nous avons lancé les tests dans les mêmes conditions que les tests précédents (à savoir sur un ordinateur Intel Pentium D Xeon 3.60GHz/32Go). Sur les 2031 règles bien typées et bien définies de la base, notre nouvelle approche en démontre 1443. Démontrer ici signifie uniquement que notre extension de Zenon réussit à trouver une preuve. En effet, la reconstruction de la preuve générée en preuve BCoq est en cours de développement. En réalité le sous-ensemble des règles que notre nouvelle approche peut traiter, parmi les règles bien typées et bien

définies et en écartant les opérateurs non traités, comprend 1534 règles. Pour comparer avec l'approche précédente, nous ne devons pas prendre le nombre final de règles démontrées mais le nombre de règles pour lesquelles Zenon avait trouvé une preuve à savoir 1426 (pour ce sous-ensemble). Nous démontrons donc la validité d'un peu plus de règles qu'avec la précédente approche.

Nous avons comparé les temps de preuve de notre approche utilisant la prénormalisation puis Zenon avec notre extension de Zenon à la méthode B (donc sans étape de normalisation). Les résultats sont présentés sous forme de graphe dans la figure 5.5. Un point représente le résultat pour une règle et les axes des abscisses et des ordonnées représentent respectivement le temps pour l'ancienne approche (reconstruction de la preuve exclue) et le temps pour la nouvelle approche. Pour chaque point situé sous la courbe tracée avec des triangles, notre extension basée sur la superdéduction est au moins 10 fois plus rapide que l'autre approche, sous la courbe avec des croix, c'est 100 fois plus rapide, et sous la courbe avec des carrés, c'est 1000 fois plus rapide. Nous pouvons observer qu'il y a de nombreuses règles pour lesquelles la preuve est de 10 à 1000 fois plus rapide avec la superdéduction. En moyenne, la superdéduction trouve les preuves 172 fois plus rapidement (le meilleur rapport étant de 7749).

En étendant Zenon à la méthode B, nous prouvons un peu plus de règles par rapport à l'approche précédente, mais dans un temps bien plus court. Néanmoins, l'extension de Zenon est complexe techniquement et le calcul (ainsi que l'implantation) des règles de superdéduction à la main se révèle difficile pour les opérateurs définis avec plusieurs quantificateurs.

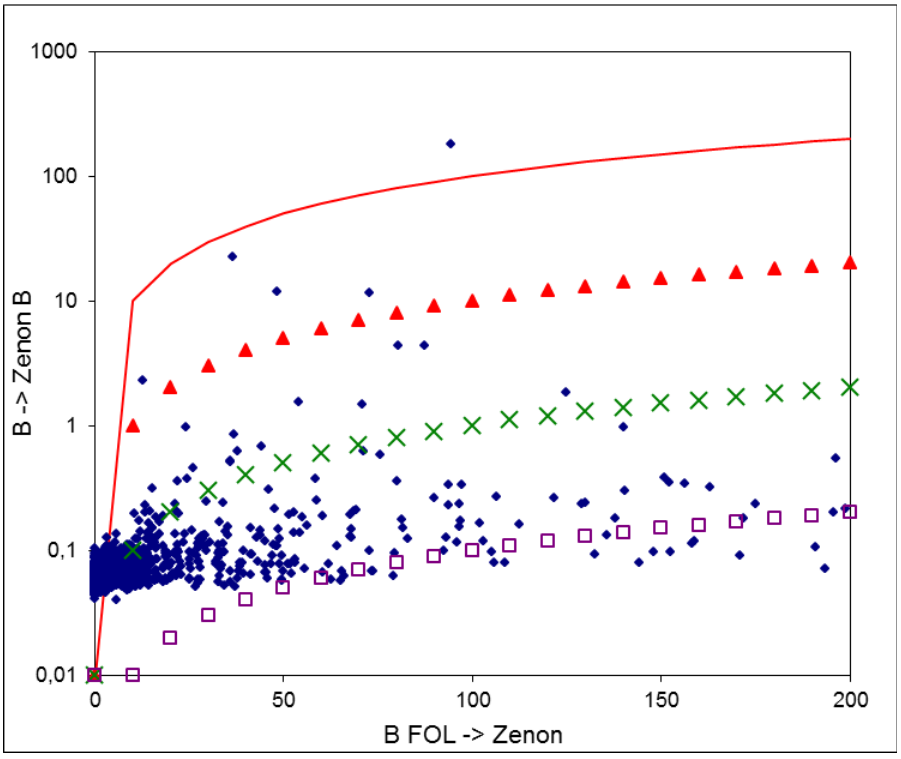

FIGURE 5.5 – Comparaison sur les temps de preuve

Chapitre 6

Extension de Zenon à toute théorie : Super Zenon

Dans le chapitre précédent, nous avons présenté une extension de Zenon [17] à la méthode B qui utilise la superdéduction [20]. Les règles de déduction de Zenon sont alors étendues avec de nouvelles règles traitant chacune d'un opérateur ensembliste de B. Ces règles ont été calculées préalablement à la main, ce qui peut être source d'erreur lorsque les définitions des opérateurs sont complexes (voir les règles associées à la fonction partielle de la figure 5.3 page 125). Pour éviter ces erreurs, nous avons intégré le calcul des super-règles à Zenon pour que celui-ci soit automatique. Pour cela, nous utilisons le moteur de preuve de Zenon ainsi qu'un sous-ensemble de ses règles (comme cela était fait pour le calcul des super-règles de la théorie de B). De plus, cela est fait dynamiquement, c'est-à-dire qu'à partir d'un fichier contenant une théorie axiomatique et un problème dans cette théorie, le prouveur automatique calcule les nouvelles super-règles puis cherche une preuve au problème avec ses règles d'origine et les nouvelles super-règles. Le nouvel outil ainsi étendu est appelé Super Zenon. Cet outil peut donc traiter la théorie B mais aussi toute théorie axiomatique comme par exemple les théories des problèmes de la bibliothèque TPTP [59].

Le principe du calcul des nouvelles super-règles, bien que maintenant automatisé, est le même que dans le chapitre précédent. Tout d'abord on identifie les axiomes avant de les orienter, puis on calcule les super-règles associées en utilisant uniquement un sous-ensemble des règles de déduction de Zenon (voir la section 6.1). La correction est assurée pour les même raisons que précédemment (voir la section 6.2) et l'implantation de l'outil

sera expliquée dans la section 6.3. Grâce aux super-règles et à la méthode des tableaux, les traces de Super Zenon sont lisibles par un humain. Cela permet, par exemple, d'aider une personne dans sa preuve, à partir des traces (voir la section 6.5). Nous avons appliqué Super Zenon sur la théorie axiomatique de B (voir la section 6.4). Enfin nous avons testé cette approche sur des problèmes issus de la bibliothèque TPTP (voir la section 6.7), qui regroupe des problèmes dédiés au test des prouveurs automatiques de théorèmes.

6.1 Calcul des super-règles

Contrairement à la situation décrite au chapitre précédent, pour une théorie, nous ne connaissons pas par avance la forme des axiomes.

Nous considérons uniquement les axiomes de la forme $\forall \bar{x}\ (P \Leftrightarrow Q)$, $\forall \bar{x}\ (P \Rightarrow Q)$ et $\forall \bar{x}\ P$ qui sont orientés ainsi (où P et Q sont des formules quelconques) :

Axiome	Règles de réécriture	
$\forall \bar{x}\ (P \Leftrightarrow Q)$	$P \to Q$	$\neg P \to \neg Q$
$\forall \bar{x}\ (P \Rightarrow Q)$	$P \to Q$	$\neg Q \to \neg P$
$\forall \bar{x}\ P$	$\neg P \to \bot$	

Par rapport au chapitre précédent, nous n'avons plus ici une règle de réécriture par axiome mais plusieurs. Auparavant, à partir de ces règles de réécriture une règle positive et une règle négative étaient calculées. Cela n'est plus possible ici à cause des nouveaux axiomes traités qui sont de la forme $\forall \bar{x}\ (P \Rightarrow Q)$ et $\forall \bar{x}\ P$. En effet pour le premier axiome, en suivant le processus de calculs des super-règles du chapitre précédent, la règle négative serait incorrecte car elle permettrait de dériver $\neg Q$ à partir de $\neg P$. Pour le deuxième axiome, il n'y a pas de règle négative puisqu'elle n'aiderait pas la recherche de preuve. Nous avons donc introduit de nouvelles règles de réécriture qui permettront de produire une seule règle de superdéduction.

Parmi ces orientations, on ne conserve que les règles qui respectent les critères suivants :

1. le membre gauche est atomique ;
2. il n'y a pas d'égalité dans le membre gauche ;
3. le membre gauche n'est pas applicable sur le membre droit d'une même règle c'est-à-dire on ne peut pas boucler sur l'application d'une règle ;

4. le membre gauche n'est pas en conflit avec le membre gauche d'une règle déjà acceptée, autrement dit il ne peut pas y avoir deux règles applicables sur une même formule ;
5. l'ensemble des variables libres du membre droit est inclus dans l'ensemble des variables libres du membre gauche.

Si pour un axiome, aucune règle n'est conservée, alors il est laissé en axiome.

Le critère 2 permet de laisser le traitement de l'égalité à Zenon, qui a des règles spécifiques. Le critère 3 évite qu'une règle s'applique indéfiniment. Grâce au critère 4, deux règles ne peuvent pas potentiellement s'appliquer sur un même but. Le critère 5 évite qu'une variable soit insérée dans la recherche de preuve, sans aucune information sur elle.

Exemple 33 (Règles rejetées)

- $\neg(range\_of(Z) = domain\_of(inverse(Z))) \rightarrow \bot$ *n'est pas conservée car il y a une égalité dans le membre gauche.*
- $a + b \rightarrow (a + b) + a$ *n'est pas conservée car le membre gauche* $a + b$ *est applicable sur le membre droit* $(a + b) + a$.
- *Supposons que nous avons la règle, déjà acceptée,* $p(X) \rightarrow q(X)$ *où* X *est une méta-variable. La règle* $p(c) \rightarrow r(c)$ *où* c *est une constante ne sera pas conservée puisque* $p(c)$ *est une instance du membre gauche* $p(X)$ *d'une autre règle.*
- $p(X, Z) \rightarrow p(X, Y) \wedge p(Y, Z)$ *n'est pas conservée car* $\{X, Y, Z\} \nsubseteq \{X, Z\}$

Ces orientations s'appliquent bien aux axiomes de B moyennant une transformation de l'égalité ensembliste en équivalence.

Une fois toutes les règles de réécriture calculées, nous appliquons le même processus que celui qui était conçu pour la théorie de la méthode B dans la section 5.3. Cela produit un ensemble de super-règles spécifique à la théorie.

6.2 Correction et complétude

Comme pour l'extension de Zenon, la correction se justifie par l'utilisation uniquement de règles de Zenon pour le calcul des nouvelles règles. Par contre, l'approche n'est pas complète puisque nous ne conservons pas toutes les règles associées à un axiome si elles ne respectent pas tous les critères énoncés dans la section 6.1. Mais nous verrons, qu'en

pratique, cela n'empêche pas de prouver de nombreux théorèmes provenant de TPTP. Dans le cas où elles sont toutes conservées, pour les axiomes de la forme $\forall \bar{x}\ (P \Leftrightarrow Q)$ et $\forall \bar{x}\ P$ où P est atomique, nous pouvons nous ramener aux preuves effectuées dans le chapitre précédent (puisque ce sont les axiomes traités pour l'extension de Zenon à la méthode B). Le nouveau cas d'axiome à considérer est donc $\forall \bar{x}\ (P \Rightarrow Q)$. Nous allons montrer que les deux règles associées ($P \to Q$ et $\neg Q \to \neg P$) sont nécessaires.

Pour calculer les règles de superdéduction, l'axiome $P \Rightarrow Q$ est orienté ainsi : $R : P \to Q$ et $\neg R : \neg Q \to \neg P$. L'axiome P est transformé en $R' : \neg P \to \bot$. Soient les règles de superdéduction associées :

$$\frac{P}{Q}\,P \qquad \frac{\neg Q}{\neg P}\,\neg Q \qquad \frac{\neg P}{\bot}\,\neg P$$

Avec les règles de superdéduction, la preuve devient la suivante :

$$\frac{\dfrac{\neg Q}{\neg P}\,\neg Q}{\odot}\,\neg P$$

On peut remarquer que sans la règle $\neg Q$, il n'y aurait pas de preuve sans coupure de Q.

6.3 Implantation

Contrairement à l'extension de Zenon à la méthode B, Super Zenon permet d'étendre les règles de superdéduction dynamiquement. Il suffit d'avoir en entrée une théorie axiomatique, Super Zenon va alors transformer certains de ses axiomes en super-règles pour créer dynamiquement une nouvelle extension. Les autres axiomes seront traités tel quels par Zenon. Toute cette extension a été implantée en OCaml et représente environ 2500 lignes. Le calcul des super-règles est effectué en utilisant le moteur de preuve de Zenon c'est-à-dire que l'application des règles, issues du sous-ensemble de règles de Zenon défini dans la définition 5, est faite automatiquement, ainsi que la formation des super-règles durant laquelle les prémisses sont collectées. Ensuite, Super Zenon permet d'appliquer directement ces super-règles sur le problème donné en entrée, et ce automatiquement. L'application des super-règles est implantée par un algorithme d'unification qui compare le membre gauche

des super-règles avec le prédicat courant à réfuter de Zenon. Cela remplace le fichier d'extension habituel. Toutefois, nous laissons la possibilité à l'utilisateur de générer ce fichier d'extension à partir du calcul des super-règles effectués par Super Zenon. Ainsi, pour l'extension de B par exemple, il semble logique de ne calculer qu'une seule fois les super-règles associées. De même, pour toute application de Zenon à une théorie particulière, cela permet de récupérer le code correspondant à une règle complexe à calculer à la main. De plus, l'extension est personnalisable si besoin : on peut choisir de transformer en super-règles qu'un sous-ensemble d'axiome, ou on peut permuter deux branches dans une super-règle pour faciliter la recherche d'instanciation par exemple.

6.4 Application à la méthode B

Pour comparer l'outil Super Zenon à l'extension de Zenon à la méthode B du chapitre 5, nous avons exprimé la théorie de B en TPTP que nous avons donnée en entrée à Super Zenon. Ainsi nous pouvons comparer les super-règles générées par Super Zenon avec celles que nous avions calculées à la main. Nous pouvons noter que certaines super-règles de l'extension B étaient difficiles à calculer et à vérifier *a posteriori* (comme par exemple la règle pour la fonction partielle). Super Zenon est donc, vis-à-vis de la méthode B, une aide au calcul des super-règles.

Exprimer la théorie de B en TPTP n'est pas immédiat. En effet, nous ne pouvons pas exprimer l'ensemble en compréhension en TPTP. En plus, en B les axiomes sont exprimés soit avec une équivalence (pour les axiomes de base de la théorie des ensembles) soit avec une égalité syntaxique (qui permet de définir les opérateurs de B telle que l'union). Le premier type d'axiome est directement adapté à Super Zenon. Seul l'axiome définissant l'ensemble en compréhension (SET 3) pose problème puisque cette construction n'est pas exprimable en TPTP. Pour ce cas précis, nous avons donc une règle précalculée dans Super Zenon, concrètement cette règle est écrite en OCaml. Le second type d'axiome nécessite, quant à lui, une transformation préliminaire. Ainsi, nous cherchons à obtenir à partir d'un axiome de la forme $E = F$ où E et F sont des expressions, un axiome de la forme $\forall \bar{x}\ (P \Leftrightarrow Q)$ où P et Q sont des prédicats. Dans les cas les plus simples, il faut appliquer l'axiome d'extensionnalité $a = b \Rightarrow \forall x.x \in a \Leftrightarrow x \in b$ puis simplifier les ensembles en compréhension

grâce à l'axiome SET 3 ($E \in \{x|x \in s \wedge P\} \Leftrightarrow (E \in s \wedge [x := E]P)$). Par exemple, pour l'union définie par $a \cup b \triangleq \{x|x \in a \wedge x \in b\}$, l'axiome sera $\forall x\ (x \in a \cup b \Leftrightarrow x \in a \wedge x \in b)$ Dans le cas des opérateurs relationnels (de type $s \times t$ par exemple), nous transformons l'axiome $E = F$ en ces deux axiomes : $\forall(x, y)\ ((x, y) \in E \Leftrightarrow (x, y) \in F)$ et $\forall x\ (x \in E \Leftrightarrow (\exists y.\exists z.x = (y, z) \wedge (y, z) \in F))$. Tout cela est nécessaire pour retrouver, après orientation des axiomes, les mêmes règles de réécriture que celles utilisées pour étendre Zenon à la méthode B présentées dans le chapitre 5 (voir la sous-section 5.3.1).

On peut remarquer qu'au lieu de faire cette transformation à la main, on pourrait adapter Super Zenon aux axiomes de la forme $E = F$ afin qu'il effectue seul l'orientation adéquate à cette forme d'axiome. Cela permettrait notamment une utilisation beaucoup plus simple de Super Zenon pour la théorie B puisque l'écriture en syntaxe TPTP serait plus directe. Pour cela, il faudrait étendre la syntaxe TPTP à une syntaxe acceptant des constructions telles que l'ensemble en compréhension.

6.5 Traces de Super Zenon

Zenon utilisant la méthode des tableaux, les traces qu'il fournit sont relativement lisibles par un humain. En effet, contrairement à une méthode de recherche comme la résolution, la méthode des tableaux traite directement de la formule et ne nécessite pas de traitement préalable de la formule comme la mise en forme clausale. De plus, les applications des règles permettent de décomposer, étape par étape, la formule tant qu'une contradiction n'a pas été trouvée. Il est donc facile de retrouver les formules initiales ainsi que leur décomposition dans la preuve générée. Super Zenon améliore ce point dans le sens où les preuves résultantes sont plus courtes et l'application d'une super-règle, à la place de l'utilisation d'un axiome, permet une preuve plus claire.

6.5.1 Preuve d'une règle ajoutée

Pour évaluer notre outil sur la théorie ensembliste de B et pour montrer qu'il peut produire des preuves assez compréhensibles pour retrouver l'intuition de ces règles, nous proposons de considérer l'exemple d'une règle B issue de la base de règles maintenue par

Siemens IC-MOL. Cette règle fait en réalité partie de l'ensemble des règles ajoutées proposé par l'Atelier B et est nommée « SimplifyRelDorXY.27 ». La définition de cette règle est la suivante :

$$\emptyset \lhd a = \emptyset$$

Lorsqu'il est appliqué sur cette formule, Super Zenon produit la preuve présentée dans la figure 6.1 qui est dans le format de preuve le plus abstrait proposé par Zenon. L'énoncé du problème, c'est-à-dire le texte commençant par « `fof` », utilise ici la syntaxe TPTP [59], et la preuve est présentée à la suite. Cette preuve est constituée de plusieurs étapes numérotées, où chacune d'entre elles est un ensemble de formule avec le nom de la règle appliquée à l'étape considérée. Les formules des étapes sont signées, celles commençant par « `-.` » sont négatives (elles sont niées). Une règle apparaît à la fin de l'étape après « `###` », et indique, après « `-->` », les étapes auxquelles elle est reliée (ces autres étapes représentent le résultat de l'application de la règle à l'ensemble courant de formules). Par exemple dans cette preuve, l'étape 1 est reliée aux étapes 2 et 3. Ce lien entre les étapes suit une preuve en forme d'arbre où les étapes avec des règles axiomatiques sont des feuilles, alors que les autres étapes sont des nœuds. Parmi ces étapes, il y a notamment des étapes d'application de super-règles, qui commencent par « `Extension` ». Dans cette preuve, les constructions B sont préfixées par « `b_` », et « `b_empty` », « `b_in` », « `b_eq` », et « `b_restg` » représentent respectivement l'ensemble vide $\emptyset$, l'opérateur d'appartenance « $\in$ », l'égalité extensionnelle « $=$ », et la construction de la restriction gauche « $\lhd$ ».

On peut observer que cette preuve peut être facilement compréhensible, non seulement grâce à la méthode des tableaux, mais aussi grâce aux règles de superdéduction qui nous permettent de raccourcir la preuve en supprimant les détails non importants pour la compréhension de la preuve. Pour justifier cela, nous allons décrire le déroulement de la preuve qui peut être extraite de cette preuve formelle et qui permet de mettre en évidence l'intuition de la preuve. La preuve formelle est construite de la manière suivante :

1. La preuve débute avec le séquent « $\vdash \emptyset \lhd a = \emptyset$ », qui correspond à l'hypothèse `H0` de l'étape 1 (où la formule initiale est niée puisque la méthode des tableaux utilise la réfutation). La règle appliquée à ce séquent est la super-règle qui traite de l'égalité

```
fof(simplifyRelDorXY_27, conjecture,
  b_eq (b_drest (b_empty, a), b_empty)).
(* PROOF-FOUND *)
   1. H0: (-. (b_eq (b_restg (b_empty) (a)) (b_empty)))
      ### [Extension/SZ/not_b_eq H0 H1 H2 H3 H4 H5 H6] --> 2 3
   2. H2: (b_in T_7 (b_empty))
      ### [Extension/SZ/b_empty H2 H7]

   3. H4: (b_in T_7 (b_restg (b_empty) (a)))
      ### [Extension/SZ/b_restg H4 H8 H9 H10 H7 H6 H11] --> 4
   4. H9: (b_in T_12 (b_empty))
      ### [Extension/SZ/b_empty H9 H12]
```

FIGURE 6.1 – Preuve de la règle « SimplifyRelDorXY.27 » de l'Atelier B

et qui corresponds à la règle de superdéduction appelée `not_b_eq` dans la preuve de Super Zenon, c'est-à-dire la négation de l'égalité, toujours car la formule initiale a été niée. La règle de superdéduction est la suivante (nous l'exprimons ici en calcul des séquents pour aider la compréhension de la preuve) :

$$\frac{\Gamma, x \in a \vdash x \in b, \Delta \qquad \Gamma, x \in b \vdash x \in a, \Delta}{\Gamma \vdash a = b, \Delta} =\text{R},\ x \notin \Gamma, \Delta$$

2. En appliquant cette règle, nous obtenons deux cas à prouver (comme montré par la règle ci-dessus). La preuve de Super Zenon commence par le côté droit de la règle, et nous devons ensuite montrer le séquent « $x \in \emptyset \vdash x \in \emptyset \lhd a$ », qui correspond à l'étape 2. Comme nous pouvons le voir, dans l'ensemble des formules de chaque étape, la preuve de Super Zenon n'affiche que les formules utiles pour la preuve. Par exemple, dans l'étape 2, la formule « $x \in \emptyset \lhd a$ » n'apparaît pas (alors qu'elle est présente dans l'ensemble des formules) car elle n'est pas utilisée dans la suite de la preuve. La preuve de Super Zenon traite ensuite l'hypothèse « $x \in \emptyset$ » et applique la règle de superdéduction appelée `b_empty` qui correspond à la super-règle suivante dans le calcul des séquents :

$$\frac{}{\Gamma, x \in \emptyset \vdash \Delta}\ \emptyset\text{L}$$

Cette règle termine cette partie car $x \in \emptyset$ est une contradiction.

3. Le second cas produit par l'application de la super-règle sur l'égalité correspond au séquent suivant « $x \in \emptyset \lhd a \vdash x \in \emptyset$ », qui est la formule de l'étape 3 dans la preuve de Super Zenon. Dans cette étape, Super Zenon traite de l'hypothèse « $x \in \emptyset \lhd a$ », et applique la règle de superdéduction `b_restg` qui correspond à la règle suivante dans le calcul des séquents :

$$\frac{\Gamma, x = (y,z), (y,z) \in b, y \in a \vdash \Delta}{\Gamma, x \in a \lhd b \vdash \Delta} \lhd\text{L},\ y, z \notin \Gamma, \Delta$$

Une fois que cette règle est appliquée, nous obtenons le séquent :

$$« x = (y,z), (y,z) \in a, y \in \emptyset \vdash x \in \emptyset »$$

Super Zenon traite ensuite l'hypothèse $y \in \emptyset$ dans l'étape 4 ce qui clôt la preuve.

À partir de la preuve de Super Zenon, il est désormais aisé de produire une preuve courte et informelle de la manière suivante :

– Pour montrer que $\emptyset \lhd a = \emptyset$, nous devons considérer les deux cas suivants : étant donné $x \in \emptyset$, nous devons montrer que $x \in \emptyset \lhd a$, et étant donné $x \in \emptyset \lhd a$, nous devons montrer que $x \in \emptyset$;

  1. Si $x \in \emptyset$ alors nous avons une contradiction ;
  2. Si $x \in \emptyset \lhd a$ alors $x = (y,z)$, $(y,z) \in a$, et $y \in \emptyset$, ce qui est absurde comme précédemment.

6.5.2 Preuve d'un problème TPTP

Comme Super Zenon peut traiter toutes les théories, il peut être utilisé dans plusieurs contextes, en particulier pour les théories de la bibliothèque de problèmes TPTP [59]. Nous allons présenter une preuve générée par Super Zenon provenant de la bibliothèque de problèmes TPTP de la catégorie géométrie. Ce problème (GEO170+3) dit que si deux points distincts appartiennent à une droite, alors cette droite relie ces deux points. La preuve de cet exemple utilise les axiomes suivants de la géométrie constructive (nous utilisons les noms donnés dans les fichiers TPTP) :

$$\forall x, y\ (\text{distinct\_points}(x, y) \Rightarrow \neg\text{apart\_point\_and\_line}(x, \text{line\_connecting}(x, y))) \quad (ci1)$$
$$\forall x, y\ (\text{distinct\_points}(x, y) \Rightarrow \neg\text{apart\_point\_and\_line}(y, \text{line\_connecting}(x, y))) \quad (ci2)$$
$$\forall x, y, u, v\ (\text{distinct\_points}(x, y) \wedge \text{distinct\_lines}(u, v) \Rightarrow$$
$$\text{apart\_point\_and\_line}(x, u) \vee \text{apart\_point\_and\_line}(x, v) \vee$$
$$\text{apart\_point\_and\_line}(y, u) \vee \text{apart\_point\_and\_line}(y, v)) \quad (cu1)$$
$$\forall x, y\ (\text{equal\_lines}(x, y) \Leftrightarrow \neg\text{distinct\_lines}(x, y)) \quad (ax2)$$
$$\forall x, y\ (\text{incident\_point\_and\_line}(x, y) \Leftrightarrow \neg\text{apart\_point\_and\_line}(x, y)) \quad (a4)$$

où :

- distinct_points(x, y) (resp. distinct_lines(x, y)) signifie que x et y sont deux points distincts (resp. deux droites distinctes) ;
- incident_point_and_line(x, y) (resp. apart_point_and_line(x, y)) signifie que le point x appartient (resp. n'appartient pas) à la droite y ;
- equal_lines(x, y) signifie que x et y représente la même droite ;
- line_connecting(x, y) représente la droite reliant les points x et y.

Parmi ces axiomes, les axiomes $(ci2)$, $(ax2)$, et $(a4)$ sont transformés en règles de superdéduction :

$$\frac{\text{distinct\_points}(x, y)}{\neg\text{apart\_point\_and\_line}(y, \text{line\_connecting}(x, y))}\ ci2$$

$$\frac{\text{apart\_point\_and\_line}(y, \text{line\_connecting}(x, y))}{\neg\text{distinct\_points}(x, y)}\ ci2ctrp$$

$$\frac{\text{equal\_lines}(x, y)}{\neg\text{distinct\_lines}(x, y)}\ ax2$$

$$\frac{\neg\text{equal\_lines}(x, y)}{\text{distinct\_lines}(x, y)}\ \neg ax2$$

$$\frac{\text{incident\_point\_and\_line}(x, y)}{\neg\text{apart\_point\_and\_line}(x, y)}\ a4$$

L'axiome $(ci1)$ est laissé en axiome car l'une des super-règles correspondantes entre en conflit avec une des super-règles déjà calculées pour l'axiome $(ci2)$. L'axiome $(cu1)$ est aussi laissé en axiome car sa forme syntaxique empêche la transformation en super-règle. Pour l'axiome $(ax2)$, une règle de superdéduction correspondant à la contraposée est générée puisque les deux formules mises en jeu dans l'équivalence sont atomiques.

Le problème est formellement exprimé comme suit :

$$\forall x, y, z\ (\text{distinct\_points}(x, y) \wedge \text{incident\_point\_and\_line}(x, z) \wedge \\ \text{incident\_point\_and\_line}(y, z) \Rightarrow \text{equal\_lines}(z, \text{line\_connecting}(x, y)))$$

Lorsqu'il est appliqué sur ce problème, Super Zenon produit la preuve présentée dans les figures 6.2 et 6.3 (nous utilisons toujours le format de preuve le plus abstrait de Zenon). Les étapes commençant par « Axiom » sont des feuilles de l'arbre de preuve. En suivant cette preuve, il est possible de construire la preuve suivante :

– Étant donnés les points `T4` (étape 2), `T6` (étape 3), et la droite `T8` (étape 4) tels que
`H10: distinct_points(T4,T6)`,
`H7: incident_point_and_line(T4,T8)`,
et `H11: incident_point_and_line(T6,T8)`,
nous devons montrer que
`H9` niée : `equal_lines(T8,line_connecting(T4,T6))` ;

– En appliquant la super-règle *a4* sur les hypothèses
`H7: incident_point_and_line(T4,T8)` et
`H11: incident_point_and_line(T6,T8)`,
nous avons
`H12: -. apart_point_and_line(T4,T8)` (étape 5) et
`H13: -. apart_point_and_line(T6,T8)` (étape 6) ;

– En utilisant l'axiome (*ax2*), le but
`H9` niée : `equal_lines(T8,line_connecting(T4,T6))` à prouver est équivalent à
`H14` niée : `-. distinct_lines(T8,line_connecting(T4,T6))` (étape 7) ;

– En utilisant l'axiome (*cu1*) (l'hypothèse `H2`) avec `T4`, `T6`, `T8`, et
`line_connecting(T4,T6)` (étapes 8 à 11), nous devons montrer le but précédent (`H14` niée) dans les cas suivants :

1. Étant donné
`H20: -. distinct_points(T4,T6)`, nous avons aussi
`H10: distinct_points(T4,T6)`, ce qui est absurde (étape 12) ;
2. Étant donné
`H21: -. distinct_lines(T8,line_connecting(T4,T6))`, c'est exactement le

```
fof(geometry_conjecture, conjecture,
  (! [X, Y, Z] : ((distinct_points (X, Y) &
    incident_point_and_line (X, Z) &
    incident_point_and_line (Y, Z)) =>
    equal_lines (Z, line_connecting(X, Y))))).
(* PROOF-FOUND *)
   1. H0: (-. (All X, (All Y, (All Z, (((distinct_points X Y) /\
            ((incident_point_and_line X Z) /\
            (incident_point_and_line Y Z))) =>
            (equal_lines Z (line_connecting X Y)))))))
      H1: (All X, (All Y, ((distinct_points X Y) =>
            (-. (apart_point_and_line X (line_connecting X Y))))))
      H2: (All X, (All Y, (All U, (All V, (((distinct_points X Y) /\
            (distinct_lines U V)) => ((apart_point_and_line X U) \/
            ((apart_point_and_line X V) \/
            ((apart_point_and_line Y U) \/
            (apart_point_and_line Y V)))))))))
      ### [NotAllEx H0] --> 2
   2. H3: (-. (All Y, (All Z, (((distinct_points T_4 Y) /\
            ((incident_point_and_line T_4 Z) /\
            (incident_point_and_line Y Z))) =>
            (equal_lines Z (line_connecting T_4 Y))))))
      ### [NotAllEx H3] --> 3
   3. H5: (-. (All Z, (((distinct_points T_4 T_6) /\
            ((incident_point_and_line T_4 Z) /\
            (incident_point_and_line T_6 Z))) =>
            (equal_lines Z (line_connecting T_4 T_6)))))
      ### [NotAllEx H5] --> 4
   4. H7: (incident_point_and_line T_4 T_8)
      H9: (-. (equal_lines T_8 (line_connecting T_4 T_6)))
      H10: (distinct_points T_4 T_6)
      H11: (incident_point_and_line T_6 T_8)
      ### [Extension/szen/a4 H7 H12 H4 H8] --> 5
   5. H12: (-. (apart_point_and_line T_4 T_8))
      ### [Extension/szen/a4 H11 H13 H6 H8] --> 6
   6. H13: (-. (apart_point_and_line T_6 T_8))
      ### [Extension/szen/not_ax2 H9 H14 H8 H15] --> 7
   7. H14: (distinct_lines T_8 (line_connecting T_4 T_6))
      ### [All H2] --> 8
   8. H16: (All Y, (All U, (All V, (((distinct_points T_4 Y) /\
            (distinct_lines U V)) => ((apart_point_and_line T_4 U) \/
            ((apart_point_and_line T_4 V) \/
            ((apart_point_and_line Y U) \/
            (apart_point_and_line Y V)))))))))
      ### [All H16] --> 9
```

FIGURE 6.2 – Preuve du problème de géométrie n° 170+3 de TPTP (Partie 1)

```
 9. H17: (All U, (All V, (((distinct_points T_4 T_6) /\
           (distinct_lines U V)) => ((apart_point_and_line T_4 U) \/
           ((apart_point_and_line T_4 V) \/
           ((apart_point_and_line T_6 U) \/
           (apart_point_and_line T_6 V)))))))
    ### [All H17] --> 10
10. H18: (All V, (((distinct_points T_4 T_6) /\
           (distinct_lines T_8 V)) =>
           ((apart_point_and_line T_4 T_8) \/
           ((apart_point_and_line T_4 V) \/
           ((apart_point_and_line T_6 T_8) \/
           (apart_point_and_line T_6 V))))))
    ### [All H18] --> 11
11. H19: (((distinct_points T_4 T_6) /\
           (distinct_lines T_8 (line_connecting T_4 T_6))) =>
           ((apart_point_and_line T_4 T_8) \/
           ((apart_point_and_line T_4 (line_connecting T_4 T_6)) \/
           ((apart_point_and_line T_6 T_8) \/
           (apart_point_and_line T_6 (line_connecting T_4 T_6))))))
    ### [DisjTree H19] --> 12 13 14 15 16 17
12. H20: (-. (distinct_points T_4 T_6))
    ### [Axiom H10 H20]
13. H21: (-. (distinct_lines T_8 (line_connecting T_4 T_6)))
    ### [Axiom H14 H21]
14. H22: (apart_point_and_line T_4 T_8)
    ### [Axiom H22 H12]
15. H23: (apart_point_and_line T_4 (line_connecting T_4 T_6))
    ### [All H1] --> 18
18. H24: (All Y, ((distinct_points T_4 Y) =>
           (-. (apart_point_and_line T_4 (line_connecting T_4 Y)))))
    ### [All H24] --> 19
19. H25: ((distinct_points T_4 T_6) =>
           (-. (apart_point_and_line T_4 (line_connecting T_4 T_6))))
    ### [Imply H25] --> 20 21
20. H20: (-. (distinct_points T_4 T_6))
    ### [Axiom H10 H20]
21. H26: (-. (apart_point_and_line T_4 (line_connecting T_4 T_6)))
    ### [Axiom H23 H26]
16. H27: (apart_point_and_line T_6 T_8)
    ### [Axiom H27 H13]
17. H28: (apart_point_and_line T_6 (line_connecting T_4 T_6))
    ### [Extension/szen/ci2ctrp H28 H20 H4 H6] --> 22
22. H20: (-. (distinct_points T_4 T_6))
    ### [Axiom H10 H20]
```

FIGURE 6.3 – Preuve du problème de géométrie n° 170+3 de TPTP (Partie 2)

but à prouver (`H14` niée), il est donc prouvé par hypothèse (étape 13) ;

3. Étant donné
   `H22: apart_point_and_line(T4,T8)`, nous avons aussi
   `H12: -. apart_point_and_line(T4,T8)` en hypothèse,
   ce qui est absurde (étape 14) ;
4. Étant donné
   `H23: apart_point_and_line(T4,line_connecting(T4,T6))`,
   nous instancions l'axiome (*ci1*) (l'hypothèse `H1`) avec `T4` (étape 18),
   `T6` (étape 19). Comme on a `distinct_points(T4,T6)` (étape 20), on en déduit
   (toujours en appliquant l'axiome (*ci1*)) :
   `H26: -. apart_point_and_line(T4,line_connecting(T4,T6))`,
   ce qui est absurde (étape 21) ;
5. Étant donné
   `H27: apart_point_and_line(T6,T8)`, nous avons aussi
   `H13: -. apart_point_and_line(T6,T8)` en hypothèse,
   ce qui est absurde (étape 16) ;
6. Étant donné
   `H28: apart_point_and_line(T6,line_connecting(T4,T6))` utilisée avec la
   contraposée de l'axiome (*ci2ctrp*), nous avons
   `H20: -. distinct_points(T4,T6)` (étape 17), ce qui est absurde puisque nous
   avons aussi
   `H10: distinct_points(T4,T6)` en hypothèse (étape 22).

Si l'on s'abstrait davantage de la preuve produite par Super Zenon, on peut donner la preuve informelle faite ci-dessous qui est plus proche d'une démonstration faite dans un livre de géométrie.

– Étant donnés les points x, y, et la droite z tels que distinct_points(x, y),
  incident_point_and_line(x, z), et incident_point_and_line(y, z),
  nous devons montrer que equal_lines$(z,$ line_connecting$(x, y))$;
– À partir des hypothèses incident_point_and_line(x, z) et
  incident_point_and_line(y, z), nous avons ¬apart_point_and_line(x, z) et

¬apart_point_and_line(y, z) en utilisant l'axiome (*a4*) ;

– En utilisant l'axiome (*ax2*), le but equal_lines$(z, \text{line_connecting}(x, y))$ à prouver est équivalent à ¬distinct_lines$(z, \text{line_connecting}(x, y))$;

– En utilisant l'axiome (*cu1*) avec x, y, z, et line_connecting(x, y), nous devons montrer le but précédent dans les cas suivants :

1. Étant donné ¬distinct_points(x, y), nous avons aussi distinct_points(x, y) en hypothèse, ce qui est absurde ;
2. Étant donné ¬distinct_lines$(z, \text{line_connecting}(x, y))$, c'est exactement le but à prouver, il est donc prouvé par hypothèse ;
3. Étant donné apart_point_and_line(x, z), nous avons aussi ¬apart_point_and_line(x, z) en hypothèse, ce qui est absurde ;
4. Étant donné apart_point_and_line$(x, \text{line_connecting}(x, y))$, nous utilisons l'axiome (*ci1*) avec x, y, et distinct_points(x, y), pour obtenir ¬apart_point_and_line$(x, \text{line_connecting}(x, y))$, ce qui est absurde ;
5. Étant donné apart_point_and_line(y, z), nous avons aussi ¬apart_point_and_line(y, z) en hypothèse, ce qui est absurde ;
6. Étant donné apart_point_and_line$(y, \text{line_connecting}(x, y))$ utilisé avec la contraposée de l'axiome (*ci2*) avec x, y, nous avons ¬distinct_points(x, y), ce qui est absurde puisque nous avons aussi distinct_points(x, y) en hypothèse.

À partir de la preuve générée par Super Zenon, nous avons donc réussi à reconstruire une preuve compréhensible par un humain.

6.6 Résultats expérimentaux sur la base de règles

Nous présentons dans cette section les résultats expérimentaux que nous avons obtenus sur la base de règles Siemens IC-MOL avec Super Zenon appliqué à la théorie de B. Nous avons lancé les tests dans les mêmes conditions que les tests précédents (à savoir sur un ordinateur Intel Pentium D Xeon 3.60GHz/32Go). Sur les 2031 règles bien typées

et bien définies de la base, notre nouvelle approche en démontre 946. Démontrer ici signifie uniquement que Super Zenon réussit à trouver une preuve. En effet, actuellement Super Zenon ne génère pas de preuve Coq mais uniquement des preuves dans des formats abstraits. En réalité le sous-ensemble des règles que notre nouvelle approche peut traiter, parmi les règles bien typées et bien définies et en écartant les opérateurs non traités, comprend 1534 règles (ce sont exactement les mêmes règles que pour l'extension de Zenon à B du chapitre précédent).

Si l'on compare avec l'approche de Zenon étendu à B (qui démontrait sur ce sous-ensemble 1443 règles), Super Zenon en démontre 497 de moins. Cela est dû à l'extension dynamique des règles qui est plus technique que l'écriture *ad hoc* des règles dans un fichier. Ainsi, l'application dynamique des règles d'instanciation contenant plusieurs métavariables dans une super-règle n'est pas encore terminée dans l'implantation de Super Zenon. Or, cela est très utilisé dans la théorie B. Même si l'implantation n'est pas finie, on peut d'ores et déjà utiliser Super Zenon pour calculer des super-règles et améliorer l'extension statique de Zenon à B.

6.7 Résultats expérimentaux sur les problèmes TPTP

Nous avons lancé Super Zenon sur les problèmes proposés dans TPTP sur un ordinateur Intel Pentium D Xeon 3.60GHz/32Go. Super Zenon s'appuyant sur la méthode des tableaux, nous avons sélectionné uniquement les problèmes de la logique du premier ordre qui ne sont pas exprimés sous forme clausale (c'est-à-dire les formules « fof » uniquement). Afin de comparer ses résultats avec Zenon, nous avons lancés les mêmes expérimentations, dans les mêmes conditions, avec Zenon. Les résultats sont résumés dans le tableau 6.1 où ils sont classés par catégorie de problème (par exemple les catégories SET et GEO regroupe respectivement des problèmes issus de la théorie des ensembles et qui concernent la géographie). Les deux premières lignes représentent les résultats de Zenon et Super Zenon dans une catégorie donnée. La troisième ligne donne le nombre de total de chaque catégorie. La première colonne résume les résultats totaux sur les deux prouveurs. Les autres colonnes représentent les résultats des prouveurs dans une catégorie.

	Total	AGT	ALG	BOO	CAT	COM	CSR	GEO	GRA
Zenon	1646	18	58	0	2	13	129	221	3
Super Zenon	1765	17	59	0	2	13	135	223	3
Total	6644	52	212	1	46	29	724	359	19
	GRP	HAL	HWV	KLE	KRS	LAT	LCL	MED	MGT
Zenon	5	1	0	7	74	12	57	7	51
Super Zenon	5	1	0	7	78	13	58	9	50
Total	185	6	20	224	87	379	264	9	67
	MSC	NLP	NUM	PRO	PUZ	REL	RNG	SCT	SET
Zenon	5	8	122	2	15	0	34	9	147
Super Zenon	5	9	141	1	16	0	38	9	202
Total	5	25	565	63	23	108	152	85	462
	SWV	SWW	SYN	SYO	TOP	SEU	SWB	SWC	
Zenon	162	18	275	1	4	102	23	61	
Super Zenon	180	17	275	2	5	105	23	64	
Total	322	290	290	7	104	899	139	422	

TABLE 6.1 – Résultats de Zenon et Super Zenon sur les formules fof de TPTP

Nous voyons dans ce tableau que Super Zenon démontre sur la totalité des problèmes fof 119 problèmes de plus que Zenon. Bien que cela ne représente que près de 2% de plus de problèmes démontrés par rapport à la totalité, cela montre que la superdéduction, implantée dans Super Zenon, permet de prouver plus de problèmes. Les meilleurs gains obtenus avec Super Zenon sont dans la catégorie MED (médecine) : on passe de 78% de problèmes démontrés par Zenon à 100% par Super Zenon). La catégorie dans laquelle il y a le plus de différence entre le nombre de problèmes prouvés par Super Zenon et Zenon est SET, dans laquelle Super Zenon démontre 55 problèmes en plus. Cela montre que Super Zenon est, en pratique, assez prometteur pour la théorie des ensembles.

Nous avons aussi évalué Super Zenon sur l'ensemble des problèmes insatisfiables de la logique de premier ordre issus de la bibliothèque TPTP. Sur ces 768 problèmes, Super Zenon n'en démontre aucun. On peut donc avoir une certaine confiance quant à la correction de l'implantation de cet outil.

Chapitre 7

Résultats expérimentaux et interface graphique

Dans les chapitres précédents, nous avons exposé différentes implantations permettant de prouver des formules issues de la théorie B. Ces implantations vont d'une approche autarcique utilisant le langage de tactique de Coq $\mathcal{L}_{\text{tac}}$, à une approche sceptique qui combine $\mathcal{L}_{\text{tac}}$ au prouveur externe Zenon (où soit $\mathcal{L}_{\text{tac}}$ normalise une formule ensembliste soit il permet uniquement de faire communiquer Coq avec Zenon). Dans ce chapitre, nous résumons les résultats obtenus avec ces approches (voir la section 7.1). Puis, nous étudions l'impact de BCARe sur les développements B fait à Siemens IC-MOL (voir la section 7.2). Enfin, nous présentons une interface graphique permettant de réaliser automatiquement le processus de vérification des règles ajoutées de l'Atelier B sur un ensemble de règles (voir la section 7.3).

7.1 Comparaison des approches

Nous avons lancé notre outil BCARe sur les 5077 règles inductives de la base de règles Siemens IC-MOL avec un ordinateur Intel Pentium D Xeon 3.60GHz/32Go, avec des limitations pour les preuves de 30 minutes et de 10 Go en consommation mémoire. Il existe plusieurs versions de cette base, celle étudiée ici n'est pas la dernière mais peut contenir des règles fausses.

Dans un premier temps, un composant de BCARe réifie les règles : sur l'ensemble de

5077 règles, 3017 règles sont traduites en lemmes BCoq à vérifier. Parmi les 2060 règles restantes, 1 règle est rejetée à cause de gardes qui ne respectent pas la syntaxe de l'Atelier B. 2059 règles ne sont pas réifiées car elles contiennent au moins une construction non traitée par BCARe ou si elles sont traitées par BCARe, elles sont utilisées dans des cas particuliers.

La règle qui ne respecte pas la syntaxe des gardes est une règle de la base de règles de l'Atelier B appelée « s1.21 » qui est la suivante :

$$(\mathsf{band}(\mathsf{binhyp}(\underline{\neg(x \in \mathsf{dom}(f)), \mathsf{bsearch}(\{x\}, A \cup B, C)})))\ \wedge$$
$$(\mathsf{blvar}(Q)) \wedge (Q \backslash (x, f)) \Rightarrow f[A \cup B] == f[C]$$

où band est une garde qui rassemble deux gardes par une conjonction (comme le $\wedge$), et bsearch est une garde qui permet de rechercher un motif dans une formule et de renommer la formule dans lequel le motif a été supprimé. Ici, il y a un problème de parenthèses (voir les parenthèses soulignées) : band doit avoir deux gardes en argument selon sa définition, or ici elle n'a comme argument que $\mathsf{binhyp}(\neg(x \in \mathsf{dom}(f)), \mathsf{bsearch}(\{x\}, A \cup B, C))$. Cela est détecté par notre composant de BCARe qui par conséquent échoue dans la réification de cette règle. Concrètement, cette règle ne peut pas être appliquée dans une preuve de l'Atelier B, mais nous la considérons comme fausse.

Pour 1996 autres règles, les constructions non traitées par BCARe sont par exemple des opérateurs arithmétiques (min), des opérateurs traitant des listes (size), des gardes (bvrb). Concernant les constructions traitées par BCARe, elles doivent être présentées sous certaines formes. Par exemple, BCARe traite de la construction λ sous la contrainte qu'elle soit de la forme $\lambda x.(x \in s|E)$, comme cela est spécifié dans le B-Book. Mais des règles ajoutées contiennent des λ mal formées, comme par exemple $\lambda x.(x \in s \vee x \in t|E)$. Les règles mettant en jeu des constructions « mal formées » comme celle-ci sont rejetées par BCARe. On parlera alors de cas particulier, cela concerne 63 règles de la base de règles.

Ensuite, un composant de BCARe essaie d'inférer un type sur les règles réifiées. Pour 2972 règles des 3017 règles réifiées, l'inférence de type s'est bien déroulée. Parmi les 45 restantes, 31 contiennent une expression non connue de BCARe (« blankrule », seules des règles de l'Atelier B sont concernées), 6 règles sont fausses et 8 ont un problème de type dans les gardes sur la non-liberté de variable ou bfresh. Par exemple, dans la règle suivante

« SimplifyRelFonXY.15 » (issue de la base de règles de l'Atelier B) :

$$(\mathsf{binhyp}(a \in \mathsf{dom}(s))) \wedge (\mathsf{blvar}(Q)) \wedge (Q \backslash (a \in \mathsf{dom}(s) \wedge r)) \Rightarrow (r \cup s)(a) == s(a)$$

$a \in \mathsf{dom}(s) \wedge r$ et $(r \cup s)(a) == s(a)$ ne permettent pas de déduire un type pour r puisque dans le premier cas r serait un prédicat et dans le second cas, il serait un ensemble. Notre composant rejette donc ce type de règle. Il semblerait que l'Atelier B, quant à lui, autorise ce genre de règle.

La tactique qui vérifie le typage réussit à prouver le lemme de typage pour 2409 règles parmi les 2972. Pour 487 règles, elle échoue car ces règles contiennent de l'arithmétique, qui n'est pas encore géré par BCARe. Enfin 76 règles contiennent des cas particuliers non traités actuellement par BCARe.

La bonne définition sans propagation est ensuite vérifiée. Cela consiste à ne pas supposer que le but est bien défini. La bonne définition avec propagation consiste à vérifier que les règles, à partir d'un but bien défini, n'introduit pas d'expression mal définie. Parmi les 2409 règles bien typées, notre outil prouve que 2031 règles sont bien définies (sans tenir compte de la propagation) et donc échoue pour 378 règles. Cela n'implique pas que les règles restantes sont mal définies car certaines règles respectent la bonne définition avec propagation et notre tactique de preuve pour la bonne définition n'est pas complète. Tous les résultats sont résumés dans les tableaux 7.1 et 7.2.

	Réification	Inférence de type	Vérification de type	Bonne définition
OK	3017	2972	2409	2031
KO	2060	45	563	378
Total	5077			

TABLE 7.1 – Résultats pour la vérification du typage et de la bonne définition

Désormais, nous considérons uniquement le sous-ensemble de règles pour lequel le typage a pu être démontré par Coq, ainsi que la bonne définition, ce qui représente 2031 règles. Notre première approche $\mathcal{L}_{\text{tac}}$ démontre la validité de 856 règles de ce sous-ensemble. L'approche utilisant Zenon trouve une preuve pour 1426 règles alors que celle utilisant l'extension de Zenon à la méthode B en prouve 1443. Enfin, Super Zenon appliqué à la méthode B prouver 946 règles. Ici, nous ne prenons pas en compte la vérification des preuves trouvées

Raison de l'échec	Nombre de règles	Réification	Inférence de type	Typage
Règles fausses	7	1	6	0
Non traitée	2514	1996	31	487
Cas particuliers	139	63	0	76
Typage	8	0	8	0

TABLE 7.2 – Analyse des échecs

dans BCoq mais uniquement le fait que Zenon (ou Zenon étendu) trouve une preuve ou non. De plus, le sous-ensemble de règles traitées par Zenon étendu à la méthode B est plus petit que celui traité par l'approche utilisant Zenon sans extension. Ce sous-ensemble est exactement le même que pour Super Zenon. Les résultats sont résumés dans le tableau 7.3.

	$\mathcal{L}_{\text{tac}}$	Zenon	Zenon-B	Super Zenon
OK	856	1426	1443	946
KO	1175	605	588	1085

TABLE 7.3 – Comparaison du nombre de règles prouvées selon l'approche

Alors qu'entre l'approche $\mathcal{L}_{\text{tac}}$ et l'approche Zenon, il y a un important écart sur le nombre de règles prouvées (570), entre les approches utilisant Zenon avec ou sans extension, il y a un important gain en temps de preuve (en moyenne est observé un coefficient de 172) comme le montre la figure 7.1. Dans ce graphe, nous comparons les temps de preuve de notre approche utilisant la prénormalisation puis Zenon avec notre extension de Zenon à la méthode B (donc sans étape de normalisation). Un point représente le résultat pour une règle et les axes des abscisses et des ordonnées représentent respectivement le temps pour Zenon (reconstruction de la preuve exclue) et le temps pour Zenon étendu à B. Pour chaque point situé sous la courbe tracée avec des triangles, notre extension basée sur la superdéduction est au moins 10 fois plus rapide que l'autre approche, sous la courbe avec des croix, c'est 100 fois plus rapide, et sous la courbe avec des carrés, c'est 1000 fois plus rapide. Nous pouvons observer qu'il y a de nombreuses règles pour lesquelles la preuve est de 10 à 1000 fois plus rapide avec la superdéduction. En moyenne, la superdéduction trouve les preuves 172 fois plus rapidement (le meilleur rapport étant de 7749). Cette dernière approche est donc très prometteuse. Mais du développement est encore nécessaire

pour la reconstruction de la preuve en BCoq.

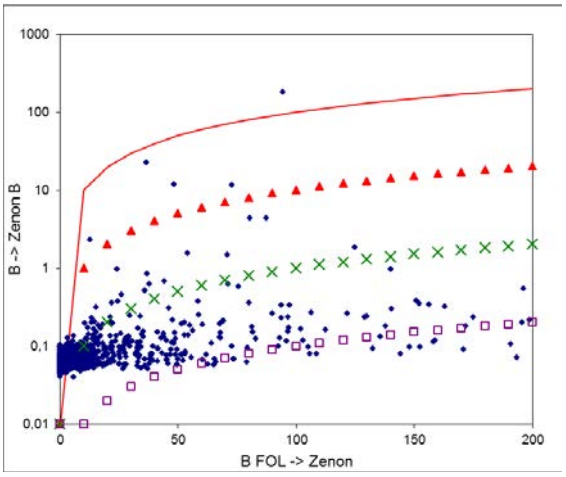

FIGURE 7.1 – Comparaison sur les temps de preuve

Concernant Super Zenon, il démontre moins de règle que Zenon étendu à B. Cela s'explique par le développement non terminé de cet outil. Ainsi, à l'heure actuelle, il ne gère pas l'application dynamique des règles d'instanciation lorsqu'il y a plusieurs métavariables.

L'ensemble des composants BCARe a permis de détecter au total 11 règles fausses dans la base de règles de Siemens IC-MOL dont 2 seulement étaient connues comme fausses. Parmi les 11 règles fausses, l'une d'entre elles ne respecte pas la syntaxe des gardes de l'Atelier B. 6 de ces règles sont fausses à cause d'un problème de typage. Elles ont été découvertes par le composant de BCARe qui essaie d'inférer des types et qui a alors échoué. Les 4 autres ont été détectées lors de la vérification de la validité des règles. En effet, Zenon a généré des contre-modèles pour ces 4 règles. Le nombre de règles fausses trouvées selon les étapes de vérification est résumé dans le tableau 7.4

Nombre	Réification	Inférence de type	Validité
11	1	6	4

TABLE 7.4 – Détection des règles fausses

7.2 Utilisation industrielle de BCARe

Lors de cette thèse, nous avons été amené à vérifier un ensemble de règles issues d'un développement en cours. Cet ensemble comprend des règles déjà inclues dans la base de règles interne à Siemens IC-MOL ainsi que des nouvelles règles ajoutées spécifiquement pour ce développement. Au total, on dénombre 298 règles dans cet ensemble.

Parmi ces règles, BCARe a permis de démontrer automatiquement la validité de 119 règles et de détecter une règle fausse (qui en réalité fait déjà partie de la base de règles de Siemens IC-MOL). Cela a permis aux développeurs de corriger directement le développement qui utilisait cette règle. Dans ce cas précis, la règle était fausse car il manquait une condition sur un ensemble qui ne devait pas être vide. L'impact de la règle fausse a donc été mineure sur le développement en cours. Néanmoins, si cela avait été découvert une fois le développement terminé, cela aurait engendré des coûts supplémentaires puisque le temps prévu initialement pour le développement aurait été rallongé pour le rejeu de la preuve concernée par la règle fausse.

Au final, durant cette thèse, BCARe a mis en évidence 11 règles fausses dans la base de règles de Siemens IC-MOL (voir tableau 7.4) dont 2 seulement étaient déjà connues par le service qui vérifie les règles. Nous pouvons remarquer que toutes les règles qui avaient été validées à tort, avaient été validées manuellement. BCARe a donc permis d'améliorer considérablement la confiance dans la base de règles interne. BCARe a aussi permis de valider de manière indépendante les outils déjà utilisés pour la vérification des règles et de les compléter si besoin. De plus, en cas d'échec de nos approches automatiques, BCARe offre un environnement de preuve interactif. Ainsi, nous avons un cadre formel (différent du papier/crayon utilisé dans ces cas là) qui permet d'effectuer des preuves sans risque d'erreur.

Lorsqu'une règle est détectée fausse, toutes les preuves dans laquelle cette règle est utilisée doivent être refaites. À ce jour, toutes les règles fausses trouvées n'ont pas engendré de redéveloppement complet de nos logiciels. Nous n'avons pas eu non plus de cas où le modèle B a dû être modifié pour pouvoir effectuer les preuves des obligations de preuve sans la règle fausse. Dans le pire des cas que nous avons rencontrés, les preuves mettant

en jeu la règle fausse doivent être modifiées considérablement. Dans le meilleur des cas, la correction de la règle n'influe pas sur la preuve. Cela arrive par exemple, lorsqu'il manque une condition dans la règle, et que cette condition est déjà en hypothèse dans l'obligation de preuve. Dans tous les cas, plus la détection des règles fausses est en amont du cycle de développement, moins le coût du développement est important.

7.3 Interface graphique

La plate-forme BCARe est destinée à aider les experts à vérifier les règles de preuve ajoutées par les développeurs. À terme, elle pourrait être également mise à disposition des développeurs de manière à ce qu'ils puissent vérifier au plus tôt ces règles. Afin de faciliter non seulement l'utilisation de BCARe mais aussi le développement de BCARe lui-même, une interface a été implantée et permet notamment une gestion élaborée de la vérification des règles.

L'interface (développée en PHP par un stagiaire) permet de sélectionner les règles à vérifier selon différents critères (comme le nom ou les opérateurs utilisés), de choisir la méthode de preuve (il s'agit uniquement de $\mathcal{L}_{\text{tac}}$ et Zenon avec prénormalisation pour le moment, mais d'autres outils pourraient être ajoutés) pour démontrer le lemme de la règle (voir section 2.3), de renseigner certains paramètres comme la limite en mémoire ou en temps. La figure 7.2 illustre la page d'accueil de l'interface. Il est ensuite possible d'afficher les résultats, et d'éditer les rapports selon différents formats comme XML, LaTeX ou PDF. Les résultats sont affichés en temps réel et sont présentés sous la forme d'un tableau dont chaque colonne représente une étape du processus décrit dans la section 2.3. La figure 7.3 donne un exemple de résultats produits par la vérification d'un ensemble de trois règles. On remarquera que la partie Zenon est divisée en un certain nombre d'étapes intermédiaires (à savoir : la mise au premier ordre, la traduction de la règle pour Zenon, la preuve par Zenon, et la reconstruction de la preuve dans BCoq) afin de faciliter le travail de mise au point de la plate-forme. Les résultats sont conservés dans une base de données MySQL pour permettre d'effectuer des statistiques via des requêtes (voir la figure 7.4). Pour cela, il suffit de faire des requêtes sur des identifiants prédéfinis (comme `capture` pour la capture de variable) et leurs valeurs possibles (`'OK'` ou `'KO'` pour `capture`). Par exemple, on peut sélectionner

toutes les règles vérifiées par l'outil pour lesquelles les temps totaux des preuves du lemme de validité avec $\mathcal{L}_{\text{tac}}$ et avec Zenon sont supérieurs à 100 secondes. Cela correspond à la requête `ltac_total > 100 && zenon_total > 100`.

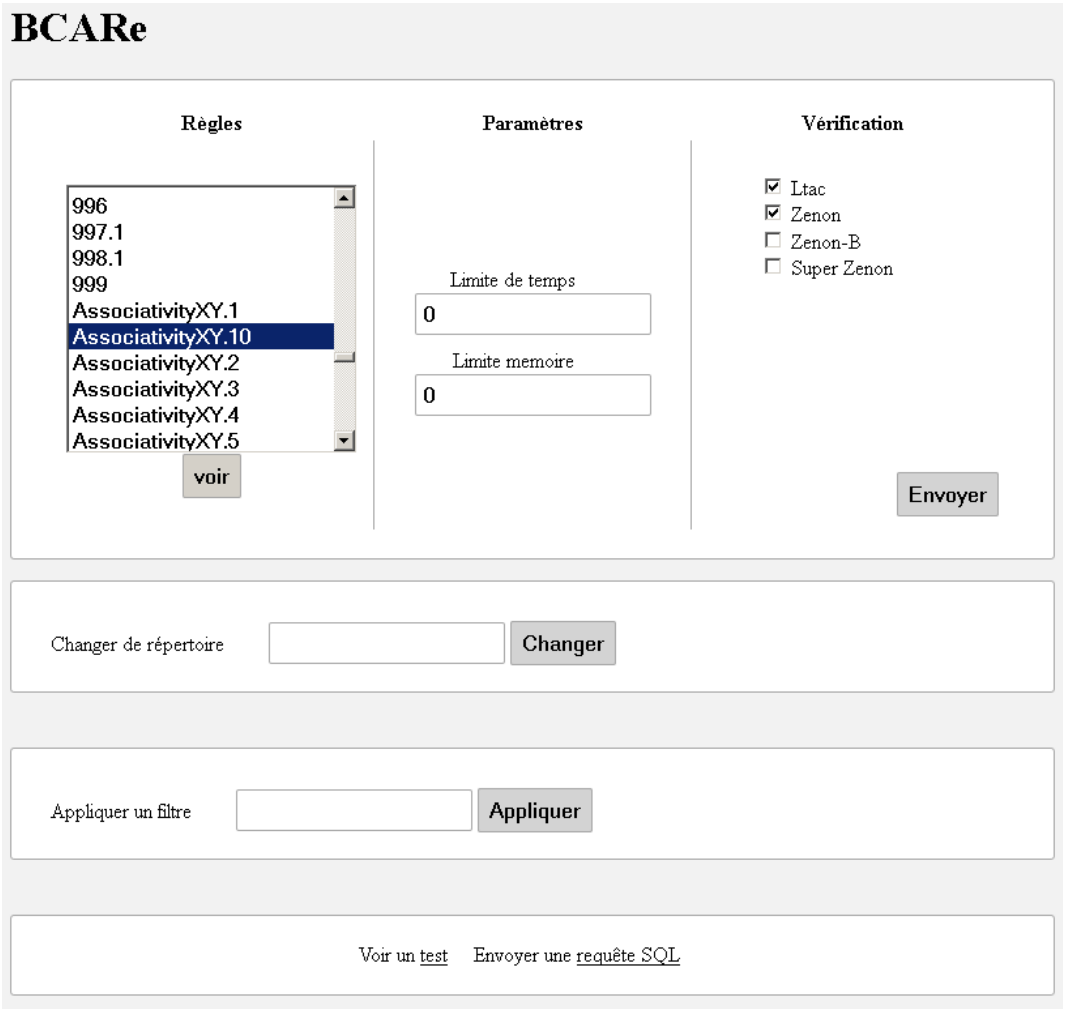

FIGURE 7.2 – Page d'accueil de l'interface BCARe

				Zenon							
Nom	Capture variable	Typage	Bonne définition	to_FOL (en s)	Interprétation (en s)	Zenon (en s)	Reconstruction (en s)	Compilation (en s)	apply (en s)	Qed (en s)	Temps total (en s)
1113	OK	KO	OK	KO	KO	KO	KO	KO	KO	KO	KO
1115.1	OK	OK	OK	0.076004	0	0.064003	0	0.904057	0.040003	0.040002	1.12407
3153	OK	OK	OK	1.78011	0	0.060003	0	13.1648	0.336021	0.584036	15.925
Nb OK :	3	2	3	2	2	2	2	2	2	2	

Statistiques		
Temps total (en s)	Nombre de règles vérifiées par Zenon	Nombre de règles vérifiées
17.04907	2	2/3

FIGURE 7.3 – Exemple de résultats produits par la vérification d'un ensemble de trois règles

Exemples de requêtes:

```
zenon_qed inferieur à 1000 :
      "zenon_qed < 1000"

ltac_total et zenon_total superieur à 100 :
     "ltac_total > 100 && zenon_total > 100"

capture de variable réussie :
     "capture = 'OK'"

identifiants inclus entre 10 et 50:
     "id_test<51 && id_test>9"
```

FIGURE 7.4 – Requêtes sur la base de données

Conclusion

Dans cette thèse, nous avons étendu l'environnement BCARe, qui offre un cadre formel et mécanisé permettant de vérifier les règles de l'Atelier B. Cet environnement repose sur un ensemble d'outils, développés par Siemens IC-MOL, et comprend un plongement profond de la théorie de B dans la logique de l'outil d'aide à la preuve Coq, appelé BCoq. BCARe permet de vérifier automatiquement que les règles ne peuvent introduire de capture de variable. Concernant la bonne définition, le typage et la validité d'une règle, BCARe génère automatiquement les obligations de preuve dans BCoq (c'est-à-dire écrites en Coq en utilisant les opérateurs de BCoq) qui sont nécessaires pour vérifier une règle. Cette thèse consolide BCARe, notamment grâce à une formalisation de la traduction d'une règle de l'Atelier B dans BCoq. Elle contribue aussi à l'automatisation de la preuve de validité d'une règle qui était totalement interactive avant cette thèse. Pour cela, trois approches ont été développées.

La première consiste en une heuristique utilisant $\mathcal{L}_{\text{tac}}$, le langage de tactique de Coq. Cela se révèle assez performant puisqu'elle permet de prouver la validité de 856 règles, mais l'algorithme de preuve ne se révèle pas assez efficace pour envisager la preuve de l'ensemble des règles.

La deuxième approche utilise le prouveur automatique de théorèmes Zenon. La formule exprimée initialement en B est alors normalisée pour obtenir une formule de la logique du premier ordre. Celle-ci est ensuite interprétée afin que Zenon essaye de la prouver. Lorsque Zenon trouve une preuve, celle-ci est traduite en preuve B afin d'être rejouée dans BCoq. Cette approche est bien plus performante que la précédente puisqu'elle démontre la validité de 1426 règles. De plus, elle permet de détecter facilement des règles fausses. Néanmoins, l'étape de prénormalisation la ralentit considérablement.

Dans la dernière approche, l'étape de prénormalisation est totalement supprimée. Pour cela, Zenon a été étendu à la théorie B grâce à la superdéduction : le noyau de règles original de Zenon a été complété par de nouvelles règles traitant des opérateurs de la théorie de B. Ces règles, appelées règles de superdéduction, ont l'avantage de raccourcir certaines étapes de déduction et donc d'améliorer le temps de recherche de preuve. Cette approche permet de démontrer la validité de 1443 règles. En plus de démontrer un peu plus de règles, cette approche permet un gain de temps considérable puisque les preuves sont trouvées en

moyenne 172 fois plus rapidement qu'avec la précédente.

Cette dernière approche montrant des améliorations, nous avons étendu Zenon à toute théorie, toujours grâce à la superdéduction. Le nouvel outil s'appelle Super Zenon et calcule dynamiquement les nouvelles règles de superdéduction à partir d'une théorie axiomatique quelconque. Il peut notamment traiter les théories proposées par l'ensemble des problèmes TPTP. Concernant la théorie de la méthode B, Super Zenon n'est pas encore assez efficace. Cela s'explique par le fait que les super-règles calculées dans ce cas contiennent plusieurs métavariables, et que l'application de ces règles en particulier n'est pas encore implantée totalement. Apparemment, ce genre de règles est moins présentes pour les théories des problèmes TPTP. Les résultats de Super Zenon sont donc prometteurs puisqu'il prouve plus de théorèmes que Zenon sur l'ensemble des problèmes énoncés avec la logique du premier ordre de la bibliothèque TPTP.

L'utilisateur de BCARe doit avoir confiance en certains de ses composants, alors que d'autres composants reposent sur Coq. Ainsi, concernant l'encodage profond de B dans Coq, on peut considérer qu'il reflète fidèlement la théorie du B-Book car le plongement profond assure une assez bonne traçabilité de la théorie encodée. La réification des règles ajoutées dans ce plongement fait, quant à elle, partie des composants auxquels l'utilisateur doit faire confiance. Ensuite, l'inférence de type n'est pas prouvée correcte, mais elle est vérifiée *a posteriori* par la preuve du lemme de typage avec le langage de tactique $\mathcal{L}_{\text{tac}}$. De même, le lemme de bonne définition est démontré avec $\mathcal{L}_{\text{tac}}$. Enfin pour le lemme de validité, dans le cas de l'approche avec $\mathcal{L}_{\text{tac}}$ et l'approche utilisant Zenon avec reconstruction de la preuve obtenue en BCoq, toute l'assurance repose sur la validation de ces preuves par Coq. Pour l'approche utilisant Zenon étendu à B, l'interprétation de la formule B vers la syntaxe d'entrée de Zenon n'est pas assurée. Néanmoins, la preuve trouvée par cette extension Zenon est vérifiable en Coq. Enfin, pour l'approche avec Super Zenon, l'interprétation n'est également pas assurée. De plus, Super Zenon ne fournit pas à ce jour de preuve Coq, mais uniquement des traces compréhensibles.

Plus généralement, cette thèse a permis d'améliorer la démonstration de la sécurité des logiciels développés par Siemens IC-MOL car elle outille une partie du développement B qui, lorsque les outils de l'Atelier B échoue, est faite à la main sans aucune assistance d'outil.

Cela permet aussi d'avoir un outil indépendant de l'Atelier B et donc de valider les outils déjà utilisés. En effet, les experts qui valident les règles ajoutées de l'Atelier B à Siemens IC-MOL utilisent l'outil CHAINE VALIDATION développé par ClearSy. Cet outil ne fournit pas de trace, qu'il trouve ou non une preuve. Ses résultats ne sont donc pas vérifiables. BCARe permet alors de valider le moteur de preuve de CHAINE VALIDATION ou de compléter l'outil lorsqu'il échoue. De plus, si la règle est fausse, la preuve dans BCoq aide à avoir une indication sur ce qu'il faut modifier pour corriger la règle. Par exemple, Zenon génère des contre-modèles qui peuvent être exploités. Enfin, l'outil peut désormais être utilisé directement par les développeurs et donc engendrer un gain de coût. En effet, auparavant les règles ajoutées de l'Atelier B n'étaient vérifiées qu'à la fin du cycle de développement et donc, les modifications des règles fausses impliquaient une nouvelle intervention des développeurs.

Du point de vue scientifique, cette thèse met en valeur l'importance des traces de preuve des outils automatiques notamment pour des utilisations industrielles qui nécessitent des justifications pour démontrer la sécurité. Zenon est actuellement l'un des seuls prouveurs automatiques qui génèrent des preuves vérifiables automatiquement, par exemple, par Coq. Cela nous permet d'utiliser ce prouveur en rejouant ses preuves (après traduction) et donc de l'utiliser sans risque dans le processus de vérification existant. Une perspective à ce travail serait de définir une syntaxe de trace commune à tous les prouveurs automatiques (et aux solveurs SMT) comme cela est fait pour la syntaxe des problèmes en entrée par TPTP et SMT-LIB. Ces traces devront au moins être vérifiables automatiquement par un assistant à la preuve. La correction (et donc l'absence de bogue d'implantation) des prouveurs serait alors vérifiée par la pratique. De plus, cela participerait à favoriser l'utilisation des prouveurs. Des recherches [6, 11] sont actuellement faites sur ce sujet, mais pour l'instant, il n'y a pas de syntaxe de trace précise qui a émergé.

Enfin, le but de cette thèse étant de vérifier les règles ajoutées de l'Atelier B, il reste à continuer le développement de BCARe :

– Tout d'abord sur le court terme, la traduction des preuves fournies par Zenon (étendu à B) en preuves B permettraient d'assurer la correction de l'approche vis-à-vis de la méthode B. Ce travail avait été fait pour l'approche utilisant Zenon sans extension.

Il faut donc traiter en plus les nouvelles règles de superdéduction associées aux opérateurs B. Pour cela, le travail déjà effectué pour la reconstruction de la preuve de Zenon vers une preuve BCoq doit être étendu avec en particulier, la preuve des lemmes BCoq correspondant à chaque règle de superdéduction de Zenon.

- À plus long terme, pour simplifier les preuves dans la théorie de B, une perspective est de valider la suppression de tout ce qui est relatif au typage, une fois que celui-ci a été vérifié. Cela est déjà fait dans l'Atelier B et cela est proposé dans [4]. Néanmoins, il n'a jamais été démontré formellement que l'on pouvait s'abstraire du typage.
- De plus, il faudrait démontrer la complétude sans coupure de Zenon étendu à B et de Super Zenon. Ces preuves de complétude ne sont pas faites dans cette thèse, même si des pistes sont mentionnées.
- Le développement de BCARe peut être continué afin de pouvoir traiter à toutes les étapes du processus ainsi qu'au niveau de l'automatisation des preuves, les règles de la base de Siemens IC-MOL contenant des substitutions (200 règles), de l'arithmétique (1900 règles), des listes (800 règles), etc. Pour cela, une plate-forme de preuve regroupant les outils actuels avec des solveurs SMT par exemple, pourrait, selon le type de la règle, sélectionner les prouveurs adéquats, voire décomposer les règles en sous problèmes pour ensuite appeler les prouveurs.
- La plate-forme précédente pourrait permettre de vérifier aussi les règles ajoutées issues des développements Event-B (de Rodin). Elle pourrait aussi traiter (ou vérifier une partie seulement) des obligations de preuves, celles-ci étant plus complexes que les règles ajoutées. L'objectif du projet ANR BWare (`http://bware.lri.fr/`), débuté en octobre dernier, est justement de développer un ensemble d'outils au sein d'une plate-forme afin de vérifier des obligations de preuves issues de développement industriels B. L'un de ses outils serait une nouvelle extension de Zenon à la déduction modulo. Puis, il serait intéressant d'étendre Super Zenon à la déduction modulo afin d'étudier la combinaison de la superdéduction et de la déduction modulo au sein de Zenon. Ce dernier outil n'est pas immédiat à implanter puisque des choix doivent être faits sur ce qui doit être traité par l'une ou l'autre technique. La comparaison de ces outils donnerait une première idée de l'efficacité de ces approches en pratique.

Bibliographie

[1] J.-R. Abrial. *The B-Book, Assigning Programs to Meanings.* Cambridge University Press, Cambridge (UK), 1996. ISBN 0521496195 17, 23, 24, 47, 48, 111

[2] J.-R. Abrial. *Modeling in Event-B - System and Software Engineering.* Cambridge University Press, Cambridge (UK), 2010. ISBN 978-0-521-89556-9. 33

[3] J.-R. Abrial and D. Cansell. Click'n Prove : Interactive Proofs within Set Theory. In D. A. Basin and B. Wolff, editors, *TPHOLs*, volume 2758 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2003. 30

[4] J.-R. Abrial and L. Mussat. On Using Conditional Definitions in Formal Theories. In Bert et al. [10], pages 242–269. 48, 69, 70, 73, 75, 83, 121, 174

[5] M. Armand, G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In J.-P. Jouannaud and Z. Shao, editors, *CPP*, volume 7086 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2011. 34

[6] M. Armand, G. Faure, B. Grégoire, C. Keller, L. Thery, and B. Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In J.-P. Jouannaud and Z. Shao, editors, *Certified Programs and Proofs 2011*, volume 7086 of *Lecture notes in computer science - LNCS*, pages 135–150, Kenting, Taiwan, Province Of China, Dec. 2011. Springer. 173

[7] H. Barendregt and A. M. Cohen. Electronic Communication of Mathematics and the Interaction of Computer Algebra Systems and Proof Assistants. *J. Symb. Comput.*, 32(1/2) :3–22, 2001. 33, 81, 100

Les nombres qui suivent la référence bibliographique sont les numéros de pages où la référence est citée.

[8] C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *Proceedings of the* 19^{th} *International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, July 2007. Berlin, Germany. 32

[9] K. Berkani, C. Dubois, A. Faivre, and J. Falampin. Validation des règles de base de l'Atelier B. *Technique et Science Informatiques (TSI)*, 23(7) :855–878, 2004. 49

[10] D. Bert, J. P. Bowen, M. C. Henson, and K. Robinson, editors. *ZB 2002 : Formal Specification and Development in Z and B, 2nd International Conference of B and Z Users, Grenoble, France, January 23-25, 2002, Proceedings*, volume 2272 of *Lecture Notes in Computer Science*. Springer, 2002. 175, 176

[11] F. Besson, P. Fontaine, and L. Théry. A Flexible Proof Format for SMT : a Proposal. In P. Fontaine and A. Stump, editors, *First International Workshop on Proof eXchange for Theorem Proving - PxTP 2011*, Wroclaw, Poland, Aug. 2011. 173

[12] E. W. Beth. Semantic Entailment and Formal Derivability. *Mededelingen der Koninklijke Nederlandse Akademie van Wetenschappen*, 18(13) :309–342, 1955. 101

[13] F. Bobot, S. Conchon, E. Contejean, and S. Lescuyer. Implementing Polymorphism in SMT solvers. In C. Barrett and L. de Moura, editors, *SMT 2008 : 6th International Workshop on Satisfiability Modulo*, 2008. 32

[14] J.-P. Bodeveix and M. Filali. Type Synthesis in B and the Translation of B to PVS. In Bert et al. [10], pages 350–369. 31

[15] S. Böhme and T. Nipkow. Sledgehammer : Judgement Day. In J. Giesl and R. Hähnle, editors, *IJCAR*, volume 6173 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2010. 34

[16] R. Bonichon. TaMeD : A Tableau Method for Deduction Modulo. In D. A. Basin and M. Rusinowitch, editors, *IJCAR*, volume 3097 of *Lecture Notes in Computer Science*, pages 445–459. Springer, 2004. 118

[17] R. Bonichon, D. Delahaye, and D. Doligez. Zenon : An Extensible Automated Theorem Prover Producing Checkable Proofs. In Dershowitz and Voronkov [35], pages 151–165. 19, 32, 99, 100, 101, 104, 107, 115, 119, 136, 141

[18] R. Bonichon and O. Hermant. A Semantic Completeness Proof for TaMeD. In M. Hermann and A. Voronkov, editors, *LPAR*, volume 4246 of *LNCS*, pages 167–181, Phnom Penh, Cambodia, 2006. Springer-Verlag. 131

[19] R. J. Boulton, A. D. Gordon, M. J. C. Gordon, J. Harrison, J. Herbert, and J. V. Tassel. Experience with Embedding Hardware Description Languages in HOL. In V. Stavridou, T. F. Melham, and R. T. Boute, editors, *TPCD*, volume A-10 of *IFIP Transactions*, pages 129–156. North-Holland, 1992. 31

[20] P. Brauner, C. Houtmann, and C. Kirchner. Principles of Superdeduction. In *Logic in Computer Science (LICS)*, pages 41–50, Wroclaw (Poland), July 2007. IEEE Computer Society Press. 19, 115, 116, 117, 118, 119, 131, 141

[21] L. Burdy. *Traitement des expressions dépourvues de sens de la théorie des ensembles : Application à la méthode B.* PhD thesis, CEDRIC Laboratory, Paris, France, 2000. 69, 96

[22] G. Burel. Unbounded Proof-Length Speed-up in Deduction Modulo. In J. Duparc and T. A. Henziger, editors, *Computer Science Logic*, volume 4646 of *Lecture Notes in Computer Science*, pages 496–511. Springer, 2007. 117

[23] P. Chartier. Formalisation of B in Isabelle/HOL. In D. Bert, editor, *B Conference*, volume 1393 of *LNCS*, pages 66–82, Montpellier (France), Apr. 1998. Springer. 31

[24] A. Ciabattoni, N. Galatos, and K. Terui. From Axioms to Analytic Rules in Nonclassical Logics. In *LICS*, pages 229–240. IEEE Computer Society, 2008. 116

[25] H. Cirstea and C. Kirchner. Using Rewriting and Strategies for Describing the B Predicate Prover. In *Strategies in Automated Deduction*, pages 25–36, Lindau (Germany), July 1998. 30, 85

[26] ClearSy. *Guide de rédaction de règles mathématiques.* 40

[27] ClearSy. *Atelier B 4.0*, Feb. 2009. `http://www.atelierb.eu/`. 37

[28] J. Coleman, C. Jones, I. Oliver, A. Romanovsky, and E. Troubitsyna. RODIN (Rigorous Open Development Environment for Complex Systems). In *EDCC-5, Budapest, Supplementary Volume*, pages 23–26, April 2005. 34

[29] S. Colin and G. Mariano. Coq, l'alpha et l'omega de la preuve pour B ? 2009. `http://hal.archives-ouvertes.fr/hal-00361302/en/`. 31

[30] L. M. de Moura and N. Bjorner. Z3 : An Efficient SMT Solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. 32

[31] D. Déharbe, P. Fontaine, Y. Guyot, and L. Voisin. SMT Solvers for Rodin. In Derrick et al. [34], pages 194–207. 34

[32] D. Delahaye. A Tactic Language for the System Coq. In M. Parigot and A. Voronkov, editors, *Logic for Programming and Automated Reasoning (LPAR)*, volume 1955 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2000. 19, 33, 81

[33] D. Delahaye and M. Jacquel. Recovering Intuition from Automated Formal Proofs using Tableaux with Superdeduction. *Electronic Journal of Mathematics & Technology (eJMT)*, 2013. À paraître. 21

[34] J. Derrick, J. A. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, and E. Riccobene, editors. *Abstract State Machines, Alloy, B, VDM, and Z - Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings*, volume 7316 of *Lecture Notes in Computer Science*. Springer, 2012. 178, 180

[35] N. Dershowitz and A. Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, volume 4790 of *Lecture Notes in Computer Science*. Springer, 2007. 176, 179

[36] G. Dowek, T. Hardin, and C. Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning (JAR)*, 31(1) :33–72, Sept. 2003. 115, 116

[37] P. Fontaine, J.-Y. Marion, S. Merz, L. P. Nieto, and A. F. Tiu. Expressiveness + Automation + Soundness : Towards Combining SMT Solvers and Interactive Proof Assistants. In H. Hermanns and J. Palsberg, editors, *TACAS*, volume 3920 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006. 34

[38] M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science*. Springer, 1979. 33

[39] K. J. J. Hintikka. Form and Content in Quantification Theory. In *IFIP TCS*, volume 8, pages 7–55. Acta Philosophia Fennica, 1955. 101

[40] C. Houtmann. *Représentation et interaction des preuves en superdéduction modulo.* PhD thesis, Université Henri Poincaré - Nancy I, Mar. 2010. 118

[41] M. Jacquel. Automatisation de la preuve B. Master's thesis, Université Pierre et Marie Curie, Sept. 2009. 82, 84

[42] M. Jacquel, K. Berkani, D. Delahaye, and C. Dubois. Verifying B Proof Rules using Deep Embedding and Automated Theorem Proving. In G. Barthe, A. Pardo, and G. Schneider, editors, *SEFM*, volume 7041 of *LNCS*, pages 253–268, Montevideo (Uruguay), Nov. 2011. Springer. 20

[43] M. Jacquel, K. Berkani, D. Delahaye, and C. Dubois. Tableaux Modulo Theories Using Superdeduction - An Application to the Verification of B Proof Rules with the Zenon Automated Theorem Prover. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 332–338. Springer, 2012. 21, 115

[44] M. Jacquel, K. Berkani, D. Delahaye, and C. Dubois. Verifying B Proof Rules using Deep Embedding and Automated Theorem Proving. *Software and Systems Modeling (SoSyM)*, 2013. À paraître. 20, 101

[45] É. Jaeger and C. Dubois. Why Would You Trust B? In Dershowitz and Voronkov [35], pages 288–302. 31, 90

[46] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008. 101

[47] É. Le Lay. Automatiser la validation des règles. Master's thesis, INSA (Rennes), Sept. 2008. 59, 64

[48] M. Leuschel and M. J. Butler. ProB : an Automated Analysis Toolset for the B Method. *STTT*, 10(2) :185–203, 2008. 31

[49] I. Maamria, M. Butler, A. Edmunds, and A. Rezazadeh. On an Extensible Rule-Based Prover for Event-B. In M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, editors, *ASM*, volume 5977 of *Lecture Notes in Computer Science*, page 407. Springer, 2010. 38, 47

[50] D. Mentré, C. Marché, J.-C. Filliâtre, and M. Asuka. Discharging Proof Obligations from Atelier B Using Multiple Automated Provers. In Derrick et al. [34], pages 238–251. 33

[51] S. Merz and H. Vanzetto. Automatic Verification of TLA + Proof Obligations with SMT Solvers. In N. Bjorner and A. Voronkov, editors, *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 289–303. Springer, 2012. 32, 34

[52] S. Negri and J. von Plato. Cut elimination in the presence of axioms. *Bulletin of Symbolic Logic*, 4(4) :418–435, 1998. 115

[53] D. Pastre. Strong and weak points of the MUSCADET theorem prover - examples from CASC-JC. *Journal of AI Communications*, 15(2-3) :147–160, 2002. 32

[54] D. Prawitz. Natural Deduction. A Proof-Theoretical Study. *Stockholm Studies in Philosophy*, 3, 1965. 116

[55] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *Journal of AI Communications*, 15(2-3) :91–110, 2002. 32, 101

[56] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2-3) :111–126, 2002. 100

[57] R. M. Smullyan. First-Order Logic . Berlin, 1968. Springer. 101

[58] STERIA. *Guide méthodologique de validation de la base de règles de l'Atelier B*, 1996. 46

[59] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure : The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4) :337–362, 2009. 141, 147, 149

[60] The Coq Development Team. *Coq, version 8.4*. Inria, Aug. 2012. `http://coq.inria.fr/`. 19, 31, 40, 49, 81

[61] B. Wack. *Typage et déduction dans le calcul de réécriture.* PhD thesis, Université Henri Poincaré - Nancy I, Oct. 2005. 117, 118

[62] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS Version 3.5. In R. A. Schmidt, editor, *CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009. 100

[63] W. Windsteiger. An Automated Prover for Zermelo-Fraenkel Set Theory in *Theorema*. *J. Symb. Comput.*, 41(3-4) :435–470, 2006. 32

Annexes

Annexe A

Règles du B-Book

A.1 Règles de preuve du B-Book

$$\frac{H \vdash P \qquad H \vdash Q}{H \vdash P \wedge Q}\ \text{REGLE 1} \qquad \frac{H \vdash P \wedge Q}{H \vdash P}\ \text{REGLE 2}$$

$$\frac{H \vdash P \wedge Q}{H \vdash Q}\ \text{REGLE 2'} \qquad \frac{H, P \vdash Q}{H \vdash P \Rightarrow Q}\ \text{REGLE 3}$$

$$\frac{H \vdash P \Rightarrow Q}{H, P \vdash Q}\ \text{REGLE 4} \qquad \frac{H, \neg Q \vdash P \qquad H, \neg Q \vdash \neg P}{H \vdash Q}\ \text{REGLE 5}$$

$$\frac{H, Q \vdash \neg P \qquad H, Q \vdash P}{H \vdash \neg Q}\ \text{REGLE 6} \qquad \frac{x \backslash E \qquad H \vdash P}{H \vdash \forall x.P}\ \text{REGLE 7}$$

$$\frac{H \vdash \forall x.P}{H \vdash [x := E]P}\ \text{REGLE 8} \qquad \frac{H \vdash E = F \qquad H \vdash [x := E]P}{H \vdash [x := F]P}\ \text{REGLE 9}$$

$$\frac{}{H \vdash E = E}\ \text{REGLE 10}$$

Les règles suivantes sont appelées règles dérivées dans le B-Book.

$$\frac{H \vdash P}{H \vdash \neg\neg P}\ \text{DR 1} \qquad \frac{H \vdash P \qquad H \vdash \neg Q}{H \vdash \neg(P \Rightarrow Q)}\ \text{DR 2}$$

$$\frac{H \vdash P \Rightarrow \neg Q}{H \vdash \neg(P \wedge Q)}\ \text{DR 3} \qquad \frac{H \vdash P \Rightarrow Q}{H \vdash (\neg\neg P) \Rightarrow Q}\ \text{DR 4}$$

$$\frac{H \vdash P \Rightarrow ((\neg Q) \Rightarrow R)}{H \vdash (\neg(P \Rightarrow Q)) \Rightarrow R}\ \text{DR 5} \qquad \frac{H \vdash \neg P \Rightarrow R \qquad H \vdash \neg Q \Rightarrow R}{H \vdash \neg(P \wedge Q) \Rightarrow R}\ \text{DR 6}$$

$$\frac{H \vdash \neg P \Rightarrow R \qquad H \vdash Q \Rightarrow R}{H \vdash (P \Rightarrow Q) \Rightarrow R}\ \text{DR 7} \qquad \frac{H \vdash P \Rightarrow (Q \Rightarrow R)}{H \vdash (P \wedge Q) \Rightarrow R}\ \text{DR 8}$$

$$\frac{x \backslash R \qquad x \backslash H \qquad H \vdash \neg P \Rightarrow R}{H \vdash \neg \forall x.P \Rightarrow R}\ \text{DR 9} \qquad \frac{H \vdash [x := E] \neg P}{H \vdash \neg \forall x.P}\ \text{DR 10}$$

$$\frac{\forall x.P \text{ est dans } H \qquad H \vdash [x := E]P \Rightarrow R}{H \vdash R}\ \text{DR 11}$$

$$\frac{x \backslash R \qquad x \backslash H \qquad H \vdash P \Rightarrow R}{H \vdash \exists x.P \Rightarrow R}\ \text{DR 12}$$

$$\frac{H \vdash [x := E]P}{H \vdash \exists x.P}\ \text{DR 13} \qquad \frac{x \backslash R \qquad x \backslash H \qquad H \vdash \forall x. \neg P \Rightarrow R}{H \vdash \neg \exists x.P \Rightarrow R}\ \text{DR 14}$$

$$\frac{H \vdash \forall x. \neg P}{H \vdash \neg \exists x.P}\ \text{DR 15} \qquad \frac{H \vdash \exists x. \neg P}{H \vdash \neg \forall x.P}\ \text{DR 16}$$

A.2 Règles de typage du B-Book

Pour chaque règle de typage Ti, il existe la règle symétrique Ti'.

$$\frac{H \vdash_\tau \mathsf{check}(P) \qquad H \vdash_\tau \mathsf{check}(Q)}{H \vdash_\tau \mathsf{check}(P \wedge Q)}\ \text{T1} \qquad \frac{H \vdash_\tau \mathsf{check}(P) \qquad H \vdash_\tau \mathsf{check}(Q)}{H \vdash_\tau \mathsf{check}(P \Rightarrow Q)}\ \text{T2}$$

$$\frac{H \vdash_\tau \mathsf{check}(P)}{H \vdash_\tau \mathsf{check}(\neg P)}\ \text{T3} \qquad \frac{x \setminus s \qquad x \setminus H \qquad H, x \in s \vdash_\tau \mathsf{check}(P)}{H \vdash_\tau \mathsf{check}(\forall x.(x \in s \Rightarrow P))}\ \text{T4}$$

$$\frac{H \vdash_\tau \mathsf{check}(\forall x.(x \in s \Rightarrow \forall y.(y \in t \Rightarrow P)))}{H \vdash_\tau \mathsf{check}(\forall (x,y).(x,y \in s \times t \Rightarrow P))}\ \text{T5} \qquad \frac{H \vdash_\tau \mathsf{check}(\forall x.(P \Rightarrow Q \wedge R))}{H \vdash_\tau \mathsf{check}(\forall x.(P \wedge Q \Rightarrow R))}\ \text{T6}$$

$$\frac{H \vdash_\tau \mathsf{type}(E) \equiv \mathsf{type}(F)}{H \vdash_\tau \mathsf{check}(E = F)}\ \text{T7} \qquad \frac{H \vdash_\tau \mathsf{type}(E) \equiv \mathsf{super}(s)}{H \vdash_\tau \mathsf{check}(E \in s)}\ \text{T8}$$

$$\frac{x \in s \text{ dans } H \qquad H \vdash_\tau \mathsf{super}(s) \equiv U}{H \vdash_\tau \mathsf{type}(x) \equiv U}\ \text{T9} \qquad \frac{H \vdash_\tau \mathsf{type}(E) \times \mathsf{type}(F) \equiv U}{H \vdash_\tau \mathsf{type}(E \mapsto F) \equiv U}\ \text{T10}$$

$$\frac{H \vdash_\tau \mathsf{super}(s) \equiv U}{H \vdash_\tau \mathsf{type}(\mathsf{choice}(s)) \equiv U}\ \text{T11} \qquad \frac{H \vdash_\tau \mathbb{P}(\mathsf{super}(s)) \equiv U}{H \vdash_\tau \mathsf{type}(s) \equiv U}\ \text{T12}$$

$$\frac{x \in s \text{ dans } H \qquad H \vdash_\tau \mathsf{super}(s) \equiv \mathbb{P}(U)}{H \vdash_\tau \mathsf{super}(x) \equiv U}\ \text{T13} \qquad \frac{H \vdash_\tau \mathsf{super}(s) \times \mathsf{super}(t) \equiv U}{H \vdash_\tau \mathsf{super}(s \times t) \equiv U}\ \text{T14}$$

$$\frac{H \vdash_\tau \mathbb{P}(\mathsf{super}(s)) \equiv U}{H \vdash_\tau \mathsf{super}(\mathbb{P}(s)) \equiv U}\ \text{T15}$$

$$\frac{H \vdash_\tau \mathsf{check}(\forall x.(x \in s \Rightarrow P)) \qquad H \vdash_\tau \mathsf{super}(s) \equiv U}{H \vdash_\tau \mathsf{super}(\{x | x \in s \wedge P\} \equiv U} \text{ T16}$$

$$\frac{\mathsf{given}(I) \text{ est dans } H \qquad H \vdash_\tau I \equiv U}{H \vdash_\tau \mathsf{super}(I) \equiv U} \text{ T17} \qquad \frac{H \vdash_\tau \mathsf{super}(s) \equiv \mathbb{P}(U)}{H \vdash_\tau \mathsf{super}(\mathsf{choice}(s)) \equiv \mathbb{P}(U)} \text{ T18}$$

$$\frac{H \vdash_\tau T \equiv U}{H \vdash_\tau \mathbb{P}(T) \equiv \mathbb{P}(U)} \text{ T19} \qquad \frac{H \vdash_\tau T \equiv U \qquad H \vdash_\tau V \equiv W}{H \vdash_\tau T \times V \equiv U \times W} \text{ T20}$$

$$\frac{\mathsf{given}(I) \in H}{H \vdash_\tau I \equiv I} \text{ T21} \qquad \frac{H \vdash_\tau \mathsf{super}(a) \equiv U \qquad H \vdash_\tau \mathsf{super}(b) \equiv U}{H \vdash_\tau \mathsf{super}(a - b) \equiv U} \text{ T24}$$

$$\frac{\mathsf{given}(I) \text{ est dans } H}{H \vdash_\tau \mathsf{super}(\emptyset) \equiv I} \text{ T27} \qquad \frac{H \vdash_\tau \mathsf{super}(r) \equiv U \times \mathsf{super}(\emptyset)}{H \vdash_\tau \mathsf{super}(\mathsf{dom}(r)) \equiv U} \text{ T29}$$

$$\frac{\mathsf{type}(E) \equiv \mathsf{super}(\mathsf{dom}(f)) \qquad \mathsf{super}(\mathsf{ran}(f)) \equiv \mathbb{P}(U)}{H \vdash_\tau \mathsf{super}(f(E)) \equiv U} \text{ T47}$$

Annexe B

Définitions des opérateurs B du B-Book

Pour simplifier la lecture de ces définitions, nous notons dans la suite un couple (a, b) au lieu de $a \mapsto b$.

Constructions dérivées

Soient u un ensemble, a une variable, E une expression appartenant à u, L une liste d'expressions appartenant à u et s et t deux ensembles inclus dans u.

$s \cup t \triangleq \{\ a \mid a \in u \wedge (a \in s \vee a \in t)\ \}$
$s \cap t \triangleq \{\ a \mid a \in u \wedge (a \in s \wedge a \in t)\ \}$
$s - t \triangleq \{\ a \mid a \in u \wedge (a \in s \wedge a \notin t)\ \}$
$\{\ E\ \} \triangleq \{\ a \mid a \in u \wedge a = E\ \}$
$\{\ L, E\ \} \triangleq \{\ L\ \} \cup \{\ E\ \}$
$\emptyset \triangleq \mathsf{BIG} - \mathsf{BIG}$

Relations binaires : première série

Soient u, v et w des ensembles, a, b, c des variables distinctes et p, q, s tel que $p \in \mathbb{P}(u \times v)$, $q \in \mathbb{P}(v \times w)$, $s \subseteq u$ et $t \subseteq v$.

$u \leftrightarrow v \triangleq \mathbb{P}(u \times v)$
$p^{-1} \triangleq \{\ (b, a) \mid (b, a) \in v \times u \wedge (a, b) \in p\ \}$
$\mathsf{dom}(p) \triangleq \{\ a \mid a \in u \wedge \exists b\ (b \in v \wedge (a, b) \in p)\ \}$
$\mathsf{ran}(p) \triangleq \mathsf{dom}(p^{-1})$
$p; q \triangleq \{\ (a, c) \mid (a, c) \in u \times w \wedge \exists b\ (b \in v \wedge (a, b) \in p \wedge (b, c) \in q)\ \}$
$q \circ p \triangleq p; q$
$\mathsf{id}(u) \triangleq \{\ (a, b) \mid (a, b) \in u \times u \wedge a = b\ \}$

$s \lhd p \triangleq \mathsf{id}(s); p$
$p \rhd t \triangleq p; \mathsf{id}(t)$
$s \lhd\!\!\!- p \triangleq (\mathsf{dom}(p) - s) \lhd p$
$p \rhd\!\!\!- t \triangleq p \rhd (\mathsf{ran}(p) - t)$

Deuxièmes définitions :
$\mathsf{ran}(p) \triangleq \{\ b \mid b \in v \wedge \exists a\ (a \in u \wedge (a,b) \in p)\}$
$s \lhd p \triangleq \{\ a,b \mid (a,b) \in p \wedge a \in s\}$
$p \rhd t \triangleq \{\ a,b \mid (a,b) \in p \wedge b \in t\}$
$s \lhd\!\!\!- p \triangleq \{\ a,b \mid (a,b) \in p \wedge a \notin s\}$
$p \rhd\!\!\!- t \triangleq \{\ a,b \mid (a,b) \in p \wedge b \notin t\}$

Relations binaires : seconde série

Soient s, t, u et v des ensemble, a, b, c des variables distinctes et p, w, q, f, g, h et k des ensembles tels que : $p \in s \leftrightarrow t$, $w \subseteq s$, $q \in s \leftrightarrow t$, $f \in s \leftrightarrow u$, $g \in s \leftrightarrow v$, $h \in s \leftrightarrow u$, $k \in t \leftrightarrow v$.
$p[w] \triangleq \mathsf{ran}(w \lhd p)$
$q \lhd\!\!+ p \triangleq (\mathsf{dom}(p) \lhd\!\!\!- q) \cup p$
$f \otimes g \triangleq \{\ (a,(b,c)) \mid (a,(b,c)) \in s \times (u \times v) \wedge (a,b) \in f \wedge (a,c) \in g\ \}$
$\mathsf{prj}_1(s,t) \triangleq (\mathsf{id}(s) \otimes (s \times t))^{-1}$
$\mathsf{prj}_2(s,t) \triangleq ((t \times s) \otimes \mathsf{id}(t))^{-1}$
$h \parallel k \triangleq (\mathsf{prj}_1(s,t); h) \otimes (\mathsf{prj}_2(s,t); k)$
Deuxièmes définitions :
$p[w] \triangleq \{\ b \mid b \in t \wedge \exists a.(a \in w \wedge (a,b) \in p)\}$
$q \lhd\!\!+ p \triangleq \{\ p,q \mid (p,q) \in s \times t \wedge (((a,b) \in q \wedge a \notin \mathsf{dom}(p)) \vee (a,b) \in p)\}$
$\mathsf{prj}_1(s,t) \triangleq \{\ a,b,c \mid (a,b,c) \in s \times t \times s \wedge c = a\}$
$\mathsf{prj}_2(s,t) \triangleq \{\ a,b,c \mid (a,b,c) \in s \times t \times t \wedge c = b\}$
$h \parallel k \triangleq \{\ (a,b),(c,d) \mid ((a,b),(c,d)) \in (s \times t) \times (u \times v) \wedge (a,c) \in h \wedge (b,d) \in k\ \}$

Fonction partielle

Soient s et t des ensembles et r et f des variables.

$$s \nrightarrow t \triangleq \{\ r \mid r \in s \leftrightarrow t \wedge (r^{-1}; r) \subseteq \mathsf{id}(t)\}$$

Deuxième définition :

$$s \nrightarrow t \triangleq \{\ r \mid t \in s \leftrightarrow t \wedge \forall (x,y,z)\ ((x,y) \in r \wedge (x,z) \in r \Rightarrow y = z)\ \}$$

le cnam

Mélanie JACQUEL

Automatisation des preuves pour la vérification des règles de l'Atelier B

le cnam

Résumé : Cette thèse porte sur la vérification des règles ajoutées de l'Atelier B en utilisant une plate-forme appelée BCARe qui repose sur un plongement de la théorie sous-jacente à la méthode B (théorie de B) dans l'assistant à la preuve Coq. En particulier, nous proposons trois approches pour prouver la validité d'une règle, ce qui revient à prouver une formule exprimée dans la théorie de B. Ces trois approches ont été évaluées sur les règles de la base de règles de Siemens IC-MOL. La première approche dite autarcique est développée avec le langage de tactiques de Coq $\mathcal{L}_{\text{tac}}$. Elle repose sur une première étape qui consiste à déplier tous les opérateurs ensemblistes pour obtenir une formule de la logique du premier ordre. Puis nous appliquons une procédure de décision qui met en œuvre une heuristique naïve en ce qui concerne les instanciations. La deuxième approche, dite sceptique, appelle le prouveur automatique de théorèmes Zenon après avoir effectué l'étape de normalisation précédente. Nous vérifions ensuite les preuves trouvées par Zenon dans le plongement profond de B en Coq. La troisième approche évite l'étape de normalisation précédente grâce à une extension de Zenon utilisant des règles d'inférence spécifiques à la théorie de B. Ces règles sont obtenues grâce à la technique de superdéduction. Cette dernière approche est généralisée en une extension de Zenon à toute théorie, appelée Super Zenon, grâce à un calcul dynamique des règles de superdéduction.

Mots clés :
Vérification, Déduction automatique, Superdéduction, Méthode B, Règles de l'Atelier B, Coq, Zenon

Abstract : The purpose of this thesis is the verification of Atelier B added rules using the framework named BCARe which relies on a deep embedding of the B theory within the logic of the Coq proof assistant. We propose especially three approaches in order to prove the validity of a rule, which amounts to prove a formula expressed in the B theory. These three approaches have been assessed on the rules coming from the rule database maintained by Siemens IC-MOL. To do so, the first approach, so-called autarkic approach, is developed thanks to the Coq tactic language, $\mathcal{L}_{\text{tac}}$. It rests upon a first step which consists in unfolding the set operators so as to obtain a first order formula. A decision procedure which implements an heuristic is applied afterwards to deal with instantiation. We propose a second approach, so-called skeptic approach, which uses the automated first order theorem prover Zenon, after the previous normalization step has been applied. Then we verify the Zenon proofs in the deep embedding of B in Coq. A third approach consists in using an extension of Zenon to the B method thanks to the superdeduction. Superdeduction allows us to add the axioms of the B theory by means of deduction rules in the proof mechanism of Zenon. This last approach is generalized in an extension of Zenon to every theory, named Super Zenon, thanks to a dynamic calculus of the superdeduction rules.

Keywords :
Verification, Automated Theorem Proving, Superdeduction, B method, Atelier B rules, Coq, Zenon