Automated Deduction and Proof Certification for the B Method

Pierre Halmagrand

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

le cnam

École doctorale Informatique, Télécommunications et Électronique (Paris)

Centre d'Études et de Recherche en Informatique et Communications

THÈSE DE DOCTORAT

présentée par : **Pierre HALMAGRAND**

soutenue le : **10 décembre 2016**

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

Spécialité : **Informatique**

Automated Deduction and Proof Certification for the B Method

THÈSE dirigée par

M. DELAHAYE David	*Professeur des Universités, Université de Montpellier*
M. DOLIGEZ Damien	*Chargé de Recherche, Inria*
M. HERMANT Olivier	*Chargé de Recherche, Mines ParisTech*

RAPPORTEURS

M. GRAHAM-LENGRAND Stéphane	*Chargé de Recherche, CNRS*
M. MÉRY Dominique	*Professeur des Universités, Université de Lorraine*

PRÉSIDENT

Mme. LALEAU Régine	*Professeur des Universités, Université Paris-Est Créteil*

EXAMINATEURS

Mme. DELEBARRE Véronique	*PDG, SafeRiver*
Mme. HARDIN Thérèse	*Professeur Émérite, Université Pierre et Marie Curie*

Abstract

The B Method is a formal method heavily used in the railway industry to specify and develop safety-critical software. It allows the development of correct-by-construction programs, thanks to a refinement process from an abstract specification to a deterministic implementation of the program. The soundness of the refinement steps depends on the validity of logical formulas called proof obligations, expressed in a specific typed set theory. Typical industrial projects using the B Method generate thousands of proof obligations, thereby relying on automated tools to discharge as many as possible proof obligations. A specific tool, called Atelier B, designed to implement the B Method and provided with a theorem prover, helps users verify the validity of proof obligations, automatically or interactively.

Improving the automated verification of proof obligations is a crucial task for the speed and ease of development. The solution developed in our work is to use Zenon, a first-order logic automated theorem prover based on the tableaux method. The particular feature of Zenon is to generate proof certificates, i.e. proof objects that can be verified by external tools. The B Method is based on first-order logic and a specific typed set theory. To improve automated theorem proving in this theory, we extend the proof-search algorithm of Zenon to polymorphism and deduction modulo theory, leading to a new tool called Zenon Modulo which is the main contribution of our work.

The extension to polymorphism allows us to deal with problems combining several sorts, like booleans and integers, and generic axioms, like B set theory axioms, without relying on encodings. Deduction modulo theory is an extension of first-order logic with rewriting both on terms and propositions. It is well suited for proof search in axiomatic theories, as it turns axioms into rewrite rules. This way, we turn proof search among

axioms into computations, avoiding unnecessary combinatorial explosion, and reducing the size of proofs by recording only their meaningful steps. To certify Zenon Modulo proofs, we choose to rely on Dedukti, a proof-checker used as a universal backend to verify proofs coming from different theorem provers, and based on deduction modulo theory.

This work is part of a larger project called BWare, which gathers academic entities and industrial companies around automated theorem proving for the B Method. These industrial partners provide to BWare a large benchmark of proof obligations coming from real industrial projects using the B Method and allowing us to test our tool Zenon Modulo. The experimental results obtained on this benchmark are particularly conclusive since Zenon Modulo proves more proof obligations than state-of-the-art first-order provers. In addition, all the proof certificates produced by Zenon Modulo on this benchmark are well checked by Dedukti, increasing our confidence in the soundness of our work.

Keywords : B Method, Set theory, Zenon Modulo, Automated deduction, Deduction modulo theory, Tableau method, Sequent calculus, Polymorphism, Dedukti, $\lambda\Pi$-calculus modulo theory, Proof certification.

Résumé

La Méthode B est une méthode formelle de spécification et de développement de logiciels critiques largement utilisée dans l'industrie ferroviaire. Elle permet le développement de programmes dit corrects par construction, grâce à une procédure de raffinements successifs d'une spécification abstraite jusqu'à une implantation déterministe du programme. La correction des étapes de raffinement est garantie par la vérification de la correction de formules mathématiques appelées obligations de preuve et exprimées dans la théorie des ensembles de la Méthode B. Les projets industriels utilisant la Méthode B génèrent généralement des milliers d'obligation de preuve. La faisabilité et la rapidité du développement dépendent donc fortement d'outils automatiques pour prouver ces formules mathématiques. Un outil logiciel, appelé Atelier B, spécialement développé pour aider au développement de projet avec la Méthode B, permet aux utilisateurs de décharger les obligations de preuve, automatiquement ou interactivement.

Améliorer la vérification automatique des obligations de preuve est donc une tâche importante. La solution que nous proposons est d'utiliser Zenon, un outils de déduction automatique pour la logique du premier ordre et qui met en œuvre la méthode des tableaux. La particularité de Zenon est de générer des certificats de preuve, des preuves écrites dans un certain format et qui peuvent être vérifiées automatiquement par un outil tiers. La théorie des ensembles de la Méthode B est une théorie des ensembles en logique du premier ordre qui fait appel à des schémas d'axiomes polymorphes. Pour améliorer la preuve automatique avec celle-ci, nous avons étendu l'algorithme de recherche de preuve de Zenon au polymorphisme et à la déduction modulo théorie. Ce nouvel outil, qui constitue le coeur de notre contribution, est appelé Zenon Modulo.

L'extension de Zenon au polymorphisme nous a permis de traiter, efficacement et sans

encodage, les problèmes utilisant en même temps plusieurs types, par exemple les booléens et les entiers, et des axiomes génériques, tels ceux de la théorie des ensembles de B. La déduction modulo théorie est une extension de la logique du premier ordre à la réécriture des termes et des propositions. Cette méthode est adaptée à la recherche de preuve dans les théories axiomatiques puisqu'elle permet de transformer des axiomes en règles de réécriture. Par ce moyen, nous passons d'une recherche de preuve dans des axiomes à du calcul, réduisant ainsi l'explosion combinatoire de la recherche de preuve en présence d'axiomes et compressant la taille des preuves en ne gardant que les étapes intéressantes. La certification des preuves de Zenon Modulo, une autre originalité de nos travaux, est faite à l'aide de Dedukti, un vérificateur universel de preuve qui permet de certifier les preuves provenant de nombreux outils différents, et basé sur la déduction modulo théorie.

Ce travail fait partie d'un projet plus large appelé BWare, qui réunit des organismes de recherche académiques et des industriels autour de la démonstration automatique d'obligations de preuve dans l'Atelier B. Les partenaires industriels ont fourni à BWare un ensemble d'obligation de preuve venant de vrais projets industriels utilisant la Méthode B, nous permettant ainsi de tester notre outil Zenon Modulo. Les résultats expérimentaux obtenus sur cet ensemble de référence sont particulièrement convaincants puisque Zenon Modulo prouve plus d'obligation de preuve que les outils de déduction automatique de référence au premier ordre. De plus, tous les certificats de preuve produits par Zenon Modulo ont été validés par Dedukti, nous permettant ainsi d'être très confiant dans la correction de notre travail.

Mots clés : Méthode B, Théorie des ensembles, Zenon Modulo, Déduction automatique, Déduction modulo théorie, Méthode des tableaux, Calcul des séquents, Polymorphisme, Dedukti, $\lambda\Pi$-calcul modulo théorie, Certification de preuve.

Acknowledgements

It has been an honor and a great pleasure to spend the three years of my PhD in the Inria Saclay team Deducteam.

My first thoughts go to my doctoral advisors, David Delahaye, Olivier Hermant and Damien Doligez. I discovered formal methods and proof theory thanks to David, four years ago as a student at Cnam. I would never have started this PhD without meeting you first David. So I will always be grateful to you for trusting me and helping me during this whole adventure. Thank you very much Olivier for your suitable advice, in particular during the last year and the writing of my manuscript, which I would never have finished without you. This PhD has been an existing journey with both of you, from South Africa to Fiji Islands and Connecticut ! Thank you very much Damien for your help on Zenon and your expert advice about OCaml. I was very lucky to have the three of you as my advisors !

I am very grateful to Stéphane Graham-Lengrand and Dominique Méry for accepting to review my thesis. Thank you very much for the time you spent on my manuscript and your encouraging remarks. I also want to thank Véronique Delebarre, Thérèse Hardin and Régine Laleau for having immediately agreed to be member of my thesis defense jury.

These three years would not have been so stimulating without my colleagues of Deducteam. I am very grateful to Gilles Dowek for welcoming me so friendly in his team. The good mood in Deducteam is really thanks to you. I also want to thank the other PhD candidates of Deducteam, Raphaël Cauderlier, Frédéric Gilbert, Guillaume Bury, Ronan

Saillard, Simon Cruanes, Ali Assaf, François Thiré and Kailiang Ji. We shared a lot of things together: small talks, debates, lunches, coffees, λ-pies, beers, ... and we even worked a little. Thank you very much guys ! I also want to thank Catherine Dubois, Guillaume Burel, Frédéric Blanqui, Simon Martiel and Thida Iem for all these moments we shared in Deducteam and all the help you gave to me.

I am very grateful to Stéphane Demri and all the members of the LSV laboratory of the ENS Paris-Saclay. Thank you very much for the friendly welcome and the good atmosphere in your lab. The free coffee policy helped me a lot !

My last thoughts go to my friends and my family. Thank you very much for your help and your patience during these three long years. Thank you very much my dear love Alexa, nothing would have been possible without you.

Contents

List of Tables

List of Figures

Introduction

The year 2016 will be remembered as a milestone in the rise of autonomous vehicles. While the web company Google has started Google Car, its project of autonomous cars, several years ago – the fleet of vehicles has already been tested on almost three million kilometers –, official disclosures of new projects of autonomous cars were released in the first half of 2016. Most of the worldwide automobile manufacturers released statements about the advent of self-driving cars within five years. For instance, the US automaker Ford announced in August a fully automated driverless car – without a steering wheel or pedals – for 2021 [Sage and Lienert 2016]. In the city of Pittsburgh, PA, the transportation network company Uber released in August a fleet of self-driving test-cars to transport clients, along with safety drivers for the moment [Chafkin 2016].

The rise of self-driving cars is surely a good news, and it will be a relief for a large number of persons, in particular those suffering from reduced mobility. Once released, this new means of transportation should quickly outpace the old-fashioned car. But all these positive aspects should not hide some legitimate concerns about safety, a self-driving car being clearly a life-critical system. An autonomous car relies on dozens of sensors, microchips and embedded software to operate it. The development of software for safety-critical system requires specific expertise and painstakingness, which seems to lack in the automotive industry.

The recent Toyota "unintended acceleration" affair, as reported by Bagnara in the 12th Workshop on Automotive Software & Systems in 2014 [Bagnara 2014], reveals questionable practices. In 2000, the car manufacturer Toyota adopted an Electronic Throttle Control System (ETCS for short) for most of its new car models, replacing a mechanically operated throttle pedal by an electronic one. In 2010, the National Highway

Traffic Safety Administration reported that 89 deaths may be linked to this affair, in addition to thousands of car accidents. In 2013, the first trial in which the plaintiffs alleged that the unintended acceleration was caused by a malfunction of the ETCS system, has used the testimony of Baar [Barr 2013], an embedded software expert, and Koopman [Koopman 2014], professor at Carnegie Mellon University, who were both allowed to investigate the source code of the ETCS embedded software. Their conclusions revealed that the software was, at least, very far from the expected standards for safety-critical systems. For instance, they reported that the development process did not follow strictly the MISRA-C guidelines – a discretionary standard developed by the Motor Industry Software Reliability Association. They also described the C source code as "spaghetti code", containing more than 10,000 read/write global variables. Finally, they pointed out the lack of certification requirements for software in safety-critical systems for US automakers.

In 2014, Toyota reached a $1.2 billion settlement with the US Department of Justice, ending a criminal investigation into the unintended acceleration affair. All the consequences of this affair are not yet fully documented sixteen years after the release of the ETCS system. But it can already be considered as a relevant case study and a turning point for functional safety in critical software systems.

An interesting conclusion of this story, which goes beyond the limited context of this case, is the lack of mandatory standards for US automotive industry when developing safety-critical software. Other transportation sectors, like aeronautic and railway industries, have successfully performed their electronic revolutions thirty years ago. For instance, the aircraft manufacturer Airbus released in 1984 the A320, the first airliner to fly with an all-digital fly-by-wire control system [Favre 1994]. In the railway industry, the first autonomous vehicles, a new fully automatic and driverless subway line in the city of Lille in France, appeared in 1983 [Lardennois 1993].

The high level of safety in these two sectors has been achieved with the application of specific mandatory standards for electronic devices and embedded software. The standard IEC 61508 is the generic international standard for electrical, electronic and programmable safety-related systems, published by the International Electrotechnical Commission. It

has been specified for each particular sector. For instance in the railway industry, the standard EN 50128 applies to safety-related software for railway control and protection systems. One of the important notions defined by this standard is the Safety Integrity Level (SIL for short), a quantity that measures the relative level of risk-reduction provided by a safety function. The standard defines four SIL levels, from SIL 1 (the lowest level of risk reduction) to SIL 4 (the highest level of risk reduction).

Safety functions of a system that requires a SIL 4 certification level are typically the most critical of the whole system, for instance the speed control system of a fully automatic driverless train. In software engineering, a large family of development methods, called formal methods, have been designed to develop highly trusted software. The general idea of formal methods is to prove that a program satisfies some particular mathematical properties. These mathematical properties translate the desired behaviour of a system, and are gathered into the specification, a formal model of the system. The notion of specification is a central concept in formal methods, because all these methods allow us to prove only relative correctness of a program with respect to a specification. Thus, specifications must be described in a formal language, typically a language without ambiguity, like logic-based languages, unlike natural languages. There exists a large number of different formal methods, covering all or part of the development cycle, from specification of a system to implementation.

The B Method is a formal method developed by Abrial and presented in its reference book, called the B-Book and published in 1996 [Abrial 1996]. The B Method is based on previous work of Hoare and Dijkstra about the correctness of programs. It is mainly used in the railway industry to specify and develop safety-critical software. For instance, it has been successfully used to develop the command control system of the fully automatic driverless trains of the subway line 14 in the city of Paris in France in 1998 [Behm, Benoit, Faivre, and Meynadier 1999]. The B Method covers all the development cycle of a program, from the formal specification of a system, called the abstract machine, to a deterministic implementation of the program. The resulting programs are said correct-by-construction, thanks to a refinement process from the abstract machine to the last and fully deterministic

B machine, called B0. The last stage of source code extraction consists in a mere syntactic translation of the B0 machine. The soundness of the refinement steps depends on the validity of logical formulæ, called proof obligations, expressed in the specific B set theory.

Common industrial projects using the B Method generate thousands of proof obligations, thereby relying on automated tools to discharge as many proof obligations as possible. A specific tool, called Atelier B [ClearSy 2013], designed to implement the B Method and provided with a theorem prover, helps users verify the validity of proof obligations, automatically or interactively. The automated theorem prover (ATP for short) of Atelier B proves around 85% of proof obligations in common industrial projects, letting thousands of proof obligations requiring a human interaction to be proved. This lack of automation in the B Method is a major cost factor for industrials, slowing its wide diffusion.

Our work aims to improve the automated verification of B proof obligations, with a particular focus on the soundness of the generated proofs. Our main contribution consists in the development of a first-order ATP called Zenon Modulo. This tool extends Zenon [Bonichon, Delahaye, and Doligez 2007], a first-order ATP based on the Tableau method. The Tableau method [D'Agostino, Gabbay, Hähnle, and Posegga 2013] is an automatic proof search algorithm for the sequent calculus without cuts. In proof theory, sequent calculus [Gentzen 1935] is a family of syntax-directed formal systems used to write formal proofs. It is defined by a set of inference rules, logical objects defining a syntactic relation between a set of formulæ called premises and another set of formulæ called conclusions, and corresponding to an elementary deduction step. These kinds of system are called proof systems. Tableau method proofs can be easily translated into sequent calculus proofs, as it is just a syntactic reformulation.

We do not need in our work to deal with all the B Method notions, in particular those related to the B language. We focus only on the mathematical reasoning of the B Method, consisting mainly in the B set theory. Improving the proof search algorithm of Zenon for the B Method leads to the development of two extensions, the former being an extension to first-order logic with polymorphic types, the latter being an extension to deduction modulo theory. The motivation of these two extensions is to deal efficiently with the B set theory.

The B Method set theory differs from other ones, like the Zermelo-Fraenkel set theory. The main difference consists in the addition of typing constraints to expressions, embedded into a set theoretic level, in the sense that there is no syntactical distinction between types and sets. To verify the well-typedness of expressions, the B-Book provides a set of typing inference rules, defining a type-checking procedure, which has to be applied once before proving. We show in Chap. 4 that B axioms and hypotheses can be seen as polymorphic formulæ, in the sense that they are defined for generic types. Once the proof obligation – which is not polymorphic – is fixed, the generic types of axioms and hypotheses have to be instantiated with types coming from the proof obligation.

The B set theory is made of six axioms, in addition to a large number of derived constructs. These derived constructs, like the union between sets, the domain of a relation and the set of total injective functions, are important in the B Method since they are well represented in proof obligations. Therefore, it is crucial to deal efficiently with these constructs. We choose to benefit from deduction modulo theory [Dowek, Hardin, and Kirchner 2003] to improve proof search in the B set theory. Deduction modulo theory is a formalism that extends first-order logic with rewrite rules on both terms and propositions, and improves proof search in axiomatic theories by turning axioms into rewrite rules. It allows us to distinguish deduction and computation steps, and to reason over equivalence classes of formulæ under a congruence generated by the rewrite system.

ATPs are generally large software, using sophisticated functionalities and complex optimizations. For instance, Zenon Modulo is made of more than 40,000 lines of OCaml code. Hazard growing up with size and complexity, potential causes of bugs and malfunctions exist in ATPs. When verifying proof obligations for the development of safety-critical software, guaranteeing the soundness of proofs is a very crucial task. Barendregt and Barendsen [Barendregt and Barendsen 2002] proposed to rely on the concept of proof certificate, a proof object that contains a statement and its formal proof, and that can be verified by an external tool. The originality of this approach is to separate the generation of the proof certificate, made by the ATP, and the verification of the soundness of the proof, made by an external proof checker. Ideally, the proof checker should be built on a light and auditable kernel. From this point of view, another important contribution

to Zenon Modulo is the development of a backend that generates proof certificates for the proof checker Dedukti [Assaf, Burel, Cauderlier, Delahaye, Dowek, Dubois, Gilbert, Halmagrand, Hermant, and Saillard 2016]. Dedukti is a lightweight implementation of the $\lambda\Pi$-calculus modulo theory, an extension of the simply typed λ-calculus with dependent types and rewriting. Dedukti is commonly used as a backend to verify proofs coming from ATPs, like Zenon Modulo, and also proof assistants, like Coq [Bertot and Castéran 2013].

Concerns may legitimately arise about the relevance of using Zenon Modulo, an ATP whose underlying logic is polymorphic first-order logic (Poly-FOL for short), to prove B proof obligations expressed in a specific set theory. We answer this issue in an original way by defining an encoding of B formulæ into Poly-FOL, where one of the particularities resides in the generation of type information in the resulting Poly-FOL expressions. This was made possible thanks to an externally defined type inference procedure for B bound variables. Then, we give a syntactic translation function of Zenon Modulo sequent calculus proofs into B natural deduction. Finally, we show that the resulting B proof is a proof of the original proof obligation. This closes the loop and gives us additional confidence in the correctness of our approach.

This work is part of the BWare project [Delahaye, Dubois, Marché, and Mentré 2014], an industrial research project supported by the "Agence Nationale de la Recherche" (French Research National Agency). BWare intends to provide a mechanized framework to help the automated verification of proof obligations coming from the development of industrial applications using the B Method. The BWare consortium gathers academic entities (Cedric, LRI and Inria) as well as industrial partners (Mitsubishi Electric R&D, ClearSy and OCamlPro). The methodology of the BWare project consists in building a generic platform of verification relying on different deduction tools, such as first-order ATPs, and Satisfiability Modulo Theory (SMT for short) solvers. This platform is built upon Why3 [Bobot, Filliâtre, Marché, and Paskevich 2011], a platform for deductive program verification. The deduction tools used in the BWare framework are the ATP Zenon Modulo, the ATP iProver Modulo [Burel 2011] and the SMT solver Alt-Ergo [Bobot, Conchon, Contejean, Iguernelala, Lescuyer, and Mebsout 2013]. The diversity of these theorem provers aims to allow a wide panel

proof obligations to be automatically verified by the platform. Beyond the multi-tool aspect of this methodology, the originality of BWare resides in the requirement for the deduction tools to produce proof certificates. To test the BWare platform, a large collection of proof obligations is provided by the industrial partners of the project, which develop tools implementing the B Method and applications involving the use of the B Method. This has allowed us to perform an experiment over this benchmark, where we have compared our tool Zenon Modulo with the other BWare tools and state-of-the-art ATPs.

This manuscript is organized as follows. In Chap. 1, we introduce the logic of the B Method. In particular, we present its proof system, set theory and type system. In Chap. 2, we present a type inference procedure for B bound variables. This procedure allows us to annotate variables with their types, an information required in the following. We also present a sound elimination procedure of sets defined by comprehension. In Chap. 3, we introduce polymorphic first-order logic, denoted Poly-FOL, and present LLproof, the typed sequent calculus used by Zenon to output proofs. In Chap. 4, we define an encoding of B formulæ into Poly-FOL, and show how to rebuild B proofs from Zenon proofs. In Chap. 5, we show the soundness of LLproof$^{\equiv}$, the extension of LLproof to deduction modulo theory, with respect to LLproof. In Chap. 6, we present Zenon and its extension to polymorphism and deduction modulo theory, resulting in our new tool Zenon Modulo. In Chap. 7, we introduce Dedukti and the $\lambda\Pi$-calculus modulo theory. Then, we present the encodings of Poly-FOL and LLproof$^{\equiv}$ into the $\lambda\Pi$-calculus modulo theory. Finally, in Chap. 8, we present the BWare project. Then, we give the rewrite system corresponding to the B set theory and used in BWare. We conclude this last chapter with the experimental results obtained over the BWare benchmark.

Chapter 1

The B Method

This chapter presents the logic of the B Method, *i.e.* its syntax, proof system, set theory and type system.

It is a faithful presentation of the core logic of the B Method as presented in the first two chapters of the B-Book [Abrial 1996], dealing with mathematical reasoning and set theory. It does not contain any new contribution.

1.1 Presentation

The B Method is a formal method that covers all the development process of programs, from the formal specification of a system to its actual implementation in a programming language. The formal specification, called an *abstract machine*, is described using the B language, a high level language that manipulates programs using the concept of *generalized substitutions*, a central notion of the B Method to describe the dynamic parts of B machines.

The evolution from a specification to an implementation is done step-by-step by a refinement process of B machines that removes indeterminism from the machines. At the end of the refinement process, the last B machine, called B0, uses fully deterministic algorithms and data structures that are close to programming language ones, allowing us to generate the source code of the program by a mere syntactic translation.

The consistency of abstract machines and the soundness of the refinement steps depend on the validity of mathematical formulæ called *proof obligations*. These formulæ are expressed in the framework of first-order logic with set theory. But, as we shall see later,

the set theory behind the B Method is rather specific, compared to common set theory like the Zermelo-Fraenkel set theory.

The work presented in this manuscript deals with the mathematical aspects of the B Method. In particular, we focus on the provability of proof obligations with respect to the B set theory, without any concern about the upstream concepts of B machines and generalized substitutions.

1.2 Logic

In this section, we present the syntax, the proof system, the set theory and the type system, which form the core logic of the B Method.

1.2.1 Syntax

We present in Fig. 1.1 the syntax of the B Method. It is made of four syntactic categories, *i.e.* formulæ, expressions, variables and sets.

A *formula* P is built from the logical connectives conjunction, implication and negation and the universal quantification. A formula may also be the result of a substitution in a formula, an equality between two expressions or membership to a set.

An *expression* E may be a variable, the result of a substitution in an expression, an ordered pair, an arbitrary element in a set or a set.

A *variable* x is either an identifier or a list of variables.

Finally, a *set* s is built using the elementary set constructs, *i.e.* the cartesian product, the powerset and the comprehension set, or may be the infinite set BIG (axiomatized below).

Remark In the B Method, common constructs like existential quantification, disjunction, equivalence and subset are defined as syntactic sugar. The B-Book gives the following rewrite rules as definitions:

$$\begin{array}{rcl@{\qquad}rcl} E \mapsto F & := & E, F & \exists x \cdot P & := & \neg \forall x \cdot \neg P \\ P \vee Q & := & \neg P \Rightarrow Q & P \Leftrightarrow Q & := & (P \Rightarrow Q) \wedge (Q \Rightarrow P) \\ s \subseteq t & := & s \in \mathbb{P}(t) & s \subset t & := & s \subseteq t \wedge s \neq t \end{array}$$

P	::=	$P_1 \wedge P_2$	(conjunction)
	|	$P_1 \Rightarrow P_2$	(implication)
	|	$\neg P$	(negation)
	|	$\forall x \cdot P$	(universal quantification)
	|	$[x := E]P$	(substitution)
	|	$E_1 = E_2$	(equality)
	|	$E \in s$	(membership)
E	::=	x	(variable)
	|	$[x := E_1]E_2$	(substitution)
	|	E_1, E_2	(ordered pair)
	|	$\mathsf{choice}(s)$	(choice function)
	|	s	(set)
x	::=	$identifier$	(variable identifier)
	|	x_1, x_2	(list of variables)
s	::=	$s_1 \times s_2$	(cartesian product)
	|	$\mathbb{P}(s)$	(powerset)
	|	$\{x \mid P\}$	(comprehension set)
	|	BIG	(infinite set)

Figure 1.1: The B Method Syntax

1.2.1.1 Non-Freeness

In the B Method, non-freeness provisos are often used. For instance, inference rules of the B proof system (see Fig. 1.3), axioms of the set theory (see Fig. 1.4) and inference rules of the type system (see Fig. 1.10) use non-freeness properties.

A variable x is said to have a *free occurrence* in a formula or in an expression if: (1) it is present in such a formula and (2) it is present in a sub-formula which is not under the scope of a quantifier ranging over x itself.

A variable x is said to be *non-free* in a formula or in an expression if: (1) it is not present in such a formula or (2) it is only present in sub-formulæ under the scope of some quantifier.

We present in Fig. 1.2 the rules that define the notion of non-freeness for all syntactic constructs of the B Method.

In the following, a *context*, denoted Γ, is a set of formulæ. If x is a variable and H a formula, $x\backslash H$ means that x is non-free in H. In addition, if Γ is a set of formulæ, $x\backslash\Gamma$ means $x\backslash H$ for each H of Γ; if Γ' is another set of formulæ, $\Gamma \sqsubset \Gamma'$ means that Γ is included in Γ'; and if P is a formula, $P \sqsubseteq \Gamma$ means that P occurs in Γ.

$$\frac{x, y \text{ distinct}}{x\backslash y}\ \mathsf{NF1} \qquad \frac{x\backslash P \quad x\backslash Q}{x\backslash(P \wedge Q)}\ \mathsf{NF2} \qquad \frac{x\backslash P \quad x\backslash Q}{x\backslash(P \Rightarrow Q)}\ \mathsf{NF3}$$

$$\frac{x\backslash P}{x\backslash\neg P}\ \mathsf{NF4} \qquad \frac{}{x\backslash\forall x \cdot P}\ \mathsf{NF5} \qquad \frac{x\backslash y \quad x\backslash P}{x\backslash\forall y \cdot P}\ \mathsf{NF6}$$

$$\frac{x\backslash E}{x\backslash[x := E]F}\ \mathsf{NF7} \qquad \frac{x\backslash y \quad x\backslash E \quad x\backslash F}{x\backslash[y := E]F}\ \mathsf{NF8} \qquad \frac{x\backslash E \quad x\backslash F}{x\backslash(E = F)}\ \mathsf{NF9}$$

$$\frac{x\backslash E \quad y\backslash E}{(x, y)\backslash E}\ \mathsf{NF10} \qquad \frac{x\backslash E \quad x\backslash F}{x\backslash(E, F)}\ \mathsf{NF11} \qquad \frac{x\backslash\forall y \cdot \forall z \cdot P}{x\backslash\forall(y, z) \cdot P}\ \mathsf{NF12}$$

$$\frac{x\backslash E \quad x\backslash s}{x\backslash(E \in s)}\ \mathsf{NF13} \qquad \frac{x\backslash s}{x\backslash\mathsf{choice}(s)}\ \mathsf{NF14} \qquad \frac{x\backslash s \quad x\backslash t}{x\backslash(s \times t)}\ \mathsf{NF15}$$

$$\frac{x\backslash s}{x\backslash\mathbb{P}(s)}\ \mathsf{NF16} \qquad \frac{x\backslash\forall y \cdot P}{x\backslash\{y \mid P\}}\ \mathsf{NF17} \qquad \frac{}{x\backslash\mathsf{BIG}}\ \mathsf{NF18}$$

Figure 1.2: Non-Freeness Rules for B Constructs

Remark The B-Book extends the notion of substitution to all expressions and formulæ by defining rewrite rules to normalize expressions. Since we have limited our field of work to the verification of proof obligations, we do not need to introduce them here. We can make the legitimate assumption that expressions are already normalized in the context of proof obligations.

1.2.2 Proof System

The proof system of the B Method is an adaptation of Natural Deduction [Gentzen 1935] with sequents to the syntax of the B Method. In addition to this proof system, the B-Book defines derived rules used in a decision procedure to prove formulæ. In the following, we focus only on the original proof system of the B Method, as presented in [Abrial 1996].

The rules are summarized in Fig. 1.3.

$$\frac{}{P \vdash_{\mathsf{B}} P}\ \mathsf{BR1} \qquad \frac{\Gamma \vdash_{\mathsf{B}} P \quad \Gamma \sqsubset \Gamma'}{\Gamma' \vdash_{\mathsf{B}} P}\ \mathsf{BR2}$$

$$\frac{P \sqsubseteq \Gamma}{\Gamma \vdash_{\mathsf{B}} P}\ \mathsf{BR3} \qquad \frac{\Gamma \vdash_{\mathsf{B}} P \quad \Gamma, P \vdash_{\mathsf{B}} Q}{\Gamma \vdash_{\mathsf{B}} Q}\ \mathsf{BR4}$$

$$\frac{\Gamma \vdash_{\mathsf{B}} P \quad \Gamma \vdash_{\mathsf{B}} P \Rightarrow Q}{\Gamma \vdash_{\mathsf{B}} Q}\ \mathsf{MP} \qquad \frac{\Gamma \vdash_{\mathsf{B}} P \quad \Gamma \vdash_{\mathsf{B}} Q}{\Gamma \vdash_{\mathsf{B}} P \wedge Q}\ \mathsf{R1}$$

$$\frac{\Gamma \vdash_{\mathsf{B}} P \wedge Q}{\Gamma \vdash_{\mathsf{B}} P}\ \mathsf{R2} \qquad \frac{\Gamma \vdash_{\mathsf{B}} P \wedge Q}{\Gamma \vdash_{\mathsf{B}} Q}\ \mathsf{R2'}$$

$$\frac{\Gamma, P \vdash_{\mathsf{B}} Q}{\Gamma \vdash_{\mathsf{B}} P \Rightarrow Q}\ \mathsf{R3} \qquad \frac{\Gamma \vdash_{\mathsf{B}} P \Rightarrow Q}{\Gamma, P \vdash_{\mathsf{B}} Q}\ \mathsf{R4}$$

$$\frac{\Gamma, \neg Q \vdash_{\mathsf{B}} P \quad \Gamma, \neg Q \vdash_{\mathsf{B}} \neg P}{\Gamma \vdash_{\mathsf{B}} Q}\ \mathsf{R5} \qquad \frac{\Gamma, Q \vdash_{\mathsf{B}} P \quad \Gamma, Q \vdash_{\mathsf{B}} \neg P}{\Gamma \vdash_{\mathsf{B}} \neg Q}\ \mathsf{R6}$$

$$\frac{x \backslash \Gamma \quad \Gamma \vdash_{\mathsf{B}} P}{\Gamma \vdash_{\mathsf{B}} \forall x \cdot P}\ \mathsf{R7} \qquad \frac{\Gamma \vdash_{\mathsf{B}} \forall x \cdot P}{\Gamma \vdash_{\mathsf{B}} [x := E]P}\ \mathsf{R8}$$

$$\frac{\Gamma \vdash_{\mathsf{B}} E = F \quad \Gamma \vdash_{\mathsf{B}} [x := E]P}{\Gamma \vdash_{\mathsf{B}} [x := F]P}\ \mathsf{R9} \qquad \frac{}{\Gamma \vdash_{\mathsf{B}} E = E}\ \mathsf{R10}$$

Figure 1.3: The Proof System of the B Method

1.2.3 B Set Theory

As presented in the B-Book, the B Method set theory is a simplification of standard set theory [Abrial 1996]. Some common axioms, like the foundation axiom, are not needed in this context (see Sec. 1.3), leading to a theory made only of six axioms. Actually, axioms presented below are axiom schemata that have to be instantiated with proper expressions. The first column represents non-freeness proviso.

It should be noted that the axiom SET3, which have an implicit quantification over a predicate symbol P, is not pure first-order logic. Such issues have already been studied, we can mention the theory of classes [Kirchner 2006] for instance. We explain in Sec. 2.3 how we deal with this axiom in our work.

	$E, F \in s \times t$	$\Leftrightarrow$	$(E \in s \wedge F \in t)$	SET1
$x \backslash (s,t)$	$s \in \mathbb{P}(t)$	$\Leftrightarrow$	$\forall x \cdot (x \in s \Rightarrow x \in t)$	SET2
$x \backslash s$	$E \in \{x \mid x \in s \wedge P\}$	$\Leftrightarrow$	$(E \in s \wedge [x := E]P)$	SET3
$x \backslash (s,t)$	$\forall x \cdot (x \in s \Leftrightarrow x \in t)$	$\Rightarrow$	$s = t$	SET4
$x \backslash s$	$\exists x \cdot (x \in s)$	$\Rightarrow$	$\mathsf{choice}(s) \in s$	SET5
			$\mathsf{infinite}(\mathsf{BIG})$	SET6

Figure 1.4: The B Set Theory

1.2.3.1 Example

As an example of a proof in the B set theory, we prove, given a set u, the property:

$$u \in \mathbb{P}(u)$$

We need the instance of the axiom SET2:

$$u \in \mathbb{P}(u) \Leftrightarrow \forall x \cdot (x \in u \Rightarrow x \in u)$$

which will be abbreviated by Ax in the following.

The resulting proof is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{Ax}, x \in u \vdash_{\mathsf{B}} x \in u}\,\mathsf{BR3}}{\mathrm{Ax} \vdash_{\mathsf{B}} x \in u \Rightarrow x \in u}\,\mathsf{R3}}{\mathrm{Ax} \vdash_{\mathsf{B}} \forall x \cdot (x \in u \Rightarrow x \in u)}\,\mathsf{R7} \qquad \dfrac{\dfrac{}{\mathrm{Ax} \vdash_{\mathsf{B}} \mathrm{Ax}}\,\mathsf{BR1}}{\mathrm{Ax} \vdash_{\mathsf{B}} \forall x \cdot (x \in u \Rightarrow x \in u) \Rightarrow u \in \mathbb{P}(u)}\,\mathsf{R2'}}{u \in \mathbb{P}(u) \Leftrightarrow \forall x \cdot (x \in u \Rightarrow x \in u) \vdash_{\mathsf{B}} u \in \mathbb{P}(u)}\,\mathsf{MP}$$

1.2.3.2 Derived Constructs, Binary Relations and Functions

The B Method relies on various usual set theory constructs, derived from the basic ones previously introduced. It should be noted that these new constructs are syntactic sugar and can always be replaced by their definitions. Thus, these definitions may be seen also as rewrite rules.

Basic Derived Constructs

First, we introduced in Fig. 1.5 the basic set theory constructs for the union, the intersection, the difference of two sets, then the empty set, sets defined by extension and finally the set of non-empty subset of a set.

In the following, s and t are two sets such that they are both subsets of the same set u.

The first four definitions use an external set called u. This set is used to guarantee that the set defined by comprehension is well typed. This notion is explained in Sec. 1.3.

The definition of the empty set uses the difference between BIG and itself, because it is the only set explicitly given so far. Using the extensionality axiom, we can show that the empty set can be defined using any set.

$$
\begin{array}{rcll}
s \cup t & := & \{a \mid a \in u \wedge (a \in s \vee a \in t)\} & \text{(union)} \\
s \cap t & := & \{a \mid a \in u \wedge (a \in s \wedge a \in t)\} & \text{(intersection)} \\
s - t & := & \{a \mid a \in u \wedge (a \in s \wedge a \notin t)\} & \text{(difference)} \\
\{E\} & := & \{a \mid a \in u \wedge a = E\} & \text{(singleton)} \\
\{L, E\} & := & \{L\} \cup \{E\} & \text{(extension)} \\
\varnothing & := & \mathsf{BIG} - \mathsf{BIG} & \text{(empty set)} \\
\mathbb{P}_1(s) & := & \mathbb{P}(s) - \{\varnothing\} & \text{(non-empty powerset)}
\end{array}
$$

Figure 1.5: Basic B Set Theory Derived Constructs

Binary Relation Constructs: First Series

In the B Method, binary relations are important to modelize data structures. We present in Fig. 1.6, the first series of constructs dealing with binary relations.

The first definition is the set of binary relations from one set s to another set t, denoted by $s \leftrightarrow t$. Since a relation is a set, all the previous constructs dealing with sets can be applied to relations. Then, we introduce the notions of the inverse of a relation, the domain and the range of a relation. Then, we present the composition and the backward composition of relations, the identity relation and various forms of relational restrictions of relations.

In the following, u, v and w are sets, a, and c are some distinct variables, and p, q, s and t are such that:

$$p \in u \leftrightarrow v \qquad q \in v \leftrightarrow w \qquad s \subseteq u \qquad t \subseteq v$$

$$
\begin{array}{rcll}
u \leftrightarrow v & := & \mathbb{P}(u \times v) & \text{(relation set)} \\
p^{-1} & := & \{b, a \mid (b, a) \in v \times u \wedge (a, b) \in p\} & \text{(inverse)} \\
\mathsf{dom}(p) & := & \{a \mid a \in u \wedge \exists b \cdot (b \in v \wedge (a, b) \in p)\} & \text{(domain)} \\
\mathsf{ran}(p) & := & \mathsf{dom}(p^{-1}) & \text{(range)} \\
p; q & := & \{a, c \mid (a, c) \in u \times w \wedge \exists b \cdot (b \in v \wedge (a, b) \in p \wedge (b, c) \in q)\} & \\
 & & & \text{(composition)} \\
q \circ p & := & p; q & \text{(backward composition)} \\
\mathsf{id}(u) & := & \{a, b \mid (a, b) \in u \times u \wedge a = b\} & \text{(identity)} \\
s \lhd p & := & \mathsf{id}(s); p & \text{(domain restriction)} \\
p \rhd t & := & p; \mathsf{id}(t) & \text{(range restriction)} \\
s \mathrel{\lhd\!\!\!-} p & := & (\mathsf{dom}(p) - s) \lhd p & \text{(domain subtraction)} \\
p \mathrel{-\!\!\!\rhd} t & := & p \rhd (\mathsf{ran}(p) - t) & \text{(range subtraction)}
\end{array}
$$

Figure 1.6: Binary Relation Constructs (Part 1)

Binary Relation Constructs: Second Series

We present in Fig. 1.7 the second series of constructs dealing with binary relations. It introduces the notion of image of a set under a relation, overriding of a relation, direct product of two relations, projections for ordered pairs, and finally parallel product of two relations.

In the following, s, t, u and v are sets, a, b and c are distinct variables, and p, w, q, f, g, h and k are such that:

$$
\begin{array}{llll}
p \in s \leftrightarrow t & w \subseteq s & q \in s \leftrightarrow t & f \in s \leftrightarrow u \\
g \in s \leftrightarrow v & h \in s \leftrightarrow u & k \in t \leftrightarrow u &
\end{array}
$$

$$
\begin{array}{rcll}
p[w] & := & \mathsf{ran}(w \lhd p) & \text{(image)} \\
q \mathrel{<\!\!\!+} p & := & (\mathsf{dom}(p) \mathrel{\lhd\!\!\!-} q) \cup p & \text{(overriding)} \\
f \otimes g & := & \{a, (b, c) \mid a, (b, c) \in s \times (u \times v) \wedge (a, b) \in f \wedge (a, c) \in g\} & \\
 & & & \text{(composition)} \\
\mathsf{prj}_1(s, t) & := & (\mathsf{id}(s) \otimes (s \times t))^{-1} & \text{(projection 1)} \\
\mathsf{prj}_2(s, t) & := & ((t \times s) \otimes \mathsf{id}(t))^{-1} & \text{(projection 2)} \\
h \mid\mid k & := & (\mathsf{prj}_1(s, t); h) \otimes (\mathsf{prj}_2(s, t); k) & \text{(parallel product)}
\end{array}
$$

Figure 1.7: Binary Relation Constructs (Part 2)

Functions

In the B Method, a function is a special case of relation where two different elements of the range cannot be related to the same element of the domain. We present in Fig. 1.8 the different sets of functions, where s and t are sets, and r and f are variables.

$$
\begin{array}{rcll}
s \pfun t & := & \{r \mid r \in s \rel t \land (r^{-1}; r) \subseteq \mathsf{id}(t)\} & \text{(partial function)} \\
s \fun t & := & \{f \mid f \in s \pfun t \land \mathsf{dom}(f) = s\} & \text{(total function)} \\
s \pinj t & := & \{f \mid f \in s \pfun t \land f^{-1} \in t \pfun s\} & \text{(partial injection)} \\
s \inj t & := & s \pinj t \cap s \fun t & \text{(total injection)} \\
s \psur t & := & \{f \mid f \in s \pfun t \land \mathsf{ran}(f) = t\} & \text{(partial surjection)} \\
s \surj t & := & s \psur t \cap s \fun t & \text{(total surjection)} \\
s \pbij t & := & s \pinj t \cap s \psur t & \text{(partial bijection)} \\
s \bij t & := & s \inj t \cap s \surj t & \text{(total bijection)}
\end{array}
$$

Figure 1.8: Sets of Functions

1.3 Type System

The B Method set theory differs from other ones, like the Zermelo-Fraenkel set theory. The main difference consists in the addition of typing constraints to expressions, and the application of a type-checking procedure before proving. This avoids ill-formed formulæ such as $\exists x \cdot (x \in x)$, whose negation is provable in Zermelo-Fraenkel set theory, thanks to the foundation axiom, unlike for the B Method.

1.3.1 A Hierarchy in Set Inclusion

The proposed typing discipline relies on the monotonicity of set inclusion. For instance, if we have an expression E and two sets s and t such that $E \in s$ and $s \subseteq t$, then $E \in t$. Going further with another set u such that $t \subseteq u$, we have then $E \in u$. The idea, as explained in the B-Book, is that, given a formula to be type checked, there exists an upper limit for such set containment. This upper limit is called the *superset* of s and the type of E. Then, if u is the superset of s, we obtain the typing information $E \in u$ and $s \in \mathbb{P}(u)$.

1.3.2 Type Checking Syntax

The type checking procedure presented below uses two syntactic categories $Type$ and $Type\_Pred$ as presented in Fig. 1.9. The former corresponds to the different kinds of types of expressions. The later may be seen as the type of propositions.

In the following, we use ty, su and ch as abbreviations for the keywords type, super and check respectively.

As stated in the B-Book, the type of an expression E is either an identifier (see the notion of given set below), the powerset of a type or the cartesian product of two types; and for the particular case of sets, the type of a set is necessarily the powerset of some type.

If E is an expression, s is a set and P a formula, $\mathsf{ty}(E)$ is the type of the expression E, $\mathsf{su}(s)$ is the superset of s – *i.e.* the largest set that contains s – and $\mathsf{ch}(P)$ verifies that P is a formula.

$Type$	::=	$\mathsf{type}(E)$	(type of expression)
	|	$\mathsf{super}(s)$	(superset of set)
	|	$Type \times Type$	(product type)
	|	$\mathbb{P}(Type)$	(powerset type)
	|	$identifier$	(given set)
$Type\_Pred$	::=	$\mathsf{check}(P)$	(type of a predicate)
	|	$Type \equiv Type$	(equality of type)

Figure 1.9: B Type-Checking Syntax

1.3.3 Type Checking

Type checking is performed by applying, in a backward way, the inference rules presented in Fig. 1.10. In addition, the B-Book requires to follow the numerical order of rules, in the sense that rules with a lower number have priority. This allows us to have a deterministic procedure. For rules T9 to T18, *i.e.* those with a particular typing expression on the left-hand side of the typing equivalence symbol $\equiv$ and with an arbitrary expression on the right-hand side, the B-Book defines the symmetric rule where the conclusion is inverted

with respect to symbol of equivalence. These rules are denoted with the same name primed and are not presented in Fig. 1.10.

For instance, the rule T9′ is defined as follows:

$$\frac{x \in s \sqsubseteq \Delta \qquad \Delta \vdash_{\mathsf{tc}} U \equiv \mathsf{su}(s)}{\Delta \vdash_{\mathsf{tc}} U \equiv \mathsf{ty}(x)}\ \mathsf{T9'}$$

If this decision procedure terminates and does not fail, then the formula is said to be well-typed.

The type system of Fig. 1.10 is divided in three categories of inference rules. The first set of inference rules, from T1 to T8′, allows us to decompose the logical connectives of formulæ. The second set of inference rules, from T9 to T18 and the primed versions, allows us to eliminate the typing constructors ty and su. Finally, the third set, made of the three last rules T19, T20 and T21, deals with the set theory constructs.

The B-Book does not give us some usual properties about the type system that we might expect, like the completeness and the unicity of typing – there exists a unique and valid typing derivation for all well-typed formulæ.

We do not need such strong properties in the context of our work, in particular completeness. But we still need to state the unicity of typing. Since the type inference procedure proposed in the next chapter occurs only after the verification of well-typedness, we can consider only formulæ that are well-typed and which have a valid typing derivation.

Proposition 1.3.1

Given a well-formed formula P, if the type checking decision procedure terminates well, then the corresponding typing derivation is unique.

Proof The ordering for rule application implies that the type checking decision procedure is deterministic, leading to the unicity of typing derivation.

1.3.4 Notion of Given Sets

A type-checking sequent like $\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(P)$ means that, within the environment Δ, the formula P is well-typed. The environment Δ is made of atomic formulæ of the form $x \in s$,

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(P) \qquad \Delta \vdash_{\mathsf{tc}} \mathsf{ch}(Q)}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(P \wedge Q)}\ \mathsf{T1} \qquad \frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(P) \qquad \Delta \vdash_{\mathsf{tc}} \mathsf{ch}(Q)}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(P \Rightarrow Q)}\ \mathsf{T2}$$

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(P)}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\neg P)}\ \mathsf{T3} \qquad \frac{x \backslash s \qquad x \backslash \Delta \qquad \Delta, x \in s \vdash_{\mathsf{tc}} \mathsf{ch}(P)}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall x \cdot (x \in s \Rightarrow P))}\ \mathsf{T4}$$

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall x \cdot (x \in s \Rightarrow \forall y \cdot (y \in t \Rightarrow P)))}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall (x, y) \cdot (x, y \in s \times t \Rightarrow P))}\ \mathsf{T5}$$

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall x \cdot (P \Rightarrow (Q \wedge R)))}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall x \cdot ((P \wedge Q) \Rightarrow R))}\ \mathsf{T6} \qquad \frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ty}(E) \equiv \mathsf{ty}(F)}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(E = F)}\ \mathsf{T7}$$

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ty}(E) \equiv \mathsf{su}(s)}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(E \in s)}\ \mathsf{T8} \qquad \frac{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv \mathsf{su}(t)}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(s \subseteq t)}\ \mathsf{T8'}$$

$$\frac{x \in s \sqsubset \Delta \qquad \Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathsf{ty}(x) \equiv U}\ \mathsf{T9} \qquad \frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ty}(E) \times \mathsf{ty}(F) \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathsf{ty}(E, F) \equiv U}\ \mathsf{T10}$$

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathsf{ty}(\mathsf{choice}(s)) \equiv U}\ \mathsf{T11} \qquad \frac{\Delta \vdash_{\mathsf{tc}} \mathbb{P}(\mathsf{su}(s)) \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathsf{ty}(s) \equiv U}\ \mathsf{T12}$$

$$\frac{x \in s \sqsubset \Delta \qquad \Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv \mathbb{P}(U)}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(x) \equiv U}\ \mathsf{T13} \qquad \frac{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \times \mathsf{su}(t) \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(s \times t) \equiv U}\ \mathsf{T14}$$

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathbb{P}(\mathsf{su}(s)) \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(\mathbb{P}(s)) \equiv U}\ \mathsf{T15} \qquad \frac{\mathsf{gi}(I) \sqsubset \Delta \qquad \Delta \vdash_{\mathsf{tc}} I \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(I) \equiv U}\ \mathsf{T17}$$

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall x \cdot (x \in s \Rightarrow P)) \qquad \Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(\{x \mid x \in s \wedge P\}) \equiv U}\ \mathsf{T16}$$

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv \mathbb{P}(U)}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(\mathsf{choice}(s)) \equiv U}\ \mathsf{T18} \qquad \frac{\Delta \vdash_{\mathsf{tc}} T \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathbb{P}(T) \equiv \mathbb{P}(U)}\ \mathsf{T19}$$

$$\frac{\Delta \vdash_{\mathsf{tc}} T \equiv U \qquad \Delta \vdash_{\mathsf{tc}} V \equiv W}{\Delta \vdash_{\mathsf{tc}} T \times V \equiv U \times W}\ \mathsf{T20} \qquad \frac{\mathsf{gi}(I) \sqsubset \Delta}{\Delta \vdash_{\mathsf{tc}} I \equiv I}\ \mathsf{T21}$$

Figure 1.10: The Type System of the B Method

where x is non-free in s. All free variables in P have to be associated with some atomic formula in Δ. The only exception is for variables in P representing some abstract *given sets*, introduced at a meta-level discourse like: "Given a set s ...". Such a given set s, which will be used to type other sets, is introduced in the environment Δ by the keyword $\mathsf{given}(s)$ ($\mathsf{gi}(s)$ for short), telling us that s is free in the formula to be type-checked, and has the specific property that $\mathsf{su}(s) = s$.

1.3.5 Example

Given two sets s and t, the formula P_{ex}:

$$P_{\text{ex}} \;:=\; \forall(a,b)\cdot(a,b \in \mathbb{P}(s\times t)\times\mathbb{P}(s\times t) \Rightarrow \{x \mid x\in a \wedge x \in b\} \subseteq s\times t)$$

will be used as a running example in the following.

We want to verify that this formula is well-typed, *i.e.* verify that the following sequent is derivable:

$$\mathsf{gi}(s), \mathsf{gi}(t) \vdash_{\mathsf{tc}} \mathsf{ch}(\forall(a,b)\cdot(a,b \in \mathbb{P}(s\times t)\times\mathbb{P}(s\times t) \Rightarrow \{x \mid x\in a \wedge x \in b\} \subseteq s\times t))$$

Using the following notations:

$$\begin{array}{lcl} \Delta & := & \mathsf{gi}(s), \mathsf{gi}(t), a \in \mathbb{P}(s\times t), b \in \mathbb{P}(s\times t) \\ \Delta' & := & \mathsf{gi}(s), \mathsf{gi}(t), a \in \mathbb{P}(s\times t), b \in \mathbb{P}(s\times t), x \in a \end{array}$$

And denoting Π_1 the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta \vdash_{\mathsf{tc}} s \equiv s}\,\mathsf{T21}}{\Delta \vdash_{\mathsf{tc}} s \equiv \mathsf{su}(s)}\,\mathsf{T17'} \quad \dfrac{\dfrac{}{\Delta \vdash_{\mathsf{tc}} t \equiv t}\,\mathsf{T21}}{\Delta \vdash_{\mathsf{tc}} t \equiv \mathsf{su}(t)}\,\mathsf{T17'}}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv \mathsf{su}(s)}\,\mathsf{T17} \quad \dfrac{}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(t) \equiv \mathsf{su}(t)}\,\mathsf{T17}}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(s)\times\mathsf{su}(t) \equiv \mathsf{su}(s)\times\mathsf{su}(t)}\,\mathsf{T20}}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(s\times t) \equiv \mathsf{su}(s)\times\mathsf{su}(t)}\,\mathsf{T14}}{\Delta \vdash_{\mathsf{tc}} \mathbb{P}(\mathsf{su}(s\times t)) \equiv \mathbb{P}(\mathsf{su}(s)\times\mathsf{su}(t))}\,\mathsf{T19}}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(\mathbb{P}(s\times t)) \equiv \mathbb{P}(\mathsf{su}(s)\times\mathsf{su}(t))}\,\mathsf{T15}}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(a) \equiv \mathsf{su}(s)\times\mathsf{su}(t)}\,\mathsf{T13}}{\Pi_1}$$

We obtain, by applying the rules of Fig. 1.10, the following typing tree:

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} s \equiv s}\ \mathsf{T21} \qquad \dfrac{}{\Delta' \vdash_{\mathsf{tc}} t \equiv t}\ \mathsf{T21}$$

$$\dfrac{\Delta' \vdash_{\mathsf{tc}} s \equiv s}{\Delta' \vdash_{\mathsf{tc}} s \equiv \mathsf{su}(s)}\ \mathsf{T17'} \qquad \dfrac{\Delta' \vdash_{\mathsf{tc}} t \equiv t}{\Delta' \vdash_{\mathsf{tc}} t \equiv \mathsf{su}(t)}\ \mathsf{T17'}$$

$$\dfrac{\Delta' \vdash_{\mathsf{tc}} s \equiv \mathsf{su}(s)}{\Delta' \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv \mathsf{su}(s)}\ \mathsf{T17} \qquad \dfrac{\Delta' \vdash_{\mathsf{tc}} t \equiv \mathsf{su}(t)}{\Delta' \vdash_{\mathsf{tc}} \mathsf{su}(t) \equiv \mathsf{su}(t)}\ \mathsf{T17}$$

$$\dfrac{\Delta' \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv \mathsf{su}(s) \qquad \Delta' \vdash_{\mathsf{tc}} \mathsf{su}(t) \equiv \mathsf{su}(t)}{\Delta' \vdash_{\mathsf{tc}} \mathsf{su}(s) \times \mathsf{su}(t) \equiv \mathsf{su}(s) \times \mathsf{su}(t)}\ \mathsf{T20}$$

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} \mathsf{su}(s) \times \mathsf{su}(t) \equiv \mathsf{su}(s \times t)}\ \mathsf{T14'}$$

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} \mathsf{su}(s \times t) \equiv \mathsf{su}(s \times t)}\ \mathsf{T14}$$

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} \mathbb{P}(\mathsf{su}(s \times t)) \equiv \mathbb{P}(\mathsf{su}(s \times t))}\ \mathsf{T19}$$

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} \mathbb{P}(\mathsf{su}(s \times t)) \equiv \mathsf{su}(\mathbb{P}(s \times t))}\ \mathsf{T15'}$$

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} \mathsf{su}(s \times t) \equiv \mathsf{su}(b)}\ \mathsf{T13'}$$

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} \mathbb{P}(\mathsf{su}(s \times t)) \equiv \mathbb{P}(\mathsf{su}(b))}\ \mathsf{T19}$$

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} \mathsf{su}(\mathbb{P}(s \times t)) \equiv \mathbb{P}(\mathsf{su}(b))}\ \mathsf{T15}$$

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} \mathsf{su}(a) \equiv \mathsf{su}(b)}\ \mathsf{T13}$$

$$\dfrac{}{\Delta' \vdash_{\mathsf{tc}} \mathsf{ty}(x) \equiv \mathsf{su}(b)}\ \mathsf{T9}$$

$$\dfrac{}{\Delta, x \in a \vdash_{\mathsf{tc}} \mathsf{ch}(x \in b)}\ \mathsf{T8}$$

$$\dfrac{}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall x \cdot (x \in a \Rightarrow x \in b))}\ \mathsf{T4}$$

$$\dfrac{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall x \cdot (x \in a \Rightarrow x \in b)) \qquad \Pi_1}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(\{x \mid x \in a \wedge x \in b\}) \equiv \mathsf{su}(s) \times \mathsf{su}(t)}\ \mathsf{T16}$$

$$\dfrac{}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(\{x \mid x \in a \wedge x \in b\}) \equiv \mathsf{su}(s \times t)}\ \mathsf{T14'}$$

$$\dfrac{}{\mathsf{gi}(s), \mathsf{gi}(t), a \in \mathbb{P}(s \times t), b \in \mathbb{P}(s \times t) \vdash_{\mathsf{tc}} \mathsf{ch}(\{x \mid x \in a \wedge x \in b\} \subseteq s \times t)}\ \mathsf{T8'}$$

$$\dfrac{}{\mathsf{gi}(s), \mathsf{gi}(t), a \in \mathbb{P}(s \times t) \vdash_{\mathsf{tc}} \mathsf{ch}(\forall b \cdot (b \in \mathbb{P}(s \times t) \Rightarrow \{x \mid x \in a \wedge x \in b\} \subseteq s \times t))}\ \mathsf{T4}$$

$$\dfrac{}{\mathsf{gi}(s), \mathsf{gi}(t) \vdash_{\mathsf{tc}} \mathsf{ch}(\forall a \cdot (a \in \mathbb{P}(s \times t) \Rightarrow \forall b \cdot (b \in \mathbb{P}(s \times t) \Rightarrow \{x \mid x \in a \wedge x \in b\} \subseteq s \times t)))}\ \mathsf{T4}$$

$$\dfrac{}{\mathsf{gi}(s), \mathsf{gi}(t) \vdash_{\mathsf{tc}} \mathsf{ch}(\forall (a, b) \cdot (a, b \in \mathbb{P}(s \times t) \times \mathbb{P}(s \times t) \Rightarrow \{x \mid x \in a \wedge x \in b\} \subseteq s \times t))}\ \mathsf{T5}$$

Chapter 2

Type Inference for B Variables

This chapter introduces some modifications of the B Method syntax that will be used in the following chapters. In particular, we present in Sec. 2.2 a type inference procedure for bound variables, and in Sec. 2.3 a skolemization procedure to eliminate sets defined by comprehension. These contributions are a personal work and they have been published in [Halmagrand 2016].

2.1 A Lack of Information

As can be seen in Sec. 1.3, the B Method is based on a typed set theory, in the sense that we have to verify that a formula is well-typed before proving it. To achieve this, the B Method provides a decision procedure made of a set of inference rules (see Fig. 1.10). Unfortunately, these rules do not provide the actual type of expressions, in particular variables. For instance, the rule T16:

$$\frac{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall x \cdot (x \in s \Rightarrow P)) \qquad \Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv U}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(\{x \mid x \in s \wedge P\}) \equiv U}\ \mathsf{T16}$$

decomposes comprehension sets and generates two branches, the former checking that a universally quantified formula is well-typed, the later verifying some typing constraints about a particular set coming from the comprehension set. As we can see, the type of the bound variable x is not given explicitly.

This is because the type system of Fig. 1.10 is a decision procedure that verifies the well-typedness of formulæ, unlike a type inference procedure that gives the type of

expressions.

For the rest of this manuscript, we need to carry more typing information about expressions in B formulæ. In particular, we need that bound variables carry their type for the embedding of B formulæ into first-order logic with polymorphic types presented in Sec. 3.2.

2.2 Type Annotation for B Variables

We present in the sequel of this section a type inference procedure for bound variables. This method is performed right after type-checking if this latter step succeeds.

2.2.1 Bound Variables

In the B syntax presented in Sec. 1.2.1, two constructs introduce new bound variables: universal quantification $\forall x \cdot P$ and comprehension set $\{x \mid P\}$. It should be noted that the typing rules T4 and T16 dealing with these two syntactical constructs use the specific forms $\forall x \cdot x \in s \Rightarrow P$ and $\{x \mid x \in s \wedge P\}$. Thus, in practice, all the comprehension sets and universally quantified formulæ have to be of this specific form to be type-checked. The reason is that they must mention explicitly the set that contains the bound variable introduced. The formula $x \in s$ is therefore used to type the bound variable x:

$$\frac{x \backslash s \qquad x \backslash \Delta \qquad \Delta, x \in s \vdash_{\mathsf{tc}} \mathsf{ch}(P)}{\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(\forall x \cdot (x \in s \Rightarrow P))}\ \mathsf{T4}$$

The rule T4 is the only one that adds new variables in typing contexts Δ – T16 relies on a further application of T4 to do so.

2.2.2 New Syntactic Category for Types

We define a new syntactic category T for types:

$$\begin{array}{rcll} T & ::= & identifier & \text{(given set)} \\ & | & T_1 \times T_2 & \text{(product type constructor)} \\ & | & \mathbb{P}(T) & \text{(powerset type constructor)} \end{array}$$

And we introduce the notation x^T meaning that the variable x has type T.

This new syntactic category is based on the type-checking syntax presented in Fig. 1.9. We just keep the three constructs that denote types, *i.e.* identifiers, product of types and powerset of type. The other cases are not needed. In particular, the two keywords type and super (which respectively compute the type of an expression and a set in the type-checking algorithm presented in Fig. 1.10) and the *Type_Pred* category are not type constructors.

2.2.3 Typing Contexts

The annotation procedure given below will use the environments Δ, also called typing contexts, to annotate variables. We need to go further than the B-Book, and formalize the structure of these typing contexts.

2.2.3.1 Syntax

Typing contexts Δ contain two kinds of declarations: given sets (see Sec. 1.3.4) and variables. They follow the syntax:

$$\begin{array}{rcll} \Delta & ::= & \varnothing & \text{(empty context)} \\ & | & \Delta, \mathsf{gi}(s) & \text{(given set)} \\ & | & \Delta, x \in s & \text{(bound variable)} \end{array}$$

In the formula $x \in s$, s is necessarily a set. It can therefore only be a composition of the two type constructors $\mathbb{P}$ and $\times$ applied to sets. In the following, we denote by the symbol k of arity n arbitrary compositions of the two type constructors $\mathbb{P}$ and $\times$. The formula $x \in s$ can be written in a more precise way:

$$x \in k(s_1, \ldots, s_n)$$

where $s_1, \ldots, s_n$ are all set variables or given sets already declared in Δ.

2.2.3.2 Well-Formedness

Typing contexts Δ are augmented only by the rule T4, thus they grow following an introduction order. Contexts are then ordered sets, thus they can be seen as lists.

We say that a typing context Δ is well-formed if it satisfies the following property. When a formula $x \in k(s_1, \ldots, s_n)$ is added, then it must not be already declared in Δ, and

all $s_1, \ldots, s_n$ have to be already declared in Δ – in particular because of rules T9 and T13 –, as a given set or in a formula like $s_i \in \mathbb{P}(k(t_1, \ldots, t_n))$ such that the context Δ at this time is well-formed.

2.2.3.3 Annotated Typing Contexts

The annotation procedure transforms all the leaf typing contexts Δ, *i.e.* the typing contexts of the leaves of a typing derivation that follows the rules of Fig. 1.10, into annotated typing contexts $\Delta^\star$, where all variables and given sets are annotated with their type. Then it uses these annotated typing contexts to rebuild the typing tree by applying the same typing derivation but in a forward way, allowing us to obtain the annotated initial formula at the end.

Here is the syntax of the annotated typing contexts $\Delta^\star$:

$$
\begin{array}{lll}
\Delta^\star ::= & \varnothing & \text{(empty context)} \\
\mid & \Delta^\star, \mathsf{gi}(s^{\mathbb{P}(s)}) & \text{(annotated given set)} \\
\mid & \Delta^\star, x^{k(T_1,\ldots,T_n)} \in k(s_1^{\mathbb{P}(T_1)}, \ldots, s_n^{\mathbb{P}(T_n)}) & \text{(annotated bound variable)}
\end{array}
$$

Remark It should be noted that the syntax presented above is purposely a restricted syntax. The reason is to annotate variables with their proper type, as we will show in Prop. 2.2.2. The type of a given set s is the powerset of itself – a direct consequence of the property of given sets, being equal to themselves – and the type of a variable $x \in k(s_1^{\mathbb{P}(T_1)}, \ldots, s_n^{\mathbb{P}(T_n)})$ is $k(T_1, \ldots, T_n)$. Also, the identifiers used in types $T_1, \ldots, T_n$ are only given sets.

2.2.4 Annotation Procedure

We can now introduce the annotation procedure:

1. For all the leaf typing contexts Δ:
   - 1.1. For all $\mathsf{gi}(s)$, we annotate s by its type $\mathbb{P}(s)$, and then substitute all occurrences of s in Δ by $s^{\mathbb{P}(s)}$;
   - 1.2. Following the introduction order in Δ, for all $x \in k(s_1^{\mathbb{P}(T_1)}, \ldots, s_n^{\mathbb{P}(T_n)})$, we annotate x with its type $k(T_1, \ldots, T_n)$, and we substitute all occurrences of x in Δ by $x^{k(T_1,\ldots,T_n)}$;

2. Rebuild the (annotated) initial formula by applying the type-checking tree in a forward way, *i.e.* from the leaves to the root.

We define inductively the relation which associates respectively to an expression E and a formula P, the expression $E^\star$ and formula $P^\star$ where all variables are annotated. We have, for instance $(P_1 \wedge P_2)^\star = P_1^\star \wedge P_2^\star$, $\mathbb{P}(s)^\star = \mathbb{P}(s^\star)$ and $x^\star = x^T$ where T is the type of x.

Proposition 2.2.1 (Conservativity of the Annotation Procedure)
The annotation procedure preserves well-typedness.
We have, for any two expressions A and B and any formula P:

1. *If $\Delta \vdash_{\mathsf{tc}} A \equiv B$, then $\Delta^\star \vdash_{\mathsf{tc}} A^\star \equiv B^\star$.*

2. *If $\Delta \vdash_{\mathsf{tc}} \mathsf{ch}(P)$, then $\Delta^\star \vdash_{\mathsf{tc}} \mathsf{ch}(P^\star)$.*

Proof
This property is correct because the annotation procedure does not change the structure of formulæ or the type of expressions. Thus, we can apply the same typing derivation on type-checking sequents where expressions have annotated variables.

Item 1. If $\Delta \vdash_{\mathsf{tc}} A \equiv B$, then there exists a typing derivation Π such that:

$$\frac{\Pi}{\Delta \vdash_{\mathsf{tc}} A \equiv B}$$

We prove by induction on the structure of Π that there exists a typing derivation $\Pi^\star$, which is exactly the same derivation than Π where all expressions have annotated variables.
The base case is when Π is an application of rule T21.

$$\Pi \;:=\; \frac{\mathsf{gi}(s) \sqsubseteq \Delta}{\Delta \vdash_{\mathsf{tc}} s \equiv s}\ \mathsf{T21}$$

Then, we obtain the following typing tree:

$$\Pi^\star \;:=\; \frac{\mathsf{gi}(s^{\mathbb{P}(s)}) \sqsubseteq \Delta^\star}{\Delta^\star \vdash_{\mathsf{tc}} s^{\mathbb{P}(s)} \equiv s^{\mathbb{P}(s)}}\ \mathsf{T21}$$

The generalization deals with rules T9 to T20. We present the case of rule T9.

$$\Pi \;:=\; \dfrac{x \in s \sqsubseteq \Delta \qquad \dfrac{\Pi'}{\Delta \vdash_{\mathsf{tc}} \mathsf{su}(s) \equiv U}}{\Delta \vdash_{\mathsf{tc}} \mathsf{ty}(x) \equiv U}\,\mathsf{T9}$$

Then, we obtain the following typing tree:

$$\Pi^\star \;:=\; \dfrac{x^T \in s^\star \sqsubseteq \Delta^\star \qquad \dfrac{\dfrac{\mathrm{IH}}{\Pi'^\star}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(s^\star) \equiv U^\star}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{ty}(x^T) \equiv U^\star}\,\mathsf{T9}$$

Item 2. The proof of the second property follows exactly the proof of item 1. The base cases are now rules T7 and T8, where premises use results of item 1. The generalization deals with rules T1 to T6. We do not present the proof which is straightforward.

Proposition 2.2.2 (Soundness of the Annotation Procedure)
The annotation is sound, in the sense that we annotate variables with their very type. We have, for a variable x:
If x^T is declared in $\Delta^\star$, then $\Delta^\star \vdash_{\mathsf{tc}} \mathsf{ty}(x^T) \equiv T^\star$.

Proof
We perform a proof by induction on the structure of Δ. We give in 1. the base case for given sets, then we present in 2. the particular case where the variable is a set typed using only given sets, and finally in 3. the general case for all kind of variables.

1. If x is a given set, we denote it by s, and it is then declared in $\Delta^\star$ with $\mathsf{gi}(s^{\mathbb{P}(s)})$. We obtain the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta^\star \vdash_{\mathsf{tc}} s^{\mathbb{P}(s)} \equiv s^{\mathbb{P}(s)}}\,\mathsf{T21}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(s^{\mathbb{P}(s)}) \equiv s^{\mathbb{P}(s)}}\,\mathsf{T17}}{\Delta^\star \vdash_{\mathsf{tc}} \mathbb{P}(\mathsf{su}(s^{\mathbb{P}(s)})) \equiv \mathbb{P}(s^{\mathbb{P}(s)})}\,\mathsf{T19}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{ty}(s^{\mathbb{P}(s)}) \equiv \mathbb{P}(s^{\mathbb{P}(s)})}\,\mathsf{T12}$$

2. If x is a variable which is a set and typed only by given sets, we have:

$$x^{\mathbb{P}(k(s_1,\ldots,s_n))} \in \mathbb{P}\big(k(s_1^{\mathbb{P}(s_1)},\ldots,s_n^{\mathbb{P}(s_n)})\big) \sqsubseteq \Delta^\star$$

   Then, we obtain the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta^\star \vdash_{\mathsf{tc}} s_1^{\mathbb{P}(s_1)} \equiv s_1^{\mathbb{P}(s_1)}}\,\mathsf{T21}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(s_1^{\mathbb{P}(s_1)}) \equiv s_1^{\mathbb{P}(s_1)}}\,\mathsf{T17} \quad \ldots \quad \dfrac{\dfrac{}{\Delta^\star \vdash_{\mathsf{tc}} s_n^{\mathbb{P}(s_n)} \equiv s_n^{\mathbb{P}(s_n)}}\,\mathsf{T21}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(s_n^{\mathbb{P}(s_n)}) \equiv s_n^{\mathbb{P}(s_n)}}\,\mathsf{T17}}{\vdots \qquad\qquad\qquad \vdots}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(k(s_1^{\mathbb{P}(s_1)}, \ldots, s_n^{\mathbb{P}(s_n)})) \equiv k(s_1^{\mathbb{P}(s_1)}, \ldots, s_n^{\mathbb{P}(s_n)})}}{\Delta^\star \vdash_{\mathsf{tc}} \mathbb{P}(\mathsf{su}(k(s_1^{\mathbb{P}(s_1)}, \ldots, s_n^{\mathbb{P}(s_n)}))) \equiv \mathbb{P}(k(s_1^{\mathbb{P}(s_1)}, \ldots, s_n^{\mathbb{P}(s_n)}))}\,\mathsf{T19}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(\mathbb{P}(k(s_1^{\mathbb{P}(s_1)}, \ldots, s_n^{\mathbb{P}(s_n)}))) \equiv \mathbb{P}(k(s_1^{\mathbb{P}(s_1)}, \ldots, s_n^{\mathbb{P}(s_n)}))}\,\mathsf{T15}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{ty}(x^{\mathbb{P}(k(s_1,\ldots,s_n))}) \equiv \mathbb{P}(k(s_1^{\mathbb{P}(s_1)}, \ldots, s_n^{\mathbb{P}(s_n)}))}\,\mathsf{T9}$$

where the decomposition of the sequent

$$\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(k(s_1^{\mathbb{P}(s_1)}, \ldots, s_n^{\mathbb{P}(s_n)})) \equiv k(s_1^{\mathbb{P}(s_1)}, \ldots, s_n^{\mathbb{P}(s_n)})$$

into the n sequents

$$\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(s_i^{\mathbb{P}(s_i)}) \equiv s_i^{\mathbb{P}(s_i)}$$

is performed by applying the chains T14-T20 for $\times$, and T15-T19 for $\mathbb{P}$.

3. Now, we make the induction hypothesis that all the variables declared in $\Delta^\star$ are well annotated, *i.e.* that we have, for all variables x_i:

$$\Delta^\star \vdash_{\mathsf{tc}} \mathsf{ty}(x_i^{T_i}) \equiv T_i^\star$$

In particular, if a variable x_i is a set, we denote it s_i and we have:

$$\Delta^\star \vdash_{\mathsf{tc}} \mathsf{ty}(s_i^{\mathbb{P}(T_i)}) \equiv \mathbb{P}(T_i^\star)$$

which reduces to:

$$\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(s_i^{\mathbb{P}(T_i)}) \equiv T_i^\star$$

because, by the deterministic nature of the type-checking procedure, we know that rules T12 and T19 have been applied.
Then, for the following well-formed context:

$$\Delta^\star, x^{k(T_1,\ldots,T_n)} \in k(s_1^{\mathbb{P}(T_1)}, \ldots, s_n^{\mathbb{P}(T_n)})$$

we obtain the derivation:

$$\dfrac{\dfrac{\dfrac{\text{IH}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(s_1^{\mathbb{P}(T_1)}) \equiv T_1^\star} \quad \ldots \quad \dfrac{\text{IH}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(s_n^{\mathbb{P}(T_n)}) \equiv T_n^\star}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{su}(k(s_1^{\mathbb{P}(T_1)}, \ldots, s_n^{\mathbb{P}(T_n)})) \equiv k(T_1^\star, \ldots, T_n^\star)}}{\Delta^\star \vdash_{\mathsf{tc}} \mathsf{ty}(x^{k(T_1,\ldots,T_n)}) \equiv k(T_1^\star, \ldots, T_n^\star)}\ \mathsf{T9}$$

Universal Closure

Once the annotation procedure is done, we take the universal closure of all free variables corresponding to given sets in axioms and hypotheses, and consider them as constants in goals.

In the following, the notation $P^\star$ denotes the universal closure of the annotated formula.

Provability

The B proof system of Fig. 1.3 is neutral with respect to variable annotation, and it is always possible to apply the same proof derivation to an annotated formula. The provability of well-typed formulæ is then preserved. We have:

$$\Gamma \vdash_{\mathsf{B}} P \quad \text{iff} \quad \Gamma^\star \vdash_{\mathsf{B}} P^\star$$

Example Going back to the running example of Sec. 1.3.5, we obtained the following environment Δ' for the leave of the left branch:

$$\Delta' := \mathsf{gi}(s), \mathsf{gi}(t), a \in \mathbb{P}(s \times t), b \in \mathbb{P}(s \times t), x \in a$$

It leads to the annotated environment $\Delta'^\star$, where only the first occurrence of a variable is annotated:

$$\Delta'^\star := \mathsf{gi}(s^{\mathbb{P}(s)}), \mathsf{gi}(t^{\mathbb{P}(t)}), a^{\mathbb{P}(s\times t)} \in \mathbb{P}(s \times t), b^{\mathbb{P}(s\times t)} \in \mathbb{P}(s \times t), x^{s\times t} \in a$$

Finally, we obtain the annotated formula $(P_{\text{ex}})^\star$:

$$\forall(a^{\mathbb{P}(s\times t)}, b^{\mathbb{P}(s\times t)}) \cdot (a, b \in \mathbb{P}(s^{\mathbb{P}(s)} \times t^{\mathbb{P}(t)}) \times \mathbb{P}(s \times t) \Rightarrow \{x^{s\times t} \mid x \in a \wedge x \in b\} \subseteq s \times t)$$

2.2.5 Annotated Set Theory

Axioms SET5 and SET6 are introduced in the B Method set theory for theoretical reasons, like defining function evaluation and building natural numbers, and are never used in practice, in particular in proof obligations. So, we remove them from our work.

We now define in Fig. 2.1 the annotated version of the axioms presented in Fig. 1.4. In addition, we take the universal closure for all free variables.

$$
\begin{array}{llll}
\forall s^{\mathbb{P}(u)} & \cdot (\forall t^{\mathbb{P}(v)} & \cdot (\forall x^u \cdot (\forall y^v \cdot (x, y \in s \times t \Leftrightarrow (x \in s \wedge y \in t))))) & \mathsf{SET1}^\star \\
\forall s^{\mathbb{P}(u)} & \cdot (\forall t^{\mathbb{P}(u)} & \cdot (s \in \mathbb{P}(t) \Leftrightarrow \forall x^u \cdot (x \in s \Rightarrow x \in t))) & \mathsf{SET2}^\star \\
\forall s^{\mathbb{P}(u)} & \cdot (\forall y^u & \cdot (y \in \{x^u \mid x \in s \wedge P\} \Leftrightarrow (y \in s \wedge [x := y]P))) & \mathsf{SET3}^\star \\
\forall s^{\mathbb{P}(u)} & \cdot (\forall t^{\mathbb{P}(u)} & \cdot (\forall x^u \cdot (x \in s \Leftrightarrow x \in t) \Rightarrow s = t)) & \mathsf{SET4}^\star
\end{array}
$$

Figure 2.1: The Annotated Axioms of the B Set Theory

2.3 Dealing with Comprehension Sets

Comprehension sets are very useful to define sets, in particular in the B Method, where many derived constructs use them. Unfortunately, comprehension sets cannot be directly embedded into first-order logic, due to the presence of a predicate symbol into it.

2.3.1 A Skolemization of Comprehension Sets

We propose an elimination procedure of comprehension sets inside formulæ, based on the definition of new function symbols. The idea to skolemize comprehension sets is not new, see for instance [Dowek and Miquel 2007; Jacquel 2013].

Given an annotated formula $P^\star$, if u is a subexpression of $P^\star$ of the form:

$$u = \{y^T \mid Q(y, s_1^{T_1}, \ldots, s_n^{T_n})\}$$

we apply the following procedure:

1. Generate a fresh function symbol $f^{\mathbb{P}(T)}$ of arity n and annotated by $\mathbb{P}(T)$;
2. Add to the B set theory, the axiom:

$$\forall {s'_1}^{T_1} \cdot (\ldots \cdot (\forall {s'_n}^{T_n} \cdot (\forall x^T \cdot (x \in f^{\mathbb{P}(T)}(s'_1, \ldots, s'_n) \Leftrightarrow Q(x, s'_1, \ldots, s'_n)))))$$

3. Replace all the occurrences of u by $f^{\mathbb{P}(T)}(s_1, \ldots, s_n)$.

The resulting skolemized formula is then denoted $P^{\star s}$.

Lemma 2.3.1

Given a well-typed and annotated formula $P^\star$ containing a subexpression $u = \{y^T \mid Q(y, s_1^{T_1}, \ldots, s_n^{T_n})\}$, if we denote f the generated fresh function symbol, we can prove, in B:

$$f^{\mathbb{P}(T)}(s_1^{T_1}, \ldots, s_n^{T_n}) = \{y^T \mid Q(y, s_1^{T_1}, \ldots, s_n^{T_n})\}$$

Proof

In the following, we denote $s_1, \ldots, s_n$ the n sets $s_1^{T_1}, \ldots, s_n^{T_n}$.

Axiom SET3* tells us that:

$$\forall x^T \cdot (x \in \{y^T \mid Q(y, s_1, \ldots, s_n)\} \Leftrightarrow [y := x]Q(y, s_1, \ldots, s_n))$$

We have:

$$\forall x^T \cdot (x \in \{y^T \mid Q(y, s_1, \ldots, s_n)\} \Leftrightarrow Q(x, s_1, \ldots, s_n))$$

In addition, we know that:

$$\forall x^T \cdot (x \in f^{\mathbb{P}(T)}(s_1, \ldots, s_n) \Leftrightarrow Q(x, s_1, \ldots, s_n))$$

We deduce that:

$$\forall x^T \cdot (x \in f^{\mathbb{P}(T)}(s_1, \ldots, s_n) \Leftrightarrow x \in \{y^T \mid Q(y, s_1, \ldots, s_n)\})$$

Using axiom SET4*, we obtain:

$$f^{\mathbb{P}(T)}(s_1, \ldots, s_n) = \{y^T \mid Q(y, s_1, \ldots, s_n)\}$$

Proposition 2.3.2 (Soundness)

The skolemization procedure of comprehension sets is sound.

Given a well-typed and annotated formula $P^\star$ and a set of well-typed and annotated formulæ $\Gamma^\star$ containing the axiom SET3*, we have:

$$\Gamma^{\star s} \vdash_{\mathsf{B}} P^{\star s} \quad \Rightarrow \quad \Gamma^\star \vdash_{\mathsf{B}} P^\star$$

where $\Gamma^{\star s}$ is the union of $\Gamma^\star$ and all the axioms added by the skolemization procedure.

Proof

Let $f_1, \ldots, f_n$ be the n function symbols defined by the skolemization procedure.

If we denote Π^s the proof:

$$\frac{\Pi^s}{\Gamma^{\star s} \vdash_{\mathsf{B}} P^{\star s}}$$

we want to replace all applications of functions symbols in the proof Π^s by the corresponding comprehension sets. This is done using rewriting.

First, we define the rewrite system:

$$\begin{array}{rcl} f_1(x_1^1, \ldots, x_{m_1}^1) & \longrightarrow & \{y \mid Q_1(y, x_1^1, \ldots, x_{m_1}^1)\} \\ & \vdots & \\ f_n(x_1^n, \ldots, x_{m_n}^n) & \longrightarrow & \{y \mid Q_n(y, x_1^n, \ldots, x_{m_n}^n)\} \end{array}$$

This rewrite system is confluent because of the absence of critical pair – fresh head symbol and trivial pattern with only variables – and it is terminating for the strategy that applies rewrite rules following the reverse numerical order, from f_n to f_1 – since $Q_1, \ldots, Q_k$ do not contain $f_k, \ldots, f_n$.

We prove by induction over the structure of Π^s that there exists a proof Π such that:

$$\frac{\Pi}{\Gamma^{\star} \vdash_{\mathsf{B}} P^{\star}}$$

where we have replaced all the applications of function symbols f_i by the corresponding comprehension sets. The induction is straightforward because the B proof system presented in Fig. 1.3 is stable with respect to the confluent and terminating rewrite system defined above.

Remark Unfortunately, the skolemization of comprehension sets is not complete: it is no more possible to define a new set by comprehension during proof search if we drop axiom SET3*, only to deal with the ones at hand (the Skolem symbols).

Example Applying skolemization to the running example leads to add the following axiom to the theory:

$$\forall a^{\mathbb{P}(s\times t)} \cdot (\forall b^{\mathbb{P}(s\times t)} \cdot (\forall x^{s\times t} \cdot (x \in f^{\mathbb{P}(s\times t)}(a,b) \Leftrightarrow x \in a \wedge x \in b)))$$

And we obtain the skolemized formula $(P_{\text{ex}})^{\star s}$:

$$\forall (a^{\mathbb{P}(s\times t)}, b^{\mathbb{P}(s\times t)}) \cdot (a, b \in \mathbb{P}(s^{\mathbb{P}(s)} \times t^{\mathbb{P}(t)}) \times \mathbb{P}(s \times t) \Rightarrow f^{\mathbb{P}(s\times t)}(a,b) \subseteq s \times t)$$

Remark The skolemization of comprehension sets presented here is applied to annotated formulæ, as for the elimination of skolem symbol in the proof above. The extension of both the skolemization and its elimination to non-annotated formulæ is straightforward.

2.4 Updated Syntax and Proof System

To conclude this chapter, we present the new version of the B syntax in Fig. 2.2, with annotated variables, function symbols and without comprehension sets, choice function or BIG. In addition, we suppose that expressions are normalized in the sense that substitutions are reduced, as it is for proof obligations. We also merge the two categories for expressions and sets in a single category called E.

To lighten the B proofs in the next chapters, we introduce the new symbols $\bot$ and $\top$ defined as follows:

$$\begin{array}{rcl} \bot & := & P \wedge \neg P \\ \top & := & \neg \bot \end{array}$$

where P is a fixed closed formula.

Finally, we enrich the B proof system of Fig. 1.3 with the two derived basic rules BR5 and BR6 dealing with $\bot$ and $\top$:

$$\frac{}{\Gamma, \bot \vdash_{\mathsf{B}} Q}\ \mathsf{BR5} \ := \ \dfrac{\dfrac{\dfrac{}{\Gamma, P \wedge \neg P, \neg Q \vdash_{\mathsf{B}} P \wedge \neg P}\ \mathsf{BR3}}{\Gamma, P \wedge \neg P, \neg Q \vdash_{\mathsf{B}} P}\ \mathsf{R2} \qquad \dfrac{\dfrac{}{\Gamma, P \wedge \neg P, \neg Q \vdash_{\mathsf{B}} P \wedge \neg P}\ \mathsf{BR3}}{\Gamma, P \wedge \neg P, \neg Q \vdash_{\mathsf{B}} \neg P}\ \mathsf{R2'}}{\Gamma, P \wedge \neg P \vdash_{\mathsf{B}} Q}\ \mathsf{R5}$$

$$\frac{}{\Gamma \vdash_{\mathsf{B}} \top}\ \mathsf{BR6} \ := \ \frac{\dfrac{}{\Gamma, \bot \vdash_{\mathsf{B}} Q}\ \mathsf{BR5} \qquad \dfrac{}{\Gamma, \bot \vdash_{\mathsf{B}} \neg Q}\ \mathsf{BR5}}{\Gamma \vdash_{\mathsf{B}} \neg \bot}\ \mathsf{R6}$$

T	::=	$identifier$	(type identifier)
	|	$T_1 \times T_2$	(product type)
	|	$\mathbb{P}(T)$	(powerset type)
P	::=	$\bot$	(false)
	|	$\top$	(true)
	|	$P_1 \wedge P_2$	(conjunction)
	|	$P_1 \Rightarrow P_2$	(implication)
	|	$\neg P$	(negation)
	|	$\forall x^T \cdot P$	(universal quantification)
	|	$E_1 = E_2$	(equality)
	|	$E_1 \in E_2$	(membership)
E	::=	x^T	(variable)
	|	E_1, E_2	(ordered pair)
	|	$E_1 \times E_2$	(product set)
	|	$\mathbb{P}(E)$	(powerset)
	|	$f^{\mathbb{P}(T)}(E_1, \ldots, E_n)$	(function symbol application)
x	::=	$identifier$	(variable identifier)
	|	x_1, x_2	(list of variables)

Figure 2.2: Modified B Method Syntax

Chapter 3

Polymorphically Typed Sequent Calculus

This chapter presents a typed sequent calculus called LLproof and polymorphically typed first-order logic, denoted by Poly-FOL, and used by LLproof.

In Sec. 3.2, we introduce the syntax and the type system of Poly-FOL. This section is a mere adaptation of [Blanchette, Böhme, Popescu, and Smallbone 2013; Blanchette and Paskevich 2013].

In Sec. 3.3, we present the typed sequent calculus LLproof. This proof system is an extension to Poly-FOL of the initial LLproof proof system of the automated theorem prover Zenon, as presented in [Bonichon, Delahaye, and Doligez 2007]. This contribution is a collaborative work and it has been published in [Cauderlier and Halmagrand 2015].

3.1 First-Order Logic and Types

Reasoning with several theories together may sometimes be necessary. For instance, in the B Method, proof obligations often combine set theory, booleans and arithmetic. Then, some theory-specific axioms can be defined; for instance, a legitimate axiom about booleans could be:

$$\forall x.\ x = \mathsf{true} \vee x = \mathsf{false}$$

Instantiating this axiom with a term that is not of type bool leads to an unsound formula. To prevent this issue, a solution is to provide type information to terms. For

instance, we can change the previous axiom by the following one:

$$\forall x : \mathsf{bool}.\ x = \mathsf{true} \vee x = \mathsf{false}$$

where the notation $x : \mathsf{bool}$ means that x is of type bool.

This kind of logic is usually called a *monomorphic* and *many-sorted* logic. In this logic, we deal with a finite set of sorts, like bool or int, and all terms are typed.

In set theory, we usually want to define some generic axioms, in the sense that we can instantiate them with different types. For instance, we could define membership to the union of two sets as follows:

$$\forall s : \mathsf{set}(\alpha).\ \forall t : \mathsf{set}(\alpha).\ \forall x : \alpha.\ x \in s \cup t \Leftrightarrow x \in s \vee x \in t$$

where α is a type variable, set a type constructor, and $s : \mathsf{set}(\alpha)$ means that s is a set of objects of type α. We call this logic a *polymorphic* logic.

In the following, we denote polymorphic first-order logic by Poly-FOL, monomorphic/many-sorted first-order logic by Sorted-FOL, and untyped first-order logic by FOL (also called monomorphic/mono-sorted first-order logic sometimes).

3.2 Poly-FOL: Polymorphic First-Order Logic

In this section, we present Poly-FOL. This presentation is inspired by [Blanchette, Böhme, Popescu, and Smallbone 2013; Blanchette and Paskevich 2013].

3.2.1 Syntax

Poly-FOL Signature

We start by fixing a countably infinite set $\mathcal{A}$ of *type variables*, usually denoted by α, and a countably infinite set $\mathcal{V}$ of *term variables* , usually denoted by x.

We call a *Poly-FOL signature* a triple $\Sigma = (\mathcal{T}, \mathcal{F}, \mathcal{P})$, where:

- $\mathcal{T}$ is a countable set of type constructors T with their arity m, denoted by $T :: m$
- $\mathcal{F}$ is a countable set of function symbols f with their type signature σ, denoted by $f : \sigma$

- $\mathcal{P}$ is a countable set of predicate symbol P with their type signature σ, denoted by $P : \sigma$

Type

The set $Type_\Sigma$ is the set of types τ in signature Σ, built inductively with type variables α from $\mathcal{A}$ and type constructors T from $\mathcal{T}$. We define Type, which is meant to be the type of the elements of $Type_\Sigma$. We suppose that all types in $Type_\Sigma$ are inhabited. By convention, nullary type constructors are called *type constants*, or *sorts*. We say that a type is *polymorphic* if it contains type variables; and *monomorphic* or *ground* otherwise.

$$\begin{array}{rcll} \tau & ::= & \alpha & \text{(type variable)} \\ & | & T(\tau_1, \ldots, \tau_m) & \text{(type constructor application)} \end{array}$$

Type Signature

Type signatures σ of function and predicate symbols are type schemes, using type quantification Π over a list of type variables $\alpha_1 \ldots \alpha_m$ – sometimes denoted by $\vec{\alpha}$ when m is known from the context – and a list of types $\tau_1 \times \ldots \times \tau_n$ – sometimes denoted by $\vec{\tau}$ –, and returning a type τ for function symbol and the *pseudo-type* omicron, denoted by o, for predicate symbols. Omicron is a pseudo-type because we do not want to instantiate type variables with it. In the following, we do not allow the overloading of different arities for function and predicate symbols. In addition, we may sometimes omit type arguments $\vec{\alpha}$ to function and predicate symbols when it is clear from context.

$$\begin{array}{rcll} \sigma & ::= & \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to \tau & \text{(function type signature)} \\ & | & \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to o & \text{(predicate type signature)} \end{array}$$

Term

A *term* t can be a (term) variable x from $\mathcal{V}$ or the application of a function symbol f from $\mathcal{F}$ to m type arguments and n term arguments at once. To help reading formulæ, we use a semicolon to separate type and term arguments in function and predicate symbols.

$$\begin{array}{rcll} t & ::= & x & \text{(variable)} \\ & | & f(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n) & \text{(function application)} \end{array}$$

Formula

A *formula* φ can be built from $\top$ and $\bot$, an equality between two terms t_1 and t_2 having

the same type τ, or the application of a predicate symbol P to m type arguments and n term arguments. It can also be built inductively by the logical connectives for negation $\neg$, conjunction $\wedge$, disjunction $\vee$, implication $\Rightarrow$ and equivalence $\Leftrightarrow$. Finally, a formula can be built using universal quantification $\forall$ and existential quantification $\exists$ over term variables. By convention and without loss of generality, we assume that (type and term) variables are bound only once in a formula.

$$
\begin{array}{rcll}
\varphi & ::= & \top \mid \bot & \text{(true, false)} \\
& \mid & t_1 =_\tau t_2 & \text{(term equality)} \\
& \mid & P(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n) & \text{(predicate application)} \\
& \mid & \neg\varphi & \text{(negation)} \\
& \mid & \varphi_1 \wedge \varphi_2 & \text{(conjunction)} \\
& \mid & \varphi_1 \vee \varphi_2 & \text{(disjunction)} \\
& \mid & \varphi_1 \Rightarrow \varphi_2 & \text{(implication)} \\
& \mid & \varphi_1 \Leftrightarrow \varphi_2 & \text{(equivalence)} \\
& \mid & \exists x : \tau.\ \varphi & \text{(existential quantification)} \\
& \mid & \forall x : \tau.\ \varphi & \text{(universal quantification)}
\end{array}
$$

Type-Quantified Formula

A *type-quantified formula* φ_T is built using universal quantification over type variables.

$$
\begin{array}{rcll}
\varphi_T & ::= & \varphi & \text{(formula)} \\
& \mid & \forall\alpha.\ \varphi_T & \text{(type quantification)}
\end{array}
$$

It should be noted that this inductive presentation guarantees that quantification over type variables is always universal and at the top of the formula. Thus, we say that type quantification is *prenex*. In the following, we may sometimes call formula both formulæ and type-quantified formulæ, when it is clear from the context. In addition, we call expression, denoted by e, both terms and formulæ.

Local Context

We also define the notion of *local context* Γ_{L}, which is a set of pairs made of a type variable α and its type Type, denoted by $\alpha : \mathsf{Type}$, or a (term) variable x and its type τ, denoted by $x : \tau$.

$$
\begin{array}{rcll}
\Gamma_{\mathrm{L}} & ::= & \varnothing & \text{(empty context)} \\
& \mid & \Gamma_{\mathrm{L}}, \alpha : \mathsf{Type} & \text{(type variable declaration)} \\
& \mid & \Gamma_{\mathrm{L}}, x : \tau & \text{(term variable declaration)}
\end{array}
$$

Global Context

Finally, a *global context* Γ_{G} is a set containing the declarations of type constructors T with their arity m and function and predicate symbols f and P with their type signatures σ.

$$\begin{array}{rcll} \Gamma_{\mathrm{G}} & ::= & \varnothing & \text{(empty context)} \\ & | & \Gamma_{\mathrm{G}}, T :: m & \text{(type constructor declaration)} \\ & | & \Gamma_{\mathrm{G}}, f : \sigma & \text{(function declaration)} \\ & | & \Gamma_{\mathrm{G}}, P : \sigma & \text{(predicate declaration)} \end{array}$$

Remark In the following, we sometimes denote by Γ the pair of contexts $\Gamma_{\mathrm{G}}; \Gamma_{\mathrm{L}}$. In addition, if f is either a function symbol or a predicate symbol, $\Gamma, f : \sigma$ denotes $\Gamma_{\mathrm{G}}, f : \sigma; \Gamma_{\mathrm{L}}$. Finally, $\Gamma, \alpha : \mathsf{Type}$ and $\Gamma, x : \tau$ denote $\Gamma_{\mathrm{G}}; \Gamma_{\mathrm{L}}, \alpha : \mathsf{Type}$ and $\Gamma_{\mathrm{G}}; \Gamma_{\mathrm{L}}, x : \tau$ respectively.

In addition, if x is either a type or a term variable, we denote by $x \in \Gamma_{\mathrm{L}}$ (respectively $x \notin \Gamma_{\mathrm{L}}$) the fact that the variable x is declared in Γ_{L} (respectively not declared). This notation is extended to global contexts Γ_{G} with type constructors T, function symbols f and predicate symbols P.

Example To illustrate this presentation of Poly-FOL, we introduce a global context Γ_{G} made of a nullary type constructor T, a unary type constructor set, a function symbol for powerset $\mathbb{P}$, a predicate symbol for membership $\in$ and a constant u. In addition, we define a polymorphic formula φ_{ax} which is an axiom defining membership to the powerset and a monomorphic formula φ_{gl} which can be seen as a goal.

$$\Gamma_{\mathrm{G}} := \left\{ \begin{array}{lcl} T & :: & 0 \\ \mathsf{set} & :: & 1 \\ \mathbb{P} & : & \Pi\alpha.\ \mathsf{set}(\alpha) \to \mathsf{set}(\mathsf{set}(\alpha)) \\ \in & : & \Pi\alpha.\ \alpha \times \mathsf{set}(\alpha) \to o \\ u & : & \mathsf{set}(T) \end{array} \right.$$

$$\begin{array}{rcl} \varphi_{ax} & := & \forall\alpha.\ \forall s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha).\ \in(\mathsf{set}(\alpha); s, \mathbb{P}(\alpha; t)) \Leftrightarrow (\forall x : \alpha.\ \in(\alpha; x, s) \Rightarrow \in(\alpha; x, t)) \\ \varphi_{gl} & := & \in(\mathsf{set}(T); u, \mathbb{P}(T; u)) \end{array}$$

Later in this manuscript, we will often use an infix notation with subscript type parameters when dealing with standard symbols, like the predicate symbol $\in$ or the function symbol $\mathbb{P}$.

The presentation of the previous example shall then be:

$$\Gamma_{\mathrm{G}} := \begin{cases} T & :: 0 \\ \mathsf{set} & :: 1 \\ \mathbb{P}(\text{-}) & : \Pi\alpha.\ \mathsf{set}(\alpha) \to \mathsf{set}(\mathsf{set}(\alpha)) \\ \text{-} \in \text{-} & : \Pi\alpha.\ \alpha \times \mathsf{set}(\alpha) \to o \\ u & : \mathsf{set}(T) \end{cases}$$

$$\begin{aligned} \varphi_{ax} &:= \forall\alpha.\ \forall s,t : \mathsf{set}(\alpha).\ s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) \Leftrightarrow (\forall x : \alpha.\ x \in_\alpha s \Rightarrow x \in_\alpha t) \\ \varphi_{gl} &:= u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \end{aligned}$$

3.2.2 Typing System

Free Variable

We denote by $\mathrm{FV_T}(e)$ the set of type variables occurring freely in an expression e, either in type arguments of polymorphic symbols or in the types of variables. We denote by $\mathrm{FV}(e)$ the set of term variables occurring freely in an expression e.

Monomorphic/Polymorphic Formula

A formula φ is said to be *monomorphic* if it is not a type-quantified formula and if $\mathrm{FV_T}(\varphi)$ is empty. Otherwise, the formula is said to be *polymorphic*. A formula φ is said to be *closed* if both $\mathrm{FV_T}(\varphi)$ and $\mathrm{FV}(\varphi)$ are empty.

Type Substitution

We call *type substitution* a mapping $\rho := [\alpha_1/\tau_1, \ldots, \alpha_m/\tau_m]$ that associates type variables $\alpha_1, \ldots, \alpha_m$ with types $\tau_1, \ldots, \tau_m$.

We define the predicate symbol $\mathsf{wf}(\Gamma_{\mathrm{G}}; \Gamma_{\mathrm{L}})$ meaning that the context $\Gamma := \Gamma_{\mathrm{G}}; \Gamma_{\mathrm{L}}$ is well-formed.

A *typing judgment* $\Gamma \vdash t : \tau$ means that the term t is well-typed of type τ in the well-formed context Γ.

A typing judgment $\Gamma \vdash \varphi : o$ means that the formula φ is well-typed in the well-formed context Γ.

We present in Fig. 3.1 the inference rules for well-formedness of contexts, and in Fig. 3.2 the inference rules of the type system of Poly-FOL.

$$\frac{}{\mathsf{wf}(\varnothing;\varnothing)}\ \mathrm{WF}_1 \qquad \frac{x \notin \Gamma_\mathrm{L} \quad \Gamma_\mathrm{G};\Gamma_\mathrm{L} \vdash \tau : \mathsf{Type}}{\mathsf{wf}(\Gamma_\mathrm{G};\Gamma_\mathrm{L}, x:\tau)}\ \mathrm{WF}_2$$

$$\frac{\alpha \notin \Gamma_\mathrm{L} \quad \mathsf{wf}(\Gamma_\mathrm{G};\Gamma_\mathrm{L})}{\mathsf{wf}(\Gamma_\mathrm{G};\Gamma_\mathrm{L}, \alpha : \mathsf{Type})}\ \mathrm{WF}_3 \qquad \frac{T \notin \Gamma_\mathrm{G} \quad \mathsf{wf}(\Gamma_\mathrm{G};\varnothing)}{\mathsf{wf}(\Gamma_\mathrm{G}, T :: m;\varnothing)}\ \mathrm{WF}_4$$

$$\frac{f \notin \Gamma_\mathrm{G} \qquad \begin{array}{c}\Gamma_\mathrm{G};\alpha_1 : \mathsf{Type},\ldots,\alpha_m : \mathsf{Type} \vdash \tau_i : \mathsf{Type}, i = 1\ldots n \\ \Gamma_\mathrm{G};\alpha_1 : \mathsf{Type},\ldots,\alpha_m : \mathsf{Type} \vdash \tau : \mathsf{Type}\end{array}}{\mathsf{wf}(\Gamma_\mathrm{G}, f : \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to \tau;\varnothing)}\ \mathrm{WF}_5$$

$$\frac{P \notin \Gamma_\mathrm{G} \quad \Gamma_\mathrm{G};\alpha_1 : \mathsf{Type},\ldots,\alpha_m : \mathsf{Type} \vdash \tau_i : \mathsf{Type}, i = 1\ldots n}{\mathsf{wf}(\Gamma_\mathrm{G}, P : \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to o;\varnothing)}\ \mathrm{WF}_6$$

Figure 3.1: Context Well-Formedness in Poly-FOL

Remark It should be noted that we explicitly impose in rules of Fig. 3.1 that variable symbols are declared only once in local contexts, and type constructor, function and predicate symbols are also declared only once in global contexts.

In addition, in rules of Fig. 3.2 dealing with term variables like Var or $\forall$, we do not impose to verify that τ is a type in $x : \tau$ since this is guaranteed by rule WF_2.

Lemma 3.2.1 (Unicity of Typing)
Given a well-formed context Γ, for all typable terms t, there exists only one type τ such that:

$$\Gamma \vdash t : \tau$$

Proof By induction on the typing relation.
If t is a variable, it is true thanks to rules Var and WF_2.
If t is the application of a function symbol $f(\tau'_1,\ldots,\tau'_m;t_1,\ldots,t_n)$, by the rule Fun, $t_1,\ldots,t_n$ are well typed, and by induction hypothesis, each $t_1,\ldots,t_n$ has a unique type. If we have $f : \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to \tau'$, then, $\tau = \tau'\rho$ where $\rho = [\alpha_1/\tau'_1,\ldots,\alpha_m/\tau'_m]$, thus τ is unique.

Example We want to verify that the formula φ_{gl} introduced in the previous example is well-typed.

$$\frac{\alpha : \mathsf{Type} \in \Gamma}{\Gamma \vdash \alpha : \mathsf{Type}}\,\mathrm{TVar} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\,\mathrm{Var}$$

$$\frac{T :: m \in \Gamma \qquad \Gamma \vdash \tau_i : \mathsf{Type}, i = 1 \ldots m}{\Gamma \vdash T(\tau_1, \ldots, \tau_m) : \mathsf{Type}}\,\mathrm{TConstr}$$

$$\frac{\begin{array}{ll} f : \Pi\alpha_1 \ldots \alpha_m.\tau_1 \times \ldots \times \tau_n \to \tau \in \Gamma & \Gamma \vdash \tau'_i : \mathsf{Type}, i = 1 \ldots m \\ \rho = [\alpha_1/\tau'_1, \ldots, \alpha_m/\tau'_m] & \Gamma \vdash t_i : \tau_i\rho, i = 1 \ldots n \end{array}}{\Gamma \vdash f(\tau'_1, \ldots, \tau'_m; t_1, \ldots, t_n) : \tau\rho}\,\mathrm{Fun}$$

$$\frac{\begin{array}{ll} P : \Pi\alpha_1 \ldots \alpha_m.\tau_1 \times \ldots \times \tau_n \to o \in \Gamma & \Gamma \vdash \tau'_i : \mathsf{Type}, i = 1 \ldots m \\ \rho = [\alpha_1/\tau'_1, \ldots, \alpha_m/\tau'_m] & \Gamma \vdash t_i : \tau_i\rho, i = 1 \ldots n \end{array}}{\Gamma \vdash P(\tau'_1, \ldots, \tau'_m; t_1, \ldots, t_n) : o}\,\mathrm{Pred}$$

$$\frac{}{\Gamma \vdash \top : o}\,\top \qquad \frac{}{\Gamma \vdash \bot : o}\,\bot$$

$$\frac{\Gamma \vdash \varphi_1 : o \qquad \Gamma \vdash \varphi_2 : o}{\Gamma \vdash \varphi_1 \wedge \varphi_2 : o}\,\wedge \qquad \frac{\Gamma \vdash \varphi_1 : o \qquad \Gamma \vdash \varphi_2 : o}{\Gamma \vdash \varphi_1 \vee \varphi_2 : o}\,\vee$$

$$\frac{\Gamma \vdash \varphi_1 : o \qquad \Gamma \vdash \varphi_2 : o}{\Gamma \vdash \varphi_1 \Rightarrow \varphi_2 : o}\,\Rightarrow \qquad \frac{\Gamma \vdash \varphi_1 : o \qquad \Gamma \vdash \varphi_2 : o}{\Gamma \vdash \varphi_1 \Leftrightarrow \varphi_2 : o}\,\Leftrightarrow$$

$$\frac{\Gamma \vdash \varphi : o}{\Gamma \vdash \neg\varphi : o}\,\neg \qquad \frac{\Gamma \vdash t_1 : \tau \qquad \Gamma \vdash t_2 : \tau}{\Gamma \vdash t_1 =_\tau t_2 : o}\,=$$

$$\frac{\Gamma, x : \tau \vdash \varphi : o}{\Gamma \vdash \exists x : \tau.\ \varphi : o}\,\exists \qquad \frac{\Gamma, x : \tau \vdash \varphi : o}{\Gamma \vdash \forall x : \tau.\ \varphi : o}\,\forall$$

$$\frac{\Gamma, \alpha : \mathsf{Type} \vdash \varphi_T : o}{\Gamma \vdash \forall\alpha.\ \varphi_T : o}\,\forall_T$$

Figure 3.2: Type System of Poly-FOL

We use the following notation $\Gamma = \Gamma_G; \varnothing$.

$$\frac{\dfrac{\dfrac{}{\Gamma \vdash T : \mathsf{Type}}\,\mathrm{TConstr}}{\Gamma \vdash \mathsf{set}(T) : \mathsf{Type}}\,\mathrm{TConstr} \qquad \dfrac{}{\Gamma \vdash u : \mathsf{set}(T)}\,\mathrm{Var} \qquad \Pi}{\Gamma \vdash\in (\mathsf{set}(T); u, \mathbb{P}(T; u))}\,\mathrm{Pred}$$

Where Π is:

$$\frac{\dfrac{}{\Gamma \vdash T : \mathsf{Type}}\,\mathrm{TConstr} \qquad \dfrac{}{\Gamma \vdash u : \mathsf{set}(T)}\,\mathrm{Var}}{\dfrac{\Gamma \vdash \mathbb{P}(T; u) : \mathsf{set}(\mathsf{set}(T))}{\Pi}}\,\mathrm{Fun}$$

Remark We say that Poly-FOL uses an explicit typing syntax in the sense that we provide to function and predicate symbols their type parameters.

This logic is a simplified version of ML-polymorphism, which is known to have a decidable type inference [Strachey 2000]. So, we could have chosen to use an implicit typing syntax, *i.e.* without type parameters.

3.3 LLproof: A Typed Sequent Calculus

We present in Fig. 3.3 and Fig. 3.4 the typed sequent calculus LLproof used by the automated theorem prover Zenon to output proofs. This proof system is an extension to polymorphic types of the initial sequent calculus LLproof of Zenon presented in [Bonichon, Delahaye, and Doligez 2007].

3.3.1 A Tableau-Like Proof System

This sequent calculus is close to a Tableau method: all formulæ are on the left-hand side of the sequent and we are looking for a contradiction, given the negation of the goal as an hypothesis along with the regular hypotheses. In addition, we always keep the main formula as an hypothesis for each rule, therefore leading to a growing context Γ, and removing the need for an explicit contraction rule.

3.3.2 Dealing With Special Rules

In Fig. 3.4, we define only one *special rule* called Subst, unlike the initial presentation of LLproof in [Bonichon, Delahaye, and Doligez 2007], where there were five distinct rules. It should be noted that the three rules called Def, Ext and Lemma in [Bonichon, Delahaye, and Doligez 2007] are not necessary for our work.

The following two rules called Pred and Fun in [Bonichon, Delahaye, and Doligez 2007] are admissible from Subst. These rules allow us to identify subterms of predicate and function symbols if their type parameters are equal.

$$\frac{\Delta, t_1 \neq_{\tau_1'} u_1 \vdash \bot \quad \cdots \quad \Delta, t_n \neq_{\tau_n'} u_n \vdash \bot}{\Gamma, P(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n), \neg P(\tau_1, \ldots, \tau_m; u_1, \ldots, u_n) \vdash \bot}\ \text{Pred}$$

Closure and Quantifier-free Rules

$$\frac{}{\Gamma, \bot \vdash \bot}\,\bot \qquad \frac{}{\Gamma, \neg\top \vdash \bot}\,\neg\top$$

$$\frac{}{\Gamma, P, \neg P \vdash \bot}\,\text{Ax} \qquad \frac{}{\Gamma, t \neq_\tau t \vdash \bot}\,\neq$$

$$\frac{}{\Gamma, t =_\tau u, u \neq_\tau t \vdash \bot}\,\text{Sym} \qquad \frac{\Gamma, P \vdash \bot \quad \Gamma, \neg P \vdash \bot}{\Gamma \vdash \bot}\,\text{Cut}$$

$$\frac{\Gamma, \neg\neg P, P \vdash \bot}{\Gamma, \neg\neg P \vdash \bot}\,\neg\neg \qquad \frac{\Gamma, P \wedge Q, P, Q \vdash \bot}{\Gamma, P \wedge Q \vdash \bot}\,\wedge$$

$$\frac{\Gamma, \neg(P \wedge Q), \neg P \vdash \bot \quad \Gamma, \neg(P \wedge Q), \neg Q \vdash \bot}{\Gamma, \neg(P \wedge Q) \vdash \bot}\,\neg\wedge$$

$$\frac{\Gamma, P \vee Q, P \vdash \bot \quad \Gamma, P \vee Q, Q \vdash \bot}{\Gamma, P \vee Q \vdash \bot}\,\vee$$

$$\frac{\Gamma, \neg(P \vee Q), \neg P, \neg Q \vdash \bot}{\Gamma, \neg(P \vee Q) \vdash \bot}\,\neg\vee$$

$$\frac{\Gamma, P \Rightarrow Q, \neg P \vdash \bot \quad \Gamma, P \Rightarrow Q, Q \vdash \bot}{\Gamma, P \Rightarrow Q \vdash \bot}\,\Rightarrow$$

$$\frac{\Gamma, \neg(P \Rightarrow Q), P, \neg Q \vdash \bot}{\Gamma, \neg(P \Rightarrow Q) \vdash \bot}\,\neg\Rightarrow$$

$$\frac{\Gamma, P \Leftrightarrow Q, \neg P, \neg Q \vdash \bot \quad \Gamma, P \Leftrightarrow Q, P, Q \vdash \bot}{\Gamma, P \Leftrightarrow Q \vdash \bot}\,\Leftrightarrow$$

$$\frac{\Gamma, \neg(P \Leftrightarrow Q), \neg P, Q \vdash \bot \quad \Gamma, \neg(P \Leftrightarrow Q), P, \neg Q \vdash \bot}{\Gamma, \neg(P \Leftrightarrow Q) \vdash \bot}\,\neg\Leftrightarrow$$

Figure 3.3: LLproof Inference Rules of Zenon (Part 1)

where $\Delta := \Gamma \cup P(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n), \neg P(\tau_1, \ldots, \tau_m; u_1, \ldots, u_n)$

$$\frac{\Delta, t_1 \neq_{\tau_1'} u_1 \vdash \bot \quad \cdots \quad \Delta, t_n \neq_{\tau_n'} u_n \vdash \bot}{\Gamma, f(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n) \neq_\tau f(\tau_1, \ldots, \tau_m; u_1, \ldots, u_n) \vdash \bot}\,\text{Fun}$$

where $\Delta := \Gamma \cup f(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n) \neq_\tau f(\tau_1, \ldots, \tau_m; u_1, \ldots, u_n)$

In the following, we give the derivation of an application of the inference rule Pred

Quantifier Rules

$$\frac{\Gamma, \forall\alpha.\ P(\alpha), P(\tau) \vdash \bot}{\Gamma, \forall\alpha.\ P(\alpha) \vdash \bot}\ \forall_{\mathsf{type}} \qquad \text{where } \tau \text{ is any ground type}$$

$$\frac{\Gamma, \exists x : \tau.\ P(x), P(c) \vdash \bot}{\Gamma, \exists x : \tau.\ P(x) \vdash \bot}\ \exists$$

$$\frac{\Gamma, \neg\forall x : \tau.\ P(x), \neg P(c) \vdash \bot}{\Gamma, \neg\forall x : \tau.\ P(x) \vdash \bot}\ \neg\forall \qquad \text{where } c : \tau \text{ is a fresh constant}$$

$$\frac{\Gamma, \forall x : \tau.\ P(x), P(t) \vdash \bot}{\Gamma, \forall x : \tau.\ P(x) \vdash \bot}\ \forall$$

$$\frac{\Gamma, \neg\exists x : \tau.\ P(x), \neg P(t) \vdash \bot}{\Gamma, \neg\exists x : \tau.\ P(x) \vdash \bot}\ \neg\exists \qquad \text{where } t : \tau \text{ is any ground term}$$

Special Rule

$$\frac{\Gamma, P(t), t \neq_\tau u \vdash \bot \qquad \Gamma, P(t), P(u) \vdash \bot}{\Gamma, P(t) \vdash \bot}\ \text{Subst}$$

Figure 3.4: LLproof Inference Rules of Zenon (Part 2)

using n applications of the rule Subst. To lighten the presentation, we do not repeat the contexts.

$$\frac{\dfrac{\Pi_1}{t_1 \neq_{\tau_1'} u_1 \vdash \bot} \quad \dfrac{\Pi_2}{t_2 \neq_{\tau_2'} u_2 \vdash \bot} \quad \cdots \quad \dfrac{\Pi_n}{t_n \neq_{\tau_n'} u_n \vdash \bot}}{P(\tau_1, \ldots, \tau_m; t_1, t_2, \ldots, t_n), \neg P(\tau_1, \ldots, \tau_m; u_1, u_2, \ldots, u_n) \vdash \bot}\ \text{Pred} \quad :=$$

$$\frac{\dfrac{\dfrac{\Pi_1}{t_1 \neq_{\tau_1'} u_1 \vdash \bot} \quad \dfrac{\dfrac{\Pi_2}{t_2 \neq_{\tau_2'} u_2 \vdash \bot} \quad \dfrac{\vdots}{P(\tau_1, \ldots, \tau_m; u_1, u_2, \ldots, t_n) \vdash \bot}}{P(\tau_1, \ldots, \tau_m; u_1, t_2, \ldots, t_n) \vdash \bot}\ \text{Subst}}{P(\tau_1, \ldots, \tau_m; t_1, t_2, \ldots, t_n), \neg P(\tau_1, \ldots, \tau_m; u_1, u_2, \ldots, u_n) \vdash \bot}\ \text{Subst}$$

$$\dfrac{\dfrac{\Pi_n}{t_n \neq_{\tau_n'} u_n \vdash \bot} \quad \dfrac{}{P(\tau_1, \ldots, \tau_m; u_1, u_2, \ldots, u_n) \vdash \bot}\ \text{Ax}}{P(\tau_1, \ldots, \tau_m; u_1, u_2, \ldots, u_{n-1}, t_n) \vdash \bot}\ \text{Subst}$$

The case of the rule Fun is also straightforward. Here is the derivation for an application of the rule Fun:

$$\dfrac{\dfrac{\Pi_1}{t_1 \neq_{\tau_1'} u_1 \vdash \bot} \quad \cdots \quad \dfrac{\Pi_n}{t_n \neq_{\tau_n'} u_n \vdash \bot}}{f(\tau_1,\ldots,\tau_m;t_1,t_2,\ldots,t_n) \neq_\tau f(\tau_1,\ldots,\tau_m;u_1,u_2,\ldots,u_n) \vdash \bot}\,\text{Fun} \quad :=$$

$$\dfrac{\dfrac{\dfrac{\Pi_1}{t_1 \neq_{\tau_1'} u_1 \vdash \bot} \quad \dfrac{\dfrac{\Pi_n}{t_n \neq_{\tau_n'} u_n \vdash \bot} \quad \dfrac{}{f(\tau_1,\ldots,\tau_m;u_1,\ldots,u_n) \neq f(\tau_1,\ldots,\tau_m;u_1,\ldots,u_n) \vdash \bot}\neq}{\vdots}}{f(\tau_1,\ldots,\tau_m;u_1,\ldots,t_n) \neq_\tau f(\tau_1,\ldots,\tau_m;u_1,\ldots,u_n) \vdash \bot}\,\text{Subst}}{f(\tau_1,\ldots,\tau_m;t_1,\ldots,t_n) \neq_\tau f(\tau_1,\ldots,\tau_m;u_1,\ldots,u_n) \vdash \bot}\,\text{Subst}$$

3.3.3 Admissibility of Rules Dealing with $\vee$, $\Leftrightarrow$ and $\exists$

In the B Method, logical connectives $\vee$ and $\Leftrightarrow$, and existential quantification $\exists$ are derived from other symbols (see Sec. 1.2.1):

$$\begin{array}{lcl} P \vee Q & := & \neg P \Rightarrow Q \\ P \Leftrightarrow Q & := & (P \Rightarrow Q) \wedge (Q \Rightarrow P) \\ \exists x \cdot P & := & \neg \forall x \cdot \neg P \end{array}$$

Thus, in Sec. 4.2.2, we consider LLproof rules except $\vee$, $\neg\vee$, $\Leftrightarrow$, $\neg\Leftrightarrow$, $\exists$ and $\neg\exists$, and we show the admissibility of these rules here. It should be noted that B goals and formulæ will never contains such connectives/quantifiers. In the following, we omit to repeat the contexts.

3.3.3.1 Inference Rules $\vee$ and $\neg\vee$

For the disjunction, we have $P \vee Q := \neg P \Rightarrow Q$. We obtain the following derivations:

$$\dfrac{\dfrac{\dfrac{P \vdash \bot}{\neg\neg P \vdash \bot}\,\neg\neg \qquad Q \vdash \bot}{\neg P \Rightarrow Q \vdash \bot}\,\Rightarrow$$

$$\dfrac{\neg P, \neg Q \vdash \bot}{\neg(\neg P \Rightarrow Q) \vdash \bot}\,\neg\Rightarrow$$

3.3.3.2 Inference Rules $\Leftrightarrow$ and $\neg\Leftrightarrow$

For the equivalence, we have $P \Leftrightarrow Q := (P \Rightarrow Q) \wedge (Q \Rightarrow P)$. We obtain the following derivations:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg P, \neg Q \vdash \bot}\,\mathrm{Ax} \quad \dfrac{}{\neg P, P \vdash \bot}\,\mathrm{Ax}}{\neg P, Q \Rightarrow P \vdash \bot}\Rightarrow \quad \dfrac{\dfrac{}{Q, \neg Q \vdash \bot}\,\mathrm{Ax} \quad \dfrac{}{Q, P \vdash \bot}\,\mathrm{Ax}}{Q, Q \Rightarrow P \vdash \bot}\Rightarrow}{P \Rightarrow Q, Q \Rightarrow P \vdash \bot}\Rightarrow}{(P \Rightarrow Q) \wedge (Q \Rightarrow P) \vdash \bot}\wedge$$

$$\dfrac{\dfrac{\dfrac{P, \neg Q \vdash \bot}{\neg(P \Rightarrow Q) \vdash \bot}\neg\Rightarrow \quad \dfrac{Q, \neg P \vdash \bot}{\neg(Q \Rightarrow P) \vdash \bot}\neg\Rightarrow}{\neg((P \Rightarrow Q) \wedge (Q \Rightarrow P)) \vdash \bot}\neg\wedge$$

3.3.3.3 Inference Rules $\exists$ and $\neg\exists$

For the existential quantification, we have $\exists x.\ P := \neg\forall x.\ \neg P$. We obtain the following derivations:

$$\dfrac{\dfrac{\dfrac{P(c) \vdash \bot}{\neg\neg P(c) \vdash \bot}\neg\neg}{\neg\forall x : \tau.\ \neg P(x) \vdash \bot}\neg\forall$$

$$\dfrac{\dfrac{\dfrac{\neg P(t) \vdash \bot}{\forall x : \tau.\ \neg P(x) \vdash \bot}\forall}{\neg\neg\forall x : \tau.\ \neg P(x) \vdash \bot}\neg\neg$$

Chapter 4

Proving in B through Sequent Calculus

This chapter shows how to use the typed sequent calculus LLproof to prove B formulæ. This is done using an encoding of B formulæ into Poly-FOL, followed by a mere syntactic translation of LLproof proofs into B Natural Deduction proofs.

This chapter is a personal contribution and it has been published in [Halmagrand 2016].

4.1 Translating B Formulæ into Poly-FOL

This section presents the encoding of B Method formulæ into Poly-FOL. We first give in Sec. 4.1.1 the type signatures of the primitive constructs, followed in Sec. 4.1.2 by the general encoding function.

4.1.1 Type Signatures of Primitive Constructs

We start by defining a general skeleton for the type signatures of the B basic constructs. We introduce two type constructors set and tup corresponding respectively to the B type constructors $\mathbb{P}$ and $\times$. Then, we define the function symbols $(\text{-},\text{-})$ for ordered pairs, $\mathbb{P}(\text{-})$ for powersets and $\text{-} \times \text{-}$ for product sets. Finally, we define a predicate symbol for membership. For easier reading, we use an infix notation with subscript type arguments (see Ex. 3.2.1).

$$\mathcal{T}_{\mathsf{ske}} := \left\{ \begin{array}{lcl} \mathsf{set} & :: & 1 \\ \mathsf{tup} & :: & 2 \\ (\text{-},\text{-}) & : & \Pi\alpha_1\alpha_2.\ \alpha_1 \times \alpha_2 \to \mathsf{tup}(\alpha_1, \alpha_2) \\ \mathbb{P}(\text{-}) & : & \Pi\alpha.\ \mathsf{set}(\alpha) \to \mathsf{set}(\mathsf{set}(\alpha)) \\ \text{-} \times \text{-} & : & \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \\ \text{-} \in \text{-} & : & \Pi\alpha.\ \alpha \times \mathsf{set}(\alpha) \to o \end{array} \right.$$

4.1.2 Encoding of B Formulæ into Poly-FOL

In the following, P, E and T denote respectively a B formula, a B expression and a B type. In addition, we suppose that formulæ are well-typed and that the annotation procedure of Sec. 2.2 and the skolemization of Sec. 2.3 have been applied successfully.

We present in Fig. 4.1 the encoding functions $\langle P \rangle_{\mathsf{f}}$, $\langle E \rangle_{\mathsf{e}}$ and $\langle T \rangle_{\mathsf{t}}$ which translate respectively B formulæ, expressions and types into Poly-FOL formulæ, terms and types. In addition, we define a function θ which returns the Poly-FOL type of a B expression.

The roles of Δ and Ω in Fig. 4.1 are detailed in Sec. 4.1.2.1.

4.1.2.1 Target Theory

During the translation of a set of B formulæ, we carry a target Poly-FOL theory $\mathcal{T}$ containing the skeleton $\mathcal{T}_{\mathsf{ske}}$ defined in Sec. 4.1.1, and previously translated formulæ. In addition, we increase it by new type constructors when translating new type identifiers in a goal, and new type signatures in two cases, when translating a symbol that is not declared in the local context Δ of bound variables – then it is a constant (a given set) –, and when translating a function symbol.

Also, for each formula to be translated, we carry a set Ω for type variables and type constructors (see Sec. 4.1.2.2), and a Poly-FOL local context Δ of bound variables and their type.

Once a B formula P is translated, we take the universal closure of the translation of P with respect to the type variables of Ω, denoted $\langle P \rangle$:

$$\langle P \rangle \ := \ \underset{\alpha \in \Omega}{\forall} \vec{\alpha}.\ \langle P \rangle_{\mathsf{f}}$$

$$
\begin{array}{lll}
\theta(E)^{\Delta} = \mathsf{match}\ E\ \mathsf{with} & & \\
\quad \mid x^T & \to \Delta(x) & \\
\quad \mid E_1, E_2 & \to \mathsf{tup}(\theta(E_1)^{\Delta}, \theta(E_2)^{\Delta}) & \\
\quad \mid E_1 \times E_2 & \to \mathsf{set}(\mathsf{tup}(\theta(E_1)^{\Delta}, \theta(E_2)^{\Delta})) & \\
\quad \mid \mathbb{P}(E) & \to \mathsf{set}(\theta(E)^{\Delta}) & \\
\quad \mid f^T(\ldots) & \to \langle T \rangle_{\mathsf{t}} & \\
\\
\langle T \rangle_{\mathsf{t}} = \mathsf{match}\ T\ \mathsf{with} & & \\
\quad \mid id\ \mathsf{when}\ flag = ax & \to \mathsf{if}\ id \notin \Omega\ \mathsf{then}\ \Omega := \Omega, (id, \alpha) & \\
 & \quad \mathsf{return}\ \Omega(id) & \\
\quad \mid id\ \mathsf{when}\ flag = gl & \to \mathsf{if}\ id \notin \Omega\ \mathsf{then}\ \mathcal{T} := \mathcal{T}, T :: 0;\ \Omega := \Omega, (id, T) & \\
 & \quad \mathsf{return}\ \Omega(id) & \\
\quad \mid T_1 \times T_2 & \to \mathsf{tup}(\langle T_1 \rangle_{\mathsf{t}}, \langle T_2 \rangle_{\mathsf{t}}) & \\
\quad \mid \mathbb{P}(T) & \to \mathsf{set}(\langle T \rangle_{\mathsf{t}}) & \\
\\
\langle E \rangle_{\mathsf{e}}^{\Delta} = \mathsf{match}\ E\ \mathsf{with} & & \\
\quad \mid x^T & \to \mathsf{if}\ x \notin \Delta\ \mathsf{then}\ \mathcal{T} := \mathcal{T}, x : \langle T \rangle_{\mathsf{t}} & \\
 & \quad \mathsf{return}\ x & \\
\quad \mid E_1, E_2 & \to (\langle E_1 \rangle_{\mathsf{e}}^{\Delta}, \langle E_2 \rangle_{\mathsf{e}}^{\Delta})_{\widetilde{\tau_1}, \widetilde{\tau_2}} & \\
\quad \mid E_1 \times E_2 & \to \langle E_1 \rangle_{\mathsf{e}}^{\Delta} \times_{\widetilde{\tau_1}, \widetilde{\tau_2}} \langle E_2 \rangle_{\mathsf{e}}^{\Delta} & \\
\quad \mid \mathbb{P}(E) & \to \mathbb{P}_{\widetilde{\tau}}(\langle E \rangle_{\mathsf{e}}^{\Delta}) & \\
\quad \mid f^T(E_1, \ldots, E_n) & \to \mathsf{if}\ f : \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to \tau \notin \mathcal{T} & \\
 & \quad \mathsf{then}\ \mathcal{T} := \mathcal{T}, f : Sig(f^T(E_1, \ldots, E_n)) & \\
 & \quad \mathsf{return}\ f(\widetilde{\tau'_1}, \ldots, \widetilde{\tau'_m}; \langle E_1 \rangle_{\mathsf{e}}^{\Delta}, \ldots, \langle E_n \rangle_{\mathsf{e}}^{\Delta}) & \\
\\
\langle P \rangle_{\mathsf{f}}^{\Delta} = \mathsf{match}\ P\ \mathsf{with} & & \\
\quad \mid \bot & \to \bot & \\
\quad \mid \top & \to \top & \\
\quad \mid P_1 \wedge P_2 & \to \langle P_1 \rangle_{\mathsf{f}}^{\Delta} \wedge \langle P_2 \rangle_{\mathsf{f}}^{\Delta} & \\
\quad \mid P_1 \Rightarrow P_2 & \to \langle P_1 \rangle_{\mathsf{f}}^{\Delta} \Rightarrow \langle P_2 \rangle_{\mathsf{f}}^{\Delta} & \\
\quad \mid \neg P & \to \neg \langle P \rangle_{\mathsf{f}}^{\Delta} & \\
\quad \mid \forall x^T \cdot P & \to \forall x : \langle T \rangle_{\mathsf{t}} \cdot \langle P \rangle_{\mathsf{f}}^{\Delta, x : \langle T \rangle_{\mathsf{t}}} & \\
\quad \mid \forall (x_1^{T_1}, x_2^{T_2}) \cdot P & \to \forall x_1 : \langle T_1 \rangle_{\mathsf{t}} . \forall x_2 : \langle T_2 \rangle_{\mathsf{t}} \cdot \langle P \rangle_{\mathsf{f}}^{\Delta, x_1 : \langle T_1 \rangle_{\mathsf{t}}, x_2 : \langle T_2 \rangle_{\mathsf{t}}} & \\
\quad \mid E_1 = E_2 & \to \langle E_1 \rangle_{\mathsf{e}}^{\Delta} =_{\widetilde{\tau}} \langle E_2 \rangle_{\mathsf{e}}^{\Delta} & \\
\quad \mid E_1 \in E_2 & \to \langle E_1 \rangle_{\mathsf{e}}^{\Delta} \in_{\widetilde{\tau}} \langle E_2 \rangle_{\mathsf{e}}^{\Delta} &
\end{array}
$$

Figure 4.1: Translation from B to Poly-FOL

4.1.2.2 B Type Identifier Translation

One important point in this encoding is the interpretation given to B type identifiers coming from the type annotation procedure (see Sec. 2.2).

When starting to translate a formula, we define a set Ω that contains pairs of B identifier and Poly-FOL type variable or type constructor.

For axioms and hypotheses, we interpret B type identifiers as type variables. For a new identifier id, we generate a fresh type variable symbol α and store the pair (id, α) into Ω. Once the translation of an axiom is done, we take the universal closure with respect to all the type variables stored in Ω.

For B goals, *i.e.* formulæ that we want to prove, B type identifiers are interpreted as type constants, *i.e.* nullary type constructors. For a new identifier id, we generate a fresh type constructor symbol T and store the pair (id, T) in Ω.

This allows us to get polymorphic axioms in Poly-FOL and a monomorphic many-sorted goal. To achieve this, we add to all B formulæ to translate a flag ax for axioms and hypotheses and gl for the goal.

4.1.2.3 Generation of Type Signature for Function Symbols

The skolemization of sets defined by comprehension presented in Sec. 2.3 generates new function symbols. In addition, it generates new axioms defining the membership to the set corresponding to (the application of) these function symbols.

We suppose that the translation of a set of formulæ starts with these axioms. During the translation of these axioms, we have to add to the Poly-FOL theory the type signatures of the new function symbols. We define a function called $Sig(f(\ldots))$, where f is a B function symbol, that computes the type signature of f.

We extend the function $\mathrm{FV}(e)$, returning the free variables of an expression e, to a list of expressions $e_1, \ldots, e_n$, denoted $\mathrm{FV}_1^n(e_i)$ and returning the union of all the sets of free variables.

$$Sig(f^T(E_1, \ldots, E_n)) = \prod_{\alpha \in \mathrm{FV}_1^n(\theta(E_i))} \vec{\alpha}.\ \theta(E_1) \times \ldots \times \theta(E_n) \to \theta(T)$$

4.1.2.4 Type Parameters for Function and Predicate Symbols

When translating function and predicate symbols, like $\mathbb{P}$, we have to find the proper type parameters for the corresponding Poly-FOL symbol. For instance, if we have to translate the B term $\mathbb{P}(a)$, where a is a variable declared in Δ such that $a : \mathsf{set}(T)$ for a type constant T, we want to find τ such that we obtain the Poly-FOL term $\mathbb{P}_\tau(a)$.

We know that the function θ will give us $\theta(a)^\Delta = \mathsf{set}(T)$, and that we have $\mathbb{P}(\text{-}) : \Pi\alpha.\ \mathsf{set}(\alpha) \to \mathsf{set}(\mathsf{set}(\alpha))$, leading to solve the equation $\mathsf{set}(\alpha) = \mathsf{set}(T)$.

This is a standard syntactic unification problem that we have to perform each time we need to find the type parameters. When we translate a B term $f(E_1, \ldots, E_n)$ into a Poly-FOL term $f(\tau'_1, \ldots, \tau'_m; \langle E_1 \rangle^\Delta_{\mathsf{e}}, \ldots, \langle E_n \rangle^\Delta_{\mathsf{e}})$, where the Poly-FOL f is such that $f : \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to \tau$, we perform a syntactic unification to solve the system:

$$\begin{cases} \theta(E_1)^\Delta &= \tau_1 \\ &\vdots \\ \theta(E_n)^\Delta &= \tau_n \end{cases}$$

in order to find the type parameters $\tau'_1, \ldots, \tau'_m$.

In Fig. 4.1, we use the notation $\widetilde{\tau}$ to denote that the type τ has been found that way.

4.1.2.5 Example

We want to apply this encoding to the B formulæ of the running example of Chap. 2.

We have a B theory made of the axioms $\mathsf{SET1}^\star$, $\mathsf{SET2}^\star$ and $\mathsf{SET4}^\star$, the axiom coming from the skolemization and the goal $(P_{\text{ex}})^{\star s}$ (see Ex. 2.3.1):

$$\begin{array}{l} \forall s^{\mathbb{P}(u)} \cdot \forall t^{\mathbb{P}(v)} \cdot \forall x^u \cdot \forall y^v \cdot (x, y \in s \times t \Leftrightarrow (x \in s \wedge y \in t)) \\ \forall s^{\mathbb{P}(u)} \cdot \forall t^{\mathbb{P}(u)} \cdot (s \in \mathbb{P}(t) \Leftrightarrow \forall x^u \cdot (x \in s \Rightarrow x \in t)) \\ \forall s^{\mathbb{P}(u)} \cdot \forall t^{\mathbb{P}(u)} \cdot (\forall x^u \cdot (x \in s \Leftrightarrow x \in t) \Rightarrow s = t) \end{array}$$

$$\begin{array}{l} \forall a^{\mathbb{P}(s\times t)} \cdot \forall b^{\mathbb{P}(s\times t)} \cdot \forall x^{s\times t} \cdot (x \in f^{\mathbb{P}(s\times t)}(a, b) \Leftrightarrow x \in a \wedge x \in b) \\ \forall (a^{\mathbb{P}(s\times t)}, b^{\mathbb{P}(s\times t)}) \cdot (a, b \in \mathbb{P}(s^{\mathbb{P}(s)} \times t^{\mathbb{P}(t)}) \times \mathbb{P}(s \times t) \Rightarrow f^{\mathbb{P}(s\times t)}(a, b) \subseteq s \times t) \end{array}$$

We first obtain the three set theory axioms $\langle \mathsf{SET1}^\star \rangle$, $\langle \mathsf{SET2}^\star \rangle$ and $\langle \mathsf{SET4}^\star \rangle$:

$$\begin{array}{l} \forall \alpha_1, \alpha_2.\ \forall s : \mathsf{set}(\alpha_1), t : \mathsf{set}(\alpha_2), x : \alpha_1, y : \alpha_2. \\ \qquad (x, y)_{\alpha_1,\alpha_2} \in_{\mathsf{tup}(\alpha_1,\alpha_2)} s \times_{\alpha_1,\alpha_2} t \Leftrightarrow (x \in_{\alpha_1} s \wedge y \in_{\alpha_2} t) \\ \forall \alpha.\ \forall s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha).\ s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) \Leftrightarrow (\forall x : \alpha.\ x \in_\alpha s \Rightarrow x \in_\alpha t) \\ \forall \alpha.\ \forall s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha).\ (\forall x : \alpha.\ x \in_\alpha s \Leftrightarrow x \in_\alpha t) \Rightarrow s =_{\mathsf{set}(\alpha)} t \end{array}$$

The remainder of the theory, *i.e.* the declaration of the two type constants T_1 and T_2 coming from the translation of the goal, the signatures of the two constants s and t (the given sets), the signature of f and the axiom defining f, is:

$$
\begin{array}{l}
T_1 :: 0 \\
T_2 :: 0 \\
s : \mathsf{set}(T_1) \\
t : \mathsf{set}(T_2) \\
f : \Pi\alpha_1\alpha_2.\ \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)) \times \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)) \to \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)) \\
\forall \alpha_1,\alpha_2.\ \forall a : \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)), b : \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)), x : \mathsf{tup}(\alpha_1,\alpha_2). \\
\qquad x \in_{\mathsf{tup}(\alpha_1,\alpha_2)} f_{\alpha_1,\alpha_2}(a,b) \Leftrightarrow (x \in_{\mathsf{tup}(\alpha_1,\alpha_2)} a \wedge x \in_{\mathsf{tup}(\alpha_1,\alpha_2)} b)
\end{array}
$$

Finally, the translation of the goal $\langle (P_{\text{ex}})^{\star s} \rangle$ – we unfold the $\subseteq$ definition and remove the subscript type arguments of function and predicate symbols to lighten the formula – is:

$$\forall a : \mathsf{set}(\mathsf{tup}(T_1,T_2)), b : \mathsf{set}(\mathsf{tup}(T_1,T_2)).\ (a,b) \in \mathbb{P}(s \times t) \times \mathbb{P}(s \times t) \Rightarrow f(a,b) \in \mathbb{P}(s \times t)$$

4.2 Translation of LLproof Proofs into B Proofs

The last section of this chapter presents the translation of LLproof proofs into B Method Natural Deduction proofs. This translation of proofs relies on an encoding of Poly-FOL into B, restricted to monomorphic formulæ.

4.2.1 Encoding of Poly-FOL into B

In the following, φ is a monomorphic Poly-FOL formula and t is a monomorphic Poly-FOL term. In Fig. 4.2, we present the two encoding functions $\langle \varphi \rangle_{\mathsf{f}}^{\text{-1}}$ and $\langle t \rangle_{\mathsf{e}}^{\text{-1}}$ which translate respectively monomorphic Poly-FOL formulæ and terms into B formulæ and terms.

Lemma 4.2.1

For any B expression E and any B formula P such that $\langle E^{\star s} \rangle_{\mathsf{e}}$ and $\langle P^{\star s} \rangle_{\mathsf{f}}$ are monomorphic, we can prove, in B:

$$\langle \langle E^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{\text{-1}} = E^s \qquad \langle \langle P^{\star s} \rangle_{\mathsf{f}} \rangle_{\mathsf{f}}^{\text{-1}} \Leftrightarrow P^s$$

where E^s and P^s correspond respectively to $E^{\star s}$ and $P^{\star s}$ where we erase type annotations on variables.

This is in particular the case for B goals.

$$
\begin{array}{rcll}
\langle t \rangle_{\mathsf{e}}^{-1} & = & \mathsf{match}\ t\ \mathsf{with} & \\
& | & x & \rightarrow\ x \\
& | & (t_1, t_2)_{\tau_1,\tau_2} & \rightarrow\ \langle t_1 \rangle_{\mathsf{e}}^{-1}, \langle t_2 \rangle_{\mathsf{e}}^{-1} \\
& | & t_1 \times_{\tau_1,\tau_2} t_2 & \rightarrow\ \langle t_1 \rangle_{\mathsf{e}}^{-1} \times \langle t_2 \rangle_{\mathsf{e}}^{-1} \\
& | & \mathbb{P}_\tau(t) & \rightarrow\ \mathbb{P}(\langle t \rangle_{\mathsf{e}}^{-1}) \\
& | & f(\tau'_1, \ldots, \tau'_m; t_1, \ldots, t_n) & \rightarrow\ f(\langle t_1 \rangle_{\mathsf{e}}^{-1}, \ldots, \langle t_n \rangle_{\mathsf{e}}^{-1}) \\
\\
\langle \varphi \rangle_{\mathsf{f}}^{-1} & = & \mathsf{match}\ \varphi\ \mathsf{with} & \\
& | & \bot & \rightarrow\ \bot \\
& | & \top & \rightarrow\ \top \\
& | & \varphi_1 \wedge \varphi_2 & \rightarrow\ \langle \varphi_1 \rangle_{\mathsf{f}}^{-1} \wedge \langle \varphi_2 \rangle_{\mathsf{f}}^{-1} \\
& | & \varphi_1 \Rightarrow \varphi_2 & \rightarrow\ \langle \varphi_1 \rangle_{\mathsf{f}}^{-1} \Rightarrow \langle \varphi_2 \rangle_{\mathsf{f}}^{-1} \\
& | & \neg \varphi & \rightarrow\ \neg \langle \varphi \rangle_{\mathsf{f}}^{-1} \\
& | & \forall x : \tau.\ \varphi & \rightarrow\ \forall x \cdot \langle \varphi \rangle_{\mathsf{f}}^{-1} \\
& | & t_1 =_\tau t_2 & \rightarrow\ \langle t_1 \rangle_{\mathsf{e}}^{-1} = \langle t_2 \rangle_{\mathsf{e}}^{-1} \\
& | & t_1 \in_\tau t_2 & \rightarrow\ \langle t_1 \rangle_{\mathsf{e}}^{-1} \in \langle t_2 \rangle_{\mathsf{e}}^{-1}
\end{array}
$$

Figure 4.2: Translation from Poly-FOL to B

Proof

The proof, by induction on the structure of expressions and formulæ, is straightforward because the translation of Fig. 4.2 just erase type information of Poly-FOL monomorphic expressions and formulæ. As we will see, the only case that modify the actual shape of expressions and formulæ is the quantification over list of variables.

For expressions, we have:

$$
\begin{array}{ll}
\left\langle \left\langle x^T \right\rangle_{\mathsf{e}} \right\rangle_{\mathsf{e}}^{-1} & \rightarrow\ x \\
\langle \langle E_1^{\star s}, E_2^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1} & \rightarrow\ \langle \langle E_1^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1}, \langle \langle E_2^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1} \\
\langle \langle E_1^{\star s} \times E_2^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1} & \rightarrow\ \langle \langle E_1^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1} \times \langle \langle E_2^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1} \\
\langle \langle \mathbb{P}(E^{\star s}) \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1} & \rightarrow\ \mathbb{P}(\langle \langle E^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1}) \\
\left\langle \left\langle f^{\mathbb{P}(T)}(E_1^{\star s}, \ldots, E_n^{\star s}) \right\rangle_{\mathsf{e}} \right\rangle_{\mathsf{e}}^{-1} & \rightarrow\ f(\langle \langle E_1^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1}, \ldots, \langle \langle E_n^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1})
\end{array}
$$

By induction on E – where x is the base case – we have $\langle \langle E^{\star s} \rangle_{\mathsf{e}} \rangle_{\mathsf{e}}^{-1} = E^s$

For formulæ, we have:

$$\begin{array}{lcl}
\langle\langle \bot \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} & \to & \bot \\
\langle\langle \top \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} & \to & \top \\
\langle\langle P_1^{\star s} \wedge P_2^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} & \to & \langle\langle P_1^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} \wedge \langle\langle P_2^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} \\
\langle\langle P_1^{\star s} \Rightarrow P_2^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} & \to & \langle\langle P_1^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} \Rightarrow \langle\langle P_2^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} \\
\langle\langle \neg P^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} & \to & \neg \langle\langle P^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} \\
\left\langle\left\langle \forall x^T \cdot P^{\star s} \right\rangle_{\mathsf{f}}\right\rangle_{\mathsf{f}}^{-1} & \to & \forall x \cdot \langle\langle P^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} \\
\left\langle\left\langle \forall (x_1^{T_1}, x_2^{T_2}) \cdot P^{\star s} \right\rangle_{\mathsf{f}}\right\rangle_{\mathsf{f}}^{-1} & \to & \forall x_1 \cdot \forall x_2 \cdot \langle\langle P^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} \\
\langle\langle E_1^{\star s} = E_2^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} & \to & \langle\langle E_1^{\star s} \rangle_{\mathsf{e}}\rangle_{\mathsf{e}}^{-1} = \langle\langle E_2^{\star s} \rangle_{\mathsf{e}}\rangle_{\mathsf{e}}^{-1} \\
\langle\langle E_1^{\star s} \in E_2^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} & \to & \langle\langle E_1^{\star s} \rangle_{\mathsf{e}}\rangle_{\mathsf{e}}^{-1} \in \langle\langle E_2^{\star s} \rangle_{\mathsf{e}}\rangle_{\mathsf{e}}^{-1}
\end{array}$$

We remark that, without considering type annotation erasure, the only case that modifies the syntax is the quantification over a list of variables, here over two variables:

$$\left\langle\left\langle \forall (x_1^{T_1}, x_2^{T_2}) \cdot P^{\star s} \right\rangle_{\mathsf{f}}\right\rangle_{\mathsf{f}}^{-1} \quad \to \quad \forall x_1 \cdot \forall x_2 \cdot \langle\langle P^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1}$$

But the Theorem 1.5.3 of the B-Book tells us that, if $x \backslash y$

$$\forall (x, y) \cdot P \Leftrightarrow \forall x \cdot \forall y \cdot P$$

So, by induction on P, we have $\langle\langle P^{\star s} \rangle_{\mathsf{f}}\rangle_{\mathsf{f}}^{-1} \Leftrightarrow P^s$

Corollary 4.2.2

For any B expression E and any B formula P such that $\langle E^{\star s} \rangle_{\mathsf{e}}$ and $\langle P^{\star s} \rangle_{\mathsf{f}}$ are polymorphic, we can prove, in B:

$$\langle\langle E^{\star s} \rangle_{\mathsf{e}}^{mono}\rangle_{\mathsf{e}}^{-1} = E^s \qquad \langle\langle P^{\star s} \rangle_{\mathsf{f}}^{mono}\rangle_{\mathsf{f}}^{-1} \Leftrightarrow P^s$$

where $\langle E^{\star s} \rangle_{\mathsf{e}}^{mono}$ and $\langle P^{\star s} \rangle_{\mathsf{f}}^{mono}$ are any monomorphic instance of $\langle E^{\star s} \rangle_{\mathsf{e}}$ and $\langle P^{\star s} \rangle_{\mathsf{f}}$ respectively.

This is in particular the case for B axioms.

4.2.2 Translation of LLproof Proofs Into B Proofs

We first extend to LLproof sequents the translation from Poly-FOL to B:

$$\langle P_1, \ldots, P_n \vdash_{\mathsf{LL}} Q \rangle^{-1} \to \langle P_1 \rangle^{-1}, \ldots, \langle P_n \rangle^{-1} \vdash_{\mathsf{B}} \langle Q \rangle^{-1}$$

Then, we give in Fig. 4.3 and Fig. 4.4 the translations for each LLproof proof node. Each node can be translated into a B derivation where all LLproof sequents are translated into B sequents, leading to a B proof tree. To lighten the presentation, we omit to indicate the context Γ and useless formulæ – removable by applying BR2 – on the left-hand side of sequents, and we use $\vdash$ for $\vdash_{\mathsf{LL}}$. For instance, the translation of the LLproof Axiom rule is actually:

$$\dfrac{\dfrac{}{\langle \Gamma, P, \neg P, \neg\bot \vdash_{\mathsf{LL}} P\rangle^{-1}}\,\textsf{BR3} \qquad \dfrac{}{\langle \Gamma, P, \neg P, \neg\bot \vdash_{\mathsf{LL}} \neg P\rangle^{-1}}\,\textsf{BR3}}{\langle \Gamma, P, \neg P \vdash_{\mathsf{LL}} \bot\rangle^{-1}}\,\textsf{R5}$$

Remark The translation is straightforward because of the use of the cut rule BR4 almost systematically. Otherwise, it would require an induction on the proof tree. The problem of cut elimination in the final B proofs is not relevant here since we only care about having valid B proofs.

In addition, we do not present the translations for LLproof rules dealing with $\vee$, $\Leftrightarrow$ and $\exists$ since these symbols are not primitive in B. We have previously given the LLproof derivations for these rules in Sec. 3.3.3.

4.2.3 Conservativity of Provability

Theorem 4.2.3 (Proof Translation)
For a set of B formulæ Γ and a B goal P, if there exists a LLproof *proof of the sequent $\langle \Gamma^{\star s}\rangle, \neg\langle P^{\star s}\rangle \vdash_{\mathsf{LL}} \bot$, then there exists a B proof of the sequent $\Gamma \vdash_{\mathsf{B}} P$.*

Proof

1. Let Γ be a set of B formulæ containing axioms and hypotheses, and P a B goal.

   We suppose that we have a LLproof proof Π such that:

   $$\dfrac{\Pi}{\langle \Gamma^{\star s}\rangle, \neg\langle P^{\star s}\rangle \vdash_{\mathsf{LL}} \bot}$$

2. Given the proof Π of the sequent $\langle \Gamma^{\star s}\rangle, \neg\langle P^{\star s}\rangle \vdash_{\mathsf{LL}} \bot$, there exists a proof Π_{Kleene} of the same sequent, starting with all applications of $\forall_{\mathsf{type}}$ rules on polymorphic

Axiom

$$\dfrac{\dfrac{}{\langle P \vdash P\rangle^{-1}}\,\mathsf{BR3} \qquad \dfrac{}{\langle \neg P \vdash \neg P\rangle^{-1}}\,\mathsf{BR3}}{\langle P, \neg P \vdash \bot\rangle^{-1}}\,\mathsf{R5}$$

$\neq$

$$\dfrac{\dfrac{}{\langle \vdash t =_\tau t\rangle^{-1}}\,\mathsf{R10} \qquad \dfrac{}{\langle \neg(t =_\tau t) \vdash \neg(t =_\tau t)\rangle^{-1}}\,\mathsf{BR3}}{\langle \neg(t =_\tau t) \vdash \bot\rangle^{-1}}\,\mathsf{R5}$$

Sym

$$\dfrac{\dfrac{}{\langle \neg(u =_\tau t) \vdash \neg(u =_\tau t)\rangle^{-1}}\,\mathsf{BR3} \qquad \dfrac{\dfrac{}{\langle t =_\tau u \vdash t =_\tau u\rangle^{-1}}\,\mathsf{BR3} \qquad \dfrac{}{\langle \vdash t =_\tau t\rangle^{-1}}\,\mathsf{R10}}{\langle t =_\tau u \vdash u =_\tau t\rangle^{-1}}\,\mathsf{R9}}{\langle t =_\tau u, \neg(u =_\tau t) \vdash \bot\rangle^{-1}}\,\mathsf{R5}$$

$\neg\neg$

$$\dfrac{\dfrac{\dfrac{}{\langle \neg P \vdash \neg P\rangle^{-1}}\,\mathsf{BR3} \qquad \dfrac{}{\langle \neg\neg P \vdash \neg\neg P\rangle^{-1}}\,\mathsf{BR3}}{\langle \neg\neg P \vdash P\rangle^{-1}}\,\mathsf{R5} \qquad \langle \neg\neg P, P \vdash \bot\rangle^{-1}}{\langle \neg\neg P \vdash \bot\rangle^{-1}}\,\mathsf{BR4}$$

$\wedge$

$$\dfrac{\dfrac{\dfrac{}{\langle P \wedge Q \vdash P \wedge Q\rangle^{-1}}\,\mathsf{BR3}}{\langle P \wedge Q \vdash P\rangle^{-1}}\,\mathsf{R2} \qquad \dfrac{\dfrac{\dfrac{}{\langle P \wedge Q \vdash P \wedge Q\rangle^{-1}}\,\mathsf{BR3}}{\langle P \wedge Q \vdash Q\rangle^{-1}}\,\mathsf{R2'} \qquad \langle P \wedge Q, P, Q \vdash \bot\rangle^{-1}}{\langle P \wedge Q, P \vdash \bot\rangle^{-1}}\,\mathsf{BR4}}{\langle P \wedge Q \vdash \bot\rangle^{-1}}\,\mathsf{BR4}$$

$\Rightarrow$

$$\dfrac{\dfrac{\langle P \Rightarrow Q, \neg P \vdash \bot\rangle^{-1} \qquad \dfrac{}{\langle \vdash \neg\bot\rangle^{-1}}\,\mathsf{BR6}}{\langle P \Rightarrow Q \vdash P\rangle^{-1}}\,\mathsf{R5} \qquad \dfrac{\dfrac{}{\langle P \Rightarrow Q \vdash P \Rightarrow Q\rangle^{-1}}\,\mathsf{BR3} \qquad \langle P \Rightarrow Q, Q \vdash \bot\rangle^{-1}}{\langle P \Rightarrow Q \vdash Q\rangle^{-1}}\,\mathsf{MP}}{\langle P \Rightarrow Q \vdash \bot\rangle^{-1}}\,\mathsf{BR4}$$

Figure 4.3: Translation of LLproof Rules into B Proof System (Part 1)

$\neg\wedge$

$$\dfrac{\dfrac{\langle \neg(P \wedge Q), \neg P \vdash \bot\rangle^{-1} \quad \dfrac{}{\langle \vdash \neg\bot\rangle^{-1}}\,\mathsf{BR6}}{\langle \neg(P \wedge Q) \vdash P\rangle^{-1}}\,\mathsf{R5} \qquad \dfrac{\dfrac{\langle \neg(P \wedge Q), \neg Q \vdash \bot\rangle^{-1} \quad \dfrac{}{\langle \vdash \neg\bot\rangle^{-1}}\,\mathsf{BR6}}{\langle \neg(P \wedge Q) \vdash Q\rangle^{-1}}\,\mathsf{R5}}{\langle \neg(P \wedge Q) \vdash P \wedge Q\rangle^{-1}}\,\mathsf{R1} \qquad \Pi}{\langle \neg(P \wedge Q) \vdash \bot\rangle^{-1}}\,\mathsf{R5}$$

where $\Pi :=$ $\dfrac{}{\langle \neg(P \wedge Q) \vdash \neg(P \wedge Q)\rangle^{-1}}\,\mathsf{BR3}$

$\neg\Rightarrow$

$$\dfrac{\dfrac{\dfrac{\langle \neg(P \Rightarrow Q), P, \neg Q \vdash \bot\rangle^{-1} \quad \dfrac{}{\langle \vdash \neg\bot\rangle^{-1}}\,\mathsf{BR6}}{\langle \neg(P \Rightarrow Q), P \vdash Q\rangle^{-1}}\,\mathsf{R5}}{\langle \neg(P \Rightarrow Q) \vdash P \Rightarrow Q\rangle^{-1}}\,\mathsf{R3} \qquad \dfrac{}{\langle \neg(P \Rightarrow Q) \vdash \neg(P \Rightarrow Q)\rangle^{-1}}\,\mathsf{BR3}}{\langle \neg(P \Rightarrow Q) \vdash \bot\rangle^{-1}}\,\mathsf{R5}$$

$\neg\forall$

$$\dfrac{\dfrac{\dfrac{\langle \neg\forall x : \tau.\ P(x), \neg P(c) \vdash \bot\rangle^{-1} \quad \dfrac{}{\langle \vdash \neg\bot\rangle^{-1}}\,\mathsf{BR6}}{\langle \neg\forall x : \tau.\ P(x) \vdash P(c)\rangle^{-1}}\,\mathsf{R5}}{\langle \neg\forall x : \tau.\ P(x) \vdash \forall x : \tau.\ P(x)\rangle^{-1}}\,\mathsf{R7} \qquad \dfrac{}{\langle \neg\forall x : \tau.\ P(x) \vdash \neg\forall x : \tau.\ P(x)\rangle^{-1}}\,\mathsf{BR3}}{\langle \neg\forall x : \tau.\ P(x) \vdash \bot\rangle^{-1}}\,\mathsf{R5}$$

$\forall$

$$\dfrac{\dfrac{\dfrac{}{\langle \forall x : \tau.\ P(x) \vdash \forall x : \tau.\ P(x)\rangle^{-1}}\,\mathsf{BR3}}{\langle \forall x : \tau.\ P(x) \vdash P(t)\rangle^{-1}}\,\mathsf{R8'} \qquad \langle \forall x : \tau.\ P(x), P(t) \vdash \bot\rangle^{-1}}{\langle \forall x : \tau.\ P(x) \vdash \bot\rangle^{-1}}\,\mathsf{BR4}$$

Subst

$$\dfrac{\dfrac{\langle P(t), \neg(t =_\tau u) \vdash \bot\rangle^{-1} \quad \dfrac{}{\langle \vdash \neg\bot\rangle^{-1}}\,\mathsf{BR6}}{\langle P(t) \vdash t =_\tau u\rangle^{-1}}\,\mathsf{R5} \qquad \dfrac{}{\langle P(t) \vdash P(t)\rangle^{-1}}\,\mathsf{BR3}}{\langle P(t) \vdash P(u)\rangle^{-1}}\,\mathsf{R9} \qquad \dfrac{\langle P(t), P(u) \vdash \bot\rangle^{-1}}{\langle P(t) \vdash \bot\rangle^{-1}}\,\mathsf{BR4}$$

Figure 4.4: Translation of LLproof Rules into B Proof System (Part 2)

formulæ.

We apply the idea of permutation of inference rules of Kleene [Kleene 1951]. It is possible because type variable quantification is prenex, thus we can permute these inference nodes to shift them down to the root. As an example, we give below the permutation for a $\neg\wedge$ node when both premises are $\forall_{\mathsf{type}}$ nodes. The generalization to all LLproof inference rules is straightforward.

The proof node (we omit to repeat the contexts):

$$\dfrac{\dfrac{\dfrac{\Pi_1}{\neg Q, P(\tau_1) \vdash_{\mathsf{LL}} \bot}}{\forall\alpha.\ P(\alpha), \neg Q, \vdash_{\mathsf{LL}} \bot}\,\forall_{\mathsf{type}} \quad \dfrac{\dfrac{\Pi_2}{\neg R, P(\tau_2) \vdash_{\mathsf{LL}} \bot}}{\forall\alpha.\ P(\alpha), \neg R \vdash_{\mathsf{LL}} \bot}\,\forall_{\mathsf{type}}}{\Gamma, \forall\alpha.\ P(\alpha), \neg(Q \wedge R) \vdash_{\mathsf{LL}} \bot}\,\neg\wedge$$

is transformed into:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_1}{P(\tau_1), P(\tau_2), \neg Q \vdash_{\mathsf{LL}} \bot} \quad \dfrac{\Pi_2}{P(\tau_1), P(\tau_2), \neg R \vdash_{\mathsf{LL}} \bot}}{\neg(Q \wedge R), P(\tau_1), P(\tau_2) \vdash_{\mathsf{LL}} \bot}\,\neg\wedge}{\forall\alpha.\ P(\alpha), \neg(Q \wedge R), P(\tau_1) \vdash_{\mathsf{LL}} \bot}\,\forall_{\mathsf{type}}}{\Gamma, \forall\alpha.\ P(\alpha), \neg(Q \wedge R) \vdash_{\mathsf{LL}} \bot}\,\forall_{\mathsf{type}}$$

3. As all the type instantiations are done at the root of Π_{Kleene}, we take the subproof Π_{mono} of Π_{Kleene}, where we removed all the $\forall_{\mathsf{type}}$ nodes at root and the remaining polymorphic formulæ at each nodes.

   So, if we denote $\langle \Gamma^{\star s} \rangle^{mono}$ the set of monomorphic instances of formulæ of $\langle \Gamma^{\star s} \rangle$, we have:

$$\dfrac{\Pi_{mono}}{\langle \Gamma^{\star s} \rangle^{mono}, \neg \langle P^{\star s} \rangle \vdash_{\mathsf{LL}} \bot}$$

   It should be noted that $\langle \Gamma^{\star s} \rangle^{mono}$ may contained several monomorphic instances of the same polymorphic formula, see the permutation example above.

4. Using the translation $\langle\text{-}\rangle^{\text{-}1}$ on monomorphic instances of the translation of the B axioms leads to erase type information inside formulæ. So, we get back the same B axiom than at the beginning, except for quantification over a list of variables (see lemma 4.2.1 and corollary 4.2.2). It should be noted that we obtain actually the

universal closure of the B axioms (not anymore the axiom schemata) as presented in Sec. 2.2.5 and without type annotations, which are no longer needed.

We obtain:

$$\frac{\langle \Pi_{mono} \rangle^{-1}}{\langle \langle \Gamma^{\star s} \rangle^{mono}, \neg \langle P^{\star s} \rangle \vdash_{\mathsf{LL}} \bot \rangle^{-1}}$$

where $\langle \Pi_{mono} \rangle^{-1}$ is a B proof.

It leads to:

$$\frac{\langle \Pi_{mono} \rangle^{-1}}{\langle \langle \Gamma^{\star s} \rangle^{mono} \rangle^{-1}, \neg \langle \langle P^{\star s} \rangle \rangle^{-1} \vdash_{\mathsf{B}} \bot}$$

Then we have (see lemma 4.2.1 and corollary 4.2.2):

$$\frac{\langle \Pi_{mono} \rangle^{-1}}{\Gamma^s, \neg P^s \vdash_{\mathsf{B}} \bot}$$

Finally, we have the proof:

$$\dfrac{\dfrac{\langle \Pi_{mono} \rangle^{-1}}{\Gamma, \neg P \vdash_{\mathsf{B}} \bot} \quad \dfrac{}{\Gamma, \neg P \vdash_{\mathsf{B}} \neg \bot}\,\mathsf{BR6}}{\Gamma \vdash_{\mathsf{B}} P}\,\mathsf{R5}$$

where we eliminate the skolem symbols from the proof by applying the procedure of Prop. 2.3.2.

4.2.4 Example of Proof Translation

The LLproof proof of the example of Sec. 4.1.2.5 is already quite large, the resulting B proof being larger, we cannot present it here. Instead, we present the proof translation of the example of Sec. 1.2.3.1.

Given a set u, we want to prove the goal P:

$$u \in \mathbb{P}(u)$$

By applying the annotation procedure of Sec. 2.2, we obtain $P^\star$:

$$u^{\mathbb{P}(u)} \in \mathbb{P}(u)$$

We also need the axiom SET2*, presented in its annotated and explicitly quantified form:

$$\forall s^{\mathbb{P}(u)} \cdot (\forall t^{\mathbb{P}(u)} \cdot (s \in \mathbb{P}(t) \Leftrightarrow \forall x^{u} \cdot (x \in s \Rightarrow x \in t)))$$

The former leads to the type declarations:

$$\begin{array}{l} T :: 0 \\ u : \mathsf{set}(T) \end{array}$$

and the monomorphic Poly-FOL formula $\langle P^\star \rangle$:

$$u \in_{\mathsf{set}(T)} \mathbb{P}_T(u)$$

And the latter leads to the polymorphic Poly-FOL formula $\langle$SET2$^\star\rangle$:

$$\forall \alpha.\ \forall s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha).\ s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) \Leftrightarrow (\forall x : \alpha.\ x \in_\alpha s \Rightarrow x \in_\alpha t)$$

We obtain the LLproof proof, given that $A \Leftrightarrow B$ is considered as a notation for $(A \Rightarrow B) \wedge (B \Rightarrow A)$, and that $x \notin s$ is an abbreviation for $\neg(x \in s)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{c \in_T u, c \notin_T u \vdash_{\mathsf{LL}} \bot}\,\mathrm{Ax}}{\neg(c \in_T u \Rightarrow c \in_T u) \vdash_{\mathsf{LL}} \bot}\,\neg\Rightarrow}{\neg(\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \vdash_{\mathsf{LL}} \bot}\,\neg\forall \quad \dfrac{}{u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \vdash_{\mathsf{LL}} \bot}\,\mathrm{Ax}}{(\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \Rightarrow u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \vdash_{\mathsf{LL}} \bot}\,\Rightarrow}{u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \Leftrightarrow (\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \vdash_{\mathsf{LL}} \bot}\,\wedge}{\forall t : \mathsf{set}(T).\ u \in_{\mathsf{set}(T)} \mathbb{P}_T(t) \Leftrightarrow (\forall x : T.\ x \in_T u \Rightarrow x \in_T t) \vdash_{\mathsf{LL}} \bot}\,\forall}{\forall s : \mathsf{set}(T), t : \mathsf{set}(T).\ s \in_{\mathsf{set}(T)} \mathbb{P}_T(t) \Leftrightarrow (\forall x : T.\ x \in_T s \Rightarrow x \in_T t) \vdash_{\mathsf{LL}} \bot}\,\forall}{\forall \alpha. \forall s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha).\ s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) \Leftrightarrow (\forall x : \alpha.\ x \in_\alpha s \Rightarrow x \in_\alpha t),\ u \notin_{\mathsf{set}(T)} \mathbb{P}_T(u) \vdash_{\mathsf{LL}} \bot}\,\forall_{\mathsf{type}}$$

We extract the monomorphic subproof by removing the quantification over type variables:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{c \in_T u, c \notin_T u \vdash_{\mathsf{LL}} \bot}\,\mathrm{Ax}}{\neg(c \in_T u \Rightarrow c \in_T u) \vdash_{\mathsf{LL}} \bot}\,\neg\Rightarrow}{\neg(\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \vdash_{\mathsf{LL}} \bot}\,\neg\forall \quad \dfrac{}{u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \vdash_{\mathsf{LL}} \bot}\,\mathrm{Ax}}{(\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \Rightarrow u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \vdash_{\mathsf{LL}} \bot}\,\Rightarrow}{u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \Leftrightarrow (\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \vdash_{\mathsf{LL}} \bot}\,\wedge}{\forall t : \mathsf{set}(T).\ u \in_{\mathsf{set}(T)} \mathbb{P}_T(t) \Leftrightarrow (\forall x : T.\ x \in_T u \Rightarrow x \in_T t) \vdash_{\mathsf{LL}} \bot}\,\forall}{\forall s : \mathsf{set}(T), t : \mathsf{set}(T).\ s \in_{\mathsf{set}(T)} \mathbb{P}_T(t) \Leftrightarrow (\forall x : T.\ x \in_T s \Rightarrow x \in_T t),\ u \notin_{\mathsf{set}(T)} \mathbb{P}_T(u) \vdash_{\mathsf{LL}} \bot}\,\forall$$

We can now translate the proof above to obtain a B proof Π_{B} such that, given a set u:

$$\frac{\Pi_{\mathsf{B}}}{\forall s \cdot (\forall t \cdot (s \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot (x \in s \Rightarrow x \in t)))) \vdash_{\mathsf{B}} u \in \mathbb{P}(u)}$$

It should be noted that the left-hand side of the sequent above is exactly the universal closure of the axiom SET2 (without type annotation) as presented in Sec. 2.2.5, and the right-hand side corresponds to the initial B goal of the example.

Finally, we give in the following the proof Π_{B} resulting in the translation of the monomorphic LLproof proof above, and using the derivations provided in Fig. 4.3 and in Fig. 4.4.

Since the proof is large, we present the translation of the LLproof nodes step-by-step, from the root to the leaves, keeping only the needed formulæ on the left-hand side of the sequents. This corresponds to implicit applications of the rule BR2. In addition, we use the symbol $\vdash$ instead of $\vdash_{\mathsf{B}}$, and we remove some parentheses.

The B proof starts with:

$$\dfrac{\dfrac{\Pi_0}{\forall s \cdot \forall t \cdot s \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in s \Rightarrow x \in t), u \notin \mathbb{P}(u) \vdash \bot} \quad \dfrac{}{\vdash \neg\bot}\,\mathsf{BR6}}{\forall s \cdot (\forall t \cdot (s \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot (x \in s \Rightarrow x \in t)))) \vdash_{\mathsf{B}} u \in \mathbb{P}(u)}\,\mathsf{R5}$$

The first node $\forall$:

$$\dfrac{\Lambda_1 \quad \dfrac{\Pi_1}{\forall t \cdot u \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in t), u \notin \mathbb{P}(u) \vdash \bot}}{\forall s \cdot \forall t \cdot s \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in s \Rightarrow x \in t), u \notin \mathbb{P}(u) \vdash \bot}\,\mathsf{BR4}$$
$$\Pi_0$$

where Λ_1 is:

$$\dfrac{\dfrac{}{\forall s \cdot \forall t \cdot s \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in s \Rightarrow x \in t) \vdash \forall s \cdot \forall t \cdot s \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in s \Rightarrow x \in t)}\,\mathsf{BR3}}{\forall s \cdot \forall t \cdot s \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in s \Rightarrow x \in t) \vdash \forall t \cdot u \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in t)}\,\mathsf{R8'}$$
$$\Lambda_1$$

The second node $\forall$:

$$\dfrac{\Lambda_2 \quad \dfrac{\Pi_2}{u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u), u \notin \mathbb{P}(u) \vdash \bot}}{\forall t \cdot u \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in t), u \notin \mathbb{P}(u) \vdash \bot}\,\mathsf{BR4}$$
$$\Pi_1$$

where Λ_2 is:

$$\dfrac{\dfrac{\dfrac{}{\forall t \cdot u \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in t) \vdash \forall t \cdot u \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in t)}\,\mathsf{BR3}}{\forall t \cdot u \in \mathbb{P}(t) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in t) \vdash u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u)}\,\mathsf{R8'}}{\Lambda_2}$$

The third node $\wedge$:

$$\dfrac{\Lambda_3^1 \qquad \dfrac{\Lambda_3^2 \qquad \dfrac{\Pi_3}{(\forall x \cdot x \in u \Rightarrow x \in u) \Rightarrow u \in \mathbb{P}(u), u \notin \mathbb{P}(u) \vdash \bot}}{u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u), u \notin \mathbb{P}(u) \vdash \bot}\,\mathsf{BR4}}{u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u), u \notin \mathbb{P}(u) \vdash \bot}\,\mathsf{BR4}}{\Pi_2}$$

where Λ_3^1 is:

$$\dfrac{\dfrac{\dfrac{}{u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u) \vdash u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u)}\,\mathsf{BR3}}{u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u) \vdash u \in \mathbb{P}(u) \Rightarrow (\forall x \cdot x \in u \Rightarrow x \in u)}\,\mathsf{R2}}{\Lambda_3^1}$$

and where Λ_3^2 is:

$$\dfrac{\dfrac{\dfrac{}{u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u) \vdash u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u)}\,\mathsf{BR3}}{u \in \mathbb{P}(u) \Leftrightarrow (\forall x \cdot x \in u \Rightarrow x \in u) \vdash (\forall x \cdot x \in u \Rightarrow x \in u) \Rightarrow u \in \mathbb{P}(u)}\,\mathsf{R2'}}{\Lambda_3^2}$$

The fourth node $\Rightarrow$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_5}{\neg \forall x \cdot x \in u \Rightarrow x \in u \vdash \bot}\quad \dfrac{}{\vdash \neg \bot}\,\mathsf{BR6}}{\vdash \forall x \cdot x \in u \Rightarrow x \in u}\,\mathsf{R5} \qquad \Lambda_4}{(\forall x \cdot x \in u \Rightarrow x \in u) \Rightarrow u \in \mathbb{P}(u) \vdash u \in \mathbb{P}(u)}\,\mathsf{MP} \qquad \dfrac{\Pi_4}{u \in \mathbb{P}(u), u \notin \mathbb{P}(u) \vdash \bot}}{(\forall x \cdot x \in u \Rightarrow x \in u) \Rightarrow u \in \mathbb{P}(u), u \notin \mathbb{P}(u) \vdash \bot}\,\mathsf{BR4}}{\Pi_3}$$

where Λ_4 is:

$$\dfrac{\dfrac{}{(\forall x \cdot x \in u \Rightarrow x \in u) \Rightarrow u \in \mathbb{P}(u) \vdash (\forall x \cdot x \in u \Rightarrow x \in u) \Rightarrow u \in \mathbb{P}(u)}\,\mathsf{BR3}}{\Lambda_4}$$

The first (and only) node of the right-hand branch Ax:

$$\dfrac{\dfrac{\dfrac{}{u \in \mathbb{P}(u) \vdash u \in \mathbb{P}(u)}\,\mathsf{BR3} \qquad \dfrac{}{u \notin \mathbb{P}(u) \vdash u \notin \mathbb{P}(u)}\,\mathsf{BR3}}{u \in \mathbb{P}(u), u \notin \mathbb{P}(u) \vdash \bot}\,\mathsf{R5}}{\Pi_4}$$

The first node of the left-hand branch $\neg\forall$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_6}{\neg(c \in u \Rightarrow c \in u) \vdash \bot} \quad \dfrac{}{\vdash \neg\bot}\,\mathsf{BR6}}{\vdash c \in u \Rightarrow c \in u}\,\mathsf{R5}}{\vdash \forall x \cdot x \in u \Rightarrow x \in u}\,\mathsf{R7} \quad \dfrac{}{\neg\forall x \cdot x \in u \Rightarrow x \in u \vdash \neg\forall x \cdot x \in u \Rightarrow x \in u}\,\mathsf{BR3}}{\dfrac{\neg\forall x \cdot x \in u \Rightarrow x \in u \vdash \bot}{\Pi_5}}\,\mathsf{R5}$$

The second node of the left-hand branch $\neg \Rightarrow$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_7}{c \in u, c \notin u \vdash \bot} \quad \dfrac{}{\vdash \neg\bot}\,\mathsf{BR6}}{c \in u \vdash c \in u}\,\mathsf{R5}}{\vdash c \in u \Rightarrow c \in u}\,\mathsf{R3} \quad \dfrac{}{\neg(c \in u \Rightarrow c \in u) \vdash \neg(c \in u \Rightarrow c \in u)}\,\mathsf{BR3}}{\dfrac{\neg(c \in u \Rightarrow c \in u) \vdash \bot}{\Pi_6}}\,\mathsf{R5}$$

Finally, the third (and last) node of the left-hand branch Ax:

$$\dfrac{\dfrac{}{c \in u \vdash c \in u}\,\mathsf{BR3} \quad \dfrac{}{c \notin u \vdash c \notin u}\,\mathsf{BR3}}{\dfrac{c \in u, c \notin u \vdash \bot}{\Pi_7}}\,\mathsf{R5}$$

Remark This proof is larger than the one given in Sec. 1.2.3.1, and it contains some unnecessary steps. This is because we follow rigorously the translation of LLproof nodes given in Figs. 4.3 and 4.4. This method is still scalable to large LLproof proofs since the size of the generated B proofs depends linearly of the size of the LLproof proofs, the growing factor being constant.

Chapter 5

Deduction Modulo B Set Theory

This chapter presents the extension to deduction modulo theory of the typed sequent calculus LLproof, denoted LLproof$^{\equiv}$. In addition, it shows that LLproof$^{\equiv}$ is sound with respect to LLproof. At the end of this chapter, it becomes possible to build a B proof from a LLproof$^{\equiv}$ proof thanks to the results of Chap. 4.

In Sec. 5.1, we give an informal introduction to the formalism of deduction modulo theory.

In Sec. 5.2, we give some definitions about deduction modulo theory, then we present the extension of LLproof to deduction modulo theory. This section is an extension of results presented in [Dowek, Hardin, and Kirchner 2003] to polymorphically typed theories. This contribution is a collaborative work and it has been published in [Bury, Delahaye, Doligez, Halmagrand, and Hermant 2015b; Cauderlier and Halmagrand 2015].

In Sec. 5.3, we show the soundness of LLproof$^{\equiv}$ with respect to LLproof. This contribution is a personal work and it is inspired by [Dowek, Hardin, and Kirchner 2003].

In Sec. 5.4, we present the extension of the Poly-FOL version of the B set theory to deduction modulo theory. In particular, we discuss in Sec. 5.4.3 the consequences for the B derived constructs presented in Chap. 1. This contribution is a personal work.

5.1 Introduction to Deduction Modulo Theory

Deduction modulo theory [Dowek, Hardin, and Kirchner 2003] is a formalism that extends first-order logic with rewrite rules on both terms and propositions. These rewrite rules can be applied anywhere in a proof and then interleaved with deduction rules. The motivation of deduction modulo is to distinguish *deduction* and *computation* in proofs.

5.1.1 Deduction: Inference Rules and Axioms

Generally, we distinguish the concept of *logic* and the one of *theory*. We usually characterize a logic by providing a syntax – a set of symbols used to write formulæ – and a proof system – a set of inference rules giving an interpretation to these symbols and used to write proofs of the formulæ. A logic is a general formalism to write formulæ and their proofs.

A theory is generally seen as a more specific framework, dealing with a particular concept, like set theory or arithmetic. It is defined by a set of axioms – logical formulæ assumed to be true.

But this distinction may not be as clear as it seems. For instance, in LLproof, we have the following $\neg\neg$ inference rule:

$$\frac{\Gamma, P \vdash \bot}{\Gamma, \neg\neg P \vdash \bot}\ \neg\neg$$

used to erase two negation symbols when trying to prove false. But this rules may be replaced by the axiom scheme, given any formula P:

$$P \Rightarrow \neg\neg P$$

which states exactly the same property.

Actually, this axiom can be proved in LLproof – with the restriction that it requires to use the Cut rule –, being no more an *assumed* statement, but a *proved* lemma:

$$\dfrac{\dfrac{\dfrac{}{P, \neg\neg\neg P, \neg P \vdash \bot}\ \text{Ax} \quad \dfrac{}{P, \neg\neg\neg P, \neg\neg P \vdash \bot}\ \text{Ax}}{P, \neg\neg\neg P \vdash \bot}\ \text{Cut}}{\neg(P \Rightarrow \neg\neg P) \vdash \bot}\ \neg\Rightarrow$$

On the contrary, we can also interpret axioms as inference rules. This is for instance the idea of *super-deduction* , proposed by Prawitz, as explained in [Brauner, Houtmann, and Kirchner 2007]. The idea is to enlarge a proof system by adding new inference rules coming from axioms. This technique allows to improve automated proof search, in particular for set theory [Jacquel, Berkani, Delahaye, and Dubois 2012].

5.1.2 Computation: Rewrite rules

Deduction modulo theory introduces a third concept in addition to inference rules and axioms: rewrite rules.

The Poincaré principle, as stated by Barendregt and Barendsen [Barendregt and Barendsen 2002], makes a distinction between deduction and computation. Deduction may be defined using a set of inference rules and axioms, while computation consists mainly in simplification and unfolding of definitions. When dealing with axiomatic theories, keeping all axioms on the deduction side leads to inefficient proof search since the proof-search space grows with the theory. For instance, proving the following statement, in FOL:

$$\mathsf{fst}(a, a) = \mathsf{snd}(a, a)$$

where a is a constant, and fst and snd are defined by:

$$\forall x, y.\ \mathsf{fst}(x, y) = x \qquad\qquad \forall x, y.\ \mathsf{snd}(x, y) = y$$

and with the reflexivity axiom:

$$\forall x.\ x = x$$

using a usual automated theorem proving method such as Tableau, will generate useless boilerplate proof steps, whereas a simple unfolding of definitions of fst and snd directly leads to the formula $a = a$, that should be provable in one reflexivity step.

Deduction modulo theory was introduced by Dowek, Hardin and Kirchner [Dowek, Hardin, and Kirchner 2003] as a logical formalism to deal with axiomatic theories in automated theorem proving. The proposed solution is to remove computational arguments from proofs by reasoning modulo a decidable congruence relation $\equiv$ on formulæ. Such a congruence may be generated by a confluent and terminating system of rewrite rules.

In our example, the two definitions may be replaced by the rewrite rules:

$$\mathsf{fst}(x, y) \longrightarrow x \qquad\qquad \mathsf{snd}(x, y) \longrightarrow y$$

And we obtain the following equivalence between propositions:

$$(\mathsf{fst}(a, a) = \mathsf{snd}(a, a)) \;\equiv\; (a = a)$$

5.1.2.1 Example in Set Theory

Deduction modulo theory strongly reduces the size of proofs in general. For instance in untyped set theory, proving the following statement:

$$a \subseteq a$$

where a is a constant set, and given the axiom defining the subset predicate:

$$\forall s, t.\ s \subseteq t \Leftrightarrow \forall x.\ x \in s \Rightarrow x \in t$$

will generate the many-step proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{c \in a, \neg(c \in a) \vdash \bot}\,\mathrm{Ax}}{\neg(c \in a \Rightarrow c \in a) \vdash \bot}\,\neg\Rightarrow}{\neg\forall x.\ x \in a \Rightarrow x \in a \vdash \bot}\,\neg\forall \qquad \dfrac{}{a \subseteq a, \neg(a \subseteq a) \vdash \bot}\,\mathrm{Ax}}{a \subseteq a \Leftrightarrow \forall x.\ x \in a \Rightarrow x \in a, \neg(a \subseteq a) \vdash \bot}\,\Leftrightarrow}{\forall t.\ a \subseteq t \Leftrightarrow \forall x.\ x \in a \Rightarrow x \in t, \neg(a \subseteq a) \vdash \bot}\,\forall}{\forall s, t.\ s \subseteq t \Leftrightarrow \forall x.\ x \in s \Rightarrow x \in t, \neg(a \subseteq a) \vdash \bot}\,\forall$$

The definition may be replaced by the rewrite rule:

$$s \subseteq t \longrightarrow \forall x.\ x \in s \Rightarrow x \in t$$

leading to the shorter non-branching proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{c \in a, \neg(c \in a) \vdash \bot}\,\mathrm{Ax}}{\neg(c \in a \Rightarrow c \in a) \vdash \bot}\,\neg\Rightarrow}{\neg\forall x.\ x \in a \Rightarrow x \in a \vdash \bot}\,\neg\forall}{\neg(a \subseteq a) \vdash \bot}\,\mathrm{Rewriting}\ \subseteq$$

5.2 LLproof$^{\equiv}$: Extension of LLproof to Deduction Modulo Theory

5.2.1 Deduction Modulo Theory

Deduction modulo theory [Dowek, Hardin, and Kirchner 2003] reasons over equivalence classes of formulæ under a congruence generated by rewrite rules. Compared to [Dowek, Hardin, and Kirchner 2003], we extend deduction modulo theory to Poly-FOL. The language is that of Sec. 3.2. In the following, we introduce the definitions of term and proposition rewrite rules, and the notion of $\mathcal{RE}$-rewriting.

Rewrite System

A *term rewrite rule* is a pair of terms l and r together with a local context Γ_{L} and denoted $l \longrightarrow_{\Gamma_{\mathrm{L}}} r$. In addition, we impose that l is not a variable and that we have $\mathrm{FV}_{\mathrm{T}}(r) \subseteq \mathrm{FV}_{\mathrm{T}}(l) \subseteq \Gamma_{\mathrm{L}}$ and $\mathrm{FV}(r) \subseteq \mathrm{FV}(l) \subseteq \Gamma_{\mathrm{L}}$.

A *proposition rewrite rule* is a pair of formulæ l and r together with a local context Γ_{L} and denoted $l \longrightarrow_{\Gamma_{\mathrm{L}}} r$. In addition, we impose that l is an atomic formula and that we have $\mathrm{FV}_{\mathrm{T}}(r) \subseteq \mathrm{FV}_{\mathrm{T}}(l) \subseteq \Gamma_{\mathrm{L}}$ and $\mathrm{FV}(r) \subseteq \mathrm{FV}(l) \subseteq \Gamma_{\mathrm{L}}$.

Rewrite rules are said to be well-formed in a global context Γ_{G} if l and r have the same type in $\Gamma_{\mathrm{G}}; \Gamma_{\mathrm{L}}$. We extend the context well-formedness rules of Fig. 3.1 by adding the following rule for rewrite rules:

$$\frac{\Gamma_{\mathrm{G}}; \Gamma_{\mathrm{L}} \vdash l : \tau \qquad \Gamma_{\mathrm{G}}; \Gamma_{\mathrm{L}} \vdash r : \tau \qquad \begin{array}{l} \mathrm{FV}_{\mathrm{T}}(r) \subseteq \mathrm{FV}_{\mathrm{T}}(l) \subseteq \Gamma_{\mathrm{L}} \\ \mathrm{FV}(r) \subseteq \mathrm{FV}(l) \subseteq \Gamma_{\mathrm{L}} \end{array}}{\mathsf{wf}(\Gamma_{\mathrm{G}}, l \longrightarrow_{\Gamma_{\mathrm{L}}} r; \varnothing)} \mathrm{WF}_7$$

A *rewrite system*, denoted $\mathcal{RE}$, consists of the union of a set of proposition rewrite rules – denoted $\mathcal{R}$ – and a set of term rewrite rules – denoted $\mathcal{E}$. It is well-formed in a global context Γ_{G} if all the rewrite rules are well-formed in Γ_{G}.

$\mathcal{RE}$-Rewriting

Given a global context Γ_{G} and a rewrite system $\mathcal{RE}$ that is well-formed in Γ_{G}, a formula

φ is said to $\mathcal{RE}$-rewrite to φ', denoted $\varphi \longrightarrow_{\mathcal{RE}} \varphi'$ if $\varphi_{|\omega} = (l\rho)\sigma$ and $\varphi' = \varphi[(r\rho)\sigma]_\omega$, for some rewrite rule $l \longrightarrow_{\Gamma_\mathrm{L}} r \in \mathcal{RE}$, some occurrence ω in φ, some type substitution ρ, some term substitution σ and where $\varphi_{|\omega}$ is the expression at occurrence ω in φ, and $\varphi[(r\rho)\sigma]_\omega$ is the expression φ where $\varphi_{|\omega}$ has been replaced by $(r\rho)\sigma$.

The reflexive-transitive closure of the relation $\longrightarrow_{\mathcal{RE}}$ is written $\longrightarrow^\star_{\mathcal{RE}}$. The relation $\equiv_{\mathcal{RE}}$ is the congruence generated by $\mathcal{RE}$.

The relation $\equiv_{\mathcal{RE}}$ is not decidable in general, but this is in particular the case when $\longrightarrow_{\mathcal{RE}}$ is confluent and (weakly) terminating [Dowek, Hardin, and Kirchner 2003].

In the following, when the sets of rewrite rules are clear from the context, we may denote by $\equiv$ the congruence relation $\equiv_{\mathcal{RE}}$.

5.2.2 Extension of LLproof to Deduction Modulo Theory

Given a global typing context Γ_G and a rewrite system $\mathcal{RE}$ well-formed in Γ_G, extending LLproof to deduction modulo theory consists in adding to the proof search rules of Fig. 3.3 and Fig. 3.4, the following conversion rule:

$$\frac{\Gamma, Q \vdash \bot}{\Gamma, P \vdash \bot}\,\mathrm{conv}_{P \equiv_{\mathcal{RE}} Q}$$

The resulting proof system is called LLproof modulo, and denoted $\mathsf{LLproof}^{\equiv}$.

5.2.3 Example

If we go back to our example of Sec. 4.2.4, given the axiom:

$$\forall\alpha.\ \forall s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha).\ s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) \Leftrightarrow \forall x : \alpha.\ (x \in_\alpha s \Rightarrow x \in_\alpha t)$$

and the goal:

$$u \in_{\mathsf{set}(T)} \mathbb{P}_T(u)$$

for a given sort T and a constant $u : \mathsf{set}(T)$, we obtained the following LLproof proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{c \in_T u, c \notin_T u \vdash_{\mathsf{LL}} \bot}\,\mathrm{Ax}}{\neg(c \in_T u \Rightarrow c \in_T u) \vdash_{\mathsf{LL}} \bot}\,\neg\Rightarrow}{\neg(\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \vdash_{\mathsf{LL}} \bot}\,\neg\forall \quad \dfrac{}{u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \vdash_{\mathsf{LL}} \bot}\,\mathrm{Ax}}{(\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \Rightarrow u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \vdash_{\mathsf{LL}} \bot}\,\Rightarrow}{u \in_{\mathsf{set}(T)} \mathbb{P}_T(u) \Leftrightarrow (\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \vdash_{\mathsf{LL}} \bot}\,\wedge}{\forall t : \mathsf{set}(T).\ u \in_{\mathsf{set}(T)} \mathbb{P}_T(t) \Leftrightarrow (\forall x : T.\ x \in_T u \Rightarrow x \in_T t) \vdash_{\mathsf{LL}} \bot}\,\forall}{\forall s : \mathsf{set}(T), t : \mathsf{set}(T).\ s \in_{\mathsf{set}(T)} \mathbb{P}_T(t) \Leftrightarrow (\forall x : T.\ x \in_T s \Rightarrow x \in_T t) \vdash_{\mathsf{LL}} \bot}\,\forall}{\forall \alpha. \forall s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha).\ s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) \Leftrightarrow (\forall x : \alpha.\ x \in_\alpha s \Rightarrow x \in_\alpha t),\ u \notin_{\mathsf{set}(T)} \mathbb{P}_T(u) \vdash_{\mathsf{LL}} \bot}\,\forall_{\mathsf{type}}$$

In deduction modulo, the previous axiom can be replaced by the rewrite rule:

$$s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) \longrightarrow_{(\alpha:\mathsf{Type}, s:\mathsf{set}(\alpha), t:\mathsf{set}(\alpha))} \forall x : \alpha.\ (x \in_\alpha s \Rightarrow x \in_\alpha t)$$

Therefore, the proof in $\mathsf{LLproof}^{\equiv}$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{c \in_T u, c \notin_T u \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\mathrm{Ax}}{\neg(c \in_T u \Rightarrow c \in_T u) \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\neg\Rightarrow}{\neg(\forall x : T.\ x \in_T u \Rightarrow x \in_T u) \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\neg\forall}{\neg(u \in_{\mathsf{set}(T)} \mathbb{P}_T(u)) \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\mathrm{conv}$$

5.3 Soundness of $\mathsf{LLproof}^{\equiv}$ with Respect to LLproof

In this section, we show that the proof system $\mathsf{LLproof}^{\equiv}$ is sound with respect to LLproof, *i.e* that if we have a $\mathsf{LLproof}^{\equiv}$ proof of a Poly-FOL statement, then we can build a LLproof proof of a certain associated statement.

5.3.1 Generating Theories from Rewrite Systems

Given a set of proposition rewrite rules $\mathcal{R}$ and a set of term rewrite rules $\mathcal{E}$, we apply the following procedure for all rewrite rules.

If we have a propositional rewrite rule of the form:

$$P \longrightarrow_{\Gamma_{\mathrm{L}}} Q$$

where P is an atomic formula, Q is any formula, and $\mathrm{FV_T}(Q) \subseteq \mathrm{FV_T}(P) \subseteq \Gamma_{\mathrm{L}}$ and $\mathrm{FV}(Q) \subseteq \mathrm{FV}(P) \subseteq \Gamma_{\mathrm{L}}$; we generate the axiom:

$$\underset{\alpha \in \mathrm{FV_T}(P)}{\forall} \vec{\alpha}. \underset{x \in \mathrm{FV}(P)}{\forall} \vec{x}.\ P(\vec{\alpha}, \vec{x}) \Leftrightarrow Q(|\vec{\alpha}|, |\vec{x}|)$$

where $|\vec{\alpha}|$ and $|\vec{x}|$ denote respectively the sets $\mathrm{FV_T}(Q)$ and $\mathrm{FV}(Q)$ and satisfy $|\vec{\alpha}| \subseteq \vec{\alpha}$ and $|\vec{x}| \subseteq \vec{x}$.

If we have a term rewrite rule of the form:

$$s \longrightarrow_{\Gamma_\mathrm{L}} t$$

where s is a term that is not a variable, t is any term, and $\mathrm{FV_T}(t) \subseteq \mathrm{FV_T}(s) \subseteq \Gamma_\mathrm{L}$ and $\mathrm{FV}(t) \subseteq \mathrm{FV}(s) \subseteq \Gamma_\mathrm{L}$; we generate the axiom:

$$\underset{\alpha \in \mathrm{FV_T}(s)}{\forall} \vec{\alpha}. \underset{x \in \mathrm{FV}(s)}{\forall} \vec{x}.\ s(\vec{\alpha}, \vec{x}) =_\tau t(|\vec{\alpha}|, |\vec{x}|)$$

where τ is the type of s and t, and where $|\vec{\alpha}|$ and $|\vec{x}|$ denote respectively the sets $\mathrm{FV_T}(t)$ and $\mathrm{FV}(t)$ and satisfy $|\vec{\alpha}| \subseteq \vec{\alpha}$ and $|\vec{x}| \subseteq \vec{x}$.

At the end, we have generated a set of axioms, which is the generated theory from the rewrite system $\mathcal{RE}$. We denote this resulting theory $\mathcal{T}$.

5.3.2 Soundness

We show that the conversion rule introduced in Sec. 5.2.2 is admissible, in the sense that we can produce a LLproof derivation of the conversion rule. More precisely, we show that, given a rewrite system $\mathcal{RE}$, the proof system $\mathsf{LLproof}^{\equiv}$ is sound with respect to LLproof with the theory $\mathcal{T}$ generated by $\mathcal{RE}$. We have the following theorem:

Theorem 5.3.1 (Soundness of $\mathsf{LLproof}^{\equiv}$)
Given a rewrite system $\mathcal{RE}$ and a set of formulæ Γ, if the sequent:

$$\Gamma \vdash_{\mathsf{LL}^{\equiv}} \bot$$

is provable and Π is a $\mathsf{LLproof}^{\equiv}$ proof of this sequent, then there exists a LLproof proof Π' of the sequent:

$$\mathcal{T}, \Gamma \vdash_{\mathsf{LL}} \bot$$

where $\mathcal{T}$ is the theory generated from $\mathcal{RE}$.

In order to prove this conservativity theorem, we will incrementally consider the different cases of rewriting. Firstly, we consider in Sec. 5.3.3 the case of one step rewriting, *i.e.* the

application of one proposition rewrite rule and for the head symbol. Then, we extend in Sec. 5.3.4 the one step and propositional rewriting to any subformula. We continue in Sec. 5.3.5 with the case of one step rewriting for terms. Finally, we conclude in Sec. 5.3.6 with the final case of multiple step rewriting for both terms and propositions. Once the admissibility of the full-fledged conv rule is proved, the soundness theorem 5.3.1 follows.

5.3.3 One Step, Propositional and Head Rewriting

Lemma 5.3.2

Given a set of proposition rewrite rules $\mathcal{R}$ and the theory $\mathcal{T}$ generated from $\mathcal{R}$, if P' and Q' are two formulæ that are one-step and head convertible with respect to $\mathcal{R}$ (denoted $P' \widehat{\equiv}^1_{\mathcal{R}} Q'$), we have:

$$\textit{If} \quad \frac{\Gamma, Q' \vdash_{\mathsf{LL}^{\equiv}} \bot}{\Gamma, P' \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\mathrm{conv}_{P' \widehat{\equiv}^1_{\mathcal{R}} Q'} \qquad \textit{Then} \quad \frac{\mathcal{T}, \Gamma, Q' \vdash_{\mathsf{LL}} \bot}{\mathcal{T}, \Gamma, P' \vdash_{\mathsf{LL}} \bot}$$

Proof

If P' and Q' are one-step and head convertible, then there exists a rewrite rule in $\mathcal{R}$ of the shape:

$$P \longrightarrow_{\Gamma_{\mathrm{L}}} Q \qquad \text{or} \qquad Q \longrightarrow_{\Gamma_{\mathrm{L}}} P$$

such that $P' \longrightarrow_{\mathcal{R}} Q'$ or $Q' \longrightarrow_{\mathcal{R}} P'$. Without loss of generality, we consider the case $P \longrightarrow_{\Gamma_{\mathrm{L}}} Q$. The theory $\mathcal{T}$ generated from $\mathcal{R}$ contains the axiom (see Sec. 5.3.1):

$$\underset{\alpha \in \mathrm{FV_T}(P)}{\forall} \vec{\alpha}. \underset{x \in \mathrm{FV}(P)}{\forall} \vec{x}.\ P(\vec{\alpha}, \vec{x}) \Leftrightarrow Q(|\vec{\alpha}|, |\vec{x}|)$$

It should be noted that, when writing $P' \longrightarrow_{\mathcal{R}} Q'$, we speak about an instance of the rewrite rule $P \longrightarrow_{\Gamma_{\mathrm{L}}} Q$, where P and Q have been instantiated with some type parameters $\vec{\tau}$ and term parameters $\vec{t}$. So, if we denote respectively $\rho = [\vec{\alpha}/\vec{\tau}]$ and $\sigma = [\vec{x}/\vec{t}]$ the type substitution and term substitution, then we have:

$$P' = (P\rho)\sigma = P(\vec{\tau}, \vec{t}) \quad \text{and} \quad Q' = (Q\rho)\sigma = Q(|\vec{\tau}|, |\vec{t}|)$$

We have to following derivation:

$$\dfrac{\dfrac{\dfrac{}{\mathcal{T},\Gamma,\neg P(\vec{\tau},\vec{t}),\neg Q(|\vec{\tau}|,|\vec{t}|),P' \vdash_{\mathsf{LL}} \bot}\,\mathrm{Ax} \quad \dfrac{\mathcal{T},\Gamma,Q',\vdash_{\mathsf{LL}} \bot}{\mathcal{T},\Gamma,P(\vec{\tau},\vec{t}),Q(|\vec{\tau}|,|\vec{t}|),P' \vdash_{\mathsf{LL}} \bot}}{\mathcal{T},\Gamma,P(\vec{\tau},\vec{t}) \Leftrightarrow Q(|\vec{\tau}|,|\vec{t}|),P' \vdash_{\mathsf{LL}} \bot}\Leftrightarrow}{\dfrac{\mathcal{T},\Gamma,\forall\vec{x}.\ P(\vec{\tau},\vec{x}) \Leftrightarrow Q(|\vec{\tau}|,|\vec{x}|),P' \vdash_{\mathsf{LL}} \bot}{\mathcal{T},\Gamma,\forall\vec{\alpha}.\forall\vec{x}.\ P(\vec{\alpha},\vec{x}) \Leftrightarrow Q(|\vec{\alpha}|,|\vec{x}|),P' \vdash_{\mathsf{LL}} \bot}\forall_{\mathsf{type}}\times m}\forall\times n$$

5.3.4 One Step, Propositional and Deep Rewriting

We generalize the previous lemma to rewriting of subformulæ. In the following, if P is a subformula of F, we denote F by $F[P]$, and the replacement of P by Q in F by $F[Q]$.

Lemma 5.3.3

Given a set of proposition rewrite rules $\mathcal{R}$ and the theory $\mathcal{T}$ generated from $\mathcal{R}$, if P and Q are two formulæ that are one-step and head convertible with respect to $\mathcal{R}$, we have:

$$\textit{If} \quad \frac{\Gamma,F[Q] \vdash_{\mathsf{LL}^{\equiv}} \bot}{\Gamma,F[P] \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\mathrm{conv}_{P\widehat{\equiv}^1_{\mathcal{R}}Q} \qquad \textit{Then} \quad \frac{\mathcal{T},\Gamma,F[Q] \vdash_{\mathsf{LL}} \bot}{\mathcal{T},\Gamma,F[P] \vdash_{\mathsf{LL}} \bot}$$

Proof

First, by applying the Cut rule, we have:

$$\frac{\mathcal{T},\Gamma,F[P],F[Q] \vdash_{\mathsf{LL}} \bot \qquad \mathcal{T},\Gamma,F[P],\neg F[Q] \vdash_{\mathsf{LL}} \bot}{\mathcal{T},\Gamma,F[P] \vdash_{\mathsf{LL}} \bot}\,\mathrm{Cut}$$

The left branch corresponds to the premise of the rule:

$$\frac{\mathcal{T},\Gamma,F[Q] \vdash_{\mathsf{LL}} \bot}{\mathcal{T},\Gamma,F[P] \vdash_{\mathsf{LL}} \bot}$$

So, we have to show that the right-hand branch:

$$\mathcal{T},\Gamma,F[P],\neg F[Q] \vdash_{\mathsf{LL}} \bot$$

can be closed. We perform a proof by induction on the structure of F. The base case is:

$$\mathcal{T},\Gamma,P,\neg Q \vdash_{\mathsf{LL}} \bot$$

and is true since it is a direct consequence of the previous lemma:

$$\frac{\dfrac{}{\mathcal{T},\Gamma,Q,\neg Q \vdash_{\mathsf{LL}} \bot}\,\mathrm{Ax}}{\mathcal{T},\Gamma,P,\neg Q \vdash_{\mathsf{LL}} \bot}$$

The induction hypothesis tells us that the following sequent can be closed:

$$\frac{\text{IH}}{\mathcal{T}, \Gamma, F'[P], \neg F'[Q] \vdash_{\mathsf{LL}} \bot}$$

for F' strict subformulæ of F.

To lighten the presentation, we may sometime omit some irrelevant parameters like contexts Γ.

1. $F[P] := \neg F'[P]$:

$$\dfrac{\dfrac{\text{IH}}{\mathcal{T}, \Gamma, \neg F'[P], F'[Q] \vdash_{\mathsf{LL}} \bot}}{\mathcal{T}, \Gamma, \neg F'[P], \neg\neg F'[Q] \vdash_{\mathsf{LL}} \bot}\,\neg\neg$$

Since $P \mathrel{\widehat{\equiv}}^1_{\mathcal{R}} Q$ implies $Q \mathrel{\widehat{\equiv}}^1_{\mathcal{R}} P$, we can apply directly the induction hypothesis in this case.

2. $F[P] := F_1[P] \wedge F_2$:

$$\dfrac{\dfrac{\dfrac{\text{IH}}{\mathcal{T}, F_1[P], F_2, \neg F_1[Q] \vdash_{\mathsf{LL}} \bot} \quad \dfrac{}{\mathcal{T}, F_1[P], F_2, \neg F_2 \vdash_{\mathsf{LL}} \bot}\,\text{Ax}}{\mathcal{T}, F_1[P], F_2, \neg(F_1[Q] \wedge F_2) \vdash_{\mathsf{LL}} \bot}\,\neg\wedge}{\mathcal{T}, F_1[P] \wedge F_2, \neg(F_1[Q] \wedge F_2) \vdash_{\mathsf{LL}} \bot}\,\wedge$$

3. $F[P] := F_1[P] \Rightarrow F_2$:

$$\dfrac{\dfrac{\dfrac{\text{IH}}{\mathcal{T}, \neg F_1[P], F_1[Q], \neg F_2 \vdash_{\mathsf{LL}} \bot} \quad \dfrac{}{\mathcal{T}, F_2, F_1[Q], \neg F_2 \vdash_{\mathsf{LL}} \bot}\,\text{Ax}}{\mathcal{T}, F_1[P] \Rightarrow F_2, F_1[Q], \neg F_2 \vdash_{\mathsf{LL}} \bot}\,\Rightarrow}{\mathcal{T}, F_1[P] \Rightarrow F_2, \neg(F_1[Q] \Rightarrow F_2) \vdash_{\mathsf{LL}} \bot}\,\neg\Rightarrow$$

4. We do not present the cases for $\vee$, $\Leftrightarrow$ and all the cases where P occurs in F_2 instead of F_1, since they are very close to the previous ones.

5. $F[P] := \forall x : \tau.\ F'[P]$

$$\dfrac{\dfrac{\dfrac{\text{IH}}{\mathcal{T}, F'[P(c)], \neg F'[Q(c)] \vdash_{\mathsf{LL}} \bot}}{\mathcal{T}, \forall x : \tau.\ F'[P(x)], \neg F'[Q(c)] \vdash_{\mathsf{LL}} \bot}\,\forall}{\mathcal{T}, \forall x : \tau.\ F'[P(x)], \neg\forall x : \tau.\ F'[Q(x)] \vdash_{\mathsf{LL}} \bot}\,\neg\forall$$

Since if $P(x) \mathrel{\widehat{\equiv}}^1_{\mathcal{R}} Q(x)$, then $P(c) \mathrel{\widehat{\equiv}}^1_{\mathcal{R}} Q(c)$.

6. We do not present the case for $\exists$ which is very close to the previous one.

5.3.5 One Step Term Rewriting

Lemma 5.3.4

Given a set of term rewrite rules $\mathcal{E}$ and the theory $\mathcal{T}$ generated from $\mathcal{E}$, if s' and t' are two terms that are one-step convertible with respect to $\mathcal{E}$ (denoted $s' \equiv^1_{\mathcal{E}} t'$), we have:

$$\textit{If} \quad \frac{\Gamma, F[t'] \vdash_{\mathsf{LL}^{\equiv}} \bot}{\Gamma, F[s'] \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\mathrm{conv}_{s' \equiv^1_{\mathcal{E}} t'} \qquad \textit{Then} \quad \frac{\mathcal{T}, \Gamma, F[t'] \vdash_{\mathsf{LL}} \bot}{\mathcal{T}, \Gamma, F[s'] \vdash_{\mathsf{LL}} \bot}$$

Proof

If s' and t' are one-step convertible, then there exists a rewrite rule in $\mathcal{E}$ of the shape:

$$s \longrightarrow_{\Gamma_{\mathrm{L}}} t \qquad \text{or} \qquad t \longrightarrow_{\Gamma_{\mathrm{L}}} s$$

such that $s' \longrightarrow_{\mathcal{E}} t'$ or $t' \longrightarrow_{\mathcal{E}} s'$. Once again, without loss of generality, we consider the case $s \longrightarrow_{\Gamma_{\mathrm{L}}} t$. The theory $\mathcal{T}$ generated from $\mathcal{R}$ contains the axiom (see Sec. 5.3.1):

$$\underset{\alpha \in \mathrm{FV_T}(s)}{\forall} \vec{\alpha}. \underset{x \in \mathrm{FV}(s)}{\forall} \vec{x}.\ s(\vec{\alpha}, \vec{x}) =_\tau t(|\vec{\alpha}|, |\vec{x}|)$$

If we denote respectively $\rho = [\vec{\alpha}/\vec{\tau}]$ and $\sigma = [\vec{x}/\vec{u}]$ the type substitution and the term substitution, we have:

$$s' = (s\rho)\sigma = s(\vec{\tau}, \vec{u}) \quad \text{and} \quad t' = (t\rho)\sigma = t(|\vec{\tau}|, |\vec{u}|)$$

We have the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{s(\vec{\tau}, \vec{u}) =_\tau t(|\vec{\tau}|, |\vec{u}|), s' \neq_\tau t' \vdash_{\mathsf{LL}} \bot}\,\mathrm{Ax}}{\forall \vec{x}.\ s(\vec{\tau}, \vec{x}) =_\tau t(|\vec{\tau}|, |\vec{x}|), s' \neq_\tau t' \vdash_{\mathsf{LL}} \bot}\,\forall \times n}{\forall \vec{\alpha}.\ \forall \vec{x}.\ s(\vec{\alpha}, \vec{x}) =_\tau t(|\vec{\alpha}|, |\vec{x}|), s' \neq_\tau t' \vdash_{\mathsf{LL}} \bot}\,\forall_{\mathsf{type}} \times m \qquad \mathcal{T}, F[t'] \vdash_{\mathsf{LL}} \bot}{\mathcal{T}, \forall \vec{\alpha}.\ \forall \vec{x}.\ s(\vec{\alpha}, \vec{x}) =_\tau t(|\vec{\alpha}|, |\vec{x}|), F[s'] \vdash_{\mathsf{LL}} \bot}\,\mathrm{Subst}$$

Corollary 5.3.5

Given a rewrite system $\mathcal{RE}$ and the theory $\mathcal{T}$ generated from $\mathcal{RE}$, if F and F' are two formulæ that are one-step convertible with respect to $\mathcal{RE}$ (denoted $F \equiv^1_{\mathcal{RE}} F'$), we have:

$$\textit{If} \quad \frac{\Gamma, F' \vdash_{\mathsf{LL}^{\equiv}} \bot}{\Gamma, F \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\mathrm{conv}_{F \equiv^1_{\mathcal{RE}} F'} \qquad \textit{Then} \quad \frac{\mathcal{T}, \Gamma, F' \vdash_{\mathsf{LL}} \bot}{\mathcal{T}, \Gamma, F \vdash_{\mathsf{LL}} \bot}$$

5.3.6 Multiple Step Rewriting

The last case of this section deals with multiple steps rewriting, both for propositions and terms.

Given a rewrite system $\mathcal{RE}$, we remind that two formulæ P and Q are convertible with respect to $\mathcal{RE}$ (denoted $P \equiv_{\mathcal{RE}} Q$), if there exists a finite number of rewrite rules of $\mathcal{RE}$ and n formulæ $R_1, \ldots, R_n$ such that:

$$P \equiv^1_{\mathcal{RE}} R_1 \equiv^1_{\mathcal{RE}} \ldots \equiv^1_{\mathcal{RE}} R_n \equiv^1_{\mathcal{RE}} Q$$

Lemma 5.3.6
Given a rewrite system $\mathcal{RE}$ and the theory $\mathcal{T}$ generated from $\mathcal{RE}$, if P and Q are two formulæ that are convertible with respect to $\mathcal{RE}$ (denoted $P \equiv_{\mathcal{RE}} Q$), we have:

$$\textit{If} \quad \frac{\Gamma, Q \vdash_{\mathsf{LL}^{\equiv}} \bot}{\Gamma, P \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\mathrm{conv}_{P \equiv_{\mathcal{RE}} Q} \qquad \textit{Then} \quad \frac{\mathcal{T}, \Gamma, Q \vdash_{\mathsf{LL}} \bot}{\mathcal{T}, \Gamma, P \vdash_{\mathsf{LL}} \bot}$$

Proof The proof is a direct consequence of the definition of conversion. If $P \equiv_{\mathcal{RE}} Q$, then there exists n formulæ $R_1, \ldots, R_n$ such that:

$$P \equiv^1_{\mathcal{RE}} R_1 \equiv^1_{\mathcal{RE}} \ldots \equiv^1_{\mathcal{RE}} R_n \equiv^1_{\mathcal{RE}} Q$$

Then, we have:

$$\dfrac{\dfrac{\dfrac{\Gamma, Q \vdash_{\mathsf{LL}^{\equiv}} \bot}{\Gamma, R_n \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\mathrm{conv}_{R_n \equiv^1_{\mathcal{RE}} Q}}{\vdots}}{\dfrac{\Gamma, R_1 \vdash_{\mathsf{LL}^{\equiv}} \bot}{\Gamma, P \vdash_{\mathsf{LL}^{\equiv}} \bot}\,\mathrm{conv}_{P \equiv^1_{\mathcal{RE}} R_1}}$$

Finally, thanks to the previous corollary 5.3.5, we obtain:

$$\dfrac{\dfrac{\dfrac{\mathcal{T}, \Gamma, Q \vdash_{\mathsf{LL}} \bot}{\mathcal{T}, \Gamma, R_n \vdash_{\mathsf{LL}} \bot}}{\vdots}}{\dfrac{\mathcal{T}, \Gamma, R_1 \vdash_{\mathsf{LL}} \bot}{\mathcal{T}, \Gamma, P \vdash_{\mathsf{LL}} \bot}}$$

5.3.7 Soundness of LLproof$^{\equiv}$ With Respect to LLproof

We can conclude this section with a straightforward proof of theorem 5.3.1.

Given a rewrite system $\mathcal{RE}$ and the theory $\mathcal{T}$ generated form $\mathcal{RE}$, we suppose given a LLproof$^{\equiv}$ proof Π such that:

$$\frac{\Pi}{\Gamma \vdash_{\mathsf{LL}^{\equiv}} \bot}$$

then, we build a LLproof proof Π' such that:

$$\frac{\Pi'}{\mathcal{T}, \Gamma \vdash_{\mathsf{LL}} \bot}$$

We perform the construction by induction on the structure of Π. All the cases are straightforward since we replace all LLproof$^{\equiv}$ nodes by their corresponding LLproof ones, and use lemma 5.3.6 in case of a conversion. For instance:

$$\frac{\dfrac{\Pi}{\Gamma, F_1, F_2 \vdash_{\mathsf{LL}^{\equiv}} \bot}}{\Gamma, F_1 \wedge F_2 \vdash_{\mathsf{LL}^{\equiv}} \bot} \wedge \quad \longrightarrow \quad \frac{\dfrac{\Pi'}{\mathcal{T}, \Gamma, F_1, F_2 \vdash_{\mathsf{LL}} \bot}}{\mathcal{T}, \Gamma, F_1 \wedge F_2 \vdash_{\mathsf{LL}} \bot} \wedge$$

5.3.8 Translation of LLproof Cut Rule into B

In Sec. 5.3.4, we used the LLproof Cut rule to translate LLproof$^{\equiv}$ proofs. So, we need to extend Figs. 4.3 and 4.4 by defining the translation of the LLproof Cut rule into B Natural Deduction.

The derivation of the Cut rule in B Natural Deduction is:

$$\dfrac{\dfrac{\langle \Gamma, P \vdash_{\mathsf{LL}} \bot \rangle^{\text{-}1} \qquad \dfrac{}{\langle \Gamma, P \vdash_{\mathsf{LL}} \neg\bot \rangle^{\text{-}1}}\,\mathsf{BR6}}{\langle \Gamma \vdash_{\mathsf{LL}} \neg P \rangle^{\text{-}1}}\,\mathsf{R6} \qquad \langle \Gamma, \neg P \vdash_{\mathsf{LL}} \bot \rangle^{\text{-}1}}{\langle \Gamma \vdash_{\mathsf{LL}} \bot \rangle^{\text{-}1}}\,\mathsf{BR4}$$

5.4 B Set Theory Modulo

Expressing the B Method set theory as a theory modulo consists in building an adequate rewrite system $\mathcal{RE}$ such that the theory $\mathcal{T}$ generated by $\mathcal{RE}$ is equivalent to the B theory. To do so, we transform whenever possible the axioms and definitions of the Poly-FOL B Method set theory into rewrite rules.

In this section, we present a B theory modulo resulting from the previous translations presented in Sec. 4.1. In this presentation, all the rules of the rewrite system come from the B syntax. Unfortunately, we will see in Chap. 8 that the resulting rules are not well suited for automated theorem proving in practice.

5.4.1 Axiomatic Set Theory

Core Theory The core B set theory consists in the translation into Poly-FOL of the three annotated axioms of Fig. 2.1. We remind in the following the Poly-FOL version of the B set theory signature and axioms as introduced in Sec. 4.1:

$$
\begin{array}{ll}
\mathsf{set} & :: 1 \\
\mathsf{tup} & :: 2 \\
(\text{-},\text{-}) & : \Pi\alpha_1\alpha_2.\ \alpha_1 \times \alpha_2 \to \mathsf{tup}(\alpha_1, \alpha_2) \\
\mathbb{P}(\text{-}) & : \Pi\alpha.\ \mathsf{set}(\alpha) \to \mathsf{set}(\mathsf{set}(\alpha)) \\
\text{-} \times \text{-} & : \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \\
\text{-} \in \text{-} & : \Pi\alpha.\ \alpha \times \mathsf{set}(\alpha) \to o
\end{array}
$$

$$
\begin{array}{ll}
\forall\alpha_1, \alpha_2.\ \forall s : \mathsf{set}(\alpha_1), t : \mathsf{set}(\alpha_2), x : \alpha_1, y : \alpha_2. & \\
\qquad (x, y)_{\alpha_1,\alpha_2} \in_{\mathsf{tup}(\alpha_1,\alpha_2)} s \times_{\alpha_1,\alpha_2} t \Leftrightarrow (x \in_{\alpha_1} s \wedge y \in_{\alpha_2} t) & \mathsf{SET1} \\
\forall\alpha.\ \forall s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha). & \\
\qquad s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) \Leftrightarrow (\forall x : \alpha.\ x \in_\alpha s \Rightarrow x \in_\alpha t) & \mathsf{SET2} \\
\forall\alpha.\ \forall s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha). & \\
\qquad (\forall x : \alpha.\ x \in_\alpha s \Leftrightarrow x \in_\alpha t) \Rightarrow s =_{\mathsf{set}(\alpha)} t & \mathsf{SET4}
\end{array}
$$

New Function Symbols In addition, the Poly-FOL version of the B set theory contains axioms coming from the skolemization of the comprehension sets (see Sec. 2.3). These axioms are of the following shape:

$$
\begin{array}{l}
\forall\alpha_1.\ldots.\alpha_m.\ \forall s_1 : \tau_1.\ldots.s_n : \tau_n.\ \forall x : \tau. \\
\qquad x \in_\tau f(\alpha_1, \ldots, \alpha_m; s_1, \ldots, s_n) \Leftrightarrow \varphi(\alpha_1, \ldots, \alpha_m; x, s_1, \ldots, s_n)
\end{array}
$$

where f is a function symbol with type signature:

$$f : \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to \mathsf{set}(\tau)$$

and where φ is a polymorphic formula.

5.4.2 Generated Rewrite System

We obtain the rewrite system (to lighten the presentation, we omit to indicate the local contexts Γ_{L}):

$$\begin{array}{ll}
(x,y)_{\alpha_1,\alpha_2} \in_{\mathsf{tup}(\alpha_1,\alpha_2)} s \times_{\alpha_1,\alpha_2} t & \longrightarrow\ x \in_{\alpha_1} s \wedge y \in_{\alpha_2} t \\
s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) & \longrightarrow\ \forall x : \alpha.\ x \in_\alpha s \Rightarrow x \in_\alpha t \\
x \in_{\alpha_1,\ldots,\alpha_{m_1}} f_1(\alpha_1,\ldots,\alpha_{m_1}; s_1,\ldots,s_{n_1}) & \longrightarrow\ \varphi_1(\alpha_1,\ldots,\alpha_{m_1}; x, s_1,\ldots,s_{n_1}) \\
\vdots & \\
x \in_{\alpha_1,\ldots,\alpha_{m_p}} f_p(\alpha_1,\ldots,\alpha_{m_p}; s_1,\ldots,s_{n_p}) & \longrightarrow\ \varphi_p(\alpha_1,\ldots,\alpha_{m_p}; x, s_1,\ldots,s_{n_p})
\end{array}$$

where $f_1,\ldots,f_p$ are the p new function symbols and $\varphi_1,\ldots,\varphi_p$ are the corresponding p polymorphic formulæ.

It should be noted that the extensionality axiom has not been turned into a rewrite rule since the left hand side of the rule would be an equality, which may not be efficient in practice. Thus, it is left as an axiom in the theory.

This presentation complies with all the previous result of conservativity presented in this manuscript. So, it is always possible to retrieve an original B proof coming from a $\mathsf{LLproof}^{\equiv}$ proof with this rewrite system.

Unfortunately, this general scheme does not fit well for derived construct, as we will see in the following section.

5.4.3 Derived Constructs

The derived constructs presented in Sec. 1.2.3.2 are quite important in the B Method since they are often used in proof obligations. The treatment of those constructs is therefore important.

Derived constructs are mostly defined using comprehension sets. Thus, they are translated as function symbols, as presented in Sec. 5.4.2. We show in the following the treatment of the derived constructs with the *union* operator as an example, where s, t and u are sets such that $s \subseteq u$ and $t \subseteq u$, and a is an element of u.

In the B Method, the basic set operators union $\cup$, intersection $\cap$, set difference $-$ and singleton $\{\ \}$ are defined by comprehension (see Sec. 1.2.3.2), therefore they have the shape $\{x \mid x \in u \wedge P\}$. We saw in Sec. 2.2 that the formula $x \in u$ represents some typing

information used to verify that a formula is well typed. For instance, the union between two sets s and t is defined as follows:

$$s \cup t \ := \ \{a \mid a \in u \wedge (a \in s \vee a \in t)\}$$

In this definition, u does not provide any logical information. It is here only to guarantee that the variable a has the proper type.

If we apply the annotation procedure described in Sec. 2.2, we obtain:

$$s^{\mathbb{P}(u)} \cup t^{\mathbb{P}(u)} \ := \ \{a^u \mid a \in u^{\mathbb{P}(u)} \wedge (a \in s^{\mathbb{P}(u)} \vee a \in t^{\mathbb{P}(u)})\}$$

Then, the skolemization of comprehension sets presented in Sec. 2.3 leads to:

$$s^{\mathbb{P}(u)} \cup t^{\mathbb{P}(u)} \ := \ f^{\mathbb{P}(u)}(u^{\mathbb{P}(u)}, s^{\mathbb{P}(u)}, t^{\mathbb{P}(u)})$$

and to add the axiom, where as usual we do not repeat the typing annotation on variables:

$$\forall u^{\mathbb{P}(u)} \cdot \forall s^{\mathbb{P}(u)} \cdot \forall t^{\mathbb{P}(u)} \cdot \forall x^u \cdot x \in f^{\mathbb{P}(u)}(u, s, t) \Leftrightarrow x \in u \wedge (x \in s \vee x \in t)$$

Finally, the translation scheme from B to Poly-FOL presented in Sec. 4.1 gives us the Poly-FOL axiom:

$$\forall \alpha.\ \forall u : \mathsf{set}(\alpha), s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha), x : \alpha.\ x \in_\alpha f(\alpha; u, s, t) \Leftrightarrow x \in_\alpha u \wedge (x \in_\alpha s \vee x \in_\alpha t)$$

where f has the type signature:

$$f : \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to \mathsf{set}(\alpha)$$

At the end, the rewrite rule generated for the union between two sets is:

$$x \in_\alpha f(\alpha; u, s, t) \ \longrightarrow_{\Gamma_\mathrm{L}} \ x \in_\alpha u \wedge (x \in_\alpha s \vee x \in_\alpha t)$$

where $\Gamma_\mathrm{L} := \alpha : \mathsf{Type}, x : \alpha, u : \mathsf{set}(\alpha), s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha)$

As we can see, the left-hand side of this rewrite rule has nothing to do with an intuitive definition of the union between two sets. The left-hand side deals with three different sets and the right-hand side starts with a conjunction.

This example illustrates that this approach – despite its usefulness in the theoretical part of our work – is not well-suited for derived constructs of the B Method in practice. From a simple definition of the union between two sets, we obtain at the end an axiom defining a construct dealing with three different variables.

Chapter 6

Automated Deduction: Zenon Modulo

This chapter presents the automated theorem prover (ATP for short) Zenon and its extensions to polymorphism and deduction modulo theory, resulting to a new tool called Zenon Modulo.

In Sec. 6.1, we present the principles of the Tableau method, the proof-search method used by Zenon. This section is inspired by [Bonichon, Delahaye, and Doligez 2007; D'Agostino, Gabbay, Hähnle, and Posegga 2013].

In Sec. 6.2, we first present the extension of the syntax and type system of Poly-FOL to deal with the proof-search format of Zenon, called MLproof. Then, we present the extension of MLproof to polymorphic formulæ and discuss the implication on the proof-search algorithm. This contribution (and its corresponding implementation) is a collaborative work and it has been published in [Bury, Cauderlier, and Halmagrand 2015a; Bury, Delahaye, Doligez, Halmagrand, and Hermant 2015b].

In Sec. 6.3, we present the extension of MLproof to deduction modulo theory, denoted MLproof$^{\equiv}$, an heuristic to automatically transform axioms into rewrite rules and the rewriting algorithm used by Zenon Modulo. This contribution (and its corresponding implementation) is a personal work and it has been published in [Bury, Delahaye, Doligez, Halmagrand, and Hermant 2015b; Delahaye, Doligez, Gilbert, Halmagrand, and Hermant 2013b,a].

In Sec. 6.4, we present the experimental results over two different benchmarks. In particular we compare the different versions of Zenon introduced in this chapter and also Zenon to other polymorphic deduction tools.

6.1 Zenon: A Tableau Method Automated Theorem Prover

In this section, we present the ATP Zenon [Bonichon, Delahaye, and Doligez 2007] and its proof-search method called the Tableau method.

6.1.1 Presentation of the Tableau Method

The Tableau method is an automatic proof search algorithm for the sequent calculus without cut. It is usually seen today as a tree method, proposed by Smullyan in 1968 [Smullyan 1995], that unifies and simplifies the analytic Tableau method of Beth in 1955 [Beth 1955], and the theory of model sets of Hintikka in 1955 [Hintikka 1955].

Tableau is a proof by contradiction method. To prove a formula, we have to show that the negation of the formula is unsatisfiable. To do so, the algorithm *breaks* the logical connectives of the formula until it reaches elementary atomic formulæ or negation of atomic formulæ. This process generates branches corresponding to different possibles cases. Once a contradiction is reached in a branch, *i.e.* the branch contains an atomic formula and its negation, the branch is said to be closed. A Tableau proof is a tree where all branches are closed.

6.1.2 Common Use of the Tableau Method

In the past few years, the popularity of the Tableau method in first-order classical logic has decreased, letting other automatic proof-search methods step in. For instance, in the CASC competition [Sutcliffe 2016], considered as the world-cup competition for first-order classical logic ATPs, tools using methods like resolution [Robinson 1965] or superposition [Nieuwenhuis and Rubio 2001] are widely represented, unlike tools using the Tableau method.

Today, the Tableau method is still often used for other kind of logics. In particular, it

is very popular for non-classical logics like modal logic [Goré 1999].

One important advantage of the Tableau method, compared to other methods like resolution or superposition, is its ability to generate strong proof traces [Bonichon, Delahaye, and Doligez 2007].

The Tableau method can be seen as a proof system which corresponds to an upside-down sequent calculus where all formulæ are on the left-hand side of the turnstile. Therefore, it is straightforward to generate proof traces in a standard sequent calculus format.

6.1.3 Key Features of Zenon

Zenon [Bonichon, Delahaye, and Doligez 2007] is an ATP for first-order classical logic with equality and based on the Tableau method. It was originally designed to be the dedicated ATP of the FoCaLiZe environment [Hardin, Pessaux, Weis, and Doligez 2009], an object-oriented algebraic specification and proof system.

The key feature of Zenon is the possibility to generate formal proof certificates that can be verified by the interactive theorem prover Coq [Bertot and Castéran 2013], used as a proof checker in that case. This feature is very specific to Zenon compared to other ATPs. The benefit provided by this approach is to guarantee by an external tool the soundness of the proofs produced by Zenon.

6.2 Extension of Zenon to Polymorphism

Most of first-order automated deduction tools do not implement polymorphism. For the time being and as far as we know, besides our version of Zenon, only three different tools deal natively with polymorphism.

Alt-Ergo, a Satisfiability Modulo Theory (SMT for short) solver released in 2008 [Bobot, Conchon, Contejean, Iguernelala, Lescuyer, and Mebsout 2013], was designed to be the dedicated tool of the verification platform Why3 [Bobot, Filliâtre, Marché, and Paskevich 2011], which the native language, called WhyML, is based on polymorphism.

The two other tools dealing with polymorphism, one called Zipperposition [Cruanes 2015] and the other one being a prototype [Wand 2014] based on the ATP SPASS [Weidenbach

1999], are both based on superposition and were both released in 2014.

It should be noted that some well-known ATPs implement monomorphic/many-sorted first-order logic (denoted Sorted-FOL), like E [Schulz 2013] or Vampire [Riazanov and Voronkov 1999].

6.2.1 Extending Poly-FOL to MLproof

Zenon has two different formats to write proofs. The first, called MLproof, is the proof search format, based on the Tableau method. The second, called LLproof (see Sec. 3.3), is based on the sequent calculus.

6.2.1.1 Syntax

We present in Fig. 6.1 and Fig. 6.2 the comprehensive syntax for types, type schemes, terms, formulæ, type quantified formulæ, local contexts and global contexts of MLproof$^{\equiv}$. This is an extension of the Poly-FOL syntax presented in Sec. 3.2.1.

It should be noted that constructs that were already defined in Sec. 3.2.1 do not change. We add only three constructs in type and term categories called term metavariables, type metavariables and ε-terms. We also add the notion of rewrite rule in global contexts, used later (see Sec. 6.3).

6.2.1.2 Type System

We present in Fig. 6.3 the rules for well-formedness. The only modification compared to the presentation in Fig. 3.1 is the addition of the rule WF_7 introduced in Sec. 5.2.1, that deals with rewrite rules.

However, we have to extend the type system of Poly-FOL of Fig. 3.2 by defining typing rules for metavariables and ε-terms. We give in Fig. 6.4 the extended type system.

6.2.2 Extension of MLproof to Polymorphism

In this section, we present the proof search system MLproof and discuss about the extension to polymorphism of its original version.

Type

τ	::=	α	(type variable)
	|	A_{Type}	(type metavariable)
	|	$T(\tau_1, \ldots, \tau_m)$	(type constructor application)

Type Scheme

σ	::=	$\Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to \tau$	(function type signature)
	|	$\Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to o$	(predicate type signature)

Term

t	::=	x	(variable)
	|	X_τ	(metavariable)
	|	$\varepsilon(x : \tau).\ \varphi(x)$	(ε − term)
	|	$f(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n)$	(function application)

Formula

φ	::=	$\top \mid \bot$	(true, false)
	|	$\neg\varphi$	(negation)
	|	$\varphi_1 \wedge \varphi_2$	(conjunction)
	|	$\varphi_1 \vee \varphi_2$	(disjunction)
	|	$\varphi_1 \Rightarrow \varphi_2$	(implication)
	|	$\varphi_1 \Leftrightarrow \varphi_2$	(equivalence)
	|	$t_1 =_\tau t_2$	(term equality)
	|	$P(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n)$	(predicate application)
	|	$\exists x : \tau.\ \varphi$	(existential quantification)
	|	$\forall x : \tau.\ \varphi$	(universal quantification)

Type Quantified Formula

φ_T	::=	φ	(formula)
	|	$\forall\alpha.\ \varphi_T$	(type quantification)

Figure 6.1: Poly-FOL Syntax Extended for $\mathsf{MLproof}^{\equiv}$ (Part 1)

Local Context

$$
\begin{array}{lcll}
\Gamma_{\mathrm{L}} & ::= & \varnothing & \text{(empty context)} \\
& | & \Gamma_{\mathrm{L}},\ \alpha : \mathsf{Type} & \text{(type variable declaration)} \\
& | & \Gamma_{\mathrm{L}},\ x : \tau & \text{(term variable declaration)}
\end{array}
$$

Global Context

$$
\begin{array}{lcll}
\Gamma_{\mathrm{G}} & ::= & \varnothing & \text{(empty context)} \\
& | & \Gamma_{\mathrm{G}},\ T :: m & \text{(type constructor declaration)} \\
& | & \Gamma_{\mathrm{G}},\ f : \sigma & \text{(function declaration)} \\
& | & \Gamma_{\mathrm{G}},\ P : \sigma & \text{(predicate declaration)} \\
& | & \Gamma_{\mathrm{G}},\ l \longrightarrow_{\Gamma_{\mathrm{L}}} r & \text{(rewrite rule)}
\end{array}
$$

Figure 6.2: Poly-FOL Syntax Extended for $\mathsf{MLproof}^{\equiv}$ (Part 2)

$$\frac{}{\mathsf{wf}(\varnothing;\varnothing)}\ \mathrm{WF}_1 \qquad \frac{x \notin \Gamma_{\mathrm{L}} \qquad \Gamma_{\mathrm{G}};\Gamma_{\mathrm{L}} \vdash \tau : \mathsf{Type}}{\mathsf{wf}(\Gamma_{\mathrm{G}};\Gamma_{\mathrm{L}}, x : \tau)}\ \mathrm{WF}_2$$

$$\frac{\alpha \notin \Gamma_{\mathrm{L}} \qquad \mathsf{wf}(\Gamma_{\mathrm{G}};\Gamma_{\mathrm{L}})}{\mathsf{wf}(\Gamma_{\mathrm{G}};\Gamma_{\mathrm{L}}, \alpha : \mathsf{Type})}\ \mathrm{WF}_3 \qquad \frac{T \notin \Gamma_{\mathrm{G}} \qquad \mathsf{wf}(\Gamma_{\mathrm{G}};\varnothing)}{\mathsf{wf}(\Gamma_{\mathrm{G}}, T :: m;\varnothing)}\ \mathrm{WF}_4$$

$$\frac{\begin{array}{c}\Gamma_{\mathrm{G}};\alpha_1 : \mathsf{Type},\ldots,\alpha_m : \mathsf{Type} \vdash \tau_i : \mathsf{Type}, i = 1\ldots n \\ f \notin \Gamma_{\mathrm{G}} \qquad \Gamma_{\mathrm{G}};\alpha_1 : \mathsf{Type},\ldots,\alpha_m : \mathsf{Type} \vdash \tau : \mathsf{Type}\end{array}}{\mathsf{wf}(\Gamma_{\mathrm{G}}, f : \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to \tau;\varnothing)}\ \mathrm{WF}_5$$

$$\frac{P \notin \Gamma_{\mathrm{G}} \qquad \Gamma_{\mathrm{G}};\alpha_1 : \mathsf{Type},\ldots,\alpha_m : \mathsf{Type} \vdash \tau_i : \mathsf{Type}, i = 1\ldots n}{\mathsf{wf}(\Gamma_{\mathrm{G}}, P : \Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to o;\varnothing)}\ \mathrm{WF}_6$$

$$\frac{\Gamma_{\mathrm{G}};\Gamma_{\mathrm{L}} \vdash l : \tau \qquad \Gamma_{\mathrm{G}};\Gamma_{\mathrm{L}} \vdash r : \tau \qquad \begin{array}{l}\mathrm{FV}_{\mathrm{T}}(r) \subseteq \mathrm{FV}_{\mathrm{T}}(l) \subseteq \Gamma_{\mathrm{L}} \\ \mathrm{FV}(r) \subseteq \mathrm{FV}(l) \subseteq \Gamma_{\mathrm{L}}\end{array}}{\mathsf{wf}(\Gamma_{\mathrm{G}}, l \longrightarrow_{\Gamma_{\mathrm{L}}} r;\varnothing)}\ \mathrm{WF}_7$$

Figure 6.3: Context Well-Formedness for Poly-FOL

$$\frac{}{\Gamma \vdash A_{\mathsf{Type}} : \mathsf{Type}}\ \text{TMeta} \qquad \frac{\Gamma \vdash \tau : \mathsf{Type}}{\Gamma \vdash X_\tau : \tau}\ \text{Meta}$$

$$\frac{\alpha : \mathsf{Type} \in \Gamma}{\Gamma \vdash \alpha : \mathsf{Type}}\ \text{TVar} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \text{Var}$$

$$\frac{\Gamma \vdash \tau : \mathsf{Type} \quad \Gamma, x : \tau \vdash \varphi(x) : o}{\Gamma \vdash \varepsilon(x : \tau).\ \varphi(x) : \tau}\ \varepsilon$$

$$\frac{T :: m \in \Gamma \quad \Gamma \vdash \tau_i : \mathsf{Type}, i = 1 \ldots m}{\Gamma \vdash T(\tau_1, \ldots, \tau_m) : \mathsf{Type}}\ \text{TConstr}$$

$$\frac{\begin{array}{ll} f : \Pi\alpha_1 \ldots \alpha_m.\tau_1 \times \ldots \times \tau_n \to \tau \in \Gamma & \Gamma \vdash \tau'_i : \mathsf{Type}, i = 1 \ldots m \\ \rho = [\alpha_1/\tau'_1, \ldots, \alpha_m/\tau'_m] & \Gamma \vdash t_i : \tau_i\rho, i = 1 \ldots n \end{array}}{\Gamma \vdash f(\tau'_1, \ldots, \tau'_m; t_1, \ldots, t_n) : \tau\rho}\ \text{Fun}$$

$$\frac{\begin{array}{ll} P : \Pi\alpha_1 \ldots \alpha_m.\tau_1 \times \ldots \times \tau_n \to o \in \Gamma & \Gamma \vdash \tau'_i : \mathsf{Type}, i = 1 \ldots m \\ \rho = [\alpha_1/\tau'_1, \ldots, \alpha_m/\tau'_m] & \Gamma \vdash t_i : \tau_i\rho, i = 1 \ldots n \end{array}}{\Gamma \vdash P(\tau'_1, \ldots, \tau'_m; t_1, \ldots, t_n) : o}\ \text{Pred}$$

$$\frac{}{\Gamma \vdash \top : o}\ \top \qquad \frac{}{\Gamma \vdash \bot : o}\ \bot$$

$$\frac{\Gamma \vdash \varphi_1 : o \quad \Gamma \vdash \varphi_2 : o}{\Gamma \vdash \varphi_1 \wedge \varphi_2 : o}\ \wedge \qquad \frac{\Gamma \vdash \varphi_1 : o \quad \Gamma \vdash \varphi_2 : o}{\Gamma \vdash \varphi_1 \vee \varphi_2 : o}\ \vee$$

$$\frac{\Gamma \vdash \varphi_1 : o \quad \Gamma \vdash \varphi_2 : o}{\Gamma \vdash \varphi_1 \Rightarrow \varphi_2 : o}\ \Rightarrow \qquad \frac{\Gamma \vdash \varphi_1 : o \quad \Gamma \vdash \varphi_2 : o}{\Gamma \vdash \varphi_1 \Leftrightarrow \varphi_2 : o}\ \Leftrightarrow$$

$$\frac{\Gamma \vdash \varphi : o}{\Gamma \vdash \neg\varphi : o}\ \neg \qquad \frac{\Gamma \vdash t_1 : \tau \quad \Gamma \vdash t_2 : \tau}{\Gamma \vdash t_1 =_\tau t_2 : o}\ =$$

$$\frac{\Gamma, x : \tau \vdash \varphi : o}{\Gamma \vdash \exists x : \tau.\ \varphi : o}\ \exists \qquad \frac{\Gamma, x : \tau \vdash \varphi : o}{\Gamma \vdash \forall x : \tau.\ \varphi : o}\ \forall$$

$$\frac{\Gamma, \alpha : \mathsf{Type} \vdash \varphi_T : o}{\Gamma \vdash \forall\alpha.\ \varphi_T : o}\ \forall_T$$

Figure 6.4: Extended Poly-FOL Type System for MLproof

Closure Rules

$$\frac{\bot}{\odot}\,\odot_{\bot} \qquad \frac{P, \neg P}{\odot}\,\odot \qquad \frac{\neg R_r(\tau_1, \ldots, \tau_m; a, a)}{\odot}\,\odot_r$$

$$\frac{\neg\top}{\odot}\,\odot_{\neg\top} \qquad \frac{R_s(\tau_1, \ldots, \tau_m; a, b), \neg R_s(\tau_1, \ldots, \tau_m; b, a)}{\odot}\,\odot_s$$

α Rules

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

β Rules

$$\frac{P \vee Q}{P \quad | \quad Q}\,\beta_{\vee} \qquad \frac{\neg(P \wedge Q)}{\neg P \quad | \quad \neg Q}\,\beta_{\neg\wedge} \qquad \frac{P \Rightarrow Q}{\neg P \quad | \quad Q}\,\beta_{\Rightarrow}$$

$$\frac{P \Leftrightarrow Q}{\neg P, \neg Q \quad | \quad P, Q}\,\beta_{\Leftrightarrow} \qquad \frac{\neg(P \Leftrightarrow Q)}{\neg P, Q \quad | \quad P, \neg Q}\,\beta_{\neg\Leftrightarrow}$$

δ Rules

$$\frac{\exists x : \tau.\ P(x)}{P(\varepsilon(x : \tau).\ P(x))}\,\delta_{\exists} \qquad \frac{\neg\forall x : \tau.\ P(x)}{\neg P(\varepsilon(x : \tau).\ \neg P(x))}\,\delta_{\neg\forall}$$

γ-Rules

$$\frac{\forall \alpha.\ P(\alpha)}{P(A_{\mathsf{Type}})}\,\gamma_{\forall M_{\mathsf{Type}}} \qquad \frac{\forall \alpha.\ P(\alpha)}{P(\tau)}\,\gamma_{\forall \mathrm{inst}_{\mathsf{Type}}}\ \tau : \mathsf{Type}$$

$$\frac{\forall x : \tau.\ P(x)}{P(X_\tau)}\,\gamma_{\forall M} \qquad \frac{\forall x : \tau.\ P(x)}{P(t)}\,\gamma_{\forall \mathrm{inst}}\ t : \tau$$

$$\frac{\neg\exists x : \tau.\ P(x)}{\neg P(X_\tau)}\,\gamma_{\neg\exists M} \qquad \frac{\neg\exists x : \tau.\ P(x)}{\neg P(t)}\,\gamma_{\neg\exists \mathrm{inst}}\ t : \tau$$

Figure 6.5: Proof Search Rules of MLproof (Part 1)

Relational Rules

$$\frac{P(\tau_1,\ldots,\tau_m;a_1,\ldots,a_n),\neg P(\tau_1,\ldots,\tau_m;b_1,\ldots,b_n)}{a_1 \neq_{\tau'_1} b_1 \quad | \quad \ldots \quad | \quad a_n \neq_{\tau'_n} b_n}\ \text{pred}$$

$$\frac{f(\tau_1,\ldots,\tau_m;a_1,\ldots,a_n) \neq f(\tau_1,\ldots,\tau_m;b_1,\ldots,b_n)}{a_1 \neq_{\tau'_1} b_1 \quad | \quad \ldots \quad | \quad a_n \neq_{\tau'_n} b_n}\ \text{fun}$$

$$\frac{R_s(\tau_1,\ldots,\tau_m;a,b),\neg R_s(\tau_1,\ldots,\tau_m;c,d)}{a \neq_\tau d \quad | \quad b \neq_\tau c}\ \text{sym}$$

$$\frac{\neg R_r(\tau_1,\ldots,\tau_m;a,b)}{a \neq_\tau b}\ \neg\text{refl}$$

$$\frac{R_t(\tau_1,\ldots,\tau_m;a,b),\neg R_t(\tau_1,\ldots,\tau_m;c,d)}{\begin{array}{c} c \neq_\tau a,\neg R_t(\tau_1,\ldots,\tau_m;c,a) \\ \mid b \neq_\tau d,\neg R_t(\tau_1,\ldots,\tau_m;b,d)\end{array}}\ \text{trans}$$

$$\frac{R_{ts}(\tau_1,\ldots,\tau_m;a,b),\neg R_{ts}(\tau_1,\ldots,\tau_m;c,d)}{\begin{array}{c} d \neq_\tau a,\neg R_t(\tau_1,\ldots,\tau_m;d,a) \\ \mid b \neq_\tau c,\neg R_{ts}(\tau_1,\ldots,\tau_m;b,c)\end{array}}\ \text{transsym}$$

$$\frac{a =_\tau b,\neg R_t(\tau_1,\ldots,\tau_m;c,d)}{\begin{array}{l} c \neq_\tau a,\neg R_t(\tau_1,\ldots,\tau_m;c,a) \\ \mid \neg R_t(\tau_1,\ldots,\tau_m;c,a),\neg R_t(\tau_1,\ldots,\tau_m;b,d) \\ \mid b \neq_\tau d,\neg R_t(\tau_1,\ldots,\tau_m;b,d)\end{array}}\ \text{transeq}$$

$$\frac{a =_\tau b,\neg R_{ts}(\tau_1,\ldots,\tau_m;c,d)}{\begin{array}{l} d \neq_\tau a,\neg R_{ts}(\tau_1,\ldots,\tau_m;d,a) \\ \mid \neg R_{ts}(\tau_1,\ldots,\tau_m;a,d),\neg R_{ts}(\tau_1,\ldots,\tau_m;b,c) \\ \mid b \neq_\tau c,\neg R_{ts}(\tau_1,\ldots,\tau_m;b,c)\end{array}}\ \text{transeqsym}$$

Figure 6.6: Proof Search Rules of MLproof (Part 2)

6.2.2.1 Inference Rules of MLproof

We present in Fig. 6.5 and Fig. 6.6 the inference rules of the proof-search system of Zenon, called MLproof.

A Tableau Method Proof System

The inference rules of MLproof are applied following the standard Tableau method, *i.e.* starting with the negation of the goal and by applying the rules in a top-down fashion to build a tree. We use the notation "|" to symbolize the separation of two branches, like in rules β.

A branch is said to be closed when it ends with an application of a closure rule, symbolized by "$\odot$". When all branches are closed, the proof tree is closed and this proof tree is a proof of the goal.

The two closure rules $\odot_r$ and $\odot_s$ deal with reflexive R_r and symmetric R_s relations respectively. For instance, if the reflexive relation R_r is the equality, the rule $\odot_r$ is then:

$$\frac{a \neq_\tau a}{\odot}\ \odot_r$$

As mentioned later, the only such relation in the context of our work is actually the equality relation.

An Ordering for Rule Application

The inference rules are divided into five distinct classes which is reflected in an ordering. The idea is to minimize the size of the proof tree to reduce the proof search space. The proof-search algorithm of Zenon will apply the rules with the following order relation $\prec$, where the relational rules of Fig. 6.6 are identified as β rules:

$$\odot \prec \alpha \prec \delta \prec \beta \prec \gamma$$

The reason of this ordering is simple. We always start by trying to close a branch with a $\odot$ closure rule. If we cannot close the branch, we try to apply the α rules that deal with logical connectives and that pursue with one branch. The δ rules also continue with

one branch and generate an ε-term. Then, we try to apply the β rules which deal with logical connectives but generate two branches. Finally, we apply the γ rules that deal with universal quantification and generate new metavariables.

The fairness of this procedure is ensured because all formulæ are decomposed into atomic formulæ at the end, thanks to rules α, β and δ which are terminating – and obviously for rules $\odot$. The only category of rules that are not terminating is the γ rules which may generate an infinite number of new formulæ.

It should be noted that this algorithm is applied in a strict depth-first order: it closes the current branch before starting to work on another branch.

Pruning

Pruning is a method to reduce the size of the proof tree. As explained in [Bonichon, Delahaye, and Doligez 2007], when a branching node N has a closed subtree as one of its branches B, it is possible to determine which formulæ are useful. If the formula introduced by N in B is not in the set of useful formulæ, Zenon removes N and grafts the subtree at its place since the subtree is a valid refutation of B without N.

A formula is said to be *useful* in a subtree if it is one of the formulæ appearing in the hypotheses of a rule application on that subtree.

We present an example of pruning in Sec. 6.2.2.2.

Metavariables

In the original and untyped version of Zenon, there was only term metavariables. These metavariables, sometimes called free variables in the Tableau-related literature, are not real variables in the sense that they are never substituted with terms inside formulæ.

With the original untyped Zenon, when meeting a universally quantified formula of the shape $\forall x.\ P(x)$, Zenon applies the $\gamma_{\forall M}$ rule

$$\frac{\forall x.\ P(x)}{P(X)}\,\gamma_{\forall M}$$

that introduces a new metavariable X, which is linked to this universal formula, and generates the formula $P(X)$. If Zenon reaches later a state where there is another formula

like $\neg P(t)$ – or one of its subformula –, where t is a closed term, then it reaches a possible contradiction with $P(X)$ – or the corresponding subformula. Another common case is ending with a formula of the form $X \neq t$. At this point, Zenon instantiates the original universally quantified formula linked to the metavariable X with the proper term t in the current branch by applying the rule $\gamma_{\forall\text{inst}}$

$$\frac{\forall x.\ P(x)}{P(t)}\ \gamma_{\forall\text{inst}}$$

Since we have extended Zenon to polymorphism, we have universal quantification over type variables. Then, we need to define also metavariables for type variables and the corresponding rules in MLproof.

We discuss in more detail in Sec. 6.2.2.2 the role of type metavariables and the difference compared to term metavariables during proof search.

Hilbert's ε-Terms

When dealing with existential quantification, Zenon uses Hilbert's ε-terms [Giese and Ahrendt 1999]. For a formula $P(x)$, the term "$\varepsilon(x).\ P(x)$" is an arbitrarily chosen term that satisfies $P(x)$, if such a term exists. The use of ε-term is an alternative to Skolem terms. The main benefit of ε-terms in the context of the proof search of Zenon is to keep the information of the linked formula, unlike for Skolem terms, allowing to reuse the same ε-term at different places.

Type Parameters in MLproof Rules

For some closure and relational rules, their application is conditioned by the fact that the predicate symbols are applied to the same list of type parameters. For instance, here is the pred rule:

$$\frac{P(\tau_1, \ldots, \tau_m; a_1, \ldots, a_n), \neg P(\tau_1, \ldots, \tau_m; b_1, \ldots, b_n)}{a_1 \neq_{\tau_1'} b_1 \quad | \quad \ldots \quad | \quad a_n \neq_{\tau_n'} b_n}\ \text{pred}$$

We see that we have $P(\tau_1, \ldots, \tau_m; a_1, \ldots, a_n)$ and $\neg P(\tau_1, \ldots, \tau_m; b_1, \ldots, b_n)$, where the type parameters $\tau_1, \ldots, \tau_m$ must be the same. Then, the application of the pred rule

generates n branches of the form $a_i \neq_{\tau_i'} b_i$. It should be noted that the type τ_i', which is the type of the two terms a_i and b_i, has no reason to be the same type than τ_i.

The fact that these type parameters have to be equal can be seen as a precondition.

Equality Reasoning

In Fig. 6.6, we present the rules dealing with relations. These rules are defined for reflexive relations denoted R_r, symmetric relations denoted R_s, transitive relations denoted R_t and finally transitive and symmetric relations denoted R_{ts}.

In practice, the only relation that is concerned by these rules in our work is the equality relation.

6.2.2.2 Dealing with Type Metavariables

In Zenon, term metavariables, introduced by the rule $\gamma_{\forall M}$ play a special role, as we have seen above. They serve to simulate a closure rule to determine a substitution by unification.

In presence of polymorphism, type metavariables may also be introduced, by the rules $\gamma_{\forall M_{\mathsf{Type}}}$ and $\gamma_{\neg\exists M_{\mathsf{Type}}}$. But the behavior of type metavariables differs from the behavior of term metavariables, it is no more possible to wait to reach a possible contradiction of the form $A_{\mathsf{Type}} \neq \tau$.

The role of type metavariables is to generate type instances that allow to apply inference rules bearing conditions on types, like rule pred. Thus, when trying to apply such a rule, we look for a type metavariable substitution that satisfies the constraints. In case of success, we instantiate the initial formula with rule $\gamma_{\forall \mathrm{inst}_{\mathsf{Type}}}$. This shortcut minimizes both the search space and the size of proof trees.

As an example, consider that we have a type $\tau : \mathsf{Type}$, two constants $a : \tau$ and $b : \tau$, and a predicate symbol P with signature $P : \Pi\alpha.\ \alpha \times \alpha \to o$. We assume:

$$\forall \alpha.\ \forall x, y : \alpha.\ P(\alpha; x, y)$$

And we want to prove:

$$P(\tau; a, b)$$

The proof, before pruning of useless formulæ, is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall\alpha.\ \forall x,y:\alpha.\ P(\alpha;x,y),\neg P(\tau;a,b)}{\forall x,y:A_{\mathsf{Type}}.\ P(A_{\mathsf{Type}};x,y)}\,\gamma_{\forall M_{\mathsf{Type}}}}{\forall y:A_{\mathsf{Type}}.\ P(A_{\mathsf{Type}};X_{A_{\mathsf{Type}}},y)}\,\gamma_{\forall M}}\,①}{P(A_{\mathsf{Type}};X_{A_{\mathsf{Type}}},Y_{A_{\mathsf{Type}}})}\,\gamma_{\forall \mathrm{inst}_{\mathsf{Type}}}}\,②}{\forall x,y:\tau.\ P(\tau;x,y)}\,\gamma_{\forall M}}{\forall y:\tau.\ P(\tau;X_\tau,y)}\,\gamma_{\forall M}}\,③}{P(\tau;X_\tau,Y_\tau)}}{X_\tau\neq_\tau a}\quad Y_\tau\neq_\tau b\ \mathrm{pred}}\,④}{\forall y:\tau.\ P(\tau;a,y)}\,\gamma_{\forall \mathrm{inst}}}{P(\tau;a,Y'_\tau)}\,\gamma_{\forall M}}{\dfrac{a\neq_\tau a}{\odot}\,\odot_r\qquad\dfrac{Y'_\tau\neq_\tau b}{\dfrac{P(\tau;a,b)}{\odot}\,\odot}\,\gamma_{\forall \mathrm{inst}}}\ \mathrm{pred}$$

We remark that, ① when we introduce the formula $P(A_{\mathsf{Type}};X_{A_{\mathsf{Type}}},Y_{A_{\mathsf{Type}}})$, we would like to apply the rule pred with $\neg P(\tau;a,b)$. But ② Zenon needs first to instantiate the type metavariable A_{Type} with the type τ. Then, ③ it generates some new (term) metavariables X_τ and Y_τ with the proper type. ④ We finally apply the rule pred and identify the subterms of $P(\tau;X_\tau,T_\tau)$ with the those of $\neg P(\tau;a,b)$, leading to generate two branches. Then, Zenon reaches a potential contradiction with the formula $X_\tau\neq a$, thus it instantiates the linked formula of X_τ with a. Doing the same with Y_τ, Zenon can finally close the local branch.

The proof search is done. Thanks to pruning of useless formulæ, the open right-hand branch can be erased, leading to the following proof tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall\alpha.\ \forall x,y:\alpha.\ P(\alpha;x,y),\neg P(\tau;a,b)}{\forall x,y:\tau.\ P(\tau;x,y)}\,\gamma_{\forall \mathrm{inst}_{\mathsf{Type}}}}{\forall y:\tau.\ P(\tau;a,y)}\,\gamma_{\forall \mathrm{inst}}}{P(\tau;a,b)}\,\gamma_{\forall \mathrm{inst}}}{\odot}\,\odot$$

6.3 Extension of Zenon to Deduction Modulo Theory

In this section, we discuss the extension of Zenon to deduction modulo theory.

6.3.1 MLproof$^{\equiv}$ and Deduction Modulo Theory

In practice, and unlike the presentation in Sec. 5.2.2 of the extension of LLproof to deduction modulo theory, we do not record the conversion steps by adding an explicit conversion rule. Instead, we merge the conversion rule with all inference rules of MLproof, leading to a deduction modulo theory proof-search system called MLproof$^{\equiv}$.

Given a rewrite system $\mathcal{RE}$ and a formula P in normal form with respect to $\mathcal{RE}$, we denote by $[P]$ any formula congruent to P modulo $\equiv_{\mathcal{RE}}$. Then, we can easily extend the inference rules of MLproof to reason modulo a congruence relation by replacing the hypothesis of a rule P by its class $[P]$.

For instance, the rule $\beta_\vee$ dealing with the disjunction is:

$$\frac{[P \vee Q]}{P \quad | \quad Q}\,\beta_\vee$$

We do not present all the MLproof$^{\equiv}$ system since it is straightforward.

6.3.2 Generation of the Rewrite System

Turning axioms into rewrite rules is a crucial point in deduction modulo theory. In Zenon Modulo, we propose two solutions to achieve that.

6.3.2.1 User-Defined Rewrite System

When dealing with a specific theory, it is possible to define manually which axiom could be turned into a rewrite rule, in the sense that it is done outside Zenon Modulo. To do that, it is possible to tag axioms in TPTP files using a special keyword "rewrite". This solution is used to prove B proof obligations.

6.3.2.2 Heuristic to Build a Rewrite System

The second solution to turn axioms into rewrite rules is to rely on a heuristic.

The main advantage of the heuristic is to be fully automatic. But it may also generate some inappropriate rewrite rules, leading to a rewrite system that does not enjoy good

properties – like confluence and termination – and that does not allow to have an efficient proof search.

We present in the following a heuristic, implemented in Zenon Modulo, that allows to generate both term and propositional rewrite rules.

The main idea is to transform into term rewrite rules, axioms of the shape:

$$\forall \vec{\alpha}.\ \forall \vec{x}.\ t = u$$

where t is a term that is not a variable and u is any term; and to transform into propositional rewrite rules, axioms of the shape:

$$\forall \vec{\alpha}.\ \forall \vec{x}.\ P \Leftrightarrow \varphi$$

where P is a predicate symbol and φ is any formula.

But we have to be more restrictive to avoid catching some particular kind of axioms, like those expressing commutativity properties of symbols – it would lead to immediate non-termination.

In the following, P denotes a predicate symbol which is not an equality, φ denotes an arbitrary formula, t a term which is not a variable and u an arbitrary term. In addition, we denote $\mathrm{FV}(\varphi)$ and $\mathrm{FV}(t)$ the union of the free term and type variables of φ and t respectively.

For propositional rewrite rules, we let:

$$\begin{array}{lcl} \forall \vec{\alpha}.\ \forall \vec{x}.\ P & \rhd & P \longrightarrow \top \\ \forall \vec{\alpha}.\ \forall \vec{x}.\ \neg P & \rhd & P \longrightarrow \bot \\ \forall \vec{\alpha}.\ \forall \vec{x}.\ P \Leftrightarrow \varphi & \rhd & P \longrightarrow \varphi \end{array}$$

The last transformation rule is under the proviso that $\mathrm{FV}(\varphi) \subseteq \mathrm{FV}(P) \subseteq \vec{\alpha} \cup \vec{x}$ and that P is not unifiable with φ or any subformula of φ.

For term rewrite rules, we let:

$$\forall \vec{\alpha}.\ \forall \vec{x}.\ t = u \quad \rhd \quad t \longrightarrow u$$

provided that $\mathrm{FV}(u) \subseteq \mathrm{FV}(t) \subseteq \vec{\alpha} \cup \vec{x}$ and that t is not unifiable with u or any subterm of u.

Verifying that the left-hand side of a rewrite rule is not unifiable with any subformula/-subterm of the right-hand side allows us to eliminate some trivial cases of non-termination. Unfortunately, it does not guarantee that our final rewrite system is terminating, since we do not test this criterion for all the rewrite rules. But this heuristic is terminating and rather efficient, thus we consider it as a good compromise.

6.3.3 Rewriting Algorithm

The implementation of deduction modulo theory into Zenon Modulo consists in performing normalization of formulæ during proof search. This step of normalization is done after the application of each inference rule to all the formulæ newly generated.

The normalization procedure goes as follows:

1. We normalize only the literals, *i.e.* atomic formulæ or their negation;
2. We normalize with respect to term rewrite rules;
3. We apply one step of propositional rewriting;
4. If the formula is still unchanged, we quit; otherwise we go back to item 1.

This algorithm allows us to normalize an atomic formula up to the point, when we have either a normal form with respect to the rewrite system, or a non-atomic formula.

6.4 Experimental Results

6.4.1 Presentation of TFF1

TPTP [Sutcliffe 2009] is a well-established project for the automated theorem proving community. It provides a large library of problems – around 20,000 problems all categories together – to test and benchmark automated deduction tools. It also promotes the use of standard syntaxes for problems and proofs. The most known format is called FOF and deals with untyped first-order logic (denote FOL). We can also cite the format TFF0 for monomorphic/many-sorted first-order logic (denoted Sorted-FOL).

The TFF1 format [Blanchette and Paskevich 2013] is a new format proposed to the TPTP community in 2013. It extends the format TFF0 to polymorphic types. The polymorphic problems of TFF1 is not yet an official category in the competition CASC.

The syntax of TFF1 is close to the presentation of Poly-FOL in Sec. 3.2.1. This format is used by Zenon Modulo as its input format for polymorphic problems.

6.4.2 Encoding of Poly-FOL into FOL

A solution to use untyped provers with polymorphic theories is to rely on encodings of Poly-FOL into FOL. This question was largely studied and leads to many different encodings [Blanchette, Böhme, Popescu, and Smallbone 2013].

The verification platform Why3 [Bobot, Filliâtre, Marché, and Paskevich 2011] implements different encodings of Poly-FOL into FOL proposed by Blanchette in [Blanchette, Böhme, Popescu, and Smallbone 2013]. The default encoding is called "featherweight tags" (denoted t??) in [Blanchette, Böhme, Popescu, and Smallbone 2013] and the second one tested in the following is called "featherweight guards" (denoted g??). These encodings are used in the rest of this chapter to compare the untyped version of Zenon to the one using the new extension to polymorphism.

In the following, we present the encodings into TFF1 and FOF of the example of Sec. 4.2.4.

6.4.2.1 A Polymorphic Problem in WhyML

The syntax of WhyML is close to the syntax of Poly-FOL. The main differences are the explicit type variable quantifications in Poly-FOL, which no longer occurs in WhyML, and the type parameters of function and predicate symbol in Poly-FOL, that are not used in WhyML. We say that WhyML uses an implicit typing notation.

The example of Sec. 4.2.4 in WhyML is:

```
theory Example
  type set 'a
  predicate mem 'a (set 'a)
  function power (set 'a) : set (set 'a)
  axiom mem_power :
    forall s t : set 'a.
```

```
    mem s (power t)
    <-> (forall x : 'a. (mem x s) -> (mem x t))
  type t
  constant u : (set t)
  goal example :
    mem u (power u)
end
```

6.4.2.2 Translation of the Problem into TFF1

The encoding of the example by Why3 into TFF1 leads to the following code. The syntax of TFF1 is very close to Poly-FOL, `$tType` and `$o` being keywords of the language and corresponding to Type and o.

```
tff(t, type,
    t: $tType).
tff(u, type,
    u: set(t)).
tff(set, type,
    set: $tType > $tType).
tff(mem, type,
    mem: !>[A : $tType]: ((A * set(A)) > $o)).
tff(power, type,
    power: !>[A : $tType]: (set(A) > set(set(A)))).
tff(mem_power, axiom,
    ![A : $tType]: ![S:set(A), T:set(A)]:
    (mem(set(A), S, power(A, T))
    <=> ![X:A]: (mem(A, X, S) => mem(A, X, T)))).
tff(example, conjecture,
    mem(set(t), u, power(t, u))).
```

6.4.2.3 Translation of the Problem into FOF using t??

The default encoding, called featherweight tags t??, of Poly-FOL into FOF defines two new function symbols `sort` – which allows to verify the *type* of expressions – and `witness` – that states the existence of a witness. Then, it changes the original axiom by adding some constraints to verify the types of expressions.

```
fof(witness_sort, axiom,
    ![A]:
    (sort(A, witness(A)) = witness(A))).
fof(power_sort, axiom,
    ![A]: ![X]:
    (sort(set(set(A)), power(A, X)) = power(A, X))).
fof(mem_power, axiom,
```

```
    ![A]: ![S, T]:
    ((mem(set(A), S, power(A, T))
      => ![X]:
         (mem(A, X, S) => mem(A, X, T)))
    & (![X]:
       ((sort(A, X) = X) => (mem(A, X, S) => mem(A, X, T)))
       => mem(set(A), S, power(A, T)))))).
fof(u_sort, axiom,
    (sort(set(t), u) = u)).
fof(example, conjecture,
    mem(set(t), u, power(t, u))).
```

6.4.2.4 Translation of the Problem into FOF using g??

The second encoding implemented in Why3, called featherweight guards g??, of Poly-FOL into FOF is very close to the previous one. This time the newly defined symbol `sort` is a predicate symbol.

```
fof(witness_sort, axiom,
    ![A]:
    sort(A, witness(A))).
fof(power_sort, axiom,
    ![A]: ![X]:
    sort(set(set(A)), power(A, X))).
fof(mem_power, axiom,
    ![A]: ![S, T]:
    ((mem(set(A), S, power(A, T))
      => ![X]:
         (mem(A, X, S) => mem(A, X, T)))
    & (![X]:
       (sort(A, X) => (mem(A, X, S) => mem(A, X, T)))
       => mem(set(A), S, power(A, T)))))).
fof(u_sort, axiom,
    sort(set(t), u)).
fof(example, conjecture,
    mem(set(t), u, power(t, u))).
```

6.4.2.5 Remarks about the Encoding

The two encodings of Poly-FOL into FOF presented above have two main consequences on the input theory:

1. It increases the number of axioms leading to a larger search space, from one to four axioms in our examples;

2. It modifies the shape of the original axioms, adding some conditions that have to be fulfilled.

3. The TFF1 version of the axiom `mem_power` is well-suited to be turned into a rewrite rule using the heuristic presented above, unlike the two FOF versions that cannot be turned into rewrite rules.

Thus, the main reason to extend Zenon to polymorphism is to be able to use deduction modulo theory.

6.4.3 Experimental Results

To test our new tool Zenon Modulo dealing with Poly-FOL and rewriting, we perform an experiment over two different benchmarks, the former coming from the TPTP TFF1 library and the later from the B set theory.

In the following, we call Zenon FOF the original untyped version of Zenon, Zenon TFF1 the extension of Zenon to polymorphism, and Zenon Modulo the extension to deduction modulo theory of Zenon TFF1.

All these versions of Zenon are actually the same tool Zenon Modulo 0.4.2, using different options of the command line. The source code of Zenon Modulo is freely accessible at: `http://zenon.gforge.inria.fr/`

6.4.3.1 TPTP TFF1

We select all the polymorphic problems of the TFF1 library with status "theorem" – those which are known to be provable – leading to a benchmark made of 356 problems. We present in Tab. 6.1 and Tab. 6.2 the experimental results.

In Tab. 6.1, we compare Zenon FOF with the two encoding t?? and g?? presented in Sec. 6.4.2, Zenon TFF1 and Zenon Modulo with the heuristic presented in Sec. 6.3.2.2. We indicate the number of problems proved by each provers, and between parentheses the number of problems that are proved only by the concerned prover. In the second line, we give the mean time spent to prove problems. Finally, the third line gives the number of proved problems that are well-checked by Dedukti [Assaf, Burel, Cauderlier,

Delahaye, Dowek, Dubois, Gilbert, Halmagrand, Hermant, and Saillard 2016]. In Tab. 6.2, we compare Zenon TFF1 to the ATP Zipperposition and the SMT solver Alt-Ergo.

356 Prob.	Zenon FOF with t??	Zenon FOF with g??	Zenon TFF1	Zenon Modulo with heuristic
Proved (only by)	115 (4)	124 (8)	124 (6)	91 (7)
Mean Time in seconds	13.3	13.5	9.7	2.7
Checked by Dedukti	113	122	117	84

Table 6.1: Experimental Results over the TPTP TFF1 Benchmark (Part 1)

356 Prob.	Zenon TFF1	Zipperposition	Alt-Ergo
Proved (only by)	124 (9)	145 (8)	225 (65)
Mean Time in seconds	9.7	18.4	4.6

Table 6.2: Experimental Results over the TPTP TFF1 Benchmark (Part 2)

These results are not conclusive yet. The results of Tab. 6.1 show that Zenon FOF with encoding g?? and Zenon TFF1 prove more problems than Zenon FOF with encoding t?? and Zenon Modulo with heuristic. Zenon Modulo with the heuristic is not efficient in this benchmark. The reason is that the heuristic generates some non-terminating rewrite system. It should be noted that each of the four versions of Zenon presented here uniquely prove some problems.

The results of Tab. 6.2 show us that Zenon TFF1 proves fewer problems than Zipperposition and Alt-Ergo. But there are nine problems that are proved only by Zenon TFF1. Also, Zenon is the only prover considered to generate proof certificates.

320 Prob.	Zenon FOF with t??	Zenon FOF with g??	Zenon TFF1	Zenon Modulo with heuristic
Proved (only by)	5 (0)	4 (0)	3 (0)	123 (117)
Mean Time in seconds	7.5	10.4	26.6	1.4
Dedukti Checks OK	5	4	3	121

Table 6.3: Experimental Results over the B-Book Lemmas Benchmark (Part 1)

320 Prob.	Zenon Modulo with heuristic	Zipperposition	Alt-Ergo
Proved (only by)	123 (60)	6 (0)	61 (0)
Mean Time in seconds	1.3	9.7	0.07

Table 6.4: Experimental Results over the B-Book Lemmas Benchmark (Part 2)

6.4.3.2 The B Set Theory

We build a benchmark in the WhyML format, made of 320 Poly-FOL problems coming from the B-Book. These problems are B set theory lemmas stated in the chapter 2 of the B-Book, and dealing with all the B operators defined. In addition, we define directly in WhyML the B set theory.

In Tab. 6.3, we compare the same versions of Zenon, than in Tab. 6.1, between each other. In Tab. 6.4, we compare Zenon Modulo with the heuristic to the ATP Zipperposition and the SMT Alt-Ergo.

The results of Tab. 6.3 show that Zenon Modulo with the heuristic is much more efficient than the other versions of Zenon. The difference is very significant, both for the number of proved formulæ and for the mean time.

The results of Tab. 6.4 show that Zenon Modulo with the heuristic is also more efficient than Zipperposition and Alt-Ergo on this benchmark.

The main reason is that the B set theory is made of axioms that fit well deduction modulo theory and our heuristic.

Chapter 7

Proof Certification Using Dedukti

This chapter presents the certification of the proofs produced by Zenon Modulo using the proof-checker Dedukti. We present in Sec. 7.2 the $\lambda\Pi$-calculus modulo theory, an extension of the simply typed λ-calculus with dependant types and rewriting, and the proof-checker Dedukti which implements the $\lambda\Pi$-calculus modulo theory. This presentation of Dedukti is inspired by [Assaf, Burel, Cauderlier, Delahaye, Dowek, Dubois, Gilbert, Halmagrand, Hermant, and Saillard 2016; Cauderlier 2016; Saillard 2015].

In Sec. 7.3, we present the encoding of Poly-FOL into the $\lambda\Pi$-calculus modulo theory. This contribution (and its corresponding implementation) is a collaborative work and it has been published in [Cauderlier and Halmagrand 2015].

In Sec. 7.4, we present the encoding of Zenon Modulo proofs in Dedukti, and we provide an example of proof certificate. This contribution (and its corresponding implementation) is a collaborative work and it has been published in [Cauderlier and Halmagrand 2015].

7.1 A Proof Checker Dealing with Rewriting

A key feature of Zenon, compared to other ATPs, is its certifying approach (see Sec. 6.1.3). It consists in generating proof certificates that can be verified by external proof-checkers.

The original version of Zenon – without polymorphism and deduction modulo theory – uses the proof assistant Coq [Bertot and Castéran 2013] as a proof checker.

The extension of Zenon to deduction modulo theory leads to introduce rewriting steps inside proofs. In addition, these rewriting steps are not recorded in order to reduce the size of proofs.

Coq is not well-suited to check proofs using rewriting techniques. This would require to rebuild all the rewriting steps formally, and provided them to the proof certificates, leading to unnecessary larger files.

Instead, we choose to use another proof checker, called Dedukti [Assaf, Burel, Cauderlier, Delahaye, Dowek, Dubois, Gilbert, Halmagrand, Hermant, and Saillard 2016], which deals natively with rewriting.

7.2 The $\lambda\Pi$-Calculus Modulo Theory and Dedukti

The $\lambda\Pi$-calculus is an extension of the simply typed λ-calculus with dependent types [Barendregt, Dekkers, and Statman 2013]. It is commonly used as a logical framework to encode logics [Harper, Honsell, and Plotkin 1993].

The $\lambda\Pi$-calculus modulo theory, denoted by $\lambda\Pi^{\equiv}$ in the following, is an extension of the $\lambda\Pi$-calculus to rewriting. We do not claim to give a rigorous and exhaustive presentation of $\lambda\Pi^{\equiv}$ here, since it is out of the scope of our work. We only introduce the basic notions needed for the certification of Zenon Modulo proofs. An inquisitive reader should have a look at [Assaf, Burel, Cauderlier, Delahaye, Dowek, Dubois, Gilbert, Halmagrand, Hermant, and Saillard 2016; Saillard 2015] for a comprehensive presentation of $\lambda\Pi^{\equiv}$.

7.2.1 Syntax of $\lambda\Pi^{\equiv}$

We present in Fig. 7.1 the syntax of $\lambda\Pi^{\equiv}$.

In this calculus, types are not syntactically distinguished from terms. Only two terms, Type and Kind, are defined in a particular syntactic category and are called *sorts*, denoted s.

Type is the sort of the types used to type terms. For instance, if we have the typing judgment $\Gamma \vdash t : A$ where t is a term and A is its type, then we have $\Gamma \vdash A : \mathsf{Type}$. In this case, we say that t is an *object*.

s	::=	Type	(sort of types)
	|	Kind	(sort of Type)
t	::=	x	(variable)
	|	$t_1\ t_2$	(term application)
	|	$\lambda x : t_1.\ t_2$	(lambda abstraction)
	|	$\Pi x : t_1.\ t_2$	(product)
	|	s	(type)
Δ	::=	$\varnothing$	(empty local context)
	|	$\Delta, x : t$	(variable declaration)
Γ	::=	$\varnothing$	(empty global context)
	|	$\Gamma, x : t$	(variable declaration)
	|	$\Gamma, t_1 \hookrightarrow_\Delta t_2$	(rewrite rule declaration)

Figure 7.1: The Syntax of the $\lambda\Pi$-Calculus Modulo Theory

The second sort Kind is introduced to type Type (see rule Sort in Fig. 7.2), and other types built using Type, like $A \to$ Type, ... Thus, if we have the two typing judgments $\Gamma \vdash A : B$ and $\Gamma \vdash B :$ Kind, we say that A is a type.

A term t is either a variable x, or built inductively from term application, lambda abstraction and product. It should be noted that the arrow type $\to$ is a particular case of the product type. For instance, $A \to B$ is actually $\Pi x : A.\ B$ where x does not occur freely in B. Finally, a term can also be a sort.

In $\lambda\Pi^{\equiv}$, local contexts, denoted Δ, are used to type variables which occur freely in terms of a rewrite rules. A local context Δ is a set of variable declarations, *i.e.* pairs made of an identifier of a variable and its type.

The last category in Fig. 7.1 is the global context Γ, containing variable declarations and rewrite rules.

7.2.2 Typing Rules

We present in Fig. 7.2 the typing rules of $\lambda\Pi^{\equiv}$.

Well-formedness

$$\frac{}{\mathsf{wf}(\varnothing)}\ \text{Empty} \qquad \frac{\mathsf{wf}(\Gamma) \quad \Gamma \vdash A : s \quad x \notin \Gamma}{\mathsf{wf}(\Gamma, x : A)}\ \text{Decl}$$

$$\frac{\begin{array}{ll}\Gamma, \Delta \vdash l : A & \Gamma, \Delta \vdash A : \mathsf{Type} \\ \Gamma, \Delta \vdash r : A & \mathrm{FV}(r) \subseteq \mathrm{FV}(l) \subseteq \Delta\end{array}}{\mathsf{wf}(\Gamma, l \hookrightarrow_\Delta r)}\ \text{Rew}$$

Typing

$$\frac{\mathsf{wf}(\Gamma)}{\Gamma \vdash \mathsf{Type} : \mathsf{Kind}}\ \text{Sort} \qquad \frac{\mathsf{wf}(\Gamma) \quad x : A \in \Gamma}{\Gamma \vdash x : A}\ \text{Var}$$

$$\frac{\Gamma \vdash t_1 : \Pi x : A\ B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : B[x/t_2]}\ \text{App}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, x : A \vdash t : B \quad \Gamma, x : A \vdash B : s}{\Gamma \vdash \lambda x : A.\ t : \Pi x : A\ B}\ \text{Abs}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A\ B : s}\ \text{Prod}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash B : s \quad A \equiv_{\beta\Gamma} B}{\Gamma \vdash t : B}\ \text{Conv}$$

Figure 7.2: The $\lambda\Pi$-Calculus Modulo Theory

A variable declaration $\Gamma, x : A$ is well-formed if x is not yet declared in Γ and if the type of A is either Type or Kind. A rewrite rule declaration $\Gamma, l \hookrightarrow_\Delta r$ is well-formed if both term l and r have have the same type and if r does not introduce new free variables.

It should be noted that we factorize some rules. For instance, in rule Abs, in the premise $\Gamma, x : A \vdash B : s$ the sort s may be either Type or Kind.

In typing rule App, the notation $B[x/t_2]$ denotes the substitution in B of x by t_2. The most interesting rule is the rule Conv which tells us that we can replace a type A by another type B if these two types are $\beta\Gamma$-convertible.

The particularity of $\lambda\Pi^{\equiv}$ resides in this rule Conv. Γ contains all the rewrite rules defined, thus the $\beta\Gamma$-convertibility deals with both β-reduction and reduction with respect to the custom rewrite system. When this rewrite system is both strongly normalizing and confluent, each term gets a unique normal form (up to α-conversion), and both conversion

and type-checking become decidable.

7.2.3 Dedukti

Dedukti is a proof-checker for the $\lambda\Pi^{\equiv}$, developed by Saillard [Saillard 2015] and made of around 2,000 lines of OCaml code. It is used as a backend to verify proofs coming from ATPs, like Zenon Modulo and iProver Modulo [Burel 2011, 2013], and from proof assistants, like Coq [Boespflug and Burel 2012], HOL [Assaf and Burel 2015] and FoCaLiZe [Cauderlier 2016].

Dedukti implements some powerful features, like higher-order rewriting, making it highly expressive. In addition, experiments show that its implementation is quite effective in practice, allowing us to verify proofs quickly.

7.3 Encoding of Poly-FOL into $\lambda\Pi^{\equiv}$

In this section, we present the embedding of Poly-FOL into $\lambda\Pi^{\equiv}$. This work is an extension of the embedding of FOL into $\lambda\Pi^{\equiv}$ proposed by Burel [Burel 2013].

This embedding relies on two encodings: a *deep* encoding, denoted $|\varphi|$ for a Poly-FOL formula φ, in which logical connectives are simply declared as Dedukti constants; and a *shallow* encoding, denoted $\|\varphi\| := \mathsf{prf}\ |\varphi|$ for a Poly-FOL formula φ, using a decoding function prf to translate connectives to their impredicative encodings. We present in Sec. 7.3.2, the deep embedding and in Sec. 7.3.3 the shallow embedding of Poly-FOL into $\lambda\Pi^{\equiv}$.

7.3.1 Remarks about Poly-FOL and LLproof$^{\equiv}$

In the following, when speaking about Poly-FOL, we are referring to the syntax extended to deduction modulo theory, with rewrite rules, but without metavariables and ε-terms. So, it corresponds to the syntax presented in Fig. 6.1 and Fig. 6.2, without type and term metavariables and without ε-terms.

In addition, the considered proof system LLproof$^{\equiv}$ does not have an explicit conversion rule (like in Sec. 5.2.2), instead we merge the conversion rule with all inference rules of

LLproof$^{\equiv}$ (as for MLproof$^{\equiv}$ in Sec. 6.3.1).

7.3.2 Deep embedding

We present in Fig. 7.3 the declarations of Poly-FOL symbols as Dedukti constants, corresponding to a deep embedding of Poly-FOL. We call Γ_0 the set of declarations of Fig. 7.3.

Primitive Types

$$\mathsf{Prop} : \mathsf{Type} \qquad \mathsf{prf} : \mathsf{Prop} \to \mathsf{Type}$$
$$\mathsf{type} : \mathsf{Type} \qquad \mathsf{term} : \mathsf{type} \to \mathsf{Type}$$

Primitive Connectives

$$\top : \mathsf{Prop}$$
$$\bot : \mathsf{Prop}$$
$$\neg\text{-} : \mathsf{Prop} \to \mathsf{Prop}$$
$$\text{-} \wedge \text{-} : \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop}$$
$$\text{-} \vee \text{-} : \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop}$$
$$\text{-} \Rightarrow \text{-} : \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop}$$
$$\text{-} \Leftrightarrow \text{-} : \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop}$$
$$\forall\text{--} : \Pi\alpha : \mathsf{type}.\ (\mathsf{term}\ \alpha \to \mathsf{Prop}) \to \mathsf{Prop}$$
$$\exists\text{--} : \Pi\alpha : \mathsf{type}.\ (\mathsf{term}\ \alpha \to \mathsf{Prop}) \to \mathsf{Prop}$$
$$\forall_{\mathsf{type}}\text{-} : (\mathsf{type} \to \mathsf{Prop}) \to \mathsf{Prop}$$
$$\text{-} =_{\_} \text{-} : \Pi\alpha : \mathsf{type}.\ \mathsf{term}\ \alpha \to \mathsf{term}\ \alpha \to \mathsf{Prop}$$

Figure 7.3: Dedukti Declarations of Poly-FOL Symbols

Our embedding uses two primitive types, Prop and type. Notice that we have type : Type – case matters. The former is the type of propositions, like $\top$: Prop. The latter corresponds to the type of Poly-FOL types, like for type variables α : type. In addition, we define two functions prf and term that embed our encoded Poly-FOL terms and formulæ into the native Dedukti type Type.

7.3.3 Shallow Embedding

We present in Fig. 7.4 the shallow definitions of the Dedukti constants declared in Fig. 7.3. In Dedukti, definitions are given as rewrite rules, denoted with the symbol $\hookrightarrow$.

$$
\begin{array}{lcl}
\mathsf{prf}\ \top & \hookrightarrow & \Pi P : \mathsf{Prop}.\ \mathsf{prf} P \to \mathsf{prf} P \\
\mathsf{prf}\ \bot & \hookrightarrow & \Pi P : \mathsf{Prop}.\ \mathsf{prf} P \\
\mathsf{prf}\ (\neg A) & \hookrightarrow & \mathsf{prf} A \to \mathsf{prf} \bot \\
\mathsf{prf}\ (A \wedge B) & \hookrightarrow & \Pi P : \mathsf{Prop}.\ (\mathsf{prf} A \to \mathsf{prf} B \to \mathsf{prf} P) \to \mathsf{prf} P \\
\mathsf{prf}\ (A \vee B) & \hookrightarrow & \Pi P : \mathsf{Prop}.\ (\mathsf{prf} A \to \mathsf{prf} P) \to (\mathsf{prf} B \to \mathsf{prf} P) \to \mathsf{prf} P \\
\mathsf{prf}\ (A \Rightarrow B) & \hookrightarrow & \mathsf{prf} A \to \mathsf{prf} B \\
\mathsf{prf}\ (A \Leftrightarrow B) & \hookrightarrow & \mathsf{prf}((A \Rightarrow B) \wedge (B \Rightarrow A)) \\
\mathsf{prf}\ (\forall\ \tau\ P) & \hookrightarrow & \Pi x : \mathsf{term}\ \tau.\ (P\ x) \\
\mathsf{prf}\ (\exists\ \tau\ P) & \hookrightarrow & \Pi P : \mathsf{Prop}.\ (\Pi x : \mathsf{term}\ \tau.\ \mathsf{prf}(P\ x) \to \mathsf{prf} P) \to \mathsf{prf} P \\
\mathsf{prf}\ (\forall_{\mathsf{type}}\ P) & \hookrightarrow & \Pi \alpha : \mathsf{type}.\ \mathsf{prf}(P\ \alpha) \\
\mathsf{prf}\ (x =_\tau y) & \hookrightarrow & \Pi P : (\mathsf{term}\ \tau \to \mathsf{Prop}).\ \mathsf{prf}(P\ x) \to \mathsf{prf}(P\ y)
\end{array}
$$

Figure 7.4: Shallow Definitions of Poly-FOL Symbols in Dedukti

We claim that this embedding is shallow in the sense that our new symbols – corresponding to the Poly-FOL symbols – are normalized into primitive symbols of Dedukti. For instance, in the following definition of an implication:

$$\mathsf{prf}(A \Rightarrow B) \quad \hookrightarrow \quad \mathsf{prf} A \to \mathsf{prf} B$$

we use the native arrow "$\to$" of Dedukti to define the encoded Poly-FOL implication "$\Rightarrow$". Similarly, for universal quantification, we are encoding "$\forall$" with the Dedukti "Π".

Remark The main benefit of a shallow encoding – compared to a deep one – is to benefit from the computational aspect of Dedukti. In addition, it helps to share proofs coming from different systems.

For instance, Burel in [Burel 2013] uses a similar encoding to verify proofs coming from the ATP iProver Modulo. So, it should be quite straightforward to combine proofs coming from Zenon Modulo with those of iProver Modulo.

7.3.4 Translation Functions from Poly-FOL into $\lambda\Pi^{\equiv}$

We present in Fig. 7.5 and in Fig. 7.6 the translation function of Poly-FOL types, terms, formulæ, local contexts and global contexts into Dedukti.

Translation of Types τ

$$
\begin{array}{rcl}
|\alpha| & := & \alpha \\
|T(\tau_1, \ldots, \tau_m)| & := & T\ |\tau_1|\ \ldots\ |\tau_m|
\end{array}
$$

Translation of Type Schemes σ

$$
\begin{array}{rcl}
|\Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to \tau| & := & \Pi\alpha_1 : \mathsf{type} \ldots \alpha_m : \mathsf{type}. \\
 & & \quad \mathsf{term}\,|\tau_1| \to \ldots \to \mathsf{term}\,|\tau_n| \to \mathsf{term}\,|\tau| \\
|\Pi\alpha_1 \ldots \alpha_m.\ \tau_1 \times \ldots \times \tau_n \to o| & := & \Pi\alpha_1 : \mathsf{type} \ldots \alpha_m : \mathsf{type}. \\
 & & \quad \mathsf{term}\,|\tau_1| \to \ldots \to \mathsf{term}\,|\tau_n| \to \mathsf{Prop}
\end{array}
$$

Translation of Terms t

$$
\begin{array}{rcl}
|x| & := & x \\
|f(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n)| & := & f\ |\tau_1|\ \ldots\ |\tau_m|\ |t_1|\ \ldots\ |t_n|
\end{array}
$$

Translation of Formulæ φ

$$
\begin{array}{rcl}
|\top| & := & \top \\
|\bot| & := & \bot \\
|\neg\varphi| & := & \neg\,|\varphi| \\
|\varphi_1 \wedge \varphi_2| & := & |\varphi_1| \wedge |\varphi_2| \\
|\varphi_1 \vee \varphi_2| & := & |\varphi_1| \vee |\varphi_2| \\
|\varphi_1 \Rightarrow \varphi_2| & := & |\varphi_1| \Rightarrow |\varphi_2| \\
|\varphi_1 \Leftrightarrow \varphi_2| & := & |\varphi_1| \Leftrightarrow |\varphi_2| \\
|t_1 =_\tau t_2| & := & |t_1| =_{|\tau|} |t_2| \\
|\forall x : \tau.\ \varphi| & := & \forall\ |\tau|\ (\lambda x : \mathsf{term}\ |\tau|.\ |\varphi|) \\
|\exists x : \tau.\ \varphi| & := & \exists\ |\tau|\ (\lambda x : \mathsf{term}\ |\tau|.\ |\varphi|) \\
|\forall\alpha.\ \varphi| & := & \forall_{\mathsf{type}}\ (\lambda\alpha : \mathsf{type}.\ |\varphi|) \\
|P(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n)| & := & P\ |\tau_1|\ \ldots\ |\tau_m|\ |t_1|\ \ldots |t_n|
\end{array}
$$

Figure 7.5: Translation Functions from Poly-FOL into $\lambda\Pi^{\equiv}$ (Part 1)

Translation of Local Contexts Γ_{L}

$$
\begin{array}{rcl}
|\varnothing| & := & \varnothing \\
|\Gamma_{\mathrm{L}},\ \alpha : \mathsf{Type}| & := & |\Gamma_{\mathrm{L}}|\,,\ \alpha : \mathsf{type} \\
|\Gamma_{\mathrm{L}},\ x : \tau| & := & |\Gamma_{\mathrm{L}}|\,,\ x : \mathsf{term}\ \ |\tau|
\end{array}
$$

Translation of Global Contexts Γ_{G}

$$
\begin{array}{rcl}
|\varnothing| & := & \Gamma_0 \\
|\Gamma_{\mathrm{G}},\ T :: m| & := & |\Gamma_{\mathrm{G}}|\,,\ T : \overbrace{\mathsf{type} \to \ldots \to \mathsf{type}}^{m \text{ times}} \to \mathsf{type} \\
|\Gamma_{\mathrm{G}},\ f : \sigma| & := & |\Gamma_{\mathrm{G}}|\,,\ f : |\sigma| \\
|\Gamma_{\mathrm{G}},\ P : \sigma| & := & |\Gamma_{\mathrm{G}}|\,,\ P : |\sigma| \\
|\Gamma_{\mathrm{G}},\ l \longrightarrow_{\Gamma_{\mathrm{L}}} r| & := & |\Gamma_{\mathrm{G}}|\,,\ |l| \hookrightarrow_{|\Gamma_{\mathrm{L}}|} |r|
\end{array}
$$

Figure 7.6: Translation Functions from Poly-FOL into $\lambda\Pi^{\equiv}$ (Part 2)

Proposition 7.3.1

The translation from Poly-FOL into $\lambda\Pi^{\equiv}$ presented in Fig. 7.5 is correct in the sense that:

1. *If* $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash \tau : \mathsf{Type}$, *then* $|\Gamma_{\mathrm{G}}|\,, |\Gamma_{\mathrm{L}}| \vdash |\tau| : \mathsf{type}$
2. *If* $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash t : \tau$, *then* $|\Gamma_{\mathrm{G}}|\,, |\Gamma_{\mathrm{L}}| \vdash |t| : \mathsf{term}\ \ |\tau|$
3. *If* $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash \varphi : o$, *then* $|\Gamma_{\mathrm{G}}|\,, |\Gamma_{\mathrm{L}}| \vdash |\varphi| : \mathsf{Prop}$

Remark It should be noted that in Prop. 7.3.1, the three typing judgments:

$$\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash \tau : \mathsf{Type} \qquad \Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash t : \tau \qquad \Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash \varphi : o$$

refer to the Poly-FOL typing system of Fig. 6.4, whereas the three others:

$$|\Gamma_{\mathrm{G}}|\,, |\Gamma_{\mathrm{L}}| \vdash |\tau| : \mathsf{type} \qquad |\Gamma_{\mathrm{G}}|\,, |\Gamma_{\mathrm{L}}| \vdash |t| : \mathsf{term}\ \ |\tau| \qquad |\Gamma_{\mathrm{G}}|\,, |\Gamma_{\mathrm{L}}| \vdash |\varphi| : \mathsf{Prop}$$

refer to $\lambda\Pi^{\equiv}$ of Fig. 7.2.

Proof

1. We perform a proof by induction on the structure of τ.

The base case is $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash \alpha : \mathsf{Type}$.

a. By TVar of Fig. 6.4 we have $\alpha : \mathsf{Type} \in \Gamma_{\mathrm{L}}$

b. The translation of Fig. 7.5 tells us that $\alpha : \mathsf{type} \in |\Gamma_{\mathrm{L}}|$, so $|\Gamma_{\mathrm{L}}| \vdash \alpha : \mathsf{type}$

The general case is $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash T(\tau_1, \ldots, \tau_m) : \mathsf{Type}$

a. TConstr of Fig. 6.4 tells us that $T :: m \in \Gamma_{\mathrm{G}}$ and $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash \tau_i : \mathsf{Type}$

b. By induction hypothesis, for $i = 1, \ldots, n$ $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash |\tau_i| : \mathsf{type}$

c. By the translation of Fig. 7.5, $T : \mathsf{type} \to \ldots \to \mathsf{type} \to \mathsf{type} \in |\Gamma_{\mathrm{G}}|$

d. Then, rule App of Fig. 7.2 tells us that $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash T\,|\tau_1| \ldots |\tau_m| : \mathsf{type}$

2. We perform a proof by induction on the structure of t.
The base case is $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash x : \tau$.

a. By rule Var $x : \tau \in \Gamma_{\mathrm{L}}$ thus the translation implies $x : \mathsf{term}\,|\tau| \in |\Gamma_{\mathrm{L}}|$

b. Thus we have $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash x : \mathsf{term}\ \ |\tau|$

The general case is $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash f(\tau_1, \ldots, \tau_m; t_1, \ldots, t_n) : \tau$

a. By rule Fun, we have $f : \Pi\alpha_1 \ldots \alpha_m.\ \tau'_1 \times \ldots \times \tau'_n \to \tau' \in \Gamma_{\mathrm{G}}$ and $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash \tau_i : \mathsf{Type}$ for $i = 1, \ldots, m$

b. By translation, we have $f : \Pi\alpha_1 : \mathsf{type} \ldots \alpha_m : \mathsf{type}.\ \mathsf{term}\,|\tau'_1| \to \ldots \to \mathsf{term}\,|\tau'_n| \to \mathsf{term}\,|\tau'| \in |\Gamma_{\mathrm{G}}|$

c. By item 1. we have $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash |\tau_i| : \mathsf{type}$

d. By rule Fun, we have also, for $i = 1, \ldots, n$ $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash t_i : \tau'_i[\alpha_1/\tau_1, \ldots, \alpha_m/\tau_m]$

e. Thus, by induction hypothesis $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash |t_i| : \mathsf{term}\,|\tau'_i[\alpha_1/\tau_1, \ldots, \alpha_m/\tau_m]|$

f. And by rule App $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash f\,|\tau_1| \ldots |\tau_m|\,|t_1| \ldots |t_n| : \mathsf{term}\,|\tau'[\alpha_1/\tau_1, \ldots, \alpha_m/\tau_m]|$

g. $\tau = \tau'[\alpha_1/\tau_1, \ldots, \alpha_m/\tau_m]$ thus, $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash f\,|\tau_1| \ldots |\tau_m|\,|t_1| \ldots |t_n| : \mathsf{term}\,|\tau|$

3. We perform a proof by induction on the structure of φ.
The base cases $\top$ and $\bot$ are direct and cases for logical connectives and the equality are straightforward.
We present the universal quantification: $\Gamma_{\mathrm{G}}, \Gamma_{\mathrm{L}} \vdash \forall x : \tau.\ \varphi : o$
We have to show that $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash \forall\, |\tau|\, (\lambda x : \mathsf{term}\, |\tau|\, .\, |\varphi|) : \mathsf{Prop}$

a. We have $\forall : \Pi\alpha : \mathsf{type}.\ (\mathsf{term}\ \alpha \to \mathsf{Prop}) \to \mathsf{Prop}$

b. By rule App it is equivalent to show that $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash \lambda x : \mathsf{term}\, |\tau|\, .\, |\varphi| : \mathsf{term}\, |\tau| \to \mathsf{Prop}$

c. By rule Abs, we have to verify that $|\Gamma_{\mathrm{G}}|, |\Gamma_{\mathrm{L}}| \vdash |\varphi| : \mathsf{Prop}$

d. Which is true by induction hypothesis

7.4 Translation of Zenon Modulo Proofs into Dedukti

In this section, we present the embedding of Zenon Modulo proofs into Dedukti. Once again, we define first the Dedukti constants corresponding to the inference rules of $\mathsf{LLproof}^{\equiv}$, then we give the definitions of these constants in Dedukti by means of rewrite rules.

7.4.1 Deep Embedding of $\mathsf{LLproof}^{\equiv}$ into $\lambda\Pi^{\equiv}$

We present in Fig. 7.7 and in Fig. 7.8 the deep embedding of $\mathsf{LLproof}^{\equiv}$ into $\lambda\Pi^{\equiv}$. This is done by defining constants for each inference rule.

Remark The types of the Dedukti constants declared in Fig. 7.7 and Fig. 7.8 translate exactly the corresponding $\mathsf{LLproof}^{\equiv}$ inference rules. If we see sequents as typing contexts, then $\Gamma \vdash \bot$ is $\mathsf{prf}A_1 \to \ldots \to \mathsf{prf}A_n \to \mathsf{prf}\bot$ and the deduction rule itself is seen as a function that associates the conclusion with the premises.

Below, we present the correspondence between the inference rule $\vee$ and the Dedukti constant $\mathsf{R}_\vee$. In this example, we do not consider contexts, the general case will be done later.

Closure Rules and Cut

$$
\begin{aligned}
\mathsf{R}_{\bot} &: \mathsf{prf}\bot \to \mathsf{prf}\bot \\
\mathsf{R}_{\neg\top} &: \mathsf{prf}(\neg\top) \to \mathsf{prf}\bot \\
\mathsf{R}_{Ax} &: \Pi P : \mathsf{Prop}.\ \mathsf{prf}P \to \mathsf{prf}(\neg P) \to \mathsf{prf}\bot \\
\mathsf{R}_{\neq} &: \Pi \alpha : \mathsf{type}.\ \Pi t : \mathsf{term}\ \alpha.\ \mathsf{prf}(t \neq_\alpha t) \to \mathsf{prf}\bot \\
\mathsf{R}_{Sym} &: \Pi \alpha : \mathsf{type}.\ \Pi t, u : \mathsf{term}\ \alpha.\ \mathsf{prf}(t =_\alpha u) \to \mathsf{prf}(u \neq_\alpha t) \to \mathsf{prf}\bot \\
\mathsf{R}_{Cut} &: \Pi P : \mathsf{Prop}.\ (\mathsf{prf}P \to \mathsf{prf}\bot) \to (\mathsf{prf}(\neg P) \to \mathsf{prf}\bot) \to \mathsf{prf}\bot
\end{aligned}
$$

Quantifier-free Rules

$$
\begin{aligned}
\mathsf{R}_{\neg\neg} &: \Pi P : \mathsf{Prop}. && (\mathsf{prf}P \to \mathsf{prf}\bot) \to \mathsf{prf}(\neg\neg P) \to \mathsf{prf}\bot \\
\mathsf{R}_{\wedge} &: \Pi P, Q : \mathsf{Prop}. && (\mathsf{prf}P \to \mathsf{prf}Q \to \mathsf{prf}\bot) \to \mathsf{prf}(P \wedge Q) \to \mathsf{prf}\bot \\
\mathsf{R}_{\vee} &: \Pi P, Q : \mathsf{Prop}. && (\mathsf{prf}P \to \mathsf{prf}\bot) \\
&&& \to (\mathsf{prf}Q \to \mathsf{prf}\bot) \to \mathsf{prf}(P \vee Q) \to \mathsf{prf}\bot \\
\mathsf{R}_{\Rightarrow} &: \Pi P, Q : \mathsf{Prop}. && (\mathsf{prf}(\neg P) \to \mathsf{prf}\bot) \\
&&& \to (\mathsf{prf}Q \to \mathsf{prf}\bot) \to \mathsf{prf}(P \Rightarrow Q) \to \mathsf{prf}\bot \\
\mathsf{R}_{\Leftrightarrow} &: \Pi P, Q : \mathsf{Prop}. && (\mathsf{prf}(\neg P) \to \mathsf{prf}(\neg Q) \to \mathsf{prf}\bot) \\
&&& \to (\mathsf{prf}P \to \mathsf{prf}Q \to \mathsf{prf}\bot) \to \mathsf{prf}(P \Leftrightarrow Q) \to \mathsf{prf}\bot \\
\mathsf{R}_{\neg\wedge} &: \Pi P, Q : \mathsf{Prop}. && (\mathsf{prf}(\neg P) \to \mathsf{prf}\bot) \\
&&& \to (\mathsf{prf}(\neg Q) \to \mathsf{prf}\bot) \to \mathsf{prf}(\neg(P \wedge Q)) \to \mathsf{prf}\bot \\
\mathsf{R}_{\neg\vee} &: \Pi P, Q : \mathsf{Prop}. && (\mathsf{prf}(\neg P) \to \mathsf{prf}(\neg Q) \to \mathsf{prf}\bot) \\
&&& \to \mathsf{prf}(\neg(P \vee Q)) \to \mathsf{prf}\bot \\
\mathsf{R}_{\neg\Rightarrow} &: \Pi P, Q : \mathsf{Prop}. && (\mathsf{prf}P \to \mathsf{prf}(\neg Q) \to \mathsf{prf}\bot) \\
&&& \to \mathsf{prf}(\neg(P \Rightarrow Q)) \to \mathsf{prf}\bot \\
\mathsf{R}_{\neg\Leftrightarrow} &: \Pi P, Q : \mathsf{Prop}. && (\mathsf{prf}(\neg P) \to \mathsf{prf}Q \to \mathsf{prf}\bot) \\
&&& \to (\mathsf{prf}P \to \mathsf{prf}(\neg Q) \to \mathsf{prf}\bot) \\
&&& \to \mathsf{prf}(\neg(P \Leftrightarrow Q)) \to \mathsf{prf}\bot
\end{aligned}
$$

Figure 7.7: Deep Embedding of $\mathsf{LLproof}^{\equiv}$ into $\lambda\Pi^{\equiv}$ (Part 1)

Given two propositions P and Q, proving the Poly-FOL sequent:

$$P \vee Q \vdash \bot$$

is done in $\mathsf{LLproof}^{\equiv}$ by applying the $\vee$ rule, resulting in the proof node:

$$\frac{P \vdash \bot \qquad Q \vdash \bot}{P \vee Q \vdash \bot}\,\vee$$

Then, we have to prove the two sequents $P \vdash \bot$ and $Q \vdash \bot$.

Quantifier and Special Rules

$$
\begin{array}{lll}
\mathsf{R}_{\forall} & : \Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ \Pi t : \mathsf{term}\ \alpha. & \\
 & \quad (\mathsf{prf}(P\ t) \to \mathsf{prf}\bot) \to \mathsf{prf}(\forall\ \alpha\ P) \to \mathsf{prf}\bot & \\
\mathsf{R}_{\neg\exists} & : \Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ \Pi t : \mathsf{term}\ \alpha. & \\
 & \quad (\mathsf{prf}(\neg(P\ t)) \to \mathsf{prf}\bot) \to \mathsf{prf}(\neg(\exists\ \alpha\ P)) \to \mathsf{prf}\bot & \\
\mathsf{R}_{\exists} & : \Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ (\Pi t : \mathsf{term}\ \alpha.\ \mathsf{prf}(P\ t) \to \mathsf{prf}\bot) & \\
 & \quad \to \mathsf{prf}(\exists\ \alpha\ P) \to \mathsf{prf}\bot & \\
\mathsf{R}_{\neg\forall} & : \Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ (\Pi t : \mathsf{term}\ \alpha.\ \mathsf{prf}(\neg(P\ t)) \to \mathsf{prf}\bot) & \\
 & \quad \to \mathsf{prf}(\neg(\forall\ \alpha\ P)) \to \mathsf{prf}\bot & \\
\mathsf{R}_{\forall_{\mathsf{type}}} & : \Pi P : (\mathsf{type} \to \mathsf{Prop}).\ \Pi\alpha : \mathsf{type}. & \\
 & \quad (\mathsf{prf}(P\ \alpha) \to \mathsf{prf}\bot) \to \mathsf{prf}(\forall_{\mathsf{type}}\ P) \to \mathsf{prf}\bot & \\
\mathsf{R}_{Subst} & : \Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ \Pi t_1, t_2 : \mathsf{term}\ \alpha.\ (\mathsf{prf}(t_1 \neq_\alpha t_2) & \\
 & \quad \to \mathsf{prf}\bot) \to (\mathsf{prf}(P\ t_2) \to \mathsf{prf}\bot) \to \mathsf{prf}(P\ t_1) \to \mathsf{prf}\bot &
\end{array}
$$

Figure 7.8: Deep Embedding of $\mathsf{LLproof}^{\equiv}$ into $\lambda\Pi^{\equiv}$ (Part 2)

The corresponding Dedukti constant $\mathsf{R}_{\vee}$ has type:

$$\mathsf{R}_{\vee} : \Pi P, Q : \mathsf{Prop}.\ (\mathsf{prf}P \to \mathsf{prf}\bot) \to (\mathsf{prf}Q \to \mathsf{prf}\bot) \to \mathsf{prf}(P \vee Q) \to \mathsf{prf}\bot$$

As an informal explanation, we can identify the turnstile $\vdash$ with the logical implication. This means that, for two given propositions P and Q, if we give to $\mathsf{R}_{\vee}$ a proof that P implies $\bot$ and a proof that Q implies $\bot$, then we obtain a proof that $P \vee Q$ implies $\bot$.

7.4.2 Shallow Embedding of $\mathsf{LLproof}^{\equiv}$ into $\lambda\Pi^{\equiv}$

We present in Fig. 7.9 and Fig. 7.10 and Fig. 7.11 the shallow embedding of $\mathsf{LLproof}^{\equiv}$ derivation rules into $\lambda\Pi^{\equiv}$. This amounts to turn the static constants just defined in Fig. 7.7 and Fig. 7.8 into rewrite rules.

Remark The shallow embedding gives a meaning to constants. For instance, if we consider the constant $\mathsf{R}_{\bot}$, the deep embedding of Fig. 7.7 is:

$$\mathsf{R}_{\bot}\ :\ \mathsf{prf}\bot \to \mathsf{prf}\bot$$

The type of this declaration corresponds to the statement "false implies false". For the moment, this statement is just an axiom.

$ExMid(P : \mathsf{Prop}) \ : \ \Pi Q : \mathsf{Prop}.\ (\mathsf{prf}P \to \mathsf{prf}Q) \to (\mathsf{prf}(\neg P) \to \mathsf{prf}Q) \to \mathsf{prf}Q$

$NNPP(P : \mathsf{Prop}) \ : \ \mathsf{prf}(\neg\neg P) \to \mathsf{prf}P$
$:= \lambda H_1 : \mathsf{prf}(\neg\neg P).\ ExMid\ P\ P\ (\lambda H_2 : \mathsf{prf}P.\ H_2)\ (\lambda H_3 : \mathsf{prf}(\neg P).\ H_1\ H_3\ P)$

$Contr(P : \mathsf{Prop}, Q : \mathsf{Prop}) \ : \ \mathsf{prf}(P \Rightarrow Q) \to \mathsf{prf}(\neg Q \Rightarrow \neg P)$
$:= \lambda H_1 : \mathsf{prf}(P \Rightarrow Q).\ \lambda H_2 : \mathsf{prf}(\neg Q).\ \lambda H_3 : \mathsf{prf}P.\ H_2\ (H_1\ H_3)$

Closure Rules and Cut

$[\,]\ \mathsf{R}_{\bot}$
$\hookrightarrow \lambda H : \mathsf{prf}\bot.\ H$

$[\,]\ \mathsf{R}_{\neg\bot}$
$\hookrightarrow \lambda H_1 : \mathsf{prf}(\neg\top).\ H_1\ (\lambda P : \mathsf{Prop}.\ \lambda H_2 : \mathsf{prf}P.\ H_2)$

$[P : \mathsf{Prop}]\ \mathsf{R}_{Ax}\ P$
$\hookrightarrow \lambda H_1 : \mathsf{prf}P.\ \lambda H_2 : \mathsf{prf}(\neg P).\ H_2\ H_1$

$[\alpha : \mathsf{type}, t : \mathsf{term}\ \alpha]\ \mathsf{R}_{\neq}\ \alpha\ t$
$\hookrightarrow \lambda H_1 : \mathsf{prf}(t \neq_\alpha t).\ H_1\ (\lambda z : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ \lambda H_2 : \mathsf{prf}\ (z\ t).\ H_2)$

$[\alpha : \mathsf{type}, t : \mathsf{term}\ \alpha, u : \mathsf{term}\ \alpha]\ \mathsf{R}_{Sym}\ \alpha\ t\ u$
$\hookrightarrow \lambda H_1 : \mathsf{prf}(t =_\alpha u).\ \lambda H_2 : \mathsf{prf}(u \neq_\alpha t).\ H_2\ (\lambda z : (\mathsf{term}\ \alpha \to \mathsf{Prop}).$
$\quad \lambda H_3 : \mathsf{prf}(z\ u).\ H_1\ (\lambda x : \mathsf{term}\ \alpha.\ (z\ x) \Rightarrow (z\ t))\ (\lambda H_4 : \mathsf{prf}(z\ t).\ H_4)\ H_3)$

$[P : \mathsf{Prop}]\ \mathsf{R}_{Cut}\ P$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}P \to \mathsf{prf}\bot).\ \lambda H_2 : (\mathsf{prf}(\neg P) \to \mathsf{prf}\bot).\ H_2\ H_1$

Quantifier-free Rules

$[P : \mathsf{Prop}]\ \mathsf{R}_{\neg\neg}\ P$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}P \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}(\neg\neg P).\ H_2\ H_1$

$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\wedge}\ P\ Q$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}P \to \mathsf{prf}Q \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}(P \wedge Q).\ H_2\ \bot\ H_1$

Figure 7.9: Shallow Embedding of $\mathsf{LLproof}^{\equiv}$ into $\lambda\Pi^{\equiv}$ (Part 1)

Quantifier-free Rules (Sequel)

$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\vee}\ P\ Q$
$\hookrightarrow \lambda H_1 : (\mathsf{prf} P \to \mathsf{prf}\bot).\ \lambda H_2 : (\mathsf{prf} Q \to \mathsf{prf}\bot).$
$\lambda H_3 : \mathsf{prf}(P \vee Q).\ H_3\ \bot\ H_1\ H_2$

$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\Rightarrow}\ P\ Q$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}(\neg P) \to \mathsf{prf}\bot).\ \lambda H_2 : (\mathsf{prf} Q \to \mathsf{prf}\bot).$
$\lambda H_3 : \mathsf{prf}(P \Rightarrow Q).\ H_1(Contr\ P\ Q\ H_3\ H_2)$

$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\Leftrightarrow}\ P\ Q$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}(\neg P) \to \mathsf{prf}(\neg Q) \to \mathsf{prf}\bot).\ \lambda H_2 : (\mathsf{prf} P \to \mathsf{prf} Q \to \mathsf{prf}\bot).$
$\lambda H_3 : \mathsf{prf}(P \Leftrightarrow Q).\ H_3\ \bot\ (\lambda H_4 : (\mathsf{prf} P \to \mathsf{prf} Q).\ \lambda H_5 : (\mathsf{prf} Q \to \mathsf{prf} P).$
$(H_1\ (Contr\ P\ Q\ H_4\ (\lambda H_6 : \mathsf{prf} Q.\ (H_2\ (H_5\ H_6))\ H_6)))$
$(\lambda H_7 : \mathsf{prf} Q.\ (H_2\ (H_5\ H_7))\ H_7))$

$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\neg\wedge}\ P\ Q$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}(\neg P) \to \mathsf{prf}\bot).\ \lambda H_2 : (\mathsf{prf}(\neg Q) \to \mathsf{prf}\bot).$
$\lambda H_3 : \mathsf{prf}(\neg(P \wedge Q)).\ H_1\ (\lambda H_5 : \mathsf{prf} P.\ H_2\ (\lambda H_6 : \mathsf{prf} Q.\ H_3\ (\lambda Z : \mathsf{Prop}.$
$\lambda H_4 : (\mathsf{prf} P \to \mathsf{prf} Q \to \mathsf{prf} Z).\ H_4\ H_5\ H_6)))$

$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\neg\vee}\ P\ Q$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}(\neg P) \to \mathsf{prf}(\neg Q) \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}(\neg(P \vee Q)).$
$H_1\ (Contr\ P\ (P \vee Q)\ (\lambda H_3 : \mathsf{prf} P.\ \lambda Z : \mathsf{Prop}.\ \lambda H_4 : (\mathsf{prf} P \to \mathsf{prf} Z).$
$\lambda H_5 : (\mathsf{prf} Q \to \mathsf{prf} Z).\ H_4\ H_3)\ H_2)\ (Contr\ Q\ (P \vee Q)\ (\lambda H_6 : \mathsf{prf} Q.$
$\lambda Z : \mathsf{Prop}.\ \lambda H_7 : (\mathsf{prf} P \to \mathsf{prf} Z).\ \lambda H_8 : (\mathsf{prf} Q \to \mathsf{prf} Z).\ H_8\ H_6)\ H_2)$

$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\neg\Rightarrow}\ P\ Q$
$\hookrightarrow \lambda H_1 : (\mathsf{prf} P \to \mathsf{prf}(\neg Q) \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}(\neg(P \Rightarrow Q)).\ H_2\ (\lambda H_3 : \mathsf{prf} P.$
$(H_1\ H_3)\ (\lambda H_4 : \mathsf{prf} Q.\ H_2\ (\lambda H_5 : \mathsf{prf} P.\ H_4))\ Q)$

Figure 7.10: Shallow Embedding of $\mathsf{LLproof}^{\equiv}$ into $\lambda\Pi^{\equiv}$ (Part 2)

Quantifier-free Rules (Sequel)

$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\neg\Leftrightarrow}\ P\ Q$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}(\neg P) \to \mathsf{prf}(\neg Q)).\ \lambda H_2 : (\mathsf{prf} P \to \mathsf{prf}(\neg\neg Q)).$
$\lambda H_3 : \mathsf{prf}(\neg(P \Leftrightarrow Q)).\ (\lambda H_4 : \mathsf{prf}(\neg P).\ H_3\ (\lambda Z : \mathsf{Prop}.$
$\lambda H_5 : (\mathsf{prf}(P \Rightarrow Q) \to \mathsf{prf}(Q \Rightarrow P) \to \mathsf{prf} Z).\ H_5\ (\lambda H_6 : \mathsf{prf} P.$
$H_4\ H_6\ Q)\ (\lambda H_7 : \mathsf{prf} Q.\ H_1\ H_4\ H_7\ P)))\ (\lambda H_8 : \mathsf{prf} P.\ H_2\ H_8\ (\lambda H_9 : \mathsf{prf} Q.$
$H_3\ (\lambda Z : \mathsf{Prop}.\ \lambda H_{10} : (\mathsf{prf}(P \Rightarrow Q) \to \mathsf{prf}(Q \Rightarrow P) \to \mathsf{prf} Z).\ H_{10}$
$(\lambda H_{11} : \mathsf{prf} P.\ H_9)\ (\lambda H_{12} : \mathsf{prf} Q.\ H_8))))$

Quantifier Rules and Subst

$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}, t : \mathsf{term}\ \alpha]\ \mathsf{R}_{\forall}\ \alpha\ P\ t$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}(P\ t) \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}(\forall\ \alpha\ P).\ H_1\ (H_2\ t)$

$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}, t : \mathsf{term}\ \alpha]\ \mathsf{R}_{\neg\exists}\ \alpha\ P\ t$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}(\neg(P\ t)) \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}(\neg(\exists\ \alpha\ P)).\ H_1\ (\lambda H_4 : \mathsf{prf}(P\ t).$
$H_2\ (\lambda Z : \mathsf{Prop}.\ \lambda H_3 : (x : \mathsf{term}\ \alpha \to \mathsf{prf}(P\ x) \to \mathsf{prf} Z).\ H_3\ t\ H_4))$

$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}]\ \mathsf{R}_{\exists}\ \alpha\ P$
$\hookrightarrow \lambda H_1 : (t : \mathsf{term}\ \alpha \to \mathsf{prf}(P\ t) \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}(\exists\ \alpha\ P).\ H_2\ \bot\ H_1$

$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}]\ \mathsf{R}_{\neg\forall}\ \alpha\ P$
$\hookrightarrow \lambda H_1 : (t : \mathsf{term}\ \alpha \to \mathsf{prf}(\neg(P\ t)) \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}(\neg(\forall\ \alpha\ P)).$
$H_2\ (\lambda t : \mathsf{term}\ \alpha.\ NNPP\ (P\ t)\ (H_1\ t))$

$[P : \mathsf{type} \to \mathsf{Prop}, \alpha : \mathsf{type}]\ \mathsf{R}_{\forall_{\mathsf{type}}}\ P\ \alpha$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}(P\ \alpha) \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}(\forall_{\mathsf{type}}\ P).\ H_1\ (H_2\ \alpha)$

$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}, t_1 : \mathsf{term}\ \alpha, t_2 : \mathsf{term}\ \alpha]\ \mathsf{R}_{Subst}\ \alpha\ P\ t_1\ t_2$
$\hookrightarrow \lambda H_1 : (\mathsf{prf}(t_1 \neq_\alpha t_2) \to \mathsf{prf}\bot).\ \lambda H_2 : (\mathsf{prf}(P\ t_2) \to \mathsf{prf}\bot).$
$\lambda H_3 : \mathsf{prf}(P\ t_1).\ H_1\ (\lambda H_4 : \mathsf{prf}(t_1 =_\alpha t_2).\ H_2\ (H_4\ P\ H_3))$

Figure 7.11: Shallow Embedding of $\mathsf{LLproof}^{\equiv}$ into $\lambda\Pi^{\equiv}$ (Part 3)

In Fig. 7.9, we turn the type declaration into the rewrite rule:

$$[\,]\ \mathsf{R}_{\bot} \hookrightarrow \lambda H : \mathsf{prf}\bot.\ H$$

where [ ] denotes the (empty) local context Δ. The term $\lambda H : \mathsf{prf}\bot.\ H$ is an inhabitant of the type $\mathsf{prf}\bot \to \mathsf{prf}\bot$. In $\lambda\Pi^{\equiv}$, providing a term of a particular type can be seen as a proof of the corresponding statement. Consequently, the statement "false implies false" is a proved lemma.

We introduce in Fig. 7.9 the constant $ExMid$, which corresponds to the law of the excluded middle, and we do not provide any definition for it – it remains as a type declaration. As $\lambda\Pi^{\equiv}$ is a constructive framework, it does not enjoy this property for free, and we must add it as an axiom. This is the only axiom that is added to Dedukti in our work. We need it to prove the lemma called $NNPP$ in Fig. 7.9, a direct corollary that allows us to prove the LLproof$^{\equiv}$ rule $\neg\forall$, a classical rule.

We also define a lemma $Contr$, corresponding to the law of the contraposition, as a convenience to prove some LLproof$^{\equiv}$ inference rules also.

All the proofs of LLproof$^{\equiv}$ inference rules given in Fig. 7.9 and Fig. 7.10 and Fig. 7.11 have been well checked by Dedukti.

7.4.3 Translation of LLproof$^{\equiv}$ Proofs

We now present the extension of the translation functions of Fig. 7.5 and Fig. 7.6 to LLproof$^{\equiv}$ proofs. We first introduce the translation of LLproof$^{\equiv}$ sequents into typing contexts.

$$|\varphi_1, \ldots, \varphi_n \vdash \bot| \ :=\ x_{\varphi_1} : \mathsf{prf}\,|\varphi_1|, \ldots, x_{\varphi_n} : \mathsf{prf}\,|\varphi_n|$$

The formulæ $\varphi_1, \ldots, \varphi_n$ correspond to the axioms and hypotheses of the problem. The translation of an axiom φ is done by defining a new constant x_φ which has the type $\mathsf{prf}\,|\varphi|$.

The general function to translate proofs is:

$$\left|\frac{\dfrac{\Pi_1}{\Gamma_1, H_1^1, \ldots, H_1^m \vdash \bot} \quad \ldots \quad \dfrac{\Pi_n}{\Gamma_n, H_n^1, \ldots, H_n^q \vdash \bot}}{\Gamma, C_1, \ldots, C_p \vdash \bot}\,\mathrm{Rule}(\mathrm{Arg}_1, \ldots, \mathrm{Arg}_r)\right|$$

$:=$

$$\begin{array}{l}
\mathsf{R}_{\mathrm{Rule}}\,|\mathrm{Arg}_1|\ \ldots\ |\mathrm{Arg}_r| \\
\quad (\lambda x_{H_1^1} : \mathsf{prf}\,|H_1^1|.\ \ldots.\ \lambda x_{H_1^m} : \mathsf{prf}\,|H_1^m|.\ |\Pi_1|) \\
\quad \vdots \\
\quad (\lambda x_{H_n^1} : \mathsf{prf}\,|H_n^1|.\ \ldots.\ \lambda x_{H_n^q} : \mathsf{prf}\,|H_n^q|.\ |\Pi_n|) \\
\quad x_{C_1}.\ \ldots.\ x_{C_p}
\end{array}$$

where $x_{C_1}, \ldots, x_{C_p}$ are variables declared of type $\mathsf{prf}\,|C_1|, \ldots, \mathsf{prf}\,|C_p|$ respectively, $H_1^1, \ldots, H_1^m, \ldots, H_n^1, \ldots, H_n^q$ are all the subformulæ of $C_1, \ldots, C_p$ generated by the application of the rule, and $\Gamma, \Gamma_1, \ldots, \Gamma_n$ are contexts.

For instance, we want to translate the following proof tree:

$$\frac{\dfrac{\Pi_P}{\Gamma, P \vee Q, P \vdash \bot} \quad \dfrac{\Pi_Q}{\Gamma, P \vee Q, Q \vdash \bot}}{\Gamma, P \vee Q \vdash \bot}\,\vee(P, Q)$$

By applying the translation of Fig. 7.5 and Fig. 7.6 and its extension to proofs given above, we obtain the Dedukti proof term:

$$\mathsf{R}_{\vee}\,|P|\,|Q|\,(\lambda x_P : \mathsf{prf}|P|.\ |\Pi_P|)\ (\lambda x_Q : \mathsf{prf}|Q|.\ |\Pi_Q|)\ x_{P \vee Q}$$

It should be noted that we do not have to make a λ abstraction over $P \vee Q$ again since we already have a constant $x_{P \vee Q}$ coming from a previous λ abstraction or an axiom.

Remark We check that a $\mathsf{LLproof}^{\equiv}$ proof Π is a valid proof of the $\mathsf{LLproof}^{\equiv}$ sequent $\Gamma \vdash \bot$, by checking the $\lambda\Pi^{\equiv}$ typing judgment $|\Gamma| \vdash |\Pi| : \mathsf{prf}\bot$.

7.5 Proof Certificate Example

To illustrate the certification of proofs with Dedukti, we present an example in B set theory. The Poly-FOL theory $\mathcal{T}$ consists of three axioms defining membership to the

empty-set, membership to set difference and equality between sets which have been turned into rewrite rules. It should be noted that the equality considered here is not the common equality $=$, but a particular predicate symbol defined only for sets. We denote it $\overset{\mathsf{set}}{=}$ and use the extensionality property of it (axiom SET4 of the B theory). Consequently, the signature of $\overset{\mathsf{set}}{=}$ is different than the one of the common equality $=$.

In the following signatures, We do not indicate type arguments, only term arguments with the symbol -:

$$\begin{array}{rcl} \mathsf{set} & :: & 1 \\ \text{- } \in \text{ -} & : & \Pi\alpha.\ \alpha \times \mathsf{set}(\alpha) \to o \\ \text{- } \overset{\mathsf{set}}{=} \text{ -} & : & \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to o \\ \varnothing & : & \Pi\alpha.\ \mathsf{set}(\alpha) \\ \text{- } - \text{ -} & : & \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to \mathsf{set}(\alpha) \end{array}$$

And the rewrite rules, where type arguments are subscripted:

$$\begin{array}{rcl} s \overset{\mathsf{set}}{=}_\alpha t & \longrightarrow_{(\alpha:\mathsf{Type},s:\mathsf{set}(\alpha),t:\mathsf{set}(\alpha))} & \forall x : \alpha.\ x \in_\alpha s \Leftrightarrow x \in_\alpha t \\ x \in_\alpha \varnothing_\alpha & \longrightarrow_{(\alpha:\mathsf{Type},x:\alpha)} & \bot \\ x \in_\alpha s -_\alpha t & \longrightarrow_{(\alpha:\mathsf{Type},s:\mathsf{set}(\alpha),t:\mathsf{set}(\alpha))} & x \in_\alpha s \wedge x \notin_\alpha t \end{array}$$

We prove the formula, given a type τ:

$$\forall s : \mathsf{set}(\tau).\ s -_\tau s \overset{\mathsf{set}}{=}_\tau \varnothing_\tau$$

The $\mathsf{LLproof}^{\equiv}$ proof tree generated by Zenon Modulo is (we omit to repeat the context Γ):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(c_2 \in_\tau c_1 -_\tau c_1), c_2 \in_\tau \varnothing_\tau \vdash \bot}\bot \quad \dfrac{\dfrac{}{c_2 \in_\tau c_1, c_2 \notin_\tau c_1 \vdash \bot}\mathrm{Ax}}{c_2 \in_\tau c_1 -_\tau c_1, \neg(c_2 \in_\tau \varnothing_\tau) \vdash \bot}\wedge}{\neg((c_2 \in_\tau c_1 -_\tau c_1) \Leftrightarrow (c_2 \in_\tau \varnothing_\tau)) \vdash \bot}\neg\Leftrightarrow}{\neg(c_1 -_\tau c_1 \overset{\mathsf{set}}{=}_\tau \varnothing_\tau) \vdash \bot}\neg\forall}{\neg(\forall s : \mathsf{set}(\tau).\ s -_\tau s \overset{\mathsf{set}}{=}_\tau \varnothing_\tau) \vdash \bot}\neg\forall$$

Remark It should be noted that some normalization steps are hidden in the proof. The first normalization occurs before the third application from the bottom of rule $\neg\forall$, where the atomic formula $c_1 -_\tau c_1 \overset{\mathsf{set}}{=}_\tau \varnothing_\tau$ has been rewritten into $\forall x : \tau.\ (x \in_\tau c_1 -_\tau c_1) \Leftrightarrow (x \in_\tau \varnothing_\tau)$.

A second normalization occurs before the application of the closure rule $\bot$ on the left-hand branch, and the last one occurs before the application of the rule $\wedge$ on the right-hand branch.

We obtain the Dedukti proof certificate of Fig. 7.12 and Fig. 7.13.

$$
\begin{aligned}
\mathsf{set}\ \text{-}\ &: \mathsf{type} \to \mathsf{type} \\
\tau\ &: \mathsf{type} \\
\text{-} \in_{\text{-}} \text{-}\ &: \Pi\alpha : \mathsf{type}.\ \mathsf{term}\ \alpha \to \mathsf{term}\ \mathsf{set}\ \alpha \to \mathsf{Prop} \\
\text{-} \overset{\mathsf{set}}{=}_{\text{-}} \text{-}\ &: \Pi\alpha : \mathsf{type}.\ \mathsf{set}\ \alpha \to \mathsf{set}\ \alpha \to \mathsf{Prop} \\
\varnothing_{\text{-}}\ &: \Pi\alpha : \mathsf{type}.\ \mathsf{set}\ \alpha \\
\text{-} -_{\alpha} \text{-}\ &: \Pi\alpha : \mathsf{type}.\ \mathsf{set}\ \alpha \to \mathsf{set}\ \alpha \to \mathsf{set}\ \alpha \\
s \overset{\mathsf{set}}{=}_{\alpha} t &\hookrightarrow \forall(\alpha)(\lambda x : (\mathsf{term}\ \alpha).\ x \in_{\alpha} s \Leftrightarrow x \in_{\alpha} t) \\
x \in_{\alpha} \varnothing_{\alpha} &\hookrightarrow \bot \\
x \in_{\alpha} s -_{\alpha} t &\hookrightarrow x \in_{\alpha} s \wedge x \notin_{\alpha} t
\end{aligned}
$$

Figure 7.12: Dedukti Proof Certificate in B Set Theory (Part 1)

$$
\begin{array}{l}
Goal : \mathsf{prf}(\neg(\forall(\mathsf{set}\ \tau)(\lambda s : (\mathsf{term\ set}\ \tau).s -_\tau s \overset{\mathsf{set}}{=}_\tau \varnothing_\tau))) \to \mathsf{prf}\bot \\
[]Goal \hookrightarrow \\
\lambda x_2 : \mathsf{prf}(\neg(\forall(\mathsf{set}\ \tau)(\lambda s : (\mathsf{term\ set}\ \tau).s -_\tau s \overset{\mathsf{set}}{=}_\tau \varnothing_\tau))). \\
\mathsf{R}_{\neg\forall}(\mathsf{set}\ \tau) \\
\quad (\lambda s : (\mathsf{term\ set}\ \tau).s -_\tau s \overset{\mathsf{set}}{=}_\tau \varnothing_\tau) \\
\quad (\lambda c_1 : (\mathsf{term\ set}\ \tau). \\
\quad \lambda x_3 : \mathsf{prf}(c_1 -_\tau c_1 \neq_{\mathsf{set}\ \tau} \varnothing_\tau). \\
\quad \mathsf{R}_{\neg\forall}(\tau) \\
\qquad (\lambda x : (\mathsf{term}\ \tau).(x \in_\tau c_1 -_\tau c_1) \Leftrightarrow (x \in_\tau \varnothing_\tau)) \\
\qquad (\lambda c_2 : (\mathsf{term}\ \tau). \\
\qquad \lambda x_4 : \mathsf{prf}(\neg((c_2 \in_\tau c_1 -_\tau c_1) \Leftrightarrow (c_2 \in_\tau \varnothing_\tau))). \\
\qquad \mathsf{R}_{\neg\Leftrightarrow}(c_2 \in_\tau c_1 -_\tau c_1) \\
\qquad\quad (c_2 \in_\tau \varnothing_\tau) \\
\qquad\quad (\lambda x_5 : \mathsf{prf}(\neg(c_2 \in_\tau c_1 -_\tau c_1)). \\
\qquad\quad \lambda x_6 : \mathsf{prf}(c_2 \in_\tau \varnothing_\tau). \\
\qquad\quad \mathsf{R}_\bot x_6) \\
\qquad\quad (\lambda x_7 : \mathsf{prf}(c_2 \in_\tau c_1 -_\tau c_1). \\
\qquad\quad \lambda x_8 : \mathsf{prf}(\neg(c_2 \in_\tau \varnothing_\tau)). \\
\qquad\quad \mathsf{R}_\wedge(c_2 \in_\tau c_1) \\
\qquad\qquad (c_2 \notin_\tau c_1) \\
\qquad\qquad (\lambda x_9 : \mathsf{prf}(c_2 \in_\tau c_1). \\
\qquad\qquad \lambda x_{10} : \mathsf{prf}(c_2 \notin_\tau c_1). \\
\qquad\qquad \mathsf{R}_{Ax}(c_2 \notin_\tau c_1) \\
\qquad\qquad\quad x_{10} \\
\qquad\qquad\quad x_9 \\
\qquad\qquad x_7) \\
\qquad\quad x_4) \\
\qquad x_3) \\
\quad x_2)
\end{array}
$$

Figure 7.13: Dedukti Proof Certificate in B Set Theory (Part 2)

Chapter 8

The BWare Project

We present in this chapter the BWare project and the experimental results of our work.

We make a short presentation of the BWare project in Sec. 8.1. In Sec. 8.2, we introduce the different tools involved in the BWare toolchain.

In Sec. 8.3, we present the B set theory expressed as a Poly-FOL rewrite system. This contribution is a personal work and it has been published in [Bury, Delahaye, Doligez, Halmagrand, and Hermant 2015b].

In Sec. 8.4, we present the experimental results obtained over the BWare benchmark. In particular, we compare our tool Zenon Modulo to other state-of-the-art automated deduction tools. These experimental results have been published in [Bury, Delahaye, Doligez, Halmagrand, and Hermant 2015b].

8.1 Presentation of the BWare Project

The BWare project [Delahaye, Dubois, Marché, and Mentré 2014] is an industrial research project which intends to provide a mechanized framework to help the automated verification of proof obligations coming from the development of industrial applications using the B Method. The BWare consortium gathers academic entities – Cedric [Centre d'Études et de Recherche en Informatique et Communications], LRI [Laboratoire de Recherche en Informatique] and Inria [Inria] – as well as industrial partners – Mitsubishi Electric R&D [Mitsubishi Electric R&D Centre Europe], ClearSy [ClearSy] and OCamlPro [OCamlPro].

The methodology of the BWare project consists in building a generic platform of verification relying on different ATPs, such as first-order provers, and SMT solvers. This platform is built upon Why3 [Bobot, Filliâtre, Marché, and Paskevich 2011], a platform for deductive program verification which provides a rich language for specification and programming, called WhyML, and that relies on external provers to discharge verification conditions. The automated deduction tools used in the BWare framework are the ATP Zenon Modulo, the ATP iProver Modulo [Burel 2011] and the SMT solver Alt-Ergo [Bobot, Conchon, Contejean, Iguernelala, Lescuyer, and Mebsout 2013]. The diversity of these theorem provers aims to allow a wide panel proof obligations to be automatically verified by the platform.

Beyond the multi-tool aspect of this methodology, the originality of BWare resides in the requirement for the verification tools to produce proof objects, which have to be checked independently.

To test the BWare platform, a large collection of proof obligations is provided by the industrial partners of the project, which develop tools implementing the B Method and applications involving the use of the B Method.

8.2 The BWare Toolchain

From B proof obligations to proof certificates, the BWare project relies on a series of tools. We present in the following the toolchain involved.

8.2.1 Generating Proof Obligations

The generation of B proof obligations involves two different tools, the B integrated development environment Atelier B [ClearSy 2013] and a translation tool called bpo2why [Mentré, Marché, Filliâtre, and Asuka 2012].

8.2.1.1 Atelier B

Atelier B [ClearSy 2013] is developed and distributed by ClearSy. This tool implements the B Method and has been designed to cover all the development stages of B projects. In

particular, it allows to write specifications and refinements, to generate the corresponding proof obligations, to prove these proof obligations automatically or interactively, and finally to extract the resulting source code of a B project.

In the context of BWare, Atelier B is used to generate the proof obligations in their native format, denoted B-PO format.

8.2.1.2 bpo2why

bpo2why [Mentré, Marché, Filliâtre, and Asuka 2012] is a tool developed by Mitsubishi Electric R&D for the BWare project. It allows to translate proof obligations from the B PO format into WhyML, the language of the Why3 platform. For the moment, bpo2why is a proprietary software.

The translation scheme implemented by bpo2why is not a simple syntactic translation. It performs some non-trivial type inferences on proof obligations to rebuild typed proof obligations in the WhyML format. It also eliminates some non-first-order B constructions, like sets defined by comprehension.

8.2.2 Proving Proof Obligations

The second stage of the BWare toolchain concerns the verification of proof obligations. Why3 is the central tool of the platform and calls the automated deduction tools Zenon Modulo, iProver Modulo and Alt-Ergo.

8.2.2.1 The Why3 Platform

Why3 [Bobot, Filliâtre, Marché, and Paskevich 2011], a project carried by LRI, is a platform dedicated to program verification and which relies on external provers. The native language of Why3, called WhyML, is based on polymorphic first-order logic and is close to Poly-FOL (see Sec. 6.4.2.1). Besides external provers, Why3 is provided with the SMT solver Alt-Ergo.

In the context of BWare, it is used to call provers on proof obligations. It manages the different input formats of provers, using particular encodings if needed through drivers.

8.2.2.2 Deduction Tools

As presented in Sec. 8.1, the three automated deduction tools of BWare are Zenon Modulo, iProver Modulo and Alt-Ergo.

iProver Modulo [Burel 2011], developed by Burel, is an extension of the resolution and instantiation based first-order logic ATP iProver [Korovin 2008] to deduction modulo theory. The input files of iProver Modulo are typically in the TPTP FOF format – it does not understand polymorphism. The proof obligations provided by Why3 to iProver Modulo are in the TPTP FOF format, resulting in an encoding of polymorphism into untyped first-order logic (see Sec. 6.4.2).

Alt-Ergo [Bobot, Conchon, Contejean, Iguernelala, Lescuyer, and Mebsout 2013], developed by OCamlPro, is the first SMT solver to natively deal with polymorphism. It was originally designed to be the dedicated deduction tool of the platform Why3, thus the proof obligations provided by Why3 to Alt-Ergo are in its native format.

Zenon Arith [Bury and Delahaye 2015] is an extension of Zenon to linear arithmetic developed by Bury. This extension can be used through Zenon Modulo, as we will see in experimental results in Sec. 8.4.

8.2.2.3 Proof Checkers

The last stage of the BWare toolchain consists in the proof checker Dedukti. Only Zenon Modulo and iProver Modulo can produce proof certificates for the moment.

8.3 The B Set Theory

As shown in Sec. 8.2, the tool bpo2why translates B proof obligations into WhyML. But we did not mention the B set theory yet. We have to provide the theory to Why3 in WhyML format. The solution chosen in BWare is to define the B set theory directly in WhyML. This solution differs from the presentation proposed in Sec. 5.4, therefore it is not consistent with the theoretical results presented in previous chapters. Nevertheless, these theoretical results ensure us that Poly-FOL is a fair candidate to define the B set theory. In addition, a hand-made B theory in WhyML allows us to choose the most effective definitions of

B operators while being faithful to the original B definitions (see the discussion at Sec. 5.4.3).

The solution proposed is to define directly the rewrite rules for the derived constructs. The resulting rewrite rules are mostly propositional and based on the membership predicate symbol. In addition, we preserve the proper typing constraints coming from the real axioms.

The translation of the definition of the union, as presented in Sec. 5.4.3, lead us to define a function symbol f such that we have the rewrite rule:

$$x \in_\alpha f(\alpha; u, s, t) \longrightarrow_{\Gamma_\mathrm{L}} x \in_\alpha u \wedge (x \in_\alpha s \vee x \in_\alpha t)$$

where $\Gamma_\mathrm{L} := (\alpha : \mathsf{Type}, x : \alpha, u : \mathsf{set}(\alpha), s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha))$.

In the rewrite rule above, the variable u is not needed anymore to preserve the well typedness, since we have the typing constraints represented in Γ_L for x, s and t. So, we choose to define the rewrite rule for set union as follows:

$$x \in_\alpha s \cup_\alpha t \longrightarrow_{\Gamma_\mathrm{L}} x \in_\alpha s \vee x \in_\alpha t$$

where $\Gamma_\mathrm{L} := (\alpha : \mathsf{Type}, x : \alpha, s : \mathsf{set}(\alpha), t : \mathsf{set}(\alpha))$, and the function symbol $\cup(\alpha; s, t)$ is noted with an infix syntax.

We now introduce the hand-made B set theory modulo used by Zenon Modulo in BWare. The definitions of these rewrite rules are done in the spirit of the example above.

The presentation follows the order of Sec. 1.2.3.2. We give, for all the introduced symbols, its type signature and the corresponding rewrite rule. In addition, we use an infix notation and subscript type parameters. For type signatures, we point out the position of term arguments with the symbol "-" and we do not explicitly write type arguments. Finally, to lighten the presentation, we do not give local contexts of the rewrite rules since types of arguments are given in type signatures.

The translation tool bpo2why uses a particular predicate symbol for set equality (see Sec. 7.5). This allows us to reduce equality between sets using the extensionality, which is quite effective in practice. In the following, we use:

$$\text{-} \stackrel{\mathsf{set}}{=} \text{-} : \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to o$$

to denote the set equality, which is supposed to be different from the usual equality

$$\text{-} = \text{-} : \Pi\alpha.\ \alpha \times \alpha \to o$$

Core Theory

First, we present the signatures of primitive symbols (see Sec. 4.1.1).

$$\begin{aligned}
\mathsf{set} &:: 1 \\
\mathsf{tup} &:: 2 \\
(\text{-},\text{-}) &: \Pi\alpha_1\alpha_2.\ \alpha_1 \times \alpha_2 \to \mathsf{tup}(\alpha_1, \alpha_2) \\
\mathbb{P}(\text{-}) &: \Pi\alpha.\ \mathsf{set}(\alpha) \to \mathsf{set}(\mathsf{set}(\alpha)) \\
\text{-} \times \text{-} &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \\
\text{-} \in \text{-} &: \Pi\alpha.\ \alpha \times \mathsf{set}(\alpha) \to o \\
\text{-} \overset{\mathsf{set}}{=} \text{-} &: \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to o
\end{aligned}$$

Then, we turn the three axioms SET1, SET2 and SET4 into rewrite rules.

$$\begin{aligned}
(x, y)_{\alpha_1,\alpha_2} \in_{\mathsf{tup}(\alpha_1,\alpha_2)} s \times_{\alpha_1,\alpha_2} t &\longrightarrow x \in_{\alpha_1} s \wedge y \in_{\alpha_2} t \\
s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) &\longrightarrow \forall x : \alpha.\ x \in_\alpha s \Rightarrow x \in_\alpha t \\
s \overset{\mathsf{set}}{=}_\alpha t &\longrightarrow \forall x : \alpha.\ x \in_\alpha s \Leftrightarrow x \in_\alpha t
\end{aligned}$$

We do not remove the axiom SET4 (with the common equality symbol $=$) from the resulting theory because it may be necessary for some proof obligations.

Set Inclusion

The two constructs for set inclusions can be seen as syntactic sugar, using the membership to the powerset.

$$\begin{aligned}
\text{-} \subseteq \text{-} &: \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to o \\
\text{-} \subset \text{-} &: \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to o
\end{aligned}$$

$$\begin{aligned}
s \subseteq_\alpha t &\longrightarrow s \in_{\mathsf{set}(\alpha)} \mathbb{P}_\alpha(t) \\
s \subset_\alpha t &\longrightarrow s \subseteq_\alpha t \wedge \neg(s \overset{\mathsf{set}}{=}_\alpha t)
\end{aligned}$$

Basic Set Theory Derived Constructs

The following rewrite rules for union, intersection, difference and singleton do not use the typing set u of the original B definition (see Sec. 5.4.3). In addition, we change the

definition of the empty-set because we do not want to use the set BIG in practice.

$$
\begin{aligned}
\text{-} \cup \text{-} &: \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to \mathsf{set}(\alpha) \\
\text{-} \cap \text{-} &: \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to \mathsf{set}(\alpha) \\
\text{-} - \text{-} &: \Pi\alpha.\ \mathsf{set}(\alpha) \times \mathsf{set}(\alpha) \to \mathsf{set}(\alpha) \\
\{\text{-}\} &: \Pi\alpha.\ \alpha \to \mathsf{set}(\alpha) \\
\varnothing &: \Pi\alpha.\ \mathsf{set}(\alpha) \\
\mathbb{P}_1(\text{-}) &: \Pi\alpha.\ \mathsf{set}(\alpha) \to \mathsf{set}(\mathsf{set}(\alpha))
\end{aligned}
$$

$$
\begin{aligned}
x \in_\alpha s \cup_\alpha t &\longrightarrow x \in_\alpha s \vee x \in_\alpha t \\
x \in_\alpha s \cap_\alpha t &\longrightarrow x \in_\alpha s \wedge x \in_\alpha t \\
x \in_\alpha s -_\alpha t &\longrightarrow x \in_\alpha s \wedge x \notin_\alpha t \\
x \in_\alpha \{a\}_\alpha &\longrightarrow x =_\alpha a \\
x \in_\alpha \varnothing_\alpha &\longrightarrow \bot \\
\mathbb{P}_{1\alpha}(s) &\longrightarrow \mathbb{P}_\alpha(s) -_{\mathsf{set}(\alpha)} \{\varnothing_\alpha\}_{\mathsf{set}(\alpha)}
\end{aligned}
$$

Binary Relations: First Series

The first series of constructs related to binary relations. In the following, we omit type parameters to enlighten notation when it is clear from the context.

$$
\begin{aligned}
\text{-} \leftrightarrow \text{-} &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2))) \\
\text{-}^{-1} &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \to \mathsf{set}(\mathsf{tup}(\alpha_2, \alpha_1)) \\
\mathsf{dom}(\text{-}) &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \to \mathsf{set}(\alpha_1) \\
\mathsf{ran}(\text{-}) &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \to \mathsf{set}(\alpha_2) \\
\text{-;-} &: \Pi\alpha_1\alpha_2\alpha_3.\ \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \times \mathsf{set}(\mathsf{tup}(\alpha_2, \alpha_3)) \to \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_3)) \\
\text{-} \circ \text{-} &: \Pi\alpha_1\alpha_2\alpha_3.\ \mathsf{set}(\mathsf{tup}(\alpha_2, \alpha_3)) \times \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \to \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_3)) \\
\mathsf{id}(\text{-}) &: \Pi\alpha.\ \mathsf{set}(\alpha) \to \mathsf{set}(\mathsf{tup}(\alpha, \alpha)) \\
\text{-} \lhd \text{-} &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \to \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \\
\text{-} \rhd \text{-} &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \\
\text{-} \trianglelefteq \text{-} &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \to \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \\
\text{-} \trianglerighteq \text{-} &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2))
\end{aligned}
$$

$$p \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} u \leftrightarrow_{\alpha_1,\alpha_2} v \longrightarrow \forall x : \alpha_1.\ \forall y : \alpha_2.\ (x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p \Rightarrow x \in_{\alpha_1} u \wedge y \in_{\alpha_2} v$$

$$(y,x) \in_{\mathsf{tup}(\alpha_2,\alpha_1)} p^{-1}_{\alpha_2,\alpha_1} \longrightarrow (x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p$$

$$x \in_{\alpha_1} \mathsf{dom}_{\alpha_1,\alpha_2}(p) \longrightarrow \exists b : \alpha_2.\ (x,b) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p$$

$$y \in_{\alpha_2} \mathsf{ran}_{\alpha_1,\alpha_2}(p) \longrightarrow \exists a : \alpha_1.\ (a,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p$$

$$(x,z) \in_{\mathsf{tup}(\alpha_1,\alpha_3)} p;_{\alpha_1,\alpha_2,\alpha_3} q \longrightarrow \exists b : \alpha_2.\ (x,b) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p \wedge (b,z) \in_{\mathsf{tup}(\alpha_2,\alpha_3)} q$$

$$q \circ_{\alpha_1,\alpha_2,\alpha_3} p \longrightarrow p;_{\alpha_1,\alpha_2,\alpha_3} q$$

$$(x,y) \in_{\mathsf{tup}(\alpha,\alpha)} \mathsf{id}_\alpha(u) \longrightarrow x \in_\alpha u \wedge x =_\alpha y$$

$$(x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} s \lhd_{\alpha_1,\alpha_2} p \longrightarrow (x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p \wedge x \in_{\alpha_1} s$$

$$(x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p \rhd_{\alpha_1,\alpha_2} t \longrightarrow (x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p \wedge y \in_{\alpha_2} t$$

$$(x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} s \lhd\!\!\!\cdot_{\alpha_1,\alpha_2} p \longrightarrow (x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p \wedge x \notin_{\alpha_1} s$$

$$(x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p \rhd\!\!\!\cdot_{\alpha_1,\alpha_2} t \longrightarrow (x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p \wedge y \notin_{\alpha_2} t$$

Binary Relations: Second Series

The second series of constructs related to binary relations.

$$\begin{aligned}
\text{-}[\text{-}] &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)) \times \mathsf{set}(\alpha_1) \to \mathsf{set}(\alpha_2)\\
\text{-}\lhd\!+\text{-} &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)) \times \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)) \to \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))\\
\text{-}\otimes\text{-} &: \Pi\alpha_1\alpha_2\alpha_3.\ \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)) \times \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_3))\\
&\qquad\qquad \to \mathsf{set}(\mathsf{tup}(\alpha_1,\mathsf{tup}(\alpha_2,\alpha_3)))\\
\mathsf{prj}_1(\text{-}) &: \Pi\alpha_1\alpha_2.\ \mathsf{tup}(\mathsf{set}(\alpha_1),\mathsf{set}(\alpha_2)) \to \mathsf{set}(\mathsf{tup}(\mathsf{tup}(\alpha_1,\alpha_2),\alpha_1))\\
\mathsf{prj}_2(\text{-}) &: \Pi\alpha_1\alpha_2.\ \mathsf{tup}(\mathsf{set}(\alpha_1),\mathsf{set}(\alpha_2)) \to \mathsf{set}(\mathsf{tup}(\mathsf{tup}(\alpha_1,\alpha_2),\alpha_2))\\
\text{-}||\text{-} &: \Pi\alpha_1\alpha_2\alpha_3\alpha_4.\ \mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2)) \times \mathsf{set}(\mathsf{tup}(\alpha_3,\alpha_4))\\
&\qquad\qquad \to \mathsf{set}(\mathsf{tup}(\mathsf{tup}(\alpha_1,\alpha_3),\mathsf{tup}(\alpha_2,\alpha_4)))
\end{aligned}$$

$$x \in_{\alpha_2} p[w]_{\alpha_1,\alpha_2} \longrightarrow \exists a : \alpha_1.\ a \in_{\alpha_1} w \wedge (a,x) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p$$

$$(x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} q \lhd\!+_{\alpha_1,\alpha_2} p \longrightarrow ((x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} q \wedge x \notin_{\alpha_1} \mathsf{dom}_{\alpha_1,\alpha_2}(p)) \vee (x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} p$$

$$(x,(y,z)) \in_{\mathsf{tup}(\alpha_1,\mathsf{tup}(\alpha_2,\alpha_3))} f \otimes_{\alpha_1,\alpha_2,\alpha_3} g \longrightarrow (x,y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} f \wedge (x,z) \in_{\mathsf{tup}(\alpha_1,\alpha_3)} g$$

$$((x,y),z) \in_{\mathsf{tup}(\mathsf{tup}(\alpha_1,\alpha_2),\alpha_1)} \mathsf{prj}_{1\alpha_1,\alpha_2}(s,t) \longrightarrow ((x,y),z) \in_{\mathsf{tup}(\mathsf{tup}(\alpha_1,\alpha_2),\alpha_1)} (s \times_{\alpha_1,\alpha_2} t) \times_{\mathsf{tup}(\alpha_1,\alpha_2),\alpha_1} s \wedge x =_{\alpha_1} z$$

$$((x,y),z) \in_{\mathsf{tup}(\mathsf{tup}(\alpha_1,\alpha_2),\alpha_2)} \mathsf{prj}_{2\alpha_1,\alpha_2}(s,t) \longrightarrow ((x,y),z) \in_{\mathsf{tup}(\mathsf{tup}(\alpha_1,\alpha_2),\alpha_2)} (s \times_{\alpha_1,\alpha_2} t) \times_{\mathsf{tup}(\alpha_1,\alpha_2),\alpha_2} t \wedge y =_{\alpha_2} z$$

$$((x,y),(z,w)) \in_{\mathsf{tup}(\mathsf{tup}(\alpha_1,\alpha_3),\mathsf{tup}(\alpha_2,\alpha_4))} h||_{\alpha_1,\alpha_2,\alpha_3,\alpha_4} k \longrightarrow (x,z) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} h \wedge (y,w) \in_{\mathsf{tup}(\alpha_3,\alpha_4)}$$

Function Constructs

The constructs related to functions.

$$\begin{aligned}
\text{- } \mapsto \text{ - } &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2))) \\
\text{- } \to \text{ - } &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2))) \\
\text{- } \rightarrowtail \text{ - } &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2))) \\
\text{- } \rightarrowtail \text{ - } &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2))) \\
\text{- } \twoheadrightarrow \text{ - } &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2))) \\
\text{- } \twoheadrightarrow \text{ - } &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2))) \\
\text{- } \rightarrowtail\!\!\!\!\!\twoheadrightarrow \text{ - } &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2))) \\
\text{- } \rightarrowtail\!\!\!\!\!\twoheadrightarrow \text{ - } &: \Pi\alpha_1\alpha_2.\ \mathsf{set}(\alpha_1) \times \mathsf{set}(\alpha_2) \to \mathsf{set}(\mathsf{set}(\mathsf{tup}(\alpha_1, \alpha_2)))
\end{aligned}$$

$$\begin{aligned}
&f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \mapsto_{\alpha_1,\alpha_2} t \longrightarrow \\
&\qquad f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \leftrightarrow_{\alpha_1,\alpha_2} t \wedge \\
&\qquad (\forall x : \alpha_1.\ \forall y, z : \alpha_2.\ (x, y) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} f \wedge (x, z) \in_{\mathsf{tup}(\alpha_1,\alpha_2)} f \Rightarrow y =_{\alpha_2} z) \\
&f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \to_{\alpha_1,\alpha_2} t \longrightarrow \\
&\qquad f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \mapsto_{\alpha_1,\alpha_2} t \wedge \mathsf{dom}_{\alpha_1,\alpha_2}(f) \stackrel{\mathsf{set}}{=}_{\alpha_1} s \\
&f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \rightarrowtail_{\alpha_1,\alpha_2} t \longrightarrow \\
&\qquad f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \mapsto_{\alpha_1,\alpha_2} t \wedge f^{-1}_{\alpha_1,\alpha_2} \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} t \mapsto_{\alpha_2,\alpha_1} s \\
&f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \rightarrowtail_{\alpha_1,\alpha_2} t \longrightarrow \\
&\qquad f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \rightarrowtail_{\alpha_1,\alpha_2} t \wedge f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \to_{\alpha_1,\alpha_2} t \\
&f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \twoheadrightarrow_{\alpha_1,\alpha_2} t \longrightarrow \\
&\qquad f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \mapsto_{\alpha_1,\alpha_2} t \wedge \mathsf{ran}_{\alpha_1,\alpha_2}(f) \stackrel{\mathsf{set}}{=}_{\alpha_2} t \\
&f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \to_{\alpha_1,\alpha_2} t \longrightarrow \\
&\qquad f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \twoheadrightarrow_{\alpha_1,\alpha_2} t \wedge f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \to_{\alpha_1,\alpha_2} t \\
&f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \rightarrowtail\!\!\!\!\!\twoheadrightarrow_{\alpha_1,\alpha_2} t \longrightarrow \\
&\qquad f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \rightarrowtail_{\alpha_1,\alpha_2} t \wedge f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \twoheadrightarrow_{\alpha_1,\alpha_2} t \\
&f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \rightarrowtail\!\!\!\!\!\twoheadrightarrow_{\alpha_1,\alpha_2} t \longrightarrow \\
&\qquad f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \rightarrowtail_{\alpha_1,\alpha_2} t \wedge f \in_{\mathsf{set}(\mathsf{tup}(\alpha_1,\alpha_2))} s \twoheadrightarrow_{\alpha_1,\alpha_2} t
\end{aligned}$$

8.4 BWare Experimental Results

We present in this section the experimental results obtained over the BWare benchmark. These results allow to test our tool Zenon Modulo.

8.4.1 The BWare Proof Obligations Benchmark

The set of B proof obligations (POs for short) provided by the industrial partners of the BWare project is a very valuable resource. It is usually difficult for academic researchers to get access to real industrial datas to test their tools.

8.4.1.1 Presentation

Mitsubishi Electric R&D and ClearSy have provided to BWare a set made of 12,876 proof obligations coming from real industrial projects. These POs have been anonymized, allowing us to use and to distribute them.

After this anonymization and the translations through bpo2why and Why3 drivers, it is difficult to understand the mathematical meaning behind the input files corresponding to POs. But the industrial partners have chosen precisely this set of POs to have a wide spectrum, that reflects well the different kinds of mathematical formulæ that appear in B projects. Then, all the B operators defined in Sec. 8.3 are represented in this benchmark.

It should be noted that one important challenge represented by these POs are their sizes and their large contexts. For instance, the mean size of input files in TPTP TFF1 format is 515 KiB – with a maximum of 2,690 KiB – which represents thousands of lines. In addition, each PO is provided with hundreds of useless axioms and hypotheses, all these formulæ being quantified over dozens of variables. Consequently, the proof search space is generally very large, requiring us to implement efficient deduction tools.

All the POs of the benchmark are provable since they have been proved inside Atelier B, automatically or interactively. The automated theorem prover of Atelier B, called "main prover" and denoted mp, is able to prove automatically 85.4 % of this benchmark, the resulting 14.6 % requiring a human interaction to be proved.

8.4.1.2 Availability

The benchmark is publicly available, under the CeCILL-B license, at: `http://bware.lri.fr/`. CeCILL is a French free software license, compatible with the GNU GPL (see: `http://www.cecill.info/licences.en.html`).

Several formats are proposed and divided into several archives. The considered formats are the following:

- TPTP FOF (regular TPTP format for mono-sorted first order logic);
- TPTP TFF1 (TPTP format for first order logic with polymorphic types);
- SMT-LIB v2 (regular SMT format for many-sorted first order logic);
- Alt-Ergo (input native format of Alt-Ergo).

8.4.2 Experimental Protocol

The experiment was run on an Intel Xeon E5-2660 v2 2.20 GHz computer, with a timeout of 120 s and a memory limit of 1 GiB. For each tool (except mp, which was tested directly over the native format of POs coming from Atelier B, thus not through the Why3 platform), the following input formats and command lines (where `%t` is the timeout, `%m` the memory limit, and `%f` the file name) were used:

- Zenon Modulo 0.4.1 (Zenon with types, deduction modulo, and arithmetic):
  Input format: TPTP TFF1;
  Command line: "`zenon_modulo -p0 -itptp -b-rwrt -rwrt -x arith -max-size %mM -max-time %ts %f`".
- iProver Modulo v0.7+0.2:
  Input format: TPTP FOF;
  Command line: "`iprover_modulo_launcher.sh %f %t --strategies 'Id;Equiv(ClausalAll)' --normalization_type dtree --omit_eq false --dedukti_out_proof false`".
- Alt-Ergo 0.99.1:
  Input format: Alt-Ergo;
  Command line: "`alt-ergo -timelimit %t %f`".
- Vampire 2.6:
  Input format: TPTP FOF;

Command line: "`vampire --proof tptp --mode casc -t %t %f`".

- E 1.8:
  Input format: TPTP FOF;
  Command line: "`eprover --auto --tptp3-format %f`".

- CVC4 1.4:
  Input format: SMT-LIB v2;
  Command line: "`cvc4 --lang=smt2 --rlimit %t000 %f`".

- Z3 4.3.2:
  Input format: SMT-LIB v2;
  Command line: "`z3 -smt2 -rs:42 %f`".

8.4.3 Experimental Results

We present in this section the experimental results obtained by Zenon Modulo over the BWare benchmark. First we compare the different extensions implemented in Zenon. Then, We compare Zenon Modulo to other deduction tools.

8.4.3.1 Zenon Extensions

We summarize in Tab. 8.1 the results obtained by the different extensions of Zenon over the 12,876 POs of the BWare benchmark.

The first column gives the results of the Atelier B prover, called mp. Then, we give in the next five columns the results for the different versions of Zenon.

The column "Zenon" corresponds to the original untyped implementation of Zenon (with the TPTP FOF input format). The column "Zenon Typed" corresponds to the extension of Zenon to polymorphism presented in Sec. 6.2. The column "Zenon Arith" presents the results obtained by the extension of Zenon to linear arithmetic done by Bury [Bury and Delahaye 2015] (this is not our work).

The next column "Zenon Modulo" gives the results obtained by our tool Zenon Modulo, *i.e.* Zenon extended to polymorphism and deduction modulo theory. Finally, the last

	All Tools (12,738 / 98.9%)					
12,876	mp	Zenon	Zenon Typed	Zenon Arith	Zenon Modulo	Zenon Mod+Ari
Proofs	10,995	337	6,251	7,406	10,340	12,281
Rate	85.4%	2.6%	48.5%	57.5%	80.3%	95.4%
Time (s)	-	6.9	2.3	2.5	3.0	2.6
Unique	329	0	0	0	34	946

Table 8.1: Experimental Results over the BWare Benchmark (Part 1)

column "Zenon Mod+Ari" corresponds to the combination of Zenon Modulo and Zenon Arith.

The lines "Proofs" and "Rate" correspond respectively to the number and the percentage of POs that have been proved by the corresponding tool. The line "Time" gives the mean times, in second, taken to prove a PO. Finally, the line "Unique" gives the number of POs proved only by the corresponding tool.

The results of Tab. 8.1 are very conclusive. We remark that each extension improves the number of POs being proved.

Our first contribution, the extension of Zenon to polymorphism, allows to improve the number of proved POs from 337 to 6,251, *i.e.* an increase of 1,755 %. We can conclude from this result that dealing natively with polymorphism is better than relying on an encoding, at least for this benchmark.

Our second contribution to Zenon, the extension to deduction modulo theory, is also very conclusive. It allows to prove 4,089 more POs, corresponding to an increase of 65 % with respect to polymorphic typed version. We can conclude from this result that deduction modulo theory is quite effective to improve proof search in the B set theory. In addition, if we combine Zenon Modulo and Zenon Arith, we raise the number of proved POs to 12,281, *i.e.* a percentage of 95.4 %. This is exactly ten percentage points more than mp, the native prover of Atelier B (which also deals with arithmetic).

The combination of Zenon Modulo and Zenon Arith performs well. We remark that Zenon

Mod+Ari proves 1,941 more POs than Zenon Modulo, and that Zenon Arith proves 1,155 POs more than Zenon Typed. Thus, we can deduce that Zenon Modulo helps Zenon Arith to prove 786 POs which need arithmetic reasoning and that it could not prove before. This is an unexpected and satisfying behavior.

It should be noted also that a mean time per PO smaller than 3 seconds is reasonable. The experimental protocol does not allow us to compare this mean time to the one of mp.

Finally, the last important information provided by these results are the number of POs proved only by one tool. We remark that mp proves 329 POs that no version of Zenon manages to prove. But it is also the case for Zenon Modulo (34 POs) and for the combination of Zenon Modulo and Zenon Arith (946 POs). An unexpected behavior of the combination of Zenon Modulo and Zenon Arith is to lose some problems that Zenon Modulo may prove alone.

We present in Fig. 8.1 the same results than in Tab. 8.1. The goal of this figure is to

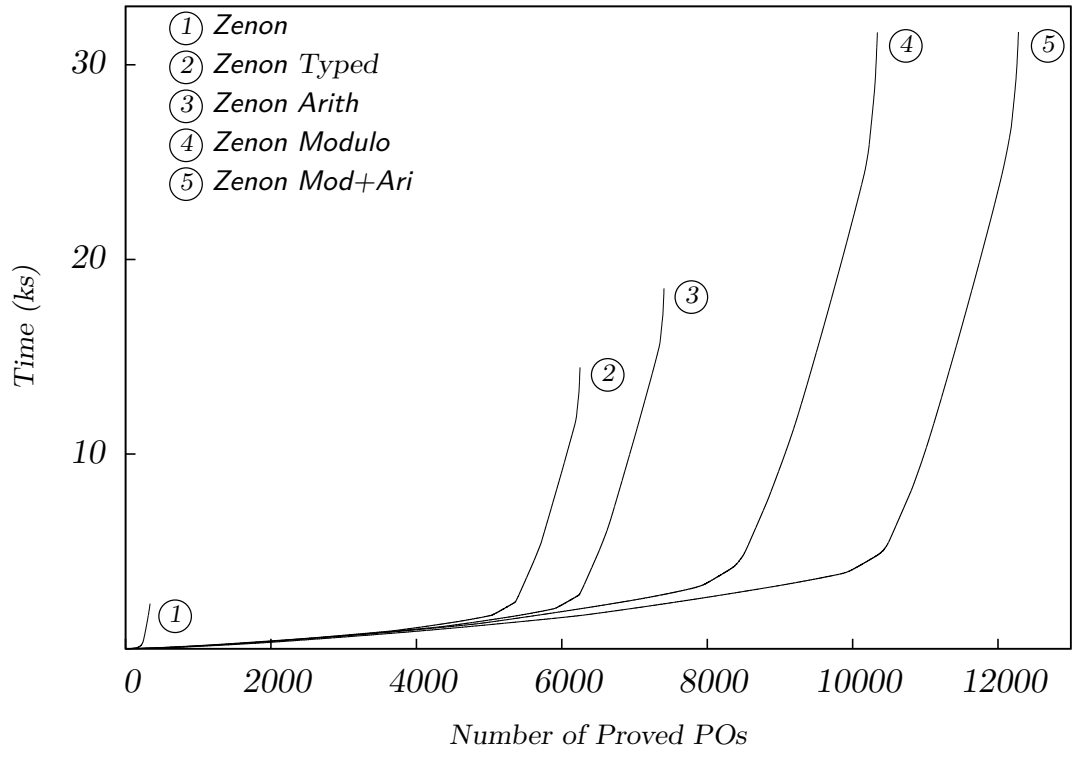

Figure 8.1: Cumulative Times According to the Numbers of Proved POs

represent the cumulative time spent to prove all the POs. We select only the POs that have been proved, and, for each tool, we ordered them from the fastest to prove to the longest.

We remark that for the four implementations of Zenon using polymorphism, the cumulative time is increasing linearly for around the first 80 % POs. In fact, the mean time spent for these 80 % POs is less than 0.5 seconds, which is much lower than the 3 seconds of Tab. 8.1.

Remark Zenon Modulo can generate proof certificates for all the 10,340 proofs found. All these 10,340 proofs certificates are checked well by Dedukti (this output was switched off for the benchmark).

This is not yet possible to generate proof certificates for Zenon Arith, thus for its combination with Zenon Modulo.

8.4.3.2 General Results

We compare in Tab. 8.2 the results of Zenon Modulo to the tools of BWare on the left-hand side, and to the state-of-the-art provers mentioned in Sec. 8.4.2 on the right-hand side.

The BWare provers are mp, the combination of Zenon Modulo and Zenon Arith (denoted Zen M+A), iProver Modulo (denoted iProv Mod) and Alt-Ergo. The state-of-the-art provers

	All Tools (12,797/99.4%)							
	BWare Tools (12,772/99.2%)				Other Tools			
12,876	mp	Zen M+A	iProv Mod	Alt Ergo	Vamp	E	CVC4	Z3
Proofs	10,995	12,281	3,695	12,620	10,154	7,919	12,173	10,880
Rate	85.4%	95.4%	28.7%	98.0%	78.9%	61.2%	94.5%	84.5%
Time	-	2.6	5.5	0.56	12	4.7	0.69	0.31
Uniq.1	109	4	0	65				
Uniq.2	84	0	0	13	0	0	1	12

Table 8.2: Experimental Results over the BWare Benchmark (Part 2)

are the ATP Vampire (denoted Vamp), the ATP E, the SMT solver CVC4 and the SMT solver Z3.

The lines "Proofs", "Rate" and "Time" are the same as in Tab. 8.1. The lines "Uniq.1" and "Uniq.2" are the same as "Unique" in Tab. 8.1, except that "Uniq.1" considers only the BWare tools, and "Uniq.2" deals with all the tools presented in Fig. 8.2.

The results of Tab. 8.2 are once again conclusive. The most important result is that, compared to all the other tools presented in this table, the combination of Zenon Modulo and Zenon Arith is the second prover that proves the most of POs, exceeded only by Alt-Ergo. The fact that this combination proves more POs than the two SMT solvers CVC4 and Z3, which both deal with arithmetic, is quite unexpected since these two tools are considered as the most efficient tools in their domain. But we remark that CVC4 and Z3 are faster than the combination of Zenon Modulo and Zenon Arith.

The two ATPs Vampire and E, which are known to be the most efficient first-order theorem provers [Sutcliffe 2016], do not deal with arithmetic reasoning. Zenon Modulo alone, which proves 10,340 POs, outperforms both Vampire and E, also when looking the time spent to prove POs.

One explanation of these results could be that the extensions of Zenon and Alt-Ergo are the only tools presented here to deal with polymorphism, whereas the other tools rely on encodings.

Finally, we present in Fig. 8.2 the cumulative times spent to prove POs for all the tools of our experiment. It should be noted that, for the SMT solvers, the cumulative time is increasing linearly, unlike for ATPs, which is an expected behavior.

We remark that for the first ten thousands POs, the combination of Zenon Modulo and Zenon Arith is faster that CVC4. This confirms that deduction modulo theory have a strong impact on proof search in axiomatic theories.

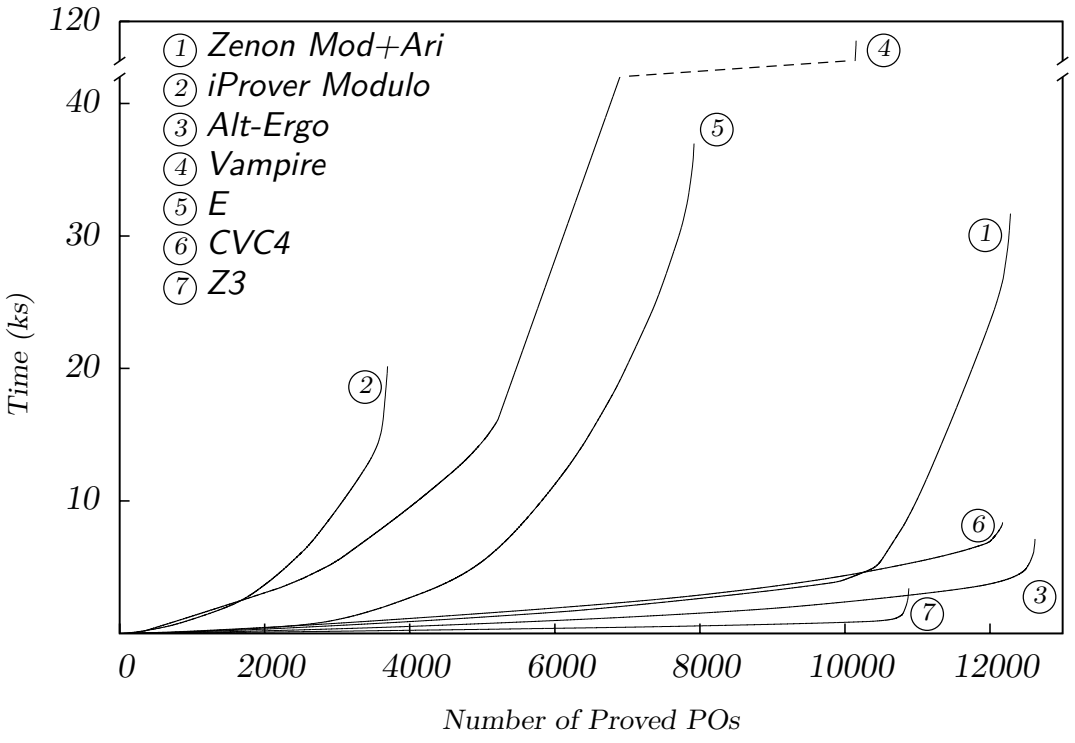

Figure 8.2: Cumulative Times According to the Numbers of Proved POs

Conclusion

Improving the automation of proof in the B Method, while ensuring the highest level of confidence in their soundness, was the guiding principle underpinning the work presented in this manuscript. Our main contribution is the development of the automated theorem prover Zenon Modulo, an extension of the Tableau-based first-order automated theorem prover Zenon to polymorphism and deduction modulo theory. We choose to implement these extensions to allow an efficient proof search in the B Method set theory in the framework of the BWare project.

The BWare framework requires having the B proof obligations encoded in the polymorphic first-order logic of WhyML, the native language of the program verification platform Why3. We have extended Zenon to deal natively with polymorphically typed fomulæ, thus avoiding to rely on external encodings of polymorphism, which tend to transform the shape of axioms. In addition, we use the formalism of deduction modulo theory, which improves proof search in axiomatic theories by turning axioms into rewrite rules, and which is well-suited for the B set theory.

The experimental results obtained over the BWare benchmark, a set made of 12,876 B proof obligations coming from real industrial projects, was very conclusive and allowed us to validate our work on Zenon Modulo. In particular, this experiment showed that each of the two extensions implemented in Zenon Modulo improves strongly the total number of proof obligations that are proved by Zenon Modulo.

To increase the confidence in the soundness of the proofs produced by Zenon Modulo, we choose to generate proof certificates, proof objects that have to be verified by external tools. We relied on Dedukti to certify our proofs, an efficient proof checker that implements

the $\lambda\Pi$-calculus modulo theory, and which is well-suited to check proofs that use rewriting techniques. This allowed us to verify all the proofs produced by Zenon Modulo in the BWare benchmark.

In addition to this development work, we have presented in this manuscript some theoretical results about the upstream chain of the BWare framework. Before being proved by Zenon Modulo, proof obligations are translated from B logic into polymorphic first-order logic. Concerns may arise about whether Zenon Modulo proofs are consistent with the original B proof obligations. In practice, the translation is done by a proprietary tool, called bpo2why, which performs some sophisticated transformations, in particular a type inference of expressions. It would have been very difficult to certify the correctness of bpo2why, even if we could have had access to its source code. Instead, we showed that we can translate Zenon Modulo proofs into B proofs. This was made possible by defining an encoding of B formulæ into polymorphic first-order logic, and a syntactic translation of the typed sequent calculus inference rules of Zenon Modulo into the B natural deduction proof system. Finally, we show that the resulting B proofs are valid proofs of the initial proof obligations. This gives us extra confidence that the approach of the BWare project to rely on the WhyML language is relevant.

We see two interesting perspectives of our work. The first one deals with the translation of Zenon Modulo proofs into B proofs, and the second with an alternative use of Zenon Modulo as a proof certificate generator.

An effective implementation of the translation of Zenon Modulo proofs into B proofs could be a very interesting project. The main benefit would be to remove the several translation steps of the proof obligation from the "trusted zone". Currently, the input formula is translated by bpo2why from B into WhyML, then by Why3 from WhyML into TFF1, and finally by Zenon Modulo from TFF1 into Dedukti. A desired solution would be to produce a B proof of the initial B formula. In this approach, Zenon Modulo would take as input the translated proof obligation, then would eventually find a proof, and finally it would generate a proof certificate that contains only the proof in the B proof system, *i.e.* without the input formula. This method would allow us to use the translation chain

and Zenon Modulo as a black-box, without any concerns about the correctness of the tools inside. Unfortunately, it is not yet possible to apply this method because it requires to have a proof checker for the B proof system.

The second perspective is an alternative use of Zenon Modulo, inspired by the work of Blanchette *et al.* in Sledgehammer and presented in [Blanchette, Böhme, Fleury, Smolka, and Steckermeier 2016]. The idea is to benefit from output of external ATPs to generate formal proof certificates through Zenon Modulo. For instance, the ATP E can generate proof traces in a specific format called TSTP. These proof traces contain the set of axioms needed to prove the statement, and a list of intermediate lemmas. Thus, these proof traces could improve the proof search of Zenon Modulo by reducing the proof-search space with the selected axioms, and by cutting a difficult proof into a list of smaller proofs of the intermediate lemmas. A proof of concept of this idea have already been tested by Pham [Pham 2016], and gave some promising results.

Finally, these two ideas could be combined to use Zenon Modulo as a generator of formal B proofs using proof traces from external tools. The main benefit from this approach would be its adaptive use in an industrial context. The certification requirements for an industrial use would only deal with the B proof checker, letting all external tools out of the "trusted zone" and allowing us to benefit from improvements made by ATP developers.

Résumé de la thèse

Introduction

L'année 2016 marque une étape importante dans le développement des véhicules autonomes. Alors que l'entreprise de technologie de l'information Google a lancé il y a plusieurs années son projet de voitures autonomes Google Car – la flotte de véhicules a déjà été testée sur près de trois millions de kilomètres –, des annonces officielles de nouveaux projets de voitures autonomes ont été publiées pendant la première moitié de l'année 2016. Un grand nombre de constructeurs automobiles de premier plan ont annoncé l'arrivé de véhicules autonomes dans les cinq prochaines années. Par exemple, le constructeur américain de voiture Ford a annoncé en août la sortie d'une voiture complètement autonome – sans volant ni pédale – pour l'année 2021 [Sage and Lienert 2016]. Dans la ville de Pittsburgh en Pennsylvanie, la société de réseau de transport Uber propose depuis le mois d'août à ses clients d'utiliser une flotte de voitures autonomes, accompagné d'un conducteur de secours pour le moment [Chafkin 2016].

L'arrivée des véhicules autonomes est certainement une bonne nouvelle. Ce sera une libération pour un grand nombre de personnes, en particulier ceux souffrant d'une mobilité réduite. Une fois lancé, ce nouveau moyen de transport prendra une place de plus en plus grande et dépassera rapidement les voitures telles que nous les connaissons. Mais tous ces points positifs ne doivent pas cacher les inquiétudes légitimes que l'on pourrait avoir quant à la sécurité de ces voitures autonomes. En effet, un véhicule autonome de transport de personne est un système critique, au sens où une défaillance de ce système pourrait mettre en danger des vies humaines. Une voiture autonome utilise des dizaines de capteurs, processeurs et programmes embarqués pour fonctionner. Le développement de logiciels

embarqués dans les systèmes critiques exige une rigueur et une expertise spécifique, ce qui semble manquer pour le moment dans l'industrie automobile.

La récente affaire des "accélérations involontaires" de Toyota, telle que rapportée par Bagnara pendant le 12th Workshop on Automotive Software & Systems en 2014, révèle des pratiques contestables [Bagnara 2014]. En l'an 2000, Le constructeur automobile japonais Toyota a adopté un système électronique de contrôle d'accélération (Electronic Throttle Control System, ETCS) pour la plupart de ses nouveaux modèles de voiture, remplaçant ainsi une pédale d'accélération mécanique par un système électronique. En 2010, l'agence gouvernementale américaine National Highway Traffic Safety Administration rapporte que 89 décès pourraient être liés à cette affaire des accélérations involontaires, en plus de milliers d'accidents de voiture. En 2013, le premier procès au cours duquel les plaignants affirment que les accélérations involontaires ont été causées par un mauvais fonctionnement du système ETCS, fait témoigner l'expert en systèmes embarqués Baar [Barr 2013] et le professeur Koopman [Koopman 2014] de l'université Carnegie Mellon, tous deux ayant pu examiner le code source du logiciel embarqué dans le système ETCS. Leurs conclusions ont révélé que le logiciel en question était très éloigné des standards attendus pour des logiciels de systèmes critiques. Par exemple, ils rapportèrent que le processus de développement n'avait pas suivit rigoureusement les recommandations MISRA-C – une norme non-contraignante proposée par la Motor Industry Software Reliability Association. Ils ont aussi décrit le code source C en question comme étant du "code spaghetti", contenant plus de 10.000 variables globales en lecture/écriture. Finalement, ils soulignèrent l'absence de contrainte de certification des logiciels dans les systèmes critiques pour les constructeurs automobiles américains.

En 2014, Toyota parvint à un accord avec le département de justice du gouvernement fédéral américain et régla une amende de 1,2 milliards de dollars américains, mettant ainsi fin à l'enquête criminelle à propos de l'affaire des accélérations involontaires. Toutes les conséquences de cette affaire ne sont toujours pas connues seize ans après la sortie du système ETCS. Mais elle peut déjà être considérée comme une étude de cas pertinente et un tournant pour la sûreté fonctionnelle des systèmes critiques.

Une conclusion intéressante de cette histoire, et qui va plus loin que son propre contexte, est la constatation d'un manque de normes obligatoires concernant les constructeurs automobiles dans le développement de logiciels pour des systèmes critiques. D'autres secteurs du transport de personnes, telles que les industries aéronautique et ferroviaire, ont effectué avec succès leur révolution électronique trente ans auparavant. Par exemple, le constructeur aéronautique européen Airbus lança en 1984 l'A320, le premier avion de ligne avec un système de contrôle de vol entièrement numérique [Favre 1994]. Dans l'industrie ferroviaire, les premiers véhicules autonomes, un nouveau système complètement automatique et sans conducteur de métro dans la ville française de Lille, apparurent en 1983 [Lardennois 1993].

Le haut niveau de sécurité de ces deux secteurs industriels a été atteint grâce à l'application de normes obligatoires et spécifiques pour les composants électroniques et les logiciels embarqués. La norme IEC 61208 est la norme générique internationale pour les systèmes critiques électriques, électroniques et programmables, et publiée par la commission internationale électrotechnique IEC. Cette norme a été spécifiée pour chaque secteur particulier. Par exemple, dans l'industrie ferroviaire, la norme EN 50128 s'applique au logiciels critiques des systèmes de contrôle et de protection. Une des notions importantes définies par cette norme est le niveau d'intégrité de sécurité SIL, une grandeur qui mesure le niveau relatif de réduction des risques fourni par une fonction de sécurité. La norme définit quatre niveaux SIL, de SIL 1 (le plus bas niveau de réduction des risques) à SIL 4 (le plus haut niveau de réduction des risques).

Les fonctions de sécurité d'un système qui exige un niveau de certification SIL 4 sont généralement les parties les plus critiques de l'ensemble du système, par exemple le système de contrôle de la vitesse d'un métro automatique sans conducteur. En génie logiciel, des méthodes de développement logiciel, appelées méthodes formelles, ont été conçues pour développer des logiciels ayant un haut niveau de fiabilité. L'idée centrale des méthodes formelles est de prouver qu'un programme informatique vérifie des propriétés mathématiques particulières. Ces propriétés mathématiques traduisent le comportement souhaité du système et sont regroupées dans la spécification, une description formelle du système. La notion de spécification est très importante dans les méthodes formelles, car toutes ces

méthodes de développement ne nous permettent que de prouver une correction relative du programme par rapport à sa spécification. Ainsi, les spécifications doivent être décrites dans un langage formel, typiquement un langage sans ambiguïté tels que les langages logiques, à l'inverse des langages naturels. Il existe un grand nombre de méthodes formelles différentes, couvrant tout ou partie du cycle de développement d'un logiciel, depuis sa spécification jusqu'à son implémentation.

La méthode B est une méthode formelle créée par Jean-Raymond Abrial et présentée dans son livre de référence, appelé le B-Book [Abrial 1996] et publié en 1996. La méthode B s'inspire des travaux antérieurs de Hoare et Dijkstra à propos de la correction des programmes. Cette méthode est principalement utilisée dans l'industrie ferroviaire pour spécifier et développer des logiciels de systèmes critiques. Par exemple, la méthode B a été utilisée avec succès pour développer le système de contrôle de commande des rames du métro automatique et sans conducteur de la ligne 14 du métro parisien en France en 1998 [Behm, Benoit, Faivre, and Meynadier 1999]. La méthode B couvre tout le cycle de développement d'un logiciel, depuis la spécification formelle du système, appelée la machine abstraite, jusqu'à son implémentation concrète. Les programmes informatiques ainsi développés sont dits corrects par construction, grâce à un processus de raffinement de la machine abstraite jusqu'à une dernière machine B complètement déterministe, appelée B0. La dernière étape d'extraction du code source consiste en une traduction pratiquement syntaxique de la machine B0 vers un sous-ensemble d'un langage impératif de bas niveau tels que les langages C ou ADA. La correction de chacune des étapes de raffinement dépend de la validité de formules logiques, appelée obligations de preuve, et exprimées dans une théorie des ensembles spécifique à la méthode B.

Les projets industriels utilisant la méthode B génèrent en général des milliers d'obligations de preuve. Ils dépendent donc fortement d'outils automatiques pour décharger le plus grand nombre possible d'obligations de preuve. Un environnement de développement intégré spécifique au développement de logiciels avec la méthode B, appelé Atelier B [ClearSy 2013], est fourni avec un outil de démonstration de théorème qui aide les utilisateurs à vérifier la validité des obligations de preuve, automatiquement ou interactivement. L'outil de

déduction automatique de l'Atelier B prouve environ 85% des obligations de preuve des projets industriels communs, laissant ainsi des milliers d'obligations de preuve à décharger interactivement, c'est-à-dire avec une intervention humaine. Le manque d'automatisation dans le développement de projets en méthode B est un facteur de coût très important pour les industriels, ralentissant ainsi sa diffusion et son utilisation.

Notre travail vise à améliorer la vérification automatique des obligations de preuve de la méthode B, avec une attention particulière portée sur la correction des preuves produites. Notre principale contribution est le développement d'un outil de déduction automatique au premier ordre appelé Zenon Modulo. Cet outil étend Zenon [Bonichon, Delahaye, and Doligez 2007], un outil de déduction automatique au premier ordre implémentant la méthode des tabeaux. La méthode des tableaux [D'Agostino, Gabbay, Hähnle, and Posegga 2013] est un algorithme de recherche automatique de preuve pour le calcul des séquents sans coupure. En théorie de la preuve, le calcul des séquents [Gentzen 1935] est une famille de systèmes formels dirigés par la syntaxe et utilisés pour écrire des preuves. Un calcul des séquents est défini par un ensemble de règles d'inférence. Une règle d'inférence est un objet logique définissant une relation syntaxique entre un ensemble de formules, appelées prémisses, et un autre ensemble de formules, appelées conclusions, et correspondant à une étape de déduction élémentaire. Ces types de systèmes sont appelés des systèmes de preuve. Les preuves obtenues par la méthode des tableaux peuvent être facilement traduites dans le calcul des séquents, puisque qu'il ne s'agit que d'une reformulation syntaxique.

Dans notre travail, nous n'avons pas besoin de toutes les notions présentes dans la méthode B, en particulier celles relatives au langage B. Nous nous concentrons uniquement sur le raisonnement mathématique dans la méthode B, ce qui consiste principalement à étudier la théorie des ensembles de la méthode B. Améliorer la recherche automatique de preuve de Zenon pour la méthode B nous a mené au développement de deux extensions. La première est une extension de la logique aux types polymorphes et la seconde est une extension à la déduction modulo théorie. La motivation de ces deux extensions est de gérer efficacement la théorie des ensembles de la méthode B.

La théorie des ensembles de la méthode B diffère des autres telle que la théorie des

ensembles de Zermelo-Fraenkel. La principale différence consiste en l'addition de contraintes de typage aux expressions du langage. Ces contraintes de typage sont exprimées à l'aide des constructions ensemblistes du langage, ce faisant il n'y a aucune distinction syntaxique entre les types et les ensembles. Pour vérifier la bon typage des expressions, le B-Book fournit un ensemble de règles d'inférence de typage, définissant une procédure de vérification de type qui doit être appliquée avant la recherche de preuve. Nous montrons dans le Chap. 4 que les formules de la méthode B qui sont des axiomes et des hypothèses peuvent être interprétées comme des formules polymorphes, au sens où elles sont définies pour des types génériques. Une fois l'obligation de preuve fixée, qui elle n'est pas polymorphe, les types génériques des axiomes et des hypothèses doivent être instanciés avec les types concrets venant de l'obligation de preuve.

La théorie des ensembles de la méthode B est composée de six axiomes, en plus d'un grand nombre de constructeurs dérivés. Ces constructeurs dérivés, comme l'union de deux ensembles, le domaine d'une relation et l'ensemble des fonctions injectives totales, ont un rôle majeur dans la méthode B car ils sont très présents dans les obligations de preuve. Ainsi, il est important de traiter efficacement ces constructeurs dérivés. Nous avons choisi d'utiliser la déduction modulo théorie [Dowek, Hardin, and Kirchner 2003] pour améliorer la recherche de preuve dans la théorie des ensembles de la méthode B. La déduction modulo théorie est un formalisme qui étend la logique du premier ordre avec des règles de réécriture sur les termes et les propositions, permettant ainsi d'améliorer la recherche de preuve dans les théories axiomatiques en transformant les axiomes en règles de réécriture. Cela nous permet de distinguer les étapes de déduction des étapes de calcul en raisonnant sur des classes d'équivalence de formules, modulo une relation de congruence générée par le système de réécriture.

Les outils de déduction automatique sont généralement des outils logiciels de taille importante, utilisant des fonctionnalités sophistiquées et implémentant des optimisations complexes. Par exemple, Zenon Modulo est composé de plus de 40.000 lignes de code OCaml. Le risque augmentant avec la taille et la complexité, des sources potentielles de mauvais comportements et de bugs peuvent apparaître dans les outils de déduction automatique. Lorsque l'on cherche à vérifier la validité d'obligations de preuve dans le

cadre d'un développement de logiciel critique, garantir la correction des preuves produites par les outils de déduction automatique est une tache fondamentale. Barendregt et Barendsen [Barendregt and Barendsen 2002] ont proposé de s'appuyer sur le concept de certificats de preuve, des objets de preuve qui contiennent un énoncé et sa preuve formelle et qui peuvent être vérifiés par des outils externes. L'originalité de cette approche est de séparer la génération des certificats de preuve, effectuée par les outils de déduction automatique, et la vérification de la correction des preuves, déléguée au vérificateur externe. Idéalement, le vérificateur de preuve utilisé doit être basé sur un noyau léger et auditable, au sens où il doit être sensiblement plus petit que l'outil de déduction automatique. De ce point de vue, une autre contribution importante à Zenon Modulo est le développement d'une sortie qui génère des certificats de preuve pour le vérificateur de preuve Dedukti. Dedukti [Assaf, Burel, Cauderlier, Delahaye, Dowek, Dubois, Gilbert, Halmagrand, Hermant, and Saillard 2016] est une implémentation légère du $\lambda\Pi$-calcul modulo théorie, une extension du λ-calcul simplement typé aux types dépendants et à la réécriture. Dedukti a été conçu pour être utilisé comme un vérificateur universel de preuve, pouvant provenir autant d'outil de déduction automatique tel que Zenon Modulo que d'assistant de preuve tel que Coq [Bertot and Castéran 2013].

L'utilisation de Zenon Modulo, dont la logique sous-jacente est la logique du premier ordre avec typage polymorphe, pour prouver des obligations de preuve exprimées dans la logique de la méthode B et sa théorie des ensembles, peut légitimement soulever des questions. Nous avons répondu à cette problématique d'une manière originale en définissant un encodage des formules B dans la logique du premier ordre avec typage polymorphe. Une des particularités de cet encodage se situe au niveau de la génération des types des expressions dans la logique polymorphe. Cet encodage repose sur une étape d'inférence des types des variables liées des formules B. De plus, nous avons défini une traduction des preuves de Zenon Modulo, exprimées dans un calcul des séquents typé, vers le système de preuve de la méthode B, une adaptation de la déduction naturelle à la syntaxe de la méthode B. Enfin, nous avons montré que la preuve B ainsi obtenue correspond bien à une preuve de l'obligation de preuve initiale. Cette méthode nous permet ainsi d'augmenter la confiance globale quant à la correction de notre approche.

Notre travail fait partie du projet BWare [Delahaye, Dubois, Marché, and Mentré 2014], un projet de recherche industrielle soutenu par l'Agence Nationale de la Recherche. BWare a pour objectif de fournir une plateforme intégrée pour la vérification automatique des obligations de preuve provenant du développement de projets industriels utilisant la méthode B. Le consortium qui compose BWare regroupe des centres publics de recherche (Cedric, LRI et Inria) et des industriels utilisant la méthode B (Mitsubishi Electric R&D, ClearSy et OCamlPro). La méthodologie du projet est de construire une plateforme générique de vérification qui repose sur différents outils de déduction automatique, tels que des outils de déduction automatique au premier ordre et des solveurs satisfiabilité modulo théorie (SMT). La plateforme du projet BWare est basée sur Why3 [Bobot, Filliâtre, Marché, and Paskevich 2011], une plateforme pour la vérification de programme. Les outils de déduction automatique utilisés dans le projet BWare sont les outils au premier ordre Zenon Modulo et iProver Modulo [Burel 2011] et le solveur SMT Alt-Ergo [Bobot, Conchon, Contejean, Iguernelala, Lescuyer, and Mebsout 2013]. La diversité de ces outils de preuve doit permettre la vérification d'un large panel d'obligations de preuve. En plus de cette approche multi-outils, une autre originalité de l'approche de BWare réside dans l'exigence pour les outils de déduction de produire des certificats de preuve. Enfin, pour tester la plateforme BWare, une large bibliothèque d'obligations de preuve a été fournie par les partenaires industriels du projet qui développent des outils implémentant la méthode B et des applications utilisant la méthode B. Cette bibliothèque nous a permis de faire une comparaison expérimentale de nos outils de déduction avec les autres outils de BWare, ainsi que des outils externes de référence.

Ce manuscrit est organisé comme suit. Dans le Chap. 1, nous introduisons la logique de la méthode B. En particulier, nous présentons son système de preuve, sa théorie des ensembles et son système de type. Dans le Chap. 2, nous présentons une procédure d'inférence de type pour les variables liées des formules B. Cette procédure nous permet d'annoter les variables avec leur type, une information nécessaire dans les chapitres suivants. Nous présentons aussi une procédure correcte d'élimination des ensembles définis par compréhension. Dans le Chap. 3, nous introduisons la logique du premier ordre avec

typage polymorphe, ainsi que le système de preuve LLproof, un calcul des séquents typé utilisé par Zenon Modulo pour produire des preuves. Dans le Chap. 4, nous définissons un encodage des formules B dans la logique du premier ordre avec typage polymorphe, puis nous montrons comment reconstruire des preuves dans le système de preuve de la méthode B à partir de preuve LLproof. Dans le Chap. 5, nous montrons la correction relative de LLproof$^{\equiv}$, l'extension de LLproof à la déduction modulo théorie, par rapport à la correction de LLproof. Dans le Chap. 6, nous présentons l'outil de déduction automatique Zenon, puis les deux extensions de Zenon au typage polymorphe et à la déduction modulo théorie, obtenant ainsi le nouvel outil Zenon Modulo. Dans le Chap. 7, nous introduisons le vérificateur de preuve Dedukti et le $\lambda\Pi$-calcul modulo théorie. Puis, nous présentons les encodages de la logique du premier ordre avec typage polymorphe et du calcul des séquents avec typage et réécriture LLproof$^{\equiv}$ dans le $\lambda\Pi$-calcul modulo théorie. Enfin, dans le Chap. 8, nous présentons le projet BWare et les outils outils le composant. Puis, nous donnons le système de réécriture correspondant à la théorie des ensembles de la méthode B et utilisé par Zenon Modulo dans BWare. Nous concluons finalement notre travail en donnant les résultats expérimentaux obtenus avec la bibliothèque d'obligations de preuve fournie par les partenaires industriels de BWare.

Conclusion

Améliorer l'automatisation des preuves pour la méthode B, tout en garantissant le plus haut niveau de confiance possible quant à leur correction, a été le principe directeur du travail présenté dans ce manuscrit. Notre principale contribution est le développement de l'outil de déduction automatique Zenon Modulo, une extension de l'outil au premier ordre implémentant la méthode des tableaux Zenon au typage polymorphe et à la déduction modulo théorie. Nous avons choisi d'implémenter ces extensions dans le but d'obtenir une recherche automatique de preuve efficace dans la théorie des ensembles de la méthode B, dans le cadre du projet BWare.

La plateforme du projet BWare nous impose d'avoir les obligations de preuve B encodées dans la logique du premier ordre avec typage polymorphe du langage WhyML, le langage de la plateforme de vérification de programme Why3. Nous avons étendu Zenon pour traiter

directement les problèmes utilisant un typage polymorphe, évitant ainsi de dépendre d'un encodage externe du polymorphisme qui tend à déstructurer la forme des formules. De plus, nous avons utilisé le formalisme de la déduction modulo théorie, conçu pour améliorer la recherche de preuve dans les théories axiomatiques, et montré que ce formalisme est adapté à la théorie des ensembles de la méthode B.

Les résultats expérimentaux obtenus à partir de la bibliothèque d'obligations de preuve du projet BWare, composé de 12.876 obligations de preuve B provenant de projets industriels, ont été particulièrement concluant, nous permettant ainsi de valider expérimentalement notre travail sur Zenon Modulo. En particulier, cette expérimentation a montré que chacune de deux extensions implémentées dans Zenon Modulo ont fortement augmenté le nombre total d'obligations de preuve prouvées.

Dans le but d'augmenter la confiance dans la correction des preuves produites par Zenon Modulo, nous avons choisi de générer des certificats de preuve, des objets de preuve qui peuvent être vérifiés par un outil externe. Nous avons décidé d'utiliser le vérificateur de preuve Dedukti pour certifier les preuves produites par Zenon Modulo, un vérificateur universel de preuve implémentant le $\lambda\Pi$-calcul modulo théorie et qui est particulièrement adapté pour vérifier les preuves utilisant de la réécriture. Cela nous a permis de vérifier toutes les preuves produites par Zenon Modulo dans l'expérimentation sur la bibliothèque d'obligations de preuve de BWare.

En plus de ce travail de développement, nous avons présenté dans ce manuscrit des résultats théoriques à propos de la chaîne amont du projet BWare. Avant d'être prouvées par Zenon Modulo, les obligations de preuve B sont traduites de la logique de la méthode B vers la logique du premier ordre avec typage polymorphe par un outil de traduction. Des questionnements sur la cohérence des preuves de Zenon Modulo par rapport aux obligations de preuve originelles peuvent légitimement apparaître. En pratique, cette traduction est faite par outil propriétaire appelé bpo2why qui effectue des transformations sophistiquées, en particulier de l'inférence de types des expressions. Il aurait été difficile de certifier formellement la correction de cet outil de traduction, même si nous avions pu avoir un accès à son code source. A la place, nous avons montré qu'il est possible de traduire les preuves de Zenon Modulo, exprimées dans un calcul des séquents typé, en preuves dans le

système de preuve de la méthode B. Cela a été rendu possible en définissant un encodage des formules de la méthode B vers la logique du premier ordre avec typage polymorphe, puis une traduction syntaxique de sous-preuves monomorphes du calcul des séquents typé de Zenon Modulo vers la déduction naturelle de la méthode B. Enfin, nous avons montré que les preuves ainsi obtenues correspondent bien à des preuves des obligations de preuve initiales. Cela nous a permis d'augmenter la confiance globale quant à l'utilisation du langage WhyML pour exprimer les obligations de preuve et la théorie des ensembles de la méthode B.

Nous voyons deux perspectives intéressantes à notre travail. La première concerne la traduction des preuves de Zenon Modulo en preuves dans le système de preuve de la méthode B. La seconde perspective concerne l'utilisation de Zenon Modulo pour certifier les traces de preuve provenant d'autres outils de déduction automatique.

Une implémentation concrète de la traduction des preuves de Zenon Modulo en preuves B pourrait être un projet très intéressant. Le principal avantage serait de retirer les différentes étapes de traduction des obligations de preuve de la "zone de confiance". Pour le moment, la formule initiale est traduite par bpo2why de la méthode B vers WhyML, puis par Why3 de WhyML vers TFF1, le format d'entrée de Zenon Modulo, puis enfin par Zenon Modulo de TFF1 vers Dedukti. Une solution serait de produire une preuve B de la formule initiale en B. Avec cette approche, Zenon Modulo recevrait en entrée l'obligation de preuve traduite, et renverrait un certificat de preuve contenant uniquement la preuve dans le système de preuve de B. Cette méthode nous permettrait d'utiliser Zenon Modulo comme une véritable boite noire, sans avoir aucune inquiétude quant à la correction des outils qui la compose. Pour cela, il faudrait avoir un vérificateur de preuve B, ce qui fait pour le moment défaut.

La seconde perspective est une utilisation alternative de Zenon Modulo, inspirée par le travail de Jasmin Blanchette sur Sledgehammer [Blanchette, Böhme, Fleury, Smolka, and Steckermeier 2016]. L'idée serait de profiter du travail d'autres outils de déduction automatique qui savent produire des traces de preuve pour générer des certificats. Par exemple, l'outil de déduction automatique E peut générer des traces de preuve dans le format TSTP, un format standard de traces de preuve. Une trace de preuve TSPT est

composée d'une liste de formules correspondant à la liste des axiomes nécessaires pour prouver le but et éventuellement à une liste de lemmes intermédiaires. La réduction de la taille de l'espace de recherche, ainsi que les étapes intermédiaires données par les lemmes, constitue une aide potentiellement très intéressante pour améliorer la performance de Zenon Modulo.

Enfin, ces deux approches pourrait être combinées pour utiliser Zenon Modulo comme un générateur de preuve formelles en B à partir d'autres outils de déduction automatique. La principale motivation de cette approche serait son adaptabilité dans un contexte industriel. Les problèmes de certification des outils logiciels dans un cadre industriel rendent souvent très contraignant les améliorations et les mises à jours des outils certifiés. En effet, il est souvent nécessaire de recommencer presque intégralement un processus de certification d'un outil modifié. Dans l'approche proposée ci-dessus, seul un vérificateur de preuve B devrait être certifié, laissant possible toutes modifications des outils de déduction automatique.

Bibliography

Jean-Raymond Abrial. *The B-Book, Assigning Programs to Meanings*. Cambridge University Press, 1996.

A. Assaf, G. Burel, R. Cauderlier, D. Delahaye, G. Dowek, C. Dubois, F. Gilbert, P. Halmagrand, O. Hermant, and R. Saillard. Dedukti: a Logical Framework based on the λΠ-Calculus Modulo Theory. manuscript, 2016.

Ali Assaf and Guillaume Burel. Translating HOL to Dedukti. *arXiv preprint arXiv:1507.08720*, 2015.

Roberto Bagnara. On the Toyota UA Case and the Redefinition of Product Liability for Embedded Software, 2014. `http://www.automotive-spin.it/uploads/12/12W_Bagnara.pdf`.

Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Cambridge University Press, 2013.

Henk Barendregt and Erik Barendsen. Autarkic Computations in Formal Proofs. *Journal of Automated Reasoning (JAR)*, 28, 2002.

Michael Barr. Bookout v. Toyota, 2005 Camry L4 Software Analysis, 2013. `http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf`.

Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. METEOR: A Successful Application of B in a Large Project. In *International Symposium on Formal Methods*, pages 369–387. Springer, 1999.

Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: the Calculus of Inductive Constructions*. Springer Science & Business Media, 2013.

Evert W Beth. Semantic Entailment and Formal Derivability. *Koninklijke Nederlandse Akademie van Wentenschappen, Proceedings of the Section of Sciences*, 18(309-342):276, 1955.

Jasmin Christian Blanchette and Andrei Paskevich. TFF1: The TPTP Typed First-Order Form with Rank-1 Polymorphism. In *Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*. Springer, 2013.

Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. *Encoding Monomorphic and Polymorphic Types*, pages 493–507. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-36742-7. doi: 10.1007/978-3-642-36742-7_34. URL `http://dx.doi.org/10.1007/978-3-642-36742-7_34`.

Jasmin Christian Blanchette, Sascha Böhme, Mathias Fleury, Steffen Juilf Smolka, and Albert Steckermeier. Semi-intelligible Isar Proofs from Machine-Generated Proofs. *Journal of Automated Reasoning*, 2016. doi: 10.1007/s10817-015-9335-3. URL `https://hal.inria.fr/hal-01211748`.

François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Why3: Shepherd Your Herd of Provers. In *International Workshop on Intermediate Verification Languages (Boogie)*, 2011.

François Bobot, Sylvain Conchon, E Contejean, Mohamed Iguernelala, Stéphane Lescuyer, and Alain Mebsout. The Alt-Ergo Automated Theorem Prover, 2008, 2013.

Mathieu Boespflug and Guillaume Burel. CoqInE: Translating the Calculus of Inductive Constructions into the λΠ-calculus modulo. In *Proof Exchange for Theorem Proving (PxTP)*, 2012.

Richard Bonichon, David Delahaye, and Damien Doligez. Zenon: An Extensible Automated Theorem Prover Producing Checkable Proofs. In *Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, volume 4790 of *LNCS/LNAI*. Springer, 2007.

Paul Brauner, Clement Houtmann, and Claude Kirchner. Principles of Superdeduction. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 41–50. IEEE, 2007.

Guillaume Burel. Experimenting with Deduction Modulo. In *Conference on Automated Deduction (CADE)*, volume 6803 of *LNCS/LNAI*. Springer, 2011.

Guillaume Burel. A Shallow Embedding of Resolution and Superposition Proofs into the λΠ-Calculus Modulo. In *International Workshop on Proof Exchange for Theorem Proving (PxTP)*, 2013.

Guillaume Bury and David Delahaye. Integrating Simplex with Tableaux. In Hans de Nivelle, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, 24th International Conference, TABLEAUX 2015, Wroclaw, Poland, September 21–24, 2015. Proceedings, pages 86–101, Wrowlaw, Poland, September 2015. Hans de Nivelle. doi: 10.1007/978-3-319-24312-2_7. URL `https://hal-mines-paristech.archives-ouvertes.fr/hal-01215490`.

Guillaume Bury, Raphaël Cauderlier, and Pierre Halmagrand. Implementing Polymorphism in Zenon. In *11th International Workshop on the Implementation of Logics (IWIL)*, Suva, Fiji, November 2015a. URL `https://hal.inria.fr/hal-01243593`.

Guillaume Bury, David Delahaye, Damien Doligez, Pierre Halmagrand, and Olivier Hermant. Automated Deduction in the B Set Theory using Typed Proof Search and Deduction Modulo. In *LPAR 20 : 20th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Suva, Fiji, November 2015b. URL `https://hal-mines-paristech.archives-ouvertes.fr/hal-01204701`.

Raphaël Cauderlier. *Object-Oriented Mechanisms for Interoperability between Proof Systems*. Theses, Conservatoire national des arts et metiers - CNAM, 2016. URL `https://who.rocq.inria.fr/Raphael.Cauderlier/thesis.pdf`.

Raphaël Cauderlier and Pierre Halmagrand. Checking Zenon Modulo Proofs in Dedukti. In *Fourth Workshop on Proof eXchange for Theorem Proving (PxTP)*, Berlin, Germany, August 2015. URL `https://hal.inria.fr/hal-01171360`.

Centre d'Études et de Recherche en Informatique et Communications. https://cedric.cnam.fr/.

Max Chafkin. Uber's First Self-Driving Fleet Arrives in Pittsburgh This Month, August 2016. http://www.bloomberg.com/news/features/2016-08-18/uber-s-first-self-driving-fleet-arrives-in-pittsburgh-this-month-is06r7on.

ClearSy. http://www.clearsy.com/.

Simon Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond*. Theses, École polytechnique, September 2015. URL https://hal.archives-ouvertes.fr/tel-01223502.

Marcello D'Agostino, Dov M Gabbay, Reiner Hähnle, and Joachim Posegga. *Handbook of Tableau Methods*. Springer Science & Business Media, 2013.

David Delahaye, Damien Doligez, Frédéric Gilbert, Pierre Halmagrand, and Olivier Hermant. Zenon Modulo: When Achilles Outruns the Tortoise using Deduction Modulo. In *Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, volume 8312 of *LNCS/ARCoSS*. Springer, 2013a.

David Delahaye, Damien Doligez, Frédéric Gilbert, Pierre Halmagrand, and Olivier Hermant. Proof Certification in Zenon Modulo: When Achilles Uses Deduction Modulo to Outrun the Tortoise with Shorter Steps. In *International Workshop on the Implementation of Logics (IWIL)*, 2013b.

David Delahaye, Catherine Dubois, Claude Marché, and David Mentré. The BWare Project: Building a Proof Platform for the Automated Verification of B Proof Obligations. In *Abstract State Machines, Alloy, B, VDM, and Z (ABZ)*, LNCS. Springer, 2014.

Gilles Dowek and Alexandre Miquel. Cut elimination for Zermelo set theory. *Archive for Mathematical Logic. Springer. Submitted*, 2007.

Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning (JAR)*, 31, 2003.

C Favre. Fly-by-wire for Commercial Aircraft: the Airbus Experience. *International Journal of Control*, 59(1):139–157, 1994.

Gerhard Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische zeitschrift*, 39(1):176–210, 1935.

Martin Giese and Wolfgang Ahrendt. Hilbert's ε-Terms in Automated Theorem Proving. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 171–185. Springer, 1999.

Rajeev Goré. Tableau Methods for Modal and Temporal Logics. In *Handbook of tableau methods*, pages 297–396. Springer, 1999.

Pierre Halmagrand. Soundly Proving B Method Formulae Using Typed Sequent Calculus. In *13th International Colloquium on Theoretical Aspects of Computing (ICTAC)*, Lecture Notes in Computer Science, Taipei, Taiwan, October 2016. Springer. URL `https://hal.archives-ouvertes.fr/hal-01342849`.

Thérèse Hardin, François Pessaux, Pierre Weis, and Damien Doligez. FoCaLiZe-Reference Manual. *LIP6-CEDRIC*, 50, 2009.

Robert Harper, Furio Honsell, and Gordon Plotkin. A Framework for Defining Logics. *Journal of the ACM*, 40, 1993.

Jaakko Hintikka. Form and Content in Quantification Theory. *Acta Philosophica Fennica*, 8(7):55, 1955.

Inria. `https://www.inria.fr/`.

Mélanie Jacquel. *Proof Automation for Atelier B Rules Verification.* Theses, Conservatoire national des arts et metiers - CNAM, April 2013. URL `https://tel.archives-ouvertes.fr/tel-00840484`.

Mélanie Jacquel, Karim Berkani, David Delahaye, and Catherine Dubois. Tableaux Modulo Theories Using Superdeduction. In *International Joint Conference on Automated Reasoning*, pages 332–338. Springer, 2012.

Florent Kirchner. A Finite First-Order Theory of Classes. In *Types for Proofs and Programs*, pages 188–202. Springer, 2006.

Stephen Cole Kleene. Permutability Of Inferences In Gentzens Calculi LK And LJ. In *Bulletin Of The American Mathematical Society*, volume 57, pages 485–485. Amer Mathematical Soc 201 Charles St, Providence, RI, 1951.

Phil Koopman. A Case Study of Toyota Unintended Acceleration and Software Safety, 2014. `https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf`.

Konstantin Korovin. iProver–an Instantiation-Based Theorem Prover for First-Order Logic (system description). In *International Joint Conference on Automated Reasoning*, pages 292–298. Springer, 2008.

Laboratoire de Recherche en Informatique. `https://www.lri.fr/`.

Regis Lardennois. VAL Automated Guided Transit Characteristics and Evolutions. *Journal of advanced transportation*, 27(1):103–120, 1993.

David Mentré, Claude Marché, Jean-Christophe Filliâtre, and Masashi Asuka. Discharging Proof Obligations from Atelier B using Multiple Automated Provers. In *Abstract State Machines, Alloy, B, VDM, and Z (ABZ)*, volume 7316 of *LNCS*. Springer, 2012.

Mitsubishi Electric R&D Centre Europe. `http://www.mitsubishielectric-rce.eu/`.

Robert Nieuwenhuis and Albert Rubio. Paramodulation-Based Theorem Proving. *Handbook of automated reasoning*, 1:371–443, 2001.

OCamlPro. `https://www.ocamlpro.com/`.

David Pham. Reconstruction de Preuves au Format TSTP pour Dedukti, 2016. Rapport de Stage – ENSIIE.

Alexandre Riazanov and Andrei Voronkov. Vampire. In *International Conference on Automated Deduction*, pages 292–296. Springer, 1999.

John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965.

Alexandria Sage and Paul Lienert. Ford Plans Self-Driving Car for Ride Share Fleets in 2021, August 2016. http://www.reuters.com/article/us-ford-autonomous-idUSKCN10R1G1.

Ronan Saillard. *Typechecking in the lambda-Pi-Calculus Modulo : Theory and Practice*. Theses, Ecole Nationale Supérieure des Mines de Paris, September 2015. URL https://pastel.archives-ouvertes.fr/tel-01299180.

Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.

Raymond M Smullyan. *First-Order Logic*. Courier Corporation, 1995.

Christopher Strachey. Fundamental Concepts in Programming Languages. *Higher-order and symbolic computation*, 13(1-2):11–49, 2000.

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.

G. Sutcliffe. The CADE ATP System Competition - CASC. *AI Magazine*, 37(2):99–101, 2016.

ClearSy. Atelier B 4.1, 2013. http://www.atelierb.eu/.

Daniel Wand. Polymorphic+Typeclass Superposition. In Boris Konev, Leonardo de Moura, and Stephan Schulz, editors, *4th Workshop on Practical Aspects of Automated Reasoning (PAAR 2014)*, page 15, Vienna, Austria, July 2014. URL https://hal.inria.fr/hal-01098078.

Christoph Weidenbach. SPASS: Combining Superposition, Sorts and Splitting. *Handbook of automated reasoning*, 2:1965–2013, 1999.

Index

le cnam

Pierre HALMAGRAND

Automated Deduction and Proof Certification for the B Method

le cnam

Abstract :
The B Method is a formal method used in the railway industry to specify and develop safety-critical software. The soundness of the development depends on the validity of logical formulas called proof obligations, expressed in a specific typed set theory. We improve the automatic verification of B Method proof obligations by developing an automated theorem prover called Zenon Modulo. This new tool is an extension to polymorphic types and deduction modulo theory of the Tableau-based automated theorem prover Zenon. We also increase the confidence in the soundness of the proof generated by Zenon Modulo by generating proof certificates, proof objects that can be verified by an external proof checker called Dedukti.

Keywords :
B Method, Set theory, Zenon Modulo, Automated deduction, Deduction modulo theory, Tableau method, Sequent calculus, Polymorphism, Dedukti, $\lambda\Pi$-calculus modulo theory, Proof certification.

Résumé :
La Méthode B est une méthode formelle de spécification et de développement de logiciels critiques utilisée dans l'industrie ferroviaire. La correction du développement est garantie par la vérification de la correction de formules mathématiques appelées obligations de preuve et exprimées dans la théorie des ensembles de la Méthode B. Nous avons amélioré la vérification automatique des obligations de preuve dans la Méthode B en développant un outil de déduction automatique appelé Zenon Modulo. Ce nouvel outil est une extension au typage polymorphe et à la déduction modulo théorie de l'outil de déduction automatique basé sur la méthode des Tableaux Zenon. De plus, Zenon Modulo génère des certificats de preuve qui peuvent être vérifiés par le vérificateur de preuve Dedukti, augmentant ainsi la confiance dans la correction des résultats obtenus.

Mots clés :
Méthode B, Théorie des ensembles, Zenon Modulo, Déduction automatique, Déduction modulo théorie, Méthode des tableaux, Calcul des séquents, Polymorphisme, Dedukti, $\lambda\Pi$-calculus modulo théorie, Certification de preuve.