université PARIS-SACLAY

Translations of Proofs between Higher-Order Logics and between Theories with Rewriting

Traductions de preuves entre logiques d'ordre supérieur et entre théories avec réécriture

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°573 – Interfaces : matériaux, systèmes, usages (INTERFACES)
Spécialité de doctorat : Informatique
Graduate School : Sciences de l'Ingénierie et des Systèmes
Référent : CentraleSupélec

Thèse préparée dans l'unité de recherche **Mathématiques et Informatique pour la Complexité et les Systèmes (Université Paris-Saclay, CentraleSupélec)**,
sous la direction de **Marc AIGUIER**, professeur des universités,
le co-encadrement de **Gilles DOWEK**, directeur de recherche,
et le co-encadrement d'**Olivier HERMANT**, professeur des universités

Thèse soutenue à Paris-Saclay, le 24 septembre 2026, par

Thomas TRAVERSIE

Composition du jury

Membres du jury avec voix délibérative

Assia MAHBOUBI Directrice de recherche, Inria, Nantes Université	Rapporteuse & Examinatrice
Frank PFENNING Professeur des universités, Carnegie Mellon University	Rapporteur & Examinateur
Delia KESNER Professeure des universités, Université Paris Cité	Examinatrice
Sara NEGRI Professeure des universités, University of Genoa	Examinatrice
Elaine PIMENTEL Professeure des universités, University College London	Examinatrice
Xavier URBAIN Professeur des universités, Lyon 1 Université	Examinateur

THESE DE DOCTORAT
NNT : 2026UPAST095

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Titre : Traductions de preuves entre logiques d'ordre supérieur et entre théories avec réécriture

Mots clés : Théorie de la preuve, Théorie des types, Logique d'ordre supérieur, Réécriture, Traduction

Résumé : Les preuves formelles et les assistants de preuve sont des outils rigoureux permettant d'exprimer des théorèmes et de vérifier leur validité. Ces preuves peuvent s'appuyer sur une grande variété de logiques, être implémentées dans divers assistants de preuve et utiliser de nombreuses structures de données ou représentations. L'objectif de cette thèse est de concevoir des traductions de preuves formelles entre ces différents formalismes.

La première partie de cette thèse s'intéresse aux traductions entre les logiques d'ordre supérieur classique, intuitionniste et minimale. Nous étendons des traductions par double négation de la logique du premier ordre à la logique d'ordre supérieur, en montrant que certains des résultats valables au premier ordre ne se vérifient pas nécessairement. De plus, nous définissons une traduction, paramétrée par des opérateurs de monade, qui peut être instanciée pour traduire la logique classique d'ordre supérieur en logique intuitionniste, ou la logique intuitionniste d'ordre supérieur en logique minimale. Cette traduction nous permet également de constructiviser les preuves de formules cohérentes d'ordre supérieur. Enfin, nous définissons une logique infinitaire d'ordre supérieur, et l'étudions tant au niveau syntaxique que sémantique. Nous étendons le théorème de Barr à cette logique, ce qui nous permet de constructiviser les preuves de formules géométriques d'ordre supérieur.

Les cadres logiques sont des méta-langages conçus pour exprimer des logiques et des systèmes déductifs. La seconde partie de cette thèse se concentre sur les traductions entre les théories du $\lambda\Pi$-calcul modulo réécriture, un cadre logique particulier, dont l'implémentation Dedukti est utilisée comme pont entre les systèmes de preuve. À titre d'étude de cas, nous adaptons une traduction par double négation à ce cadre logique, et nous développons un outil qui l'implémente dans Dedukti. De plus, nous étendons les morphismes de théories et les relations logiques au $\lambda\Pi$-calcul modulo réécriture. Ces patrons de traduction peuvent être instanciés pour générer mécaniquement des traductions entre théories. La correction de ces traductions est garantie, à condition que les paramètres fournis par l'utilisateur soient corrects. Nous montrons comment tirer parti de la réécriture lors de l'utilisation de ces patrons, en prenant l'exemple d'une traduction d'une logique typée vers une logique non typée. Nous développons un outil qui implémente ces patrons de traduction dans Dedukti.

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Title: Translations of proofs between higher-order logics and between theories with rewriting

Keywords: Proof theory, Type theory, Higher-order logic, Rewriting, Translation

Abstract: Formal proofs and proof assistants are rigorous tools for expressing theorems and verifying their validity. Formal proofs can be based on a wide variety of logics, implemented in various proof assistants, and use numerous data structures or representations. The goal of this thesis is to design translations of formal proofs between such formalisms.

The first part of this thesis focuses on the translations between higher-order classical, intuitionistic and minimal logics. We extend double-negation translations from first-order logic to higher-order logic, showing that some results, that are valid in first-order logic, do not necessarily hold. Moreover, we define a translation, parameterized by monad operators, that can be instantiated to translate higher-order classical logic into intuitionistic logic, or higher-order intuitionistic logic into minimal logic. This translation also allows us to constructivize proofs of higher-order coherent formulas. Finally, we define a higher-order infinitary logic, and study it at both the syntactic and semantic level. We extend Barr's theorem to this logic, allowing us to constructivize proofs of higher-order geometric formulas.

Logical frameworks are meta-languages designed for expressing logics and deductive systems. The second part of this thesis focuses on the translations between theories of the $\lambda\Pi$-calculus modulo rewriting, a specific logical framework, whose implementation Dedukti is used as a middleware between proof systems. As a case study, we adapt a double-negation translation to this logical framework, and develop a tool that implements it in Dedukti. Furthermore, we extend theory morphisms and logical relations to the $\lambda\Pi$-calculus modulo rewriting. These translation templates can be instantiated to mechanically generate translations between theories. Provided that the parameters supplied by the user are correct, the correctness of the generated translations is guaranteed. We show how rewriting can be leveraged when applying these templates, using the example of a translation from a typed logic to an untyped logic. We develop a tool that implements such translation templates in Dedukti.

Contents

Résumé en français

Introduction

Preuves formelles et assistants de preuve. Les résultats mathématiques sont souvent démontrés en écrivant des ***preuves à la main***. Ces dernières sont en apparence rigoureuses, mais peuvent contenir des erreurs (un théorème est appliqué sans vérifier que ses hypothèses sont satisfaites, un cas est oublié, etc). De plus, la notion de preuve n'est pas absolue : une même preuve peut être considérée comme correcte par certains, mais comme incomplète ou fausse par d'autres.

Au XIXème siècle, les mathématiciens ont commencé à formaliser le raisonnement et la logique, aboutissant à la théorie de la preuve, qui considère la démonstration comme un objet mathématique à part entière. Les ***preuves formelles*** sont une combinaison d'assertions derivées les unes des autres par des règles d'inférence. Cette notion est absolue : vérifier qu'une preuve formelle est correcte est une tâche mécanique.

Le développement de l'informatique a permis des avancées majeures dans le domaine des preuves formelles. Puisque vérifier une preuve est une tâche mécanique, elle peut être accomplie par un ordinateur. De nombreux ***assistants de preuve*** ont été développés, comme Rocq [BC04], Lean [dMKA+15] ou Isabelle [Pau94]. Ces outils facilitent l'écriture des preuves par l'utilisateur, et permettent leur vérification automatique. Parmi les résultats majeurs, on peut citer la preuve du théorème des quatre couleurs [Gon08], ou encore la certification d'un compilateur pour C [LBK+16].

Interrelations et interopérabilité. Lors de l'écriture d'une preuve formelle, il faut choisir la logique dans laquelle on raisonne, les règles d'inférence avec lesquelles on construit la preuve, les structures de données auxquelles on peut faire appel, et éventuellement l'assistant de preuve dans lequel on va formaliser la preuve. Cette grande variété de choix génère plusieurs défis, aussi bien théoriques que pratiques.

Du point de vue théorique, on cherche à savoir dans quelle mesure le choix d'un formalisme particulier affecte la validité d'une preuve formelle. En d'autres termes, on s'intéresse aux ***interrelations*** entre plusieurs logiques, systèmes de preuves, structures de données et assistants de preuve.

Du point de vue pratique, on cherche à échanger des preuves entre différents outils. C'est l'***interopérabilité*** entre assistants de preuve. Cela permet notamment de réutiliser des librairies de preuves déjà développées dans d'autres outils (simplifiant ainsi la tâche de l'utilisateur), ou encore de re-vérifier les preuves dans plusieurs outils (augmentant ainsi la confiance dans les preuves et dans les assistants de preuve).

Objectifs. Dans cette thèse, nous proposons d'établir des traductions entre preuves formelles et d'approfondir leur compréhension. Nous traitons ce problème aussi bien au niveau théorique que pratique, renforçant ainsi les interrelations et l'interopérabilité. Cette thèse présente deux axes de recherche principaux, unifiés par cette même ligne directrice.

Le travail présenté dans ce manuscrit a fait l'objet de 6 articles, correspondant aux références suivantes [Tra26, Tra24a, Tra24b, Tra25, TR25, THA26].

Traductions entre logiques d'ordre supérieur

Motivation. Au XXème siècle, les réflexions liées aux fondements des mathématiques ont amené au questionnement du principe du tiers exclu (selon lequel, pour toute formule, soit celle-ci est vraie soit sa négation l'est) et du principle d'explosion (selon lequel toute formule peut être déduite d'une contradiction). On distingue la logique classique (qui admet ces deux principes), la logique intuitionniste (qui rejette le tiers exclu), et la logique minimale (qui rejette ces deux principes). En logique intuitionniste et en logique minimale, les preuves sont ***constructives***.

Toute preuve valide en logique minimale est aussi valide en logique intuitionniste, et toute preuve valide en logique intuitionniste est aussi valide en logique classique. Le sens inverse, beaucoup plus complexe, peut se traiter par deux approches. La première consiste à constructiviser une preuve classique lorsque cela est possible. D'après le théorème de Barr [Bar74], toute formule cohérente ou géométrique prouvable en logique classique admet une preuve constructive. La seconde approche consiste à traduire la logique classique (respectivement intuitionniste) dans la logique intuitionniste (respectivement minimale). Cela fonctionne pour toutes les preuves, mais modifie les énoncés prouvés. On peut notamment citer les traductions par double négation [Gli28, Kol25, Gö33, Gen36, Kur51], qui insèrent des double négations à différents endroits dans les formules.

À l'exception de [BR14], tous ces résultats ont été développés pour la logique propositionnelle et la logique du premier ordre. La logique d'ordre supérieur est très expressive, dans la mesure où elle permet de quantifier sur des fonctions, des prédicats, des propositions, etc. Elle est modélisée par le λ-calcul simplement typé.

Dans la première partie de cette thèse, nous nous focalisons sur les traductions entre logiques classique, intuitionniste et minimale d'ordre supérieur.

Traductions par double négation. Dans le Chapitre 4, nous étendons la traduction de Kuroda à l'ordre supérieur, permettant ainsi de traduire la logique classique d'ordre supérieur dans la logique intuitionniste. Nous montrons que certaines propriétés satisfaites par la traduction au premier ordre ne se vérifient plus à l'ordre supérieur. En particulier, une formule et sa traduction ne sont pas nécessairement équivalentes en logique classique. Nous montrons que ces propriétés sont vérifiées en supposant l'extensionnalité fonctionnelle et l'extensionnalité propositionnelle.

Nous montrons également que la traduction de Kolmogorov peut être étendue à l'ordre supérieur, grâce à une simple reformulation. Cette traduction va de la logique classique d'ordre supérieur à la logique minimale.

Traductions par monade. Dans le Chapitre 5, nous généralisons la traduction de Kuroda du Chapitre 4, en utilisant des opérateurs de monade à la place des double négations. En fonction de l'opérateur de monade choisi, la traduction va de la logique classique d'ordre supérieur à la logique intuitionniste, ou de la logique intuitionniste d'ordre supérieur à la logique minimale.

De plus, nous montrons, grâce à cette traduction, comment constructiviser les preuves de formules cohérentes d'ordre supérieur. En particulier, nous étendons le théorème de Barr aux formules cohérentes d'ordre supérieur.

Logique infinitaire d'ordre supérieur. Dans le Chapitre 6, nous définissons et étudions une logique infinitaire d'ordre supérieur. Elle possède des quantificateurs d'ordre supérieur, des conjonctions infinies et des disjonctions infinies.

Nous donnons une sémantique correcte à cette logique, en se basant sur les algèbres de Heyting. Nous montrons que cette sémantique est complète, en construisant un modèle intensionnel particulier. Nous en déduisons le théorème d'élimination des coupures pour la déduction naturelle.

Enfin, nous étendons le théorème de Barr aux formules géométriques d'ordre supérieur, en tirant parti de la traduction développée dans le Chapitre 5.

Traductions entre théories avec réécriture

Motivation. Les ***cadres logiques*** sont des méta-langages pour exprimer des logiques et des systèmes déductifs. On peut notamment citer le cadre logique LF [HHP93], basé sur du λ-calcul avec types dépendants. LF s'appuie sur la ***correspondance de Curry-Howard*** : les jugements sont représentés par des types, et les preuves de ces jugements sont représentés par des termes de ces types. Vérifier la validté d'une preuve revient donc à vérifier que le terme qui la représente a le bon type.

Le $\lambda\Pi$-calcul modulo réécriture, noté $\lambda\Pi/\mathcal{R}$, étend LF avec des règles de réécriture définissables par l'utilisateur. Dedukti est le langage de preuve associé à $\lambda\Pi/\mathcal{R}$. Les théories de $\lambda\Pi/\mathcal{R}$ sont un ensemble de constantes typées et de règles de réécritures. De nombreuses logiques peuvent être encodées dans $\lambda\Pi/\mathcal{R}$, en particulier les logiques sous-jacentes de divers assistants de preuve. Cela permet d'utiliser Dedukti comme un pont entre systèmes, renforçant ainsi l'interopérabilité.

Dans la seconde partie de cette thèse, nous nous focalisons sur les traductions de preuves entre différentes théories de $\lambda\Pi/\mathcal{R}$.

Traduction par double négation. Dans le Chapitre 9, nous nous intéressons aux théories encodées dans $\lambda\Pi/\mathcal{R}$ qui se basent sur logique classique d'ordre supérieur. Nous étendons la traduction de Kuroda, telle que définie dans le Chapitre 4, en prenant en compte les types dépendants, la réécriture, et la correspondance de Curry-Howard.

Nous développons un outil qui permet de traduire des preuves Dedukti d'une théorie basée sur la logique classique d'ordre supérieur à sa version intutionniste. Nous testons cet outil sur une centaine de preuves Dedukti.

Patrons de traduction. Après l'étude de cas dans le Chapitre 9, nous nous intéressons dans le Chapitre 10 à des mécanismes de traductions entre différentes théories de $\lambda\Pi/\mathcal{R}$. Nous étendons les morphismes de théories [HST94] et les relations logiques [RS13] de LF à $\lambda\Pi/\mathcal{R}$. Ces patrons de traduction peuvent être instanciés pour générer mécaniquement des traductions entre théories. Nous établissons des méta-théorèmes qui garantissent la correction de ces traductions, à condition que les paramètres fournis par l'utilisateur soient corrects.

Nous utilisons ces patrons pour formaliser des traductions entre plusieurs théories de $\lambda\Pi/\mathcal{R}$. En particulier, nous établissons une traduction d'une logique fortement typée à une logique faiblement typée puis à une logique non typée. Tous ces exemples nous permettent d'analyser le rôle de la réécriture vis-à-vis de ces patrons de traductions. La réécriture dans la théorie de départ crée de nouvelles obligations de preuves. À l'inverse, la réécriture dans la théorie d'arrivée permet de décharger certaines obligations de preuves au système plutôt qu'à l'utilisateur, facilitant ainsi la traduction.

Nous développons un outil qui implémente ces patrons de traduction dans Dedukti, et encodons les exemples développés dans le chapitre.

Conclusion et perspectives

Résumé. Cette thèse se consacre à l'étude des interrelations entre logiques d'ordre supérieur et de l'interopérabilité entre théories de $\lambda\Pi/\mathcal{R}$. Ces deux axes de travail ont été menés à bien en établissant des traductions et en montrant qu'elles sont correctes. Nous avons défini des traductions individuelles (Chapitres 4 et 9), mais aussi des traductions paramétrées qui peuvent être instanciées (Chapitres 5 et 10). Nous avons également défini et étudié une logique infinitaire d'ordre supérieure (Chapitre 6), permettant de combiner l'expressivité de la logique infinitaire et de la logique d'ordre supérieur. En somme, cette thèse fournit des outils théoriques et pratiques pour traduire des preuves formelles entre différents formalismes.

Au-delà des perspectives propres à chaque partie, deux perpespectives sur le long terme se dégagent de cette thèse.

Alignement des concepts. Lorsqu'on traduit une librairie de preuves d'un assistant de preuve vers un autre, on obtient généralement des théorèmes et preuves qui se basent sur des concepts tels que définis dans le système de départ. Afin d'obtenir les résultats basés sur les concepts tels que définis dans le système d'arrivée, il faut procéder à un alignement des concepts [MGK+17].

Les morphismes et les relations logiques sont des outils permettant d'aligner des concepts, et ce de manière mécanique à partir de paramètres fournis par l'utilisateur. En dehors de $\lambda\Pi/\mathcal{R}$, on peut par exemple citer Trocq [CCM24, CCM25], un outil basé sur la paramétricité et qui permet l'échange de preuves dans Rocq.

Impact de l'Intelligence Artificielle. Les récents développements de l'IA générative bouleversent la manière de prouver des résultats en mathématiques, mais aussi la manière de programmer en informatique. Les résultats générés par IA n'ayant pas de garantie de fiabilité, cela renforce l'intérêt des méthodes formelles.

De plus, l'IA apparaît prometeuse pour la formalisation des preuves [YPH+26]. De nombreux travaux montrent son potentiel pour formaliser des preuves à partir de preuves à la main, pour aider les utilisateurs des assistants de preuve, ou encore pour traduire entre différents langages.

Les méthodes et outils développés dans cette thèse pourraient bénéficier de l'IA. On peut par exemple imaginer qu'une IA fournisse les paramètres des morphismes et relations logiques, plutôt que les demander à l'utilisateur.

Chapter 1

Introduction

In common speech, a proof is an evidence or a reasoning that establishes the truth of a statement. The evidence or argument must be sufficiently convincing to be accepted by everyone. Whether a proof is valid or not may vary from person to person. In that sense, the notion of proof is not absolute.

Pen-and-Paper Proofs

In mathematics, a ***pen-and-paper proof*** is a structured reasoning. It makes use of the assumptions of the problem or of known facts, and combines them using implicit reasoning rules to derive the result. A pen-and-paper proof is in appearance rigorous, but it is not free from errors and imprecisions: using a theorem without checking its assumptions, forgetting a case, saying that a non-trivial argument is trivial, or missing subtle details. Checking a proof may be difficult: if it is too short, the missing details complicate the understanding, and if it is too long, the reasoning may be cumbersome. Actually, the same proof can be viewed as satisfactory by some and unsatisfactory by others, depending on the expected level of detail. A proof can even be considered correct by the community, before being later disproved.

The limitations of pen-and-paper proofs can be illustrated by the history of the Abel–Ruffini theorem, or Abel's impossibility theorem, which states that polynomial equations of degree five or higher do not have solutions in radicals. In 1799, Ruffini established a 500-page proof. Cauchy wrote that Ruffini's memoir "proves conclusively" the theorem, while Abel felt that the "memoir is so complicated that it is very difficult to determine the validity of [the] argument" [Kie71]. The correctness of Ruffini's proof was disputed, and it later turned out that it was incomplete. In 1824, Abel established a new proof. As he had to pay for the printing himself, he condensed the proof into 6 pages. This resulted in an abrupt reasoning, that was difficult to understand [Pes04].

Formal Proofs

In the 19th century, mathematicians started to formalize logic and reasoning, leading to the development of proof theory, a branch of logic that considers proofs as mathematical objects. A ***formal proof*** [AH14] is a sequence of propositions that are axioms, assumptions, or follow from the previous ones using inference rules. Every step

of the reasoning is made explicit. It is therefore easy to check a formal proof: it suffices to check that each step is correct. In particular, formal proof is not an abstract notion that depends on the judgment of the reader, and the correctness of a formal proof is defined by an objective criterion.

However, turning a pen-and-paper proof into a formal proof is often burdensome. It requires us to identify, clarify and write all the implicit details of the pen-and-paper proof. For Bourbaki [Bou70], formalizing mathematical proofs is not doable in practice:

> *"It does not take much experience to realize that such a project is utterly unfeasible; even the simplest proof at the beginning of* Theory of Sets *would already require hundreds of characters to be fully formalized."*

Such limitations were in part tackled by the development of computer science over the course of the 20th century.

Proof Assistants

The Automath project [dB80], launched in 1966 by de Bruijn, aimed at developing a system for writing and checking mathematical proofs. The project relied on the following observation: checking the correctness of a proof is a mechanical task that can be done by a computer. This idea gave rise to ***proof assistants*** [Geu09], also called interactive theorem provers. Since the pioneering Automath project, many proof assistants have been developed, including Agda [Nor05], HOL [Gor88], HOL Light [Har96], Isabelle [Pau94], Lean [dMKA+15], Nuprl [CAB+86], Matita [ACTZ07], Mizar [TB85], PVS [ORS92], Rocq (formerly Coq) [BC04], or Twelf [PS99]. Proofs that are formalized in proof assistants are called ***computer-verified proofs***.

Modern proof assistants are software tools that help humans for both the writing and the checking of formal proofs. The user formalizes the proof, assisted by the software that displays the goals to prove and the hypotheses that can be assumed. This information is updated as the user progresses through the proof. The software mechanically checks each step of the proof and signals errors to the user. Proof assistants simplify the work of the user, ensure trust in the proof, but also reveal critical when formalizing long and complicated proofs.

Proof assistants help the user in the writing of the formal proof, but they do not produce the proof. On the contrary, automated theorem provers are software tools that try to generate a formal proof of a given statement, without human intervention. Automated theorem provers follow decision or semi-decision procedures, and only succeed for a restricted class of formulas. Proof assistants may feature the possibility to call an external automated theorem prover, as Isabelle does with Sledgehammer.

Applications of Proofs Assistants

Formal proofs and proof assistants are used to show mathematical results and to certify programs. We highlight four pioneering works, about graph coloring, sphere packing, metro driving, and code compilation. They illustrate the need to formalize proofs, as well as the challenges of formalization.

The Four Color theorem states that only four colors are sufficient to color a map such that two adjacent regions have different colors. Two pen-and-paper proofs were proposed, by Kempe in 1879 and by Tait in 1880. They were considered correct at that time, but they have both been proved to be incorrect eleven years later [Tho98]. Nearly a century after these failed attempts, Appel and Haken proposed in 1976 a new proof, that mixes pen-and-paper arguments and hundreds of computations performed by computer programs. These extensive computations cannot be reasonably checked by hand, which raised controversy about the correctness of the proof. In 2004, Gonthier formalized a proof of the Four Color theorem in the Rocq proof assistant [Gon08]. All the logical steps of the proof were checked by Rocq, leading to a high level of confidence in the proof.

The Kepler conjecture states that the maximum density when packing spheres in three dimensions is 0.74. In 1953, Fejes Tóth gave a proof strategy that required a large number of computations, and he suggested solving the problem using computers. In 1998, Ferguson and Hales proved the Kepler conjecture. Their proof mixes hundreds of pages of pen-and-paper arguments and thousands of computations performed by computer programs. The proof was published in 2006, after years of difficult reviewing and without checking the entire proof. In 2003, Hales launched the Flyspeck project, which resulted in a formal proof of the Kepler conjecture in 2014 [HAB+17]. The proof used two proof assistants, HOL Light and Isabelle.

Formal proofs are also used in the industry in order to verify safety-critical systems. In the railway sector, several metro lines are fully automated and operated without human drivers, leading to safety concerns. The programs that run the automatic driverless trains need to be free of bugs, and must respect some specification, for example the impossibility of collision between two trains. Those safety requirements are addressed using formal proofs. One of the prominent examples is the Paris metro line 14, which has been developed using the B method [BKK+20].

Formally verifying a program is not enough, as the compilation phase may also go wrong: a correct source program can be compiled into an incorrect executable code. As a case in point, several bugs were found in C compilers [ER08, YCER11], and such miscompilation issues may jeopardize safety-critical systems running on C. This led to the development of CompCert C [Ler09, LBK+16], a formally verified optimizing compiler for a large subset of C, based on Rocq. It ensures that a correct source program is compiled into a correct executable code. The project, started in 2005, has been commercially available since 2015, and received the 2021 ACM Software System Award.

The Landscape of Formal Proofs

When formalizing a proof, we have a number of technical decisions to make, and many different options are available.

— We must reason inside a logic. Many logics have been defined, depending on their axioms (classical logic, intuitionistic logic, minimal logic), on the connectives they consider (propositional logic, modal logic, linear logic, infinitary logic), or on the expressive power of their quantifiers (first-order logic, higher-order logic).

— We must build the formal proof using the inference rules of a proof calculus.

Common proof calculus are Hilbert-style systems, natural deduction or sequent calculus (with or without cuts).

— We may express our results using the data structures of a theory, such as natural numbers, integers, sets, or graphs. For a same theory, we may choose between different data representations. For instance, natural numbers can be represented by Peano's natural numbers or by binary natural numbers. Peano's natural numbers are more adapted to human reasoning, while binary natural numbers allow fast computations.

— We may implement the formalization inside a proof assistant. They all have their own syntax, but also their own logical foundations, such as Martin-Löf Type Theory [ML84] or the Calculus of (Inductive) Constructions [CH88, PM93].

This large variety of formalization choices—at the level of logics, proof calculi, data structures, data representations and proof assistants—raises theoretical questions as well as practical challenges.

The Interrelation Challenge

On the theoretical side, we want to determine to what extent the choice of a specific setting affects the validity of the proof we are formalizing. In other words, we are interested in the ***interrelation*** between the different logics, proof calculi, data structures, data representations, and foundations of proof assistants. Different relations may be established between two settings: they may be equivalent, one may be an extension of the other, one may be embedded into the other, etc. For example, natural deduction and sequent calculus are equivalent, higher-order logic extends first-order logic, and classical logic can be embedded into intuitionistic logic.

Understanding the relations between two settings is typically achieved by establishing translations between them, therefore allowing the exchange of proofs. For instance, Peano's natural numbers and binary natural numbers are two equivalent data representations, and such relation can be used to translate the arithmetic library of Rocq from the former representation to the latter [MB02, Mag03].

The Interoperability Challenge

On the practical side, we are interested in the ***interoperability*** between proof assistants, that is the ability to exchange proofs across different systems. In addition to having different syntaxes and logical foundations, proof assistants may also federate different communities, that develop and maintain numerous libraries of formal proofs. When large libraries of proofs are available in a given system, they cannot be directly re-used in other proof systems, and we have to do the formalization again—which is often cumbersome. In the same way, when a proof system is not maintained anymore, all the results formalized in it become unusable. Interoperability between proof systems therefore allows the ***re-usability*** of proof libraries.

Moreover, like any other program, proof assistants may contain errors, raising doubt about their reliability. One solution to this problem is to certify the proof assistants

themselves. For instance, the MetaRocq project [SAB+20] aims at formalizing Rocq in Rocq. Alternatively, interoperability between proof systems allows the ***cross-checking*** of formal proofs by different systems, therefore strengthening the confidence we have in computer-verified proofs.

This Thesis

The goal of this thesis is to deepen the understanding, the design and the analysis of the translations of formal proofs. We investigate the topic both theoretically and practically, strengthening the interrelations and interoperability between proof systems. This thesis presents two main research axes, unified by this common thread.

The motivation, goals and contributions of each part will be detailed in their own introductory chapters (Chapters 2 and 7). They will be followed by preliminary chapters (Chapters 3 and 8). We briefly present here the main contributions of this manuscript.

In **Part I**, we study ***translations between logics***. In particular, we focus on higher-order classical logic, higher-order intuitionistic logic and higher-order minimal logic.

In Chapter 4, we extend double-negation translations to higher-order logic, and we show that straightforward properties about them in first-order logic do not hold anymore in higher-order logic.

Moreover, in Chapter 5, we define a more general translation, parameterized by monad operators, that can be instantiated to translate higher-order classical logic into intuitionistic logic, or higher-order intuitionistic logic into minimal logic, depending on the chosen monad operator. This translation also allows us to show that higher-order coherent formulas correspond to a constructive fragment. In particular, we extend Barr's theorem to higher-order coherent formulas.

Finally, in Chapter 6, we define and investigate a higher-order infinitary logic, that features higher-order quantifiers, infinite conjunctions and infinite disjunctions. We investigate this logic at both the syntactic and semantic level. In particular, we extend Barr's theorem to higher-order geometric formulas.

In **Part II**, we study ***translations between theories***. In particular, we focus on theories of the $\lambda\Pi$-calculus modulo rewriting, a logical framework with rewriting, and its implementation Dedukti.

As a case study in Chapter 9, we extend a double-negation translation from higher-order logic to theories encoded in higher-order logic in the $\lambda\Pi$-calculus modulo rewriting. We develop a tool that implements such a translation in Dedukti.

Furthermore, in Chapter 10, we extend two translation templates to the $\lambda\Pi$-calculus modulo rewriting, namely theory morphisms and logical relations. These translation templates can be instantiated to generate various translations between theories. Their correctness is guaranteed, provided that the parameters supplied by the users are correct. We analyze the advantages and drawbacks of rewriting for applying such translation templates. In particular, we apply theory morphisms to formalize a translation from a typed logic to an untyped logic. We develop a tool that implements these translation templates in Dedukti.

This thesis work resulted in the following papers, on which this manuscript is based:

1. **Kuroda's Translation for Higher-Order Logic**. Thomas Traversié. To appear in *The Journal of Symbolic Logic*, 2026. [Tra26]
2. **Monad Translations for Higher-Order Logic**. Thomas Traversié. In proceedings of the *10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025)*. [Tra25]
3. **Kuroda's Translation for the $\lambda\Pi$-Calculus Modulo Theory and Dedukti**. Thomas Traversié. In proceedings of the *Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2024)*. [Tra24a]
4. **Investigations on Higher-Order Logic**. Thomas Traversié, Olivier Hermant and Marc Aiguier. In proceedings of the *11th International Conference on Formal Structures for Computation and Deduction (FSCD 2026)*. [THA26]
5. **Proofs for Free in the $\lambda\Pi$-Calculus Modulo Theory**. Thomas Traversié. In proceedings of the *Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2024)*. [Tra24b]
6. **Formalizing Representation Theorems for a Logical Framework with Rewriting**. Thomas Traversié and Florian Rabe. Under review at *ACM Transactions on Computational Logic*, 2025. [TR25]

We have developed the following tools:

1. **Construkti**, which translates Dedukti proofs from higher-order classical logic to higher-order intuitionistic logic.

   `https://github.com/Deducteam/Construkti`

2. **TranslationTemplates**, which implements generic translations between Dedukti theories that can be instantiated by the user.

   `https://github.com/Deducteam/TranslationTemplates`

We present in Figure 1.1 an overview of the main contributions of each chapter, along with the paper it is adapted from. For clarity, the introductory chapters (Chapters 1, 2 and 7) and the concluding chapter (Chapter 11) are omitted.

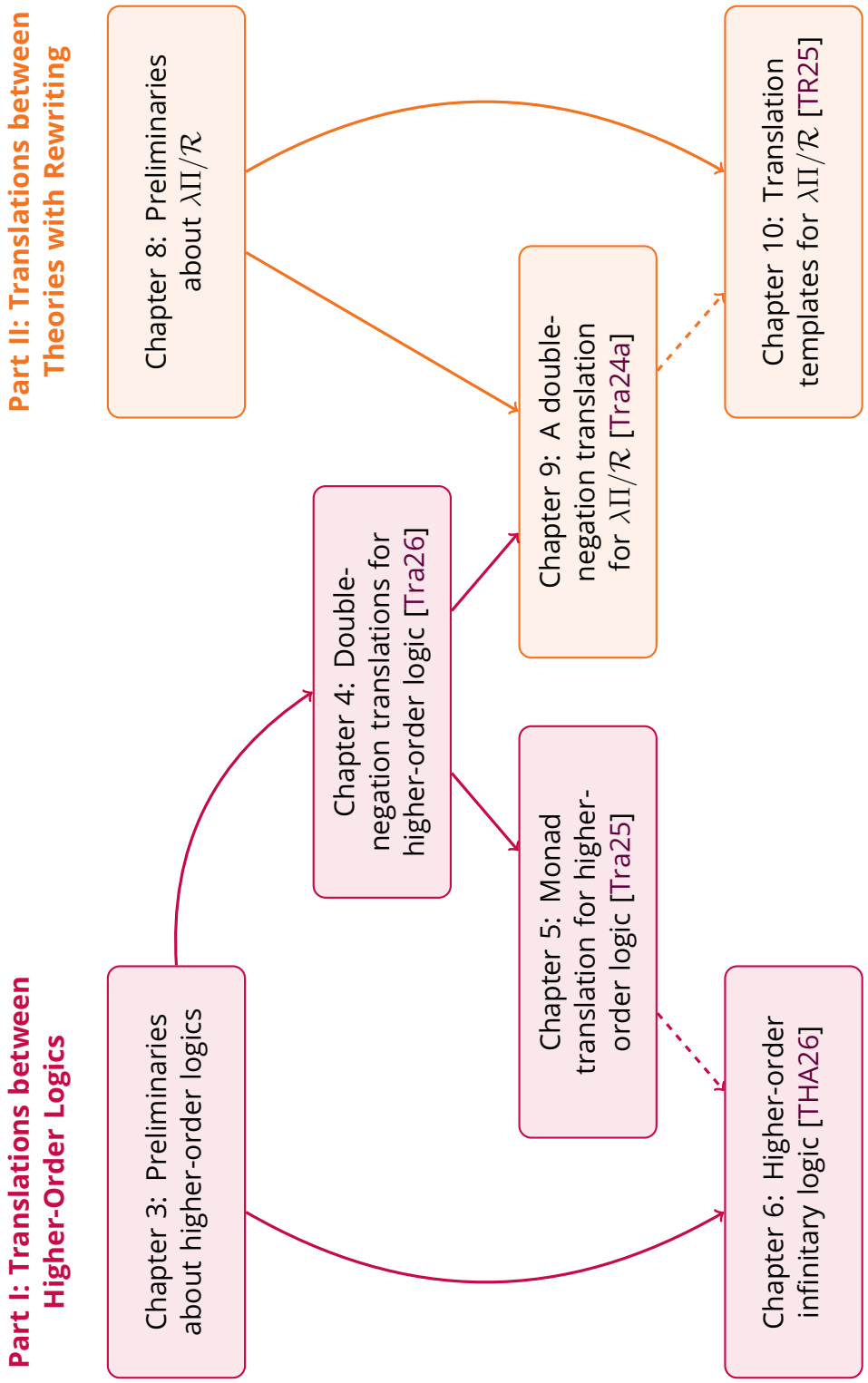


Figure 1.1: Overview of the main contributions of the thesis. Solid arrows indicate a strong dependency, and dashed arrows indicate a weak dependency. The $\lambda\Pi$-calculus modulo rewriting is abbreviated $\lambda\Pi/\mathcal{R}$.

Part I

TRANSLATIONS BETWEEN HIGHER-ORDER LOGICS

Chapter 2

Introduction of Part I

Classical Logic, Intuitionistic Logic and Minimal Logic

In the 20th century, the Brouwer-Hilbert controversy questioned the foundations of mathematical logic and reasoning, and in particular the validity of the principle of excluded middle.

$$A \vee \neg A \qquad \text{(principle of excluded middle)}$$

This principle states that any proposition is either true or its negation is true. In other words, any proposition is either true or false.

On the one hand, Brouwer defended the idea that the existence of mathematical objects should only be proved by explicitly building them. This ***constructive*** aspect is specified by the disjunction property and the witness property. The disjunction property states that if $A \vee B$ holds, then we know which of A and B holds. The witness property states that if $\exists x.A$ holds, then we can exhibit such an x. When using the principle of excluded middle, proofs are not constructive: $A \vee \neg A$ holds, but we do not know which of A or $\neg A$ is true. Brouwer consequently rejected the principle of excluded middle.

On the other hand, Hilbert [Hil27] considered the principle of excluded middle as a crucial axiom, leading him to criticize constructivism:

"Taking the principle of excluded middle from the mathematician would be the same, say, as proscribing the telescope to the astronomer or to the boxer the use of his fists. To prohibit existence statements and the principle of excluded middle is tantamount to relinquishing the science of mathematics altogether."

The principle of excluded middle is indeed a keystone tool for mathematicians. It is equivalent to the double negation elimination, which states that any proposition is true if the negation of its negation is true.

$$\neg\neg A \Rightarrow A \qquad \text{(double-negation elimination)}$$

It is also equivalent to the principle of proof by contradiction, which states that a proposition is true if its negation entails a contradiction.

$$(\neg A \Rightarrow \bot) \Rightarrow A \qquad \text{(proof by contradiction)}$$

The divergences between Brouwer and Hilbert led to the formalization of two different logics: ***classical logic***—which admits the principle of excluded middle—and ***intuitionistic logic***—which does not. Intuitionistic logic is constructive, as it satisfies the disjunction property and the witness property.

The idea of constructivism can be pushed further by rejecting the principle of explosion, which states that any formula can be derived from a contradiction.

$$\bot \Rightarrow A \qquad \text{(principle of explosion)}$$

Johansson developed ***minimal logic***, which rejects both the principle of excluded middle and the principle of explosion.

Relation between Logics

Classical logic extends intuitionistic logic, which itself extends minimal logic. It follows that provability in minimal logic entails provability in intuitionistic logic, which itself entails provability in classical logic. The reverse is however not true. A challenging problem is to investigate when it is possible to shift classical provability to intuitionistic provability, and intuitionistic provability to minimal provability.

One approach is to turn classical proofs into intuitionistic proofs without modifying the proven statement. This is called the ***constructivization*** of proofs. Such a process is not possible for any formula, but we can study particular ***fragments*** of logics that can be constructivized. For example, coherent formulas are formulas that can only be built using conjunctions, disjunctions, and existential quantifiers. Classical provability entails intuitionistic provability for coherent formulas [Neg03]. Geometric formulas are an extension of coherent formulas with infinite disjunctions. Following Barr's theorem [Bar74], classical provability entails intuitionistic provability for geometric formulas [Neg21, Rat16].

Another approach is to define ***embeddings*** between logics. This technique translates any proof from one logic to another, but in doing so it modifies the proven statement. Many different translations have been proposed over the past century, using double negations [Gli28, Kol25, Gö33, Gen36, Kur51, Kri90, Avi06] or various other operators [PM68, Fri78, Acz01, EO12, vdB19]. They embed classical logic into intuitionistic logic [Gli28, Kur51], intuitionistic logic into minimal logic [PM68], or directly classical logic into minimal logic [Kol25, Gö33, Gen36].

All those results have been established for propositional logic—in which there are no quantifiers—or for first-order logic—in which the quantifiers range over individual objects. In ***higher-order logic***, the quantifiers not only range over individual objects, but also abstract over functions, relations, predicates and propositions. This results in an expressive logic, well-suited for formalizing mathematics. Apart from [BR14], that partially embedded higher-order classical logic into intuitionistic logic, the connections between higher-order classical logic, higher-order intuitionistic logic and higher-order minimal logic remain uncharted territory.

Related Work: Ecumenical Logic

The logical connectives and quantifiers do not have the same behavior in classical logic and in intuitionistic logic. Dowek [Dow15] and Prawitz [Pra15] developed the idea that, instead of defining distinct logics, we should define distinct connectives and quantifiers—one version that corresponds to classical logic and one version that corresponds to intuitionistic logic.

This idea gave rise to ***ecumenical logic*** [PR17, PPdP18, PPdP19, PP23, Gri25], a logic in which classical and intuitionistic connectives and quantifiers coexist. In other words, formulas can contain both classical and intuitionistic symbols. The principle of excluded middle hols with the classical connectives, but does not with their intuitionistic version. A formula that only contains classical (respectively intuitionistic) symbols is provable in ecumenical logic if and only if it is provable in classical (respectively intuitionistic) logic.

Ecumenical logic is deeply related to double-negation translations. Dowek's classical connectives and quantifiers [Dow15] are directly defined as double negations of the intuitionistic ones. Prawitz's ecumenical system [Pra15] has been used to revisit Glivenko's theorem in [PBNP26] and has been related to the Gödel-Gentzen translation in [PPdP25].

While double-negation translations embed classical logic into intuitionistic logic by systematically inserting double negations, ecumenical logic offers more flexibility—we can precisely pinpoint where classical features are necessary.

Contributions of Part I

In Part I, our goal is to explore the connections between higher-order classical logic, higher-order intuitionistic logic, and higher-order minimal logic.

In **Chapter 3**, we present the preliminaries to address Part I. We give an overview of the existing double-negation translations for propositional logic and first-order logic. We recall the syntax and inference rules of higher-order logic.

In **Chapter 4**, we extend Kuroda's translation so that it embeds higher-order classical logic into higher-order intuitionistic logic. We show that some properties satisfied by the translation in first-order logic do not hold in higher-order logic, and we investigate several conditions under which they hold. Furthermore, we show that Kolmogorov's translation can be extended to higher-order logic following the same line, provided a small reformulation.

In **Chapter 5**, we generalize Kuroda's translation so that it inserts monad operators instead of double negations. Depending on the chosen monad operator, the translation embeds higher-order classical logic into higher-order intuitionistic logic, or higher-order intuitionistic logic into higher-order minimal logic. Moreover, we identify a constructive fragment of higher-order logic: for such formulas, we can transform a classical proof into an intuitionistic proof without modifying the proven statement. We prove Barr's theorem for higher-order coherent formulas.

In **Chapter 6**, we define an higher-order infinitary logic, that features higher-order quantifiers, infinite conjunctions and infinite disjunctions. We study the semantics of this logic, and prove the cut-elimination theorem. Building on the monad translation of Chapter 5, we extend Barr's theorem to higher-order geometric formulas.

Chapter 3

Preliminaries: Logics and Double Negations

Summary

In this chapter, we present the necessary preliminaries to apprehend Part I. We recall the basic knowledge about propositional logic, first-order logic and higher-order logic. We describe the important role that double negations play in intuitionistic logic, and we detail the standard double-negation translations.

3.1 Propositional and First-Order Logics

We recall the syntax and inference rules of propositional logic and first-order logic.

3.1.1 Propositional Logic

Formulas in propositional logic are built using propositional variables, implication $\Rightarrow$, conjunction $\wedge$, disjunction $\vee$ and contradiction $\bot$.

Definition 3.1: Propositional Logic

Formulas in propositional logic are inductively defined by:

$$A, B ::= p \mid A \Rightarrow B \mid A \wedge B \mid A \vee B \mid \bot$$

where p is a propositional variable.

Negation $\neg$, biconditional $\Leftrightarrow$ and tautology $\top$ are definable from these previous connectives:

$$\begin{array}{lcl} \neg A & \coloneqq & A \Rightarrow \bot \\ A \Leftrightarrow B & \coloneqq & (A \Rightarrow B) \wedge (B \Rightarrow A) \\ \top & \coloneqq & \bot \Rightarrow \bot \end{array}$$

Example 3.2 (Graph coloring). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ a graph with $\mathcal{V}$ the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ the set of edges. We want to color the graph with 4 colors so that two neighbors have different*

colors. For every $v \in \mathcal{V}$ and $i \in [\![1,4]\!]$, we declare the propositional variable $p_{v,i}$ stating that v has color i. The problem is modeled thanks to the following constraints:

— *each vertex has at least one color: for every $v \in \mathcal{V}$,*

$$p_{v,1} \vee p_{v,2} \vee p_{v,3} \vee p_{v,4}$$

— *each vertex has at most one color: for every $v \in \mathcal{V}$ and $i \neq j \in [\![1,4]\!]$,*

$$\neg(p_{v,i} \wedge p_{v,j})$$

— *two neighbors have different colors: for every $(u,v) \in \mathcal{E}$ and $i \in [\![1,4]\!]$,*

$$\neg(p_{u,i} \wedge p_{v,i})$$

The Four Color theorem ensures that it is possible to find such a coloring.

Contexts Γ are finite sequences of formulas. In natural deduction, judgments are of the form $\Gamma \vdash A$, where A is a formula and Γ a context. We write $\vdash A$ when the context is empty. Inference rules

$$\frac{\Gamma_1 \vdash A_1 \qquad \dots \qquad \Gamma_n \vdash A_n}{\Gamma \vdash A}$$

specify that the judgment $\Gamma \vdash A$ can be deduced from the judgments $\Gamma_1 \vdash A_1, \dots, \Gamma_n \vdash A_n$. The judgment $\Gamma \vdash A$ is provable when it can be obtained using the allowed inference rules. The formula A is provable when the judgment $\vdash A$ is provable.

The natural deduction rules for propositional logic are presented in Figure 3.1. Each connective is associated to introduction rules and elimination rules. The rule Bot-E corresponds to the principle of explosion. The rule PEM corresponds to the principle of excluded middle. We consider three variants of propositional logic: classical logic CL allows all these rules, intuitionistic logic IL rules out PEM, and minimal logic ML rules out PEM and Bot-E. We write $\Gamma \vdash_{\mathsf{L}} A$ when $\Gamma \vdash A$ is provable in the logic $\mathsf{L} \in \{\mathsf{CL}, \mathsf{IL}, \mathsf{ML}\}$.

Example 3.3. *The conjunction is commutative.*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge B \vdash_{\mathsf{ML}} A \wedge B}\,\textit{Ax}}{A \wedge B \vdash_{\mathsf{ML}} B}\,\textit{And-ER} \qquad \dfrac{\dfrac{}{A \wedge B \vdash_{\mathsf{ML}} A \wedge B}\,\textit{Ax}}{A \wedge B \vdash_{\mathsf{ML}} A}\,\textit{And-EL}}{A \wedge B \vdash_{\mathsf{ML}} B \wedge A}\,\textit{And-I}}{\vdash_{\mathsf{ML}} (A \wedge B) \Rightarrow (B \wedge A)}\,\textit{Imp-I}$$

Apart from the inference rules of Figure 3.1, it is also possible to show that some inference rules are admissible. We say that the inference rule

$$\frac{\Gamma_1 \vdash A_1 \qquad \dots \qquad \Gamma_n \vdash A_n}{\Gamma \vdash A}$$

is admissible when $\Gamma \vdash A$ is provable if all the premises $\Gamma_i \vdash A_i$ are provable. The natural deduction rules for tautology, negation and biconditional are admissible in minimal logic.

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\,\text{Not-I} \qquad\qquad \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot}\,\text{Not-E}$$

$$\frac{}{\Gamma, A, \Delta \vdash A}\ \text{Ax}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \text{Imp-I} \qquad \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \text{Imp-E}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \text{And-I} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \text{And-EL} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \text{And-ER}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \text{Or-IL} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \text{Or-IR}$$

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\ \text{Or-E}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \text{Bot-E} \qquad \frac{}{\Gamma \vdash A \vee \neg A}\ \text{PEM}$$

Figure 3.1: Natural deduction rules for propositional logic.

The inference rules Not-I and Not-E are direct use of Imp-I and Imp-E.

Structural rules are inference rules that are not related to a specific logical connective, like the rules of Figure 3.1. For instance, the weakening rule Weak inserts an hypothesis in the context, the contraction rule Contr removes a formula from the context if it appears two consecutive times, and the exchange rule Exch exchanges two consecutive formulas in the context.

$$\frac{\Gamma, \Delta \vdash B}{\Gamma, A, \Delta \vdash B}\ \text{Weak} \qquad \frac{\Gamma, A, A, \Delta \vdash B}{\Gamma, A, \Delta \vdash B}\ \text{Contr} \qquad \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}\ \text{Exch}$$

These three structural rules are admissible in propositional logic.

3.1.2 First-Order Logic

First-order logic extends propositional logic with the universal quantifier $\forall$ and the existential quantifier $\exists$. These quantifiers range over terms, that are built using variables and function symbols. Formulas are built using connectives, quantifiers, and predicate symbols applied to terms.

Definition 3.4: First-Order Logic

Terms in first-order logic are inductively defined by:

$$t_1, \ldots, t_n ::= x \mid c \mid f(t_1, \ldots, t_n)$$

where x is a variable, c is a constant and f is a n-ary function symbol.
Formulas in first-order logic are inductively defined by:

$$A, B ::= P(t_1, \ldots, t_n) \mid A \Rightarrow B \mid A \wedge B \mid A \vee B \mid \bot \mid \forall x.A \mid \exists x.A$$

where P is a n-ary predicate symbol.

Atomic formulas are formulas of the form $P(t_1, \ldots, t_n)$.

Example 3.5 (Equality)**.** *The equality relation $=$ is defined by a binary predicate that satisfies the reflexivity axiom*

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

and Leibniz's law (also called substitution property): for any formula $\varphi(x)$, we have

$$\forall x.\forall y.\ (x = y) \Rightarrow (\varphi(x) \Rightarrow \varphi(y))$$

In particular, Leibniz's law is not an axiom, but a axiom schema, as we cannot quantify over the formula $\varphi(x)$ in first-order logic.

Example 3.6 (Natural numbers)**.** *Natural numbers can be axiomatized in first-order logic using the Peano axioms. We take the constant 0 and the unary function* succ*, called successor function. The natural number 0 is not the successor of a natural number:*

$$\forall x.\ \neg(\mathsf{succ}(x) = 0)$$

Two natural numbers that have the same successors are equal.

$$\forall x.\forall y.\ (\mathsf{succ}(x) = \mathsf{succ}(y)) \Rightarrow x = y$$

The induction principle on natural numbers is expressed using the following axiom schema, for any formula $\varphi(x)$:

$$\Big(\varphi(0) \wedge \big(\forall x.\ \varphi(x) \Rightarrow \varphi(\mathsf{succ}(x))\big)\Big) \Rightarrow \forall x.\varphi(x)$$

This is an axiom schema and not only an axiom, as we cannot quantify over the formula φ in first-order logic.

A free variable is a variable that is not bound by a quantifier. We write $\mathrm{fv}(A_1, \ldots, A_n)$ for the free variables occurring in the formulas $A_1, \ldots, A_n$. We write $A[x \leftarrow t]$ for the formula A in which the free occurrences of x have been substituted by the term t (possibly renaming the bound variables of A to avoid capturing free variables). The natural deduction rules for first-order logic are those of propositional logic (Figure 3.1) along with those for the quantifiers (Figure 3.2).

As for propositional logic, we distinguish versions for classical logic, intuitionistic logic and minimal logic, depending on the PEM and Bot-E rules. The structural rules are admissible in first-order logic.

$$\frac{\Gamma \vdash A \qquad x \notin \mathsf{fv}(\Gamma)}{\Gamma \vdash \forall x.A}\ \text{All-I} \qquad \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[x \leftarrow t]}\ \text{All-E}$$

$$\frac{\Gamma \vdash A[x \leftarrow t]}{\Gamma \vdash \exists x.A}\ \text{Ex-I} \qquad \frac{\Gamma \vdash \exists x.A \qquad \Gamma, A \vdash C \qquad x \notin \mathsf{fv}(\Gamma, C)}{\Gamma \vdash C}\ \text{Ex-E}$$

Figure 3.2: Natural deduction rules for the quantifiers.

3.2 Classical, Intuitionistic and Minimal Logics

Classical logic admits the double-negation elimination $\neg\neg A \Rightarrow A$, while intuitionistic logic does not. In particular, in intuitionistic logic, a formula A and its double negation $\neg\neg A$ are not necessarily equivalent. The behavior of double negations is therefore at the core of the distinction between classical logic and intuitionistic logic.

3.2.1 The Role of Double Negations

The following properties specify the relation between negation and the logical connectives. They hold in propositional logic and first-order logic.

Proposition 3.7

Let A and B be formulas.

1. $\vdash_{\mathsf{ML}} A \Rightarrow \neg\neg A$
2. $\vdash_{\mathsf{ML}} \neg\neg\neg A \Leftrightarrow \neg A$
3. $\vdash_{\mathsf{ML}} \neg\neg(A \Rightarrow B) \Rightarrow (\neg\neg A \Rightarrow \neg\neg B)$
4. $\vdash_{\mathsf{IL}} (\neg\neg A \Rightarrow \neg\neg B) \Rightarrow \neg\neg(A \Rightarrow B)$
5. $\vdash_{\mathsf{ML}} \neg\neg(A \wedge B) \Leftrightarrow (\neg\neg A \wedge \neg\neg B)$
6. $\vdash_{\mathsf{ML}} \neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$

Proof. See for example [TvD88, Chapter 2] or [DNR04, Chapter 4]. □

We also specify the relation between negation and the first-order quantifiers.

Proposition 3.8

Let A and B be formulas.

1. $\vdash_{\mathsf{ML}} (\neg\neg\forall x.A) \Rightarrow (\forall x.\neg\neg A)$
2. $\vdash_{\mathsf{ML}} (\neg\exists x.A) \Leftrightarrow (\forall x.\neg A)$

Proof. See for example [TvD88, Chapter 2] or [DNR04, Chapter 4]. □

The principle of excluded middle does not hold in intuitionistic logic. However, its double negation hold in intuitionistic logic—and even in minimal logic.

Proposition 3.9

Let A be a formula. We have $\vdash_{\mathsf{ML}} \neg\neg(A \vee \neg A)$.

Proof. The following proof only uses the rules of minimal logic.

$$\dfrac{\dfrac{}{\neg(A \vee \neg A) \vdash \neg(A \vee \neg A)}\,\text{Ax} \qquad \dfrac{\dfrac{\dfrac{}{\neg(A \vee \neg A), A \vdash \neg(A \vee \neg A)}\,\text{Ax} \qquad \dfrac{\dfrac{}{\neg(A \vee \neg A), A \vdash A}\,\text{Ax}}{\neg(A \vee \neg A), A \vdash A \vee \neg A}\,\text{Or-IL}}{\neg(A \vee \neg A), A \vdash \bot}\,\text{Not-E}}{\neg(A \vee \neg A) \vdash \neg A}\,\text{Not-I}}{\neg(A \vee \neg A) \vdash A \vee \neg A}\,\text{Or-IR}}{\vdash \neg\neg(A \vee \neg A)}\,\text{Not-E}}\,\text{Not-I}$$

□

It follows that double negations are particularly interesting when translating proofs from classical logic to intuitionistic logic.

3.2.2 The Standard Double-Negation Translations

For any translation on formulas $A \mapsto A^\star$, and whenever Γ is the context $A_1, \ldots, A_n$, we write $\Gamma^\star$ for the context $A_1^\star, \ldots, A_n^\star$.

Definition 3.10: Embedding of Classical Logic into Intuitionistic Logic

Translations $A \mapsto A^\star$ embed classical logic into intuitionistic logic when they satisfy two properties:

(i) if $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma^\star \vdash_{\mathsf{IL}} A^\star$ (soundness),

(ii) $\vdash_{\mathsf{CL}} A^\star \Leftrightarrow A$ (characterization).

Both properties [TvD88, Gas13] are essential to fully characterize translations that embed classical logic into intuitionistic logic. The soundness property ensures that judgments are translated from classical logic to intuitionistic logic. The characterization property ensures that the meaning of formulas is preserved by the translation—this is captured by checking that A and $A^\star$ are equivalent in classical logic. The characterization property is sometimes replaced [TvD88, FO12b] by:

(iii) if $\Gamma^\star \vdash_{\mathsf{IL}} A^\star$ then $\Gamma \vdash_{\mathsf{CL}} A$ (reverse translation).

The reverse translation property ensures that if a translated judgment is provable in intuitionistic logic, then the original judgment is provable in classical logic. It is a consequence of the characterization property.

A standard way to embed classical logic into intuitionistic logic is to resort to double-negation translations, that are translations that insert double negations inside formulas. The most basic double-negation translation is Glivenko's translation [Gli28] for propositional logic. It only inserts a double negation in front of formulas. Whenever Γ is the context $A_1, \dots, A_n$, we write $\neg\neg\Gamma$ for the context $\neg\neg A_1, \dots, \neg\neg A_n$.

Theorem 3.11: Glivenko's Theorem

Let A be a formula and Γ be a context in propositional logic. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\neg\neg\Gamma \vdash_{\mathsf{IL}} \neg\neg A$.

Proof. By induction on the derivation, using Proposition 3.7. Glivenko's theorem is a direct consequence of Theorem 3.19, that we will prove in detail below. □

For first-order logic, Glivenko's translation does not apply anymore, and we have to insert double negations at various places in the formulas. Several such translations have been proposed. We present here the three main ones: Kolmogorov's translation [Kol25], the Gödel-Gentzen translation [Gö33, Gen36] and Kuroda's translation [Kur51].

Kolmogorov's translation [Kol25] inserts double negations in front of *every subformula*.

Definition 3.12: Kolmogorov's Translation

Kolmogorov's translation $A \mapsto A^{\mathsf{ko}}$ is defined by induction:

$$\begin{array}{lcll}
(A \Rightarrow B)^{\mathsf{ko}} & := & \neg\neg(A^{\mathsf{ko}} \Rightarrow B^{\mathsf{ko}}) & \\
(A \wedge B)^{\mathsf{ko}} & := & \neg\neg(A^{\mathsf{ko}} \wedge B^{\mathsf{ko}}) & \\
(A \vee B)^{\mathsf{ko}} & := & \neg\neg(A^{\mathsf{ko}} \vee B^{\mathsf{ko}}) & \\
A^{\mathsf{ko}} & := & \neg\neg A & \text{if } A \text{ is atomic or } \bot \\
(\forall x.A)^{\mathsf{ko}} & := & \neg\neg\forall x.A^{\mathsf{ko}} & \\
(\exists x.A)^{\mathsf{ko}} & := & \neg\neg\exists x.A^{\mathsf{ko}}. &
\end{array}$$

Some of those double negations are not necessary to embed classical logic into intuitionistic logic. The Gödel-Gentzen translation [Gö33, Gen36] refines Kolmogorov's translation, only inserting double negations in front of disjunctions, existential quantifiers and atomic formulas.

Definition 3.13: Gödel-Gentzen Translation

The Gödel-Gentzen translation $A \mapsto A^{\mathsf{gg}}$ is defined by induction:

$$\begin{array}{lcll}
(A \Rightarrow B)^{\mathsf{gg}} & := & A^{\mathsf{gg}} \Rightarrow B^{\mathsf{gg}} & \\
(A \wedge B)^{\mathsf{gg}} & := & A^{\mathsf{gg}} \wedge B^{\mathsf{gg}} & \\
(A \vee B)^{\mathsf{gg}} & := & \neg\neg(A^{\mathsf{gg}} \vee B^{\mathsf{gg}}) & \\
A^{\mathsf{gg}} & := & \neg\neg A & \text{if } A \text{ is atomic or } \bot \\
(\forall x.A)^{\mathsf{gg}} & := & \forall x.A^{\mathsf{gg}} & \\
(\exists x.A)^{\mathsf{gg}} & := & \neg\neg\exists x.A^{\mathsf{gg}}. &
\end{array}$$

Remark 3.14. *The Gödel-Gentzen translation is sometimes defined with*

$$\begin{array}{lcl}
(A \vee B)^{\mathsf{gg}} & := & \neg(\neg A^{\mathsf{gg}} \wedge \neg B^{\mathsf{gg}}) \\
(\exists x.A)^{\mathsf{gg}} & := & \neg\forall x.\neg A^{\mathsf{gg}}
\end{array}$$

Both presentations are equivalent, due to Proposition 3.7(6) and Proposition 3.8(2).

Alternatively, Kuroda's translation [Kur51] extends Glivenko's approach to first-order logic. It inserts double negations in front of formulas, and additionally inserts double negations after universal quantifiers. This results in a two-step translation.

Definition 3.15: Kuroda's Translation

Kuroda's translation $A \mapsto A^{\mathsf{ku}}$ is defined by $A^{\mathsf{ku}} = \neg\neg A_{\mathsf{ku}}$, where $A \mapsto A_{\mathsf{ku}}$ is defined by induction:

$$\begin{array}{lcll} (A \Rightarrow B)_{\mathsf{ku}} & := & A_{\mathsf{ku}} \Rightarrow B_{\mathsf{ku}} & \\ (A \wedge B)_{\mathsf{ku}} & := & A_{\mathsf{ku}} \wedge B_{\mathsf{ku}} & \\ (A \vee B)_{\mathsf{ku}} & := & A_{\mathsf{ku}} \vee B_{\mathsf{ku}} & \\ A_{\mathsf{ku}} & := & A & \text{if } A \text{ is atomic or } \bot \\ (\forall x.A)_{\mathsf{ku}} & := & \forall x.\neg\neg A_{\mathsf{ku}} & \\ (\exists x.A)_{\mathsf{ku}} & := & \exists x.A_{\mathsf{ku}}. & \end{array}$$

Example 3.16. *The double-negation translations of* $A := \forall x.(P(x) \Rightarrow \exists y.(R(x,y) \wedge P(y)))$ *are:*

$$\begin{array}{lcl} A^{\mathsf{ko}} & := & \neg\neg\forall x.\neg\neg(\neg\neg P(x) \Rightarrow (\neg\neg\exists y.\neg\neg(\neg\neg R(x,y) \wedge \neg\neg P(y)))) \\ A^{\mathsf{gg}} & := & \forall x.(\neg\neg P(x) \Rightarrow (\neg\neg\exists y.(\neg\neg R(x,y) \wedge \neg\neg P(y)))) \\ A^{\mathsf{ku}} & := & \neg\neg\forall x.\neg\neg(P(x) \Rightarrow (\exists y.(R(x,y) \wedge P(y)))) \end{array}$$

Kolmogorov's translation is always the one that inserts the most double negations. In general, Kuroda's translation is the most lightweight translation.

As we are in first-order logic, substitutions are used in the natural deduction rules of the quantifiers. The translations commute with substitution.

Lemma 3.17. *For* $\star \in \{\mathsf{ko}, \mathsf{gg}, \mathsf{ku}\}$*, we have* $(A[x \leftarrow t])^\star = A^\star[x \leftarrow t]$.

Proof. Let us detail the case for Kuroda's translation. First, we prove $(A[x \leftarrow t])_{\mathsf{ku}} = A_{\mathsf{ku}}[x \leftarrow t]$ by induction on the formula A.

— If A is atomic or $\bot$, we directly have $(A[x \leftarrow t])_{\mathsf{ku}} = A_{ku}[x \leftarrow t]$.

— If $\circ \in \{\Rightarrow, \wedge, \vee\}$, we have

$$\begin{array}{lll} ((A \circ B)[x \leftarrow t])_{\mathsf{ku}} & = (A[x \leftarrow t] \circ B[x \leftarrow t])_{\mathsf{ku}} & \\ & = (A[x \leftarrow t])_{\mathsf{ku}} \circ (B[x \leftarrow t])_{\mathsf{ku}} & \\ & = A_{\mathsf{ku}}[x \leftarrow t] \circ B_{\mathsf{ku}}[x \leftarrow t] & \text{by induction hypotheses} \\ & = (A_{\mathsf{ku}} \circ B_{\mathsf{ku}})[x \leftarrow t] & \\ & = (A \circ B)_{\mathsf{ku}}[x \leftarrow t] & \end{array}$$

— For the universal quantifier, we have

$$\begin{array}{lll} ((\forall z.A)[x \leftarrow t])_{\mathsf{ku}} & = (\forall z.A[x \leftarrow t])_{\mathsf{ku}} & \\ & = \forall z.\neg\neg(A[x \leftarrow t])_{\mathsf{ku}} & \\ & = \forall z.\neg\neg A_{\mathsf{ku}}[x \leftarrow t] & \text{by induction hypothesis} \\ & = (\forall z.\neg\neg A_{\mathsf{ku}})[x \leftarrow t] & \\ & = (\forall z.A)_{\mathsf{ku}}[x \leftarrow t] & \end{array}$$

The case for the existential quantifier is treated similarly.

Then we get $(A[x \leftarrow t])^{\mathsf{ku}} = \neg\neg(A[x \leftarrow t])_{\mathsf{ku}} = \neg\neg(A_{\mathsf{ku}}[x \leftarrow t]) = (\neg\neg A_{\mathsf{ku}})[x \leftarrow t] = A^{\mathsf{ku}}[x \leftarrow t]$.

For Kolmogorov's translation and the Gödel-Gentzen translation, we directly prove $(A[x \leftarrow t])^\star = A^\star[x \leftarrow t]$ by induction on the formula A. □

Kolmogorov's translation, the Gödel-Gentzen translation and Kuroda's translation satisfy both the soundness property and the characterization property. We can actually go further and refine the soundness property in two different ways.

First, Kolmogorov's translation and the Gödel-Gentzen translation not only embed classical logic into intuitionistic logic, but also in minimal logic.

Theorem 3.18

Let Γ, A be first-order formulas. Let $\star \in \{\mathsf{ko}, \mathsf{gg}\}$.

1. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma^\star \vdash_{\mathsf{ML}} A^\star$.
2. We have $\vdash_{\mathsf{CL}} A^\star \Leftrightarrow A$.

Proof. See for example [TvD88, Chapter 2] or [DNR04, Chapter 4]. □

Second, the double negations added by Kuroda's translation in front of the formulas in the context can be dropped from the statement of the soundness property. In other words, we state the soundness property with Γ_{ku} instead of Γ^{ku}, hence introducing less double negations. The version with Γ^{ku} is provable as well.

Theorem 3.19

Let Γ, A be first-order formulas.

1. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} A^{\mathsf{ku}}$.
2. We have $\vdash_{\mathsf{CL}} A^{\mathsf{ku}} \Leftrightarrow A$.

Proof of Theorem 3.19(1). By induction on the derivation.

— Ax: Using Ax, we get $\Gamma_{\mathsf{ku}}, A_{\mathsf{ku}}, \Delta_{\mathsf{ku}} \vdash_{\mathsf{IL}} A_{\mathsf{ku}}$. We conclude $\Gamma_{\mathsf{ku}}, A_{\mathsf{ku}}, \Delta_{\mathsf{ku}} \vdash_{\mathsf{IL}} A^{\mathsf{ku}}$ using Proposition 3.7(1).

— Imp-I: By induction hypothesis, we have $\Gamma_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg B_{\mathsf{ku}}$. Using Imp-I, we get $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} A_{\mathsf{ku}} \Rightarrow \neg\neg B_{\mathsf{ku}}$. We successively use Proposition 3.7(1) to get $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \Rightarrow \neg\neg B_{\mathsf{ku}})$, Proposition 3.7(3) to get $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}} \Rightarrow \neg\neg\neg\neg B_{\mathsf{ku}}$, and Proposition 3.7(2) to get $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}} \Rightarrow \neg\neg B_{\mathsf{ku}}$. We conclude $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \Rightarrow B_{\mathsf{ku}})$ using Proposition 3.7(4).

— Imp-E: By induction hypotheses, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \Rightarrow B_{\mathsf{ku}})$ and $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}$. Using Proposition 3.7(3), we derive $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}} \Rightarrow \neg\neg B_{\mathsf{ku}}$. Using Imp-E, we conclude $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg B_{\mathsf{ku}}$.

— And-I: By induction hypotheses, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}$ and $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg B_{\mathsf{ku}}$. Using And-I, we derive $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}} \wedge \neg\neg B_{\mathsf{ku}}$. We conclude $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \wedge B_{\mathsf{ku}})$ using Proposition 3.7(5).

— And-EL: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \wedge B_{\mathsf{ku}})$. Using Proposition 3.7(5), we derive $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}} \wedge \neg\neg B_{\mathsf{ku}}$. Using And-ER, we conclude $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg B_{\mathsf{ku}}$. We proceed similarly for And-ER.

— Or-IL: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Hypothesis}}{\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}}}{\Gamma_{\mathsf{ku}}, \neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}}) \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}}\,\text{Weak} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_{\mathsf{ku}}, \neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}}) \vdash_{\mathsf{IL}} \neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}})}\,\text{Ax}}{\Gamma_{\mathsf{ku}}, \neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}}) \vdash_{\mathsf{IL}} \neg A_{\mathsf{ku}} \wedge \neg B_{\mathsf{ku}}}\,\text{Proposition 3.7(6)}}{\Gamma_{\mathsf{ku}}, \neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}}) \vdash_{\mathsf{IL}} \neg A_{\mathsf{ku}}}\,\text{And-EL}}\,\text{Not-E}}{\Gamma_{\mathsf{ku}}, \neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}}) \vdash_{\mathsf{IL}} \bot}\,\text{Not-I}}{\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}})}$$

We proceed similarly for Or-IR.

— Or-E: By induction hypotheses, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}})$, and $\Gamma_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg C_{\mathsf{ku}}$, and $\Gamma_{\mathsf{ku}}, B_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg C_{\mathsf{ku}}$. We want to derive $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg C_{\mathsf{ku}}$. Using Not-I, we have to prove $\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \bot$. We prove that contradiction using Not-E with

$$\dfrac{\dfrac{\dfrac{\text{Hypothesis}}{\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}})}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \vee B_{\mathsf{ku}})}\,\text{Weak}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg(\neg A_{\mathsf{ku}} \wedge \neg B_{\mathsf{ku}})}\,\text{Proposition 3.7(6)}$$

and with

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Hypothesis}}{\Gamma_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg C_{\mathsf{ku}}}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg C_{\mathsf{ku}}}\,\text{Weak} \qquad \dfrac{}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg C_{\mathsf{ku}}}\,\text{Ax}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \bot}\,\text{Not-E}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg A_{\mathsf{ku}}}\,\text{Not-I} \qquad \dfrac{\dfrac{\text{Idem}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg B_{\mathsf{ku}}}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg A_{\mathsf{ku}} \wedge \neg B_{\mathsf{ku}}}\,\text{And-I}}$$

— Bot-E: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\bot$. By weakening, we get $\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\bot$, that is $\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} (\bot \Rightarrow \bot) \Rightarrow \bot$. We derive $\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \bot$ using Imp-E and the tautology $\bot \Rightarrow \bot$. We conclude $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}$ using Not-I.

— All-I: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}$. We derive $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \forall x.\neg\neg A_{\mathsf{ku}}$ using All-I. Using Proposition 3.7(1), we conclude $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\forall x.\neg\neg A_{\mathsf{ku}}$.

— All-E: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\forall x.\neg\neg A_{\mathsf{ku}}$. Using Proposition 3.8(1), we derive $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \forall x.\, \neg\neg\neg\neg A_{\mathsf{ku}}$. Using All-E, we obtain $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\neg\neg A_{\mathsf{ku}}[x \leftarrow t]$. Thanks to Proposition 3.7(2), we get $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}[x \leftarrow t]$. Using Lemma 3.17, we conclude $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} (A[x \leftarrow t])^{\mathsf{ku}}$.

— Ex-I: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} (A[x \leftarrow t])^{\mathsf{ku}}$, that is $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}[x \leftarrow t]$ using Lemma 3.17. We want to prove $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\exists x.A_{\mathsf{ku}}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Hypothesis}}{\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}[x \leftarrow t]}}{\Gamma_{\mathsf{ku}}, \forall x.\neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}[x \leftarrow t]}\,\text{Weak} \quad \dfrac{\dfrac{}{\Gamma_{\mathsf{ku}}, \forall x.\neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \forall x.\neg A_{\mathsf{ku}}}\,\text{Ax}}{\Gamma_{\mathsf{ku}}, \forall x.\neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg A_{\mathsf{ku}}[x \leftarrow t]}\,\text{All-E}}{\Gamma_{\mathsf{ku}}, \forall x.\neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \bot}\,\text{Not-E}}{\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\forall x.\neg A_{\mathsf{ku}}}\,\text{Not-I}}{\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\exists x.A_{\mathsf{ku}}}\,\text{Proposition 3.8(2)}$$

— Ex-E: By induction hypotheses, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\exists x.A_{\mathsf{ku}}$ and $\Gamma_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg C_{\mathsf{ku}}$. We want to prove $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg C_{\mathsf{ku}}$. Using Not-I, we have to prove $\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \bot$. We prove that contradiction using Not-E with

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Hypothesis}}{\Gamma_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg C_{\mathsf{ku}}}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg C_{\mathsf{ku}}}\,\text{Weak} \quad \dfrac{}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg C_{\mathsf{ku}}}\,\text{Ax}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}}, A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \bot}\,\text{Not-E}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg A_{\mathsf{ku}}}\,\text{Not-I}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \forall x.\neg A_{\mathsf{ku}}}\,\text{All-I}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\forall x.\neg A_{\mathsf{ku}}}\,\text{Proposition 3.7(1)}$$

and with

$$\dfrac{\dfrac{\dfrac{\text{Hypothesis}}{\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\exists x.A_{\mathsf{ku}}}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\exists x.A_{\mathsf{ku}}}\,\text{Weak}}{\Gamma_{\mathsf{ku}}, \neg C_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\forall x.\neg A_{\mathsf{ku}}}\,\text{Proposition 3.8(2)}$$

— PEM: Using Proposition 3.9 and weakening, we get $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \vee \neg A_{\mathsf{ku}})$.

□

Remark 3.20. *Unlike Kolmogorov's translation and the Gödel-Gentzen translation, Kuroda's translation embeds classical logic into intuitionistic logic and not into minimal logic. The only place in the proof where Bot-E is required is when we use Proposition 3.7(4) in the Imp-I case. Kuroda's translation therefore embed classical logic into minimal logic for the fragment of first-order logic that does not contain implication.*

Proof of Theorem 3.19(2). We first prove $\vdash_{\mathsf{CL}} A_{\mathsf{ku}} \Leftrightarrow A$ by induction on the formula A.

— If A is atomic or $\bot$, we directly have $\vdash_{\mathsf{CL}} A \Leftrightarrow A$.

— If $\circ \in \{\Rightarrow, \wedge, \vee\}$, we easily derive $\vdash_{\mathsf{CL}} (A_{\mathsf{ku}} \circ B_{\mathsf{ku}}) \Leftrightarrow (A \circ B)$ using the induction hypotheses $\vdash_{\mathsf{CL}} A_{\mathsf{ku}} \Leftrightarrow A$ and $\vdash_{\mathsf{CL}} B_{\mathsf{ku}} \Leftrightarrow B$.

— For the case of the universal quantifier, we have $\vdash_{\mathsf{CL}} A_{\mathsf{ku}} \Leftrightarrow A$ by induction hypothesis. Therefore we derive $\vdash_{\mathsf{CL}} \neg\neg A_{\mathsf{ku}} \Leftrightarrow A$ using the double-negation elimination of classical logic and Proposition 3.7(1). From this, we easily derive $\vdash_{\mathsf{CL}} \forall x.\neg\neg A_{\mathsf{ku}} \Leftrightarrow \forall x.A$.

— For the case of the existential quantifier, we have $\vdash_{\mathsf{CL}} A_{\mathsf{ku}} \Leftrightarrow A$ by induction hypothesis. From this, we easily derive $\vdash_{\mathsf{CL}} \exists x.A_{\mathsf{ku}} \Leftrightarrow \exists x.A$.

We conclude $\vdash_{\mathsf{CL}} A^{\mathsf{ku}} \Leftrightarrow A$ using the double-negation elimination of classical logic and Proposition 3.7(1). □

Different techniques were developed to reduce the number of double negations inserted by the previous translations. Boudard and Hermant [BH13] refined Kolmogorov's translation and the Gödel-Gentzen translation by polarizing formulas. Gilbert [Gil15] proposed a syntax-directed template of double-negation translation, that can be instantiated to yield Kolmogorov's translation, the Gödel-Gentzen translation and Kuroda's translation. In particular, he defined a minimal syntax-directed translation.

3.3 Towards Higher-Order Logic

The quantifiers of first-order logic range over terms built using constants and functions. Higher-order logic extends first-order logic, by allowing quantifiers to abstract over functions, predicates and propositions. It follows that higher-order logic is highly expressive [Far08]. A standard way to treat these quantifications is to resort to Church's simple type theory [Chu40], also called simply typed λ-calculus. We will use the term "higher-order logic" to refer to higher-order logic modeled using simple type theory.

3.3.1 Simple Type Theory

Definition 3.21: Simple Type Theory

Types are defined inductively by:

$$\tau, \sigma ::= \iota \mid o \mid \tau \to \sigma$$

where ι is the type of individuals and o is the type of propositions.
Terms are defined by:

$$t, u ::= x \mid c \mid t\ u \mid \lambda x.\ t$$

where x is a variable, c is a constant, $t\ u$ is an application and $\lambda x.\ t$ is an abstraction.

Terms $(t\ u)\ v$ are written $t\ u\ v$, and types $\tau_1 \to (\tau_2 \to \tau_3)$ are written $\tau_1 \to \tau_2 \to \tau_3$.

Remark 3.22. *It is also possible to define simple type theory with multiple base types $\iota_1, \ldots, \iota_n$.*

Typing contexts Γ are sequences of typed variables. Typing judgments $\Gamma \vdash t : \tau$ specify that the term t is of type τ in the typing context Γ. The type of the constants are declared in a signature Σ. The typing rules are presented in Figure 3.3. The typing judgments $\Gamma \vdash t : \tau$ should not be confuse with the provability judgments $\Gamma \vdash A$.

We write $\mathsf{fv}(t_1, \ldots, t_n)$ for the free variables occurring in the terms $t_1, \ldots, t_n$. While free variables are defined relatively to the quantifiers in first-order logic, they are defined relatively to the λ-abstractions in higher-order logic. We write $t[x \leftarrow u]$ for the term t in which the free occurrences of x have been substituted by u (possibly renaming bound variables of t to avoid capturing free variables).

Computation is introduced in this λ-calculus through the β-reduction rule

$$(\lambda x.\ t)\ u \hookrightarrow t[x \leftarrow u].$$

$$\frac{c : \tau \in \Sigma}{\Gamma \vdash c : \tau}\ \text{Const} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \text{Var} \qquad \frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x.\ t : \sigma \to \tau}\ \text{Abs}$$

$$\frac{\Gamma \vdash t : \sigma \to \tau \qquad \Gamma \vdash u : \sigma}{\Gamma \vdash t\ u : \tau}\ \text{App}$$

Figure 3.3: Typing rules of simple type theory.

Definition 3.23: β-Reduction and β-Conversion

The β-reduction relation $\hookrightarrow_\beta$ is the smallest relation such that:

— $(\lambda x.\ t)\ u \hookrightarrow_\beta t[x \leftarrow u]$.

— If $t \hookrightarrow_\beta t'$ then $t\ u \hookrightarrow_\beta t'\ u$ and $u\ t \hookrightarrow_\beta u\ t'$ and $\lambda x.\ t \hookrightarrow_\beta \lambda x.\ t'$.

The relation $\hookrightarrow_\beta^*$ is the reflexive and transitive closure of $\hookrightarrow_\beta$. The relation $\equiv_\beta$ is the reflexive, symmetric and transitive closure of $\hookrightarrow_\beta$.

Example 3.24. *We have* $c\ u\ ((\lambda z.\ z)\ v) \hookrightarrow_\beta c\ u\ v$ *and* $c\ ((\lambda z.\ z)\ u)\ v \hookrightarrow_\beta c\ u\ v$. *Therefore* $c\ u\ ((\lambda z.\ z)\ v) \equiv_\beta c\ ((\lambda z.\ z)\ u)\ v$. *We have* $(\lambda x.\ \lambda y.\ c\ x\ y)\ u\ ((\lambda z.\ z)\ v) \hookrightarrow_\beta^* c\ u\ v$.

Substitution and the β-reduction relation preserve typing.

Lemma 3.25 (Substitution)**.** *If* $\Gamma, x : \sigma \vdash t : \tau$ *and* $\Gamma \vdash u : \sigma$, *then* $\Gamma \vdash t[x \leftarrow u] : \tau$.

Proof. By induction on $\Gamma, x : \sigma \vdash t : \tau$. □

Lemma 3.26 (Subject Reduction)**.** *If* $\Gamma \vdash t : \tau$ *and* $t \hookrightarrow_\beta u$, *then* $\Gamma \vdash u : \tau$.

Proof. By induction on $t \hookrightarrow_\beta u$. For the β-reduction rule, we use Lemma 3.25. □

We now briefly introduce two major properties satisfied by the β-reduction relation, namely strong normalization and confluence.

Definition 3.27: Strong Normalization

A term is in β-normal form when no β-reduction is possible. A term strongly normalizes (or terminates) when there are no infinite reduction from it.

In simple type theory, the β-reduction relation is strongly normalizing.

Lemma 3.28. *Every typed term has a* β*-normal form.*

Proof. See the reducibility technique [Tai67, Gir72]. □

Definition 3.29: Confluence

The β-reduction relation is confluent when, for any terms t, u_1 and u_2, if $t \hookrightarrow_\beta^* u_1$ and $t \hookrightarrow_\beta^* u_2$ then there exists some term v such that $u_1 \hookrightarrow_\beta^* v$ and $u_2 \hookrightarrow_\beta^* v$.

In other words, confluence states that the order in which we apply the β-reductions does not matter, as we can always find a common reduct.

Lemma 3.30. *The β-reduction relation is confluent.*

Proof. See the parallel reduction technique [Bar84, Chapter 3]. □

Example 3.31. *The term $c\ ((\lambda z.\ z)\ u)\ ((\lambda z.\ z)\ v)$ β-reduces to $c\ u\ ((\lambda z.\ z)\ v)$ but also to $c\ ((\lambda z.\ z)\ u)\ v$. Both $c\ u\ ((\lambda z.\ z)\ v)$ and $c\ ((\lambda z.\ z)\ u)\ v$ β-reduce to $c\ u\ v$.*

Additionally, it is possible to consider η-conversion: $\lambda x.\ t\ x \equiv_\eta t$ whenever x is not free in t. Intuitively, it states that there is no difference between the abstraction $\lambda x.\ t\ x$ and t. η-conversion can be generated by the η-contraction rule $\lambda x.\ t\ x \hookrightarrow t$ or the η-expansion rule $t \hookrightarrow \lambda x.\ t\ x$.

Definition 3.32: $\beta\eta$-Reduction and $\beta\eta$-Conversion

The $\beta\eta$-reduction relation $\hookrightarrow_{\beta\eta}$ is defined like $\hookrightarrow_\beta$, with an additional case for η-contraction:

$$\lambda x.\ t\ x \hookrightarrow t, \text{ where } x \text{ is not free in } t$$

The relation $\equiv_{\beta\eta}$ is the reflexive, symmetric and transitive closure of $\hookrightarrow_{\beta\eta}$.

We use the subscript $\beta(\eta)$ to indicate that either variant can be used if that variant is globally fixed.

Lemma 3.33. *The conversion $\equiv_{\beta(\eta)}$ is a congruence relation.*

Proof. Let us prove the following case: if $t \equiv_{\beta(\eta)} t'$ and $u \equiv_{\beta(\eta)} u'$, then $t\ u \equiv_{\beta(\eta)\mathcal{R}} t'\ u'$. We easily prove by induction on $t \equiv_{\beta(\eta)} t'$ that $t \equiv_{\beta(\eta)} t'$ entails $t\ u \equiv_{\beta(\eta)} t'\ u$. Similarly, we prove by induction on $u \equiv_{\beta(\eta)} u'$ that $u \equiv_{\beta(\eta)} u'$ entails $t'\ u \equiv_{\beta(\eta)} t'\ u'$. The combination of the two results achieves the case. The other cases are proved similarly. □

3.3.2 Higher-Order Logic

Higher-order logic is introduced thanks to particular constants that define the logical connectives and quantifiers. There are contradiction $\bot$ of type o and implication $\Rightarrow$, conjunction $\wedge$ and disjunction $\vee$ of type $o \to o \to o$. For every type τ, there are quantifiers that range over terms of type τ: the universal quantifier $\forall_\tau$ and the existential quantifier $\exists_\tau$ of type $(\tau \to o) \to o$. For convenience, terms of the form $\forall_\tau\ (\lambda x.\ A)$ and $\exists_\tau\ (\lambda x.\ A)$ are abbreviated as $\forall x.A$ and $\exists x.A$. Again, negation is defined by $\neg A := A \Rightarrow \bot$, the logical biconditional $\Leftrightarrow$ is defined by $A \Leftrightarrow B := (A \Rightarrow B) \wedge (B \Rightarrow A)$, and tautology is defined by $\top := \bot \Rightarrow \bot$. Formulas are terms of type o.

Higher-order logic is particularly helpful for encoding the *axiom schemas* of Example 3.5 and Example 3.6 directly as *axioms* in higher-order logic.

Example 3.34 (Equality)**.** *We set the constant $=$ of type $\iota \to \iota \to o$. Leibniz's law is directly expressible using the higher-order axiom*

$$\forall P.\forall x.\forall y.\ (x = y) \Rightarrow (P\ x \Rightarrow P\ y)$$

while it requires an axiom schema in first-order logic.

Example 3.35 (Natural numbers)**.** *We set the constant 0 of type ι and the constant* succ *of type $\iota \to \iota$. The induction principle is directly expressible using the higher-order axiom*

$$\forall P.\ \Big(P\ 0 \wedge \big(\forall x.\ P\ x \Rightarrow P\ (\mathsf{succ}\ x)\big)\Big) \Rightarrow \forall x.\ P\ x$$

while it requires an axiom schema in first-order logic.

The natural deduction rules for higher-order logic are those of propositional logic (Figure 3.1) along with the rules of Figure 3.4. The Conv rule allows terms to be considered modulo the $\beta(\eta)$-conversion. Note that the Conv rule implicitly requires that B is of type o. The rules for the higher-order quantifiers are more general than the rules for the first-order quantifiers.

$$\frac{\Gamma \vdash P\ x \qquad x \notin \mathsf{fv}(\Gamma, P)}{\Gamma \vdash \forall\ P}\ \text{All-I} \qquad \frac{\Gamma \vdash \forall\ P}{\Gamma \vdash P\ t}\ \text{All-E}$$

$$\frac{\Gamma \vdash P\ t}{\Gamma \vdash \exists\ P}\ \text{Ex-I} \qquad \frac{\Gamma \vdash \exists\ P \qquad \Gamma, P\ x \vdash A \qquad x \notin \mathsf{fv}(\Gamma, P, A)}{\Gamma \vdash A}\ \text{Ex-E}$$

$$\frac{\Gamma \vdash A \qquad A \equiv_{\beta(\eta)} B}{\Gamma \vdash B}\ \text{Conv}$$

Figure 3.4: Natural deduction rules for the higher-order quantifiers and for conversion.

We distinguish versions for classical logic, intuitionistic logic and minimal logic, depending on the PEM and Bot-E rules.

Propositions 3.7 and 3.9 also hold in higher-order logic, and their proofs remain unchanged. Proposition 3.8 can be simply refined using the higher-order quantifiers.

Proposition 3.36

Let P be a predicate.

1. $\vdash_{\mathsf{ML}} \neg\neg\forall\ P \Rightarrow \forall x.\neg\neg(P\ x)$
2. $\vdash_{\mathsf{ML}} \neg\exists\ P \Leftrightarrow \forall x.\neg(P\ x)$

3.3.3 Extensions with Equality and Extensionality

In Example 3.34, we defined an equality between terms of type ι. We can also define, for every type τ, an equality symbol $=_\tau$ of type $\tau \to \tau \to o$. The symbols are infix, and we write $t = u$ when there is no ambiguity on the type τ. Equality is included in the logic using the natural deduction rules given in Figure 3.5. The introduction rule of equality Eq-I is reflexivity. The elimination rule of equality Eq-E corresponds to Leibniz's law: if the predicate P holds for u and $u = v$, then the predicate P also holds for v. Functional extensionality FunExt states that two pointwise-equal functions are equal. Propositional extensionality PropExt states that two equivalent propositions are equal.

$$\frac{}{\Gamma \vdash u = u}\ \text{Eq-I} \qquad \frac{\Gamma \vdash P\ u \qquad \Gamma \vdash u = v}{\Gamma \vdash P\ v}\ \text{Eq-E}$$

$$\frac{\Gamma \vdash f\ x = g\ x \qquad x \notin \mathsf{fv}(\Gamma, f, g)}{\Gamma \vdash f = g}\ \text{FunExt} \qquad \frac{\Gamma \vdash A \Leftrightarrow B}{\Gamma \vdash A = B}\ \text{PropExt}$$

Figure 3.5: Natural deduction rules for equality.

Given $\mathsf{L} \in \{\mathsf{CL}, \mathsf{IL}, \mathsf{ML}\}$ and $* \in \{\mathfrak{e}, \mathfrak{ep}, \mathfrak{ef}, \mathfrak{efp}\}$, we write $\Gamma \vdash^*_{\mathsf{L}} A$ when $\Gamma \vdash_{\mathsf{L}} A$ is derivable with additional inference rules:

— with Eq-I and Eq-E when $\mathfrak{e}$ occurs in $*$,

— with PropExt when $\mathfrak{p}$ occurs in $*$,

— with FunExt when $\mathfrak{f}$ occurs in $*$.

Note that FunExt and PropExt only make sense if we have Eq-I and Eq-E.

Example 3.37. *The inverse of FunExt is admissible*

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} f\ x = f\ x}\ \textit{Eq-I}}{\Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} (\lambda h.\ f\ x = h\ x)\ f}\ \textit{Conv} \qquad \Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} f = g}{\dfrac{\Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} (\lambda h.\ f\ x = h\ x)\ g}{\Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} f\ x = g\ x}\ \textit{Conv}}\ \textit{Eq-E}$$

as well as the inverse of PropExt

$$\dfrac{\dfrac{\Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} A \Leftrightarrow A}{\Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} (\lambda C.\ A \Leftrightarrow C)\ A}\ \textit{Conv} \qquad \Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} A = B}{\dfrac{\Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} (\lambda C.\ A \Leftrightarrow C)\ B}{\Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} A \Leftrightarrow B}\ \textit{Conv}}\ \textit{Eq-E}$$

since we easily derive $\Gamma \vdash^{\mathfrak{e}}_{\mathsf{ML}} A \Leftrightarrow A$.

Chapter 4

Double-Negation Translations

Summary

In this chapter, we elaborate on the work of Brown and Rizkallah [BR14]. They extended Kuroda's double-negation translation to higher-order logic, but did not prove the characterization property or the reverse translation property. We show that these properties, although straightforward in first-order logic, do not hold anymore. We prove that functional extensionality and propositional extensionality are sufficient conditions to recover them. Furthermore, Brown and Rizkallah showed that the translation fails in the presence of functional extensionality. We investigate several conditions under which Kuroda's translation works with functional extensionality. Finally, they stated that Kolmogorov's translation cannot be extended *in a natural way* to higher-order logic. We show that it is possible to do so, using a reformulation of Komogorov's translation by Dowek and Werner [DW03].

This chapter is adapted from [Tra26].

To our knowledge, the only investigation on double-negation translations for higher-order logic has been made by Brown and Rizkallah [BR14]. They showed that Kolmogorov's translation and the Gödel-Gentzen translation cannot be directly extended to higher-order logic, as they do not preserve β-conversion. For example, $(\lambda p.\ p)\ q$ and q are β-convertible, but their translations are not:

$$\begin{array}{lcll} ((\lambda p.\ p)\ q)^{\mathsf{ko}} & := & (\lambda p.\ \neg\neg p)\ \neg\neg q & \equiv_\beta \quad \neg\neg\neg\neg q \\ q^{\mathsf{ko}} & := & \neg\neg q & \end{array}$$

The problem here is that the double negations are inserted twice, for the bound variable p and for the argument q.

Brown and Rizkallah showed that Kuroda's translation can be extended to higher-order logic so that it translates classical proofs into intuitionistic profs. But they did not examine the characterization property and reverse translation property. Moreover, they looked into different forms of extensionality, and showed that the soundness property fails in the presence of functional extensionality.

Contributions. In this chapter, we fill these gaps and propose an in-depth investigation of double-negation translations for higher-order logic.

First, we prove that the soundness property holds in the presence of functional extensionality, assuming a weak form of functional extensionality. We discuss alternative conditions, including the principle of double-negation shift.

Second, we show that the characterization property and the reverse translation property—which are routine in first-order logic—do not necessarily hold in higher-order logic. We identify a necessary and sufficient condition for the characterization property. We prove that both properties hold under functional extensionality and propositional extensionality.

Third, we show that Kolmogorov's translation can be extended to higher-order logic using a reformulation by Dowek and Werner [DW03], allowing us to translate higher-order classical logic directly into higher-order minimal logic.

Outline. We extend Kuroda's translation to higher-order logic in Section 4.1. We investigate the soundness property in Section 4.2, the characterization property in Section 4.3, and the reverse translation property in Section 4.4. We extend Kolmogorov's translation to higher-order logic in Section 4.5.

4.1 Kuroda's Translation for Higher-Order Logic

Kuroda's translation for first-order logic [Kur51] inserts a double negation in front of formulas and a double negation after every universal quantifier. We express Kuroda's translation in terms of higher-order logic.

Definition 4.1: Kuroda's Translation for Higher-Order Logic

Let A be a formula in higher-order logic. Its Kuroda's translation is $A^{\mathsf{ku}} := \neg\neg A_{\mathsf{ku}}$, where A_{ku} is inductively defined by:

$$\begin{array}{lcll} x_{\mathsf{ku}} & := & x & \\ c_{\mathsf{ku}} & := & \begin{cases} \lambda p.\, \forall x. \neg\neg (p\ x) & \text{if } c = \forall \\ c & \text{otherwise} \end{cases} & \\ (\lambda x.\ t)_{\mathsf{ku}} & := & \lambda x.\ t_{\mathsf{ku}} & \\ (t\ u)_{\mathsf{ku}} & := & t_{\mathsf{ku}}\ u_{\mathsf{ku}} & \end{array}$$

While in first-order logic we have $(A[z \leftarrow w])^{\mathsf{ku}} = A^{\mathsf{ku}}[z \leftarrow w]$, this result does not hold anymore in higher-order logic. Indeed, w may contain universal quantifiers and may therefore be modified by the translation. We would intuitively expect $(A[z \leftarrow w])^{\mathsf{ku}} = A^{\mathsf{ku}}[z \leftarrow w^{\mathsf{ku}}]$. Instead, we have $(A[z \leftarrow w])^{\mathsf{ku}} = A^{\mathsf{ku}}[z \leftarrow w_{\mathsf{ku}}]$, because the double negation in front of the formula is inserted after the inductive translation.

Proposition 4.2

For any terms t and w, we have $(t[z \leftarrow w])_{\mathsf{ku}} = t_{\mathsf{ku}}[z \leftarrow w_{\mathsf{ku}}]$.

Proof. By induction on the term t.

— Constant: $(c[z \mapsto w])_{\mathsf{ku}} = c_{\mathsf{ku}} = c_{\mathsf{ku}}[z \mapsto w_{\mathsf{ku}}]$ since c and c_{ku} are closed terms.

— Variable: If $x \neq z$ then $(x[z \mapsto w])_{\mathsf{ku}} = x_{\mathsf{ku}} = x_{\mathsf{ku}}[z \mapsto w_{\mathsf{ku}}]$.
If $x = z$ then $(x[z \mapsto w])_{\mathsf{ku}} = w_{\mathsf{ku}} = x[z \mapsto w_{\mathsf{ku}}] = x_{\mathsf{ku}}[z \mapsto w_{\mathsf{ku}}]$.

— Abstraction: We have

$$\begin{array}{rcll} ((\lambda x.\ t)[z \mapsto w])_{\mathsf{ku}} & = & (\lambda x.\ t[z \mapsto w])_{\mathsf{ku}} & \\ & = & \lambda x.\ (t[z \mapsto w])_{\mathsf{ku}} & \\ & = & \lambda x.\ t_{\mathsf{ku}}[z \mapsto w_{\mathsf{ku}}] & \text{by induction hypothesis} \\ & = & (\lambda x.\ t_{\mathsf{ku}})[z \mapsto w_{\mathsf{ku}}] & \\ & = & (\lambda x.\ t)_{\mathsf{ku}}[z \mapsto w_{\mathsf{ku}}] & \end{array}$$

— Application: We have

$$\begin{array}{rcll} ((t\ u)[z \mapsto w])_{\mathsf{ku}} & = & (t[z \mapsto w]\ u[z \mapsto w])_{\mathsf{ku}} & \\ & = & (t[z \mapsto w])_{\mathsf{ku}}\ (u[z \mapsto w])_{\mathsf{ku}} & \\ & = & t_{\mathsf{ku}}[z \mapsto w_{\mathsf{ku}}]\ u_{\mathsf{ku}}[z \mapsto w_{\mathsf{ku}}] & \text{by induction hypotheses} \\ & = & (t_{\mathsf{ku}}\ u_{\mathsf{ku}})[z \mapsto w_{\mathsf{ku}}] & \\ & = & (t\ u)_{\mathsf{ku}}[z \mapsto w_{\mathsf{ku}}] & \end{array}$$

□

Corollary 4.3. *For any higher-order formula A, we have $(A[z \leftarrow w])^{\mathsf{ku}} = A^{\mathsf{ku}}[z \leftarrow w_{\mathsf{ku}}]$.*

Proof. We have $(A[z \leftarrow w])_{\mathsf{ku}} = A_{\mathsf{ku}}[z \leftarrow w_{\mathsf{ku}}]$ by Proposition 4.2. Therefore we get $(A[z \leftarrow w])^{\mathsf{ku}} = \neg\neg(A[z \leftarrow w])_{\mathsf{ku}} = \neg\neg(A_{\mathsf{ku}}[z \leftarrow w_{\mathsf{ku}}]) = (\neg\neg A_{\mathsf{ku}})[z \leftarrow w_{\mathsf{ku}}]$. □

Higher-order logic is defined using simple type theory, so $\beta(\eta)$-conversions may be used in the derivations. As Kuroda's proof relies on the fact that we can translate each step of the derivation, each time we have $A \equiv_{\beta(\eta)} B$ in the classical derivation, we want to use $A^{\mathsf{ku}} \equiv_{\beta(\eta)} B^{\mathsf{ku}}$ in the intuitionistic derivation.

Proposition 4.4

For any terms t and u, if $t \hookrightarrow_{\beta(\eta)} u$ then $t_{\mathsf{ku}} \hookrightarrow_{\beta(\eta)} u_{\mathsf{ku}}$.

Proof. By induction on $t \hookrightarrow_{\beta(\eta)} u$ (following Definition 3.23).

— We have $((\lambda x.\ t)\ u)_{\mathsf{ku}} = (\lambda x.\ t_{\mathsf{ku}})\ u_{\mathsf{ku}}$ and $(\lambda x.\ t_{\mathsf{ku}})\ u_{\mathsf{ku}} \hookrightarrow_{\beta} t_{\mathsf{ku}}[x \leftarrow u_{\mathsf{ku}}]$. Therefore $((\lambda x.\ t)\ u)_{\mathsf{ku}} \hookrightarrow_{\beta} (t[x \leftarrow u])_{\mathsf{ku}}$ using Proposition 4.2.

— Suppose that x is not free in t. We have $(\lambda x.\ t\ x)_{\mathsf{ku}} = \lambda x.\ t_{\mathsf{ku}}\ x \hookrightarrow_{\beta\eta} t_{\mathsf{ku}}$.

— By induction hypothesis we get $t_{\mathsf{ku}} \hookrightarrow_{\beta(\eta)} t'_{\mathsf{ku}}$. We directly have $t_{\mathsf{ku}}\ u_{\mathsf{ku}} \hookrightarrow_{\beta(\eta)} t'_{\mathsf{ku}}\ u_{\mathsf{ku}}$ and $u_{\mathsf{ku}}\ t_{\mathsf{ku}} \hookrightarrow_{\beta(\eta)} u_{\mathsf{ku}}\ t'_{\mathsf{ku}}$ and $\lambda x.\ t_{\mathsf{ku}} \hookrightarrow_{\beta(\eta)} \lambda x.\ t'_{\mathsf{ku}}$. □

Corollary 4.5. *For any terms t and u, if $t \equiv_{\beta(\eta)} u$ then $t_{\mathsf{ku}} \equiv_{\beta(\eta)} u_{\mathsf{ku}}$.*

Proof. Directly follows from Proposition 4.4, as $\equiv_{\beta(\eta)}$ is the reflexive, symmetric and transitive closure of $\hookrightarrow_{\beta(\eta)}$. □

Corollary 4.6. *For any higher-order formulas A and B, if $A \equiv_{\beta(\eta)} B$ then $A_{\mathsf{ku}} \equiv_{\beta(\eta)} B_{\mathsf{ku}}$ and $A^{\mathsf{ku}} \equiv_{\beta(\eta)} B^{\mathsf{ku}}$.*

Proof. We have $A_{\mathsf{ku}} \equiv_{\beta(\eta)} B_{\mathsf{ku}}$ by Corollary 4.5, and therefore $\neg\neg A_{\mathsf{ku}} \equiv_{\beta(\eta)} \neg\neg B_{\mathsf{ku}}$. □

4.2 Soundness Property

In this section, we investigate the soundness property: if $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} A^{\mathsf{ku}}$. We prove it with and without the extensionality principles. In particular, we identify different conditions that are sufficient to derive the soundness property in the presence of functional extensionality.

Note that we state and prove the soundness property with Γ_{ku} instead of Γ^{ku}, thus introducing fewer double negations in the context. All the results can be easily adapted to have the version with Γ^{ku}.

4.2.1 Without Extensionality

To prove the soundness property, Brown and Rizkallah [BR14] proceed in two steps—transforming A into a formula A' that does not contain any universal quantifier and then applying to A' an extension of Glivenko's theorem to higher-order logic without universal quantifiers. Instead of adopting this method, we proceed to a more direct proof that follows the intuition of the first-order case.

Theorem 4.7: Soundness Property

Let A be a formula and Γ be a context in higher-order logic. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} A^{\mathsf{ku}}$.

Proof. By induction on the derivation. We only show the cases specific to higher-order logic, the remaining cases are the same as in Theorem 3.19.

— All-I: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(P_{\mathsf{ku}}\ x)$. Using Conv, we derive $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} (\lambda y.\ \neg\neg(P_{\mathsf{ku}}\ y))\ x$. Using All-I, we get $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \forall x.\neg\neg(P_{\mathsf{ku}}\ x)$. We conclude using Proposition 3.7(1).

— All-E: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\forall x.\neg\neg(P_{\mathsf{ku}}\ x)$. Using Proposition 3.36(1), we get $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \forall x.\neg\neg\neg\neg(P_{\mathsf{ku}}\ x)$. Using All-E, we obtain $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\neg\neg(P_{\mathsf{ku}}\ t_{\mathsf{ku}})$. We conclude using Proposition 3.7(2).

— Ex-I: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(P_{\mathsf{ku}}\ t_{\mathsf{ku}})$. We write Δ for the context $\Gamma_{\mathsf{ku}}, \forall x.\neg(P_{\mathsf{ku}}\ x)$.

$$\dfrac{\dfrac{\dfrac{\text{Hypothesis}}{\Delta \vdash_{\mathsf{IL}} \neg\neg(P_{\mathsf{ku}}\ t_{\mathsf{ku}})}\ \text{Weakening} \qquad \dfrac{\dfrac{}{\Delta \vdash_{\mathsf{IL}} \forall x.\neg(P_{\mathsf{ku}}\ x)}\ \text{Ax}}{\Delta \vdash_{\mathsf{IL}} \neg(P_{\mathsf{ku}}\ t_{\mathsf{ku}})}\ \text{All-E}}{\Delta \vdash_{\mathsf{IL}} \bot}\ \text{Not-E}}{\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\forall x.\neg(P_{\mathsf{ku}}\ x)}\ \text{Not-I}}{\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\exists P_{\mathsf{ku}}}\ \text{Proposition 3.36(2)}$$

— Ex-E: By induction hypotheses, we get $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\exists P_{\mathsf{ku}}$ and $\Gamma_{\mathsf{ku}}, P_{\mathsf{ku}}\ x \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}$. We want to prove $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}$. We use Not-I and then Not-E with $\neg\forall x.\neg(P_{\mathsf{ku}}\ x)$. The proof of the first subgoal is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Hypothesis}}{\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}}, P_{\mathsf{ku}}\ x \vdash_{\mathsf{IL}} \neg\neg A_{\mathsf{ku}}}\text{Weakening} \quad \dfrac{}{\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}}, P_{\mathsf{ku}}\ x \vdash_{\mathsf{IL}} \neg A_{\mathsf{ku}}}\text{Ax}}{\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}}, P_{\mathsf{ku}}\ x \vdash_{\mathsf{IL}} \bot}\text{Not-E}}{\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg(P_{\mathsf{ku}}\ x)}\text{Not-I}}{\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \forall x.\neg(P_{\mathsf{ku}}\ x)}\text{All-I}}{\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\forall x.\neg(P_{\mathsf{ku}}\ x)}\text{Proposition 3.7(1)}$$

The second subgoal $\Gamma_{\mathsf{ku}}, \neg A_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\forall x.\neg(P_{\mathsf{ku}}\ x)$ derives from Proposition 3.36(2), weakening and $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg\exists P_{\mathsf{ku}}$.

— Conv derives from the induction hypothesis and Corollary 4.6. □

4.2.2 With Extensionality

We now investigate the soundness property in the setting of extensionality. We want to prove that $\Gamma \vdash^{*}_{\mathsf{CL}} A$ implies $\Gamma_{\mathsf{ku}} \vdash^{*}_{\mathsf{IL}} A^{\mathsf{ku}}$ whatever $* \in \{\mathfrak{e}, \mathfrak{ep}, \mathfrak{ef}, \mathfrak{efp}\}$. Brown and Rizkallah showed that it fails in the presence of the FunExt rule. To get around this problem, it is sufficient to assume an axiom stating a weak form of functional extensionality, where double negations have been inserted in front of equality predicates:

$$\forall f \forall g.(\forall x.\neg\neg(f\ x = g\ x)) \Rightarrow \neg\neg(f = g) \qquad (\Delta^{\mathfrak{ef}})$$

Remark 4.8. *If we consider functional extensionality as an axiom (that is a formula in the context) instead an inference rule, then its translation is equivalent to $\Delta^{\mathfrak{ef}}$ using Proposition 3.7(2), Proposition 3.7(3), Proposition 3.7(4) and Proposition 3.36(1).*

Theorem 4.9: Soundness Property

Let A be a formula and Γ be a context in higher-order logic.

1. For $* \in \{\mathfrak{e}, \mathfrak{ep}\}$, if $\Gamma \vdash^{*}_{\mathsf{CL}} A$ then $\Gamma_{\mathsf{ku}} \vdash^{*}_{\mathsf{IL}} A^{\mathsf{ku}}$.
2. For $* \in \{\mathfrak{ef}, \mathfrak{efp}\}$, if $\Gamma \vdash^{*}_{\mathsf{CL}} A$ then $\Delta^{\mathfrak{ef}}, \Gamma_{\mathsf{ku}} \vdash^{*}_{\mathsf{IL}} A^{\mathsf{ku}}$.

Proof. For the first item, we complete the proof of Theorem 4.7 with the additional cases:

— Eq-I derives from Eq-I and Proposition 3.7(1).

— Eq-E: We have $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(P_{\mathsf{ku}}\ u_{\mathsf{ku}})$ and $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(u_{\mathsf{ku}} = v_{\mathsf{ku}})$ by induction hypotheses. We directly have $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} P_{\mathsf{ku}}\ u_{\mathsf{ku}} \Rightarrow (u_{\mathsf{ku}} = v_{\mathsf{ku}}) \Rightarrow P_{\mathsf{ku}}\ v_{\mathsf{ku}}$ using Eq-E. Thus we derive $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(P_{\mathsf{ku}}\ u_{\mathsf{ku}}) \Rightarrow \neg\neg(u_{\mathsf{ku}} = v_{\mathsf{ku}}) \Rightarrow \neg\neg(P_{\mathsf{ku}}\ v_{\mathsf{ku}})$ using Proposition 3.7(1) and Proposition 3.7(3). We conclude $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(P_{\mathsf{ku}}\ v_{\mathsf{ku}})$.

— PropExt: By induction hypothesis, we have $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \Leftrightarrow B_{\mathsf{ku}})$. Using PropExt, we directly have $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} (A_{\mathsf{ku}} \Leftrightarrow B_{\mathsf{ku}}) \Rightarrow (A_{\mathsf{ku}} = B_{\mathsf{ku}})$. Using Proposition 3.7(1) and Proposition 3.7(3), we get $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} \Leftrightarrow B_{\mathsf{ku}}) \Rightarrow \neg\neg(A_{\mathsf{ku}} = B_{\mathsf{ku}})$. We conclude $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(A_{\mathsf{ku}} = B_{\mathsf{ku}})$.

For the second item, we reuse the cases of the first item with $\Delta^{\mathfrak{ep}}$ in the context, and we prove the additional FunExt case. We use the assumption $\Delta^{\mathfrak{ef}}$ on the induction hypothesis $\Delta^{\mathfrak{ef}}, \Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(f_{\mathsf{ku}}\ x = g_{\mathsf{ku}}\ x)$ to get $\Delta^{\mathfrak{ef}}, \Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} \neg\neg(f_{\mathsf{ku}} = g_{\mathsf{ku}})$. □

In classical logic, $\Delta^{\mathfrak{ef}}$ is equivalent to functional extensionality. We link $\Delta^{\mathfrak{ef}}$ to the well-known principle of double-negation shift.

$$(\forall x.\neg\neg A) \Rightarrow (\neg\neg\forall x.A) \qquad \text{(DNS)}$$

Such a principle implies $\Delta^{\mathfrak{ef}}$ in intuitionistic logic with functional extensionality.

Proposition 4.10

DNS $\vdash^{\mathfrak{ef}}_{\mathsf{IL}} \Delta^{\mathfrak{ef}}$.

Proof. We have DNS $\vdash^{\mathfrak{ef}}_{\mathsf{IL}} \forall f\forall g.(\forall x.f\ x = g\ x) \Rightarrow (f = g)$ using FunExt. We successively apply Proposition 3.7(1), Proposition 3.36(1) and Proposition 3.7(3) to derive DNS $\vdash^{\mathfrak{ef}}_{\mathsf{IL}} \forall f\forall g.\neg\neg(\forall x.f\ x = g\ x) \Rightarrow \neg\neg(f = g)$. Using DNS, we conclude DNS $\vdash^{\mathfrak{ef}}_{\mathsf{IL}} \Delta^{\mathfrak{ef}}$. □

The double-negation shift is derivable in classical logic but not in intuitionistic logic. It is deeply connected to Glivenko's theorem—which is generalized by Kuroda's translation. More precisely, such a principle is required to extend Glivenko's theorem to first-order logic [Ume59, Gab72] and to substructural first-order logic [FO12a]. Moreover, the modal counterpart of the double-negation shift $\Box\neg\neg A \Rightarrow \neg\neg\Box A$ plays an important role when studying Glivenko's theorem for modal logic [LPR17]. Here, we have emphasized a new connection between the double-negation shift and Kuroda's translation.

To prove the soundness property in the presence of functional extensionality, it is sufficient to assume the weak form of functional extensionality or the double-negation shift.

Remark 4.11. *We could also assume the double-negation elimination on equality predicates* $\forall x\forall y.\neg\neg(x = y) \Rightarrow x = y$*, written* $\mathsf{DNE}^{\mathfrak{e}}$*. Indeed, it implies* $\Delta^{\mathfrak{ef}}$ *in intuitionistic logic with functional extensionality. However, such a principle must be handled with care: when combined with the comprehension axiom of set theory or with propositional extensionality, it entails the double-negation elimination*[1]*. For instance, let us prove* $\mathsf{DNE}^{\mathfrak{e}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg P \Rightarrow P$ *for any proposition* P*. We easily derive* $\vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg P \Rightarrow \neg\neg(P = \top)$ *and* $\vdash^{\mathfrak{e}}_{\mathsf{IL}} (P = \top) \Rightarrow P$*. The double-negation elimination then follows from* $\mathsf{DNE}^{\mathfrak{e}}$ *applied to* P *and* $\top$*.*

Another approach to get around the problem of the soundness property in presence of functional extensionality is to apply a translation that eliminates extensionality [Gan56, Tak53, BR13]. To do so, the Takeuti-Gandy translation [Gan56, Tak53] resorts to relativization and quantifies over extensional elements. This translation was later refined [BR13] and coupled with a double-negation translation. The main advantage of our approach is that the translated statement

[1] I would like to thank Dominik Kirst for pointing that out.

remains close to the original statement, as no predicate has been inserted to account for the elimination of functional extensionality. In exchange, however, we assume an additional axiom.

4.3 Characterization Property

We have shown that, if a formula A is provable from a context Γ, then there exists an intuitionistic proof of A^{ku} from Γ_{ku}. We now want to prove the characterization property: A and A^{ku} are classically equivalent. Such a result is routine in first-order logic, but does not generally hold in higher-order logic.

4.3.1 Counter-Example

Before developing a counter-example of the characterization property, let us illustrate the *intensional* (or *non-extensional*) nature of higher-order logic. Consider a constant P of type $o \to o$. Even though the propositions $\bot$ and $\bot \wedge \bot$ are equivalent, it is impossible to prove that $P\ \bot$ and $P\ (\bot \wedge \bot)$ are equivalent without further assumptions. This can be shown using the intensional models[2] of higher-order logic developed by Takahashi [Tak67] and Prawitz [Pra68], based on the V-complex construction [And71]. In such models, the semantic value of a proposition is not its truth value, but a pair consisting of a syntactic value and its truth value. $\bot$ and $\bot \wedge \bot$ have the same truth value, but different syntactic values; therefore they have different semantic values. The truth value of $P\ (\bot \wedge \bot)$ (respectively $P\ \bot$) depends on the whole semantic value of $\bot \wedge \bot$ (respectively $\bot$). It follows that $P\ \bot$ and $P\ (\bot \wedge \bot)$ can be assigned different truth values. Thus, in the absence of extensionality, we cannot prove that $P\ \bot$ and $P\ (\bot \wedge \bot)$ are equivalent, even though $\bot$ and $\bot \wedge \bot$ are. Such a lack of extensionality is a feature of the λProlog logic programming language [MN12].

Consider a constant P of type $o \to o$ and a proposition B that contains at least one occurrence of the quantifier $\forall$. We take $A := P\ B$. We would like to show $\vdash_{\mathsf{CL}} A^{\mathsf{ku}} \Leftrightarrow A$, that is $\vdash_{\mathsf{CL}} \neg\neg(P\ B_{\mathsf{ku}}) \Leftrightarrow P\ B$. As we are in classical logic, this is equivalent to derive $\vdash_{\mathsf{CL}} P\ B_{\mathsf{ku}} \Leftrightarrow P\ B$. Using the intensional models of higher-order logic, this does not hold, even when B_{ku} and B are equivalent. More intuitively, the problem here is that the translation may insert double negations inside the argument B of the predicate P, but we cannot replace B_{ku} by B in the absence of extensionality.

4.3.2 Necessary and Sufficient Condition

When trying to adapt the proof of the characterization property from first-order logic to higher-order logic, we identify a necessary and sufficient condition, that we call Kuroda-equivalence.

[2] A complete development of intensional models will be shown in Chapter 6 for a higher-order infinitary logic.

Definition 4.12: Kuroda-Equivalence

For any constant or variable t of type $\tau_1 \to \ldots \to \tau_n \to o$ and any terms $u_1, \ldots, u_n$ of type $\tau_1, \ldots, \tau_n$, we have $\vdash_{\mathsf{CL}} (t\ u_1\ \ldots\ u_n)_{\mathsf{ku}} \Leftrightarrow t\ u_1\ \ldots\ u_n$.

Theorem 4.13

The characterization property holds if and only if the Kuroda-equivalence holds.

Proof. For the direct implication, we apply the characterization property to the formula $t\ u_1\ \ldots\ u_n$ to get $\vdash_{\mathsf{CL}} (t\ u_1\ \ldots\ u_n)^{\mathsf{ku}} \Leftrightarrow t\ u_1\ \ldots\ u_n$, so we derive $\vdash_{\mathsf{CL}} (t\ u_1\ \ldots\ u_n)_{\mathsf{ku}} \Leftrightarrow t\ u_1\ \ldots\ u_n$.

For the reverse implication, we only have to show $\vdash_{\mathsf{CL}} A_{\mathsf{ku}} \Leftrightarrow A$ as we are in classical logic. Let A_β be the β-normal form of A. We know that $A_\beta \equiv_\beta A$, therefore we derive $(A_\beta)_{\mathsf{ku}} \equiv_\beta A_{\mathsf{ku}}$ by Corollary 4.6. We do a proof by cases on A_β. If A_β is a variable or a constant, $(A_\beta)_{\mathsf{ku}}$ corresponds to A_β, therefore we directly have $\vdash_{\mathsf{CL}} (A_\beta)_{\mathsf{ku}} \Leftrightarrow A_\beta$ and we conclude $\vdash_{\mathsf{CL}} A_{\mathsf{ku}} \Leftrightarrow A$. Otherwise, A_β is an application $t\ u_1\ \ldots\ u_n$, where t is a variable or a constant of type $\tau_1 \to \ldots \to \tau_n \to o$ and $u_1, \ldots, u_n$ are terms of type $\tau_1, \ldots, \tau_n$. By the Kuroda-equivalence, we have $\vdash_{\mathsf{CL}} (A_\beta)_{\mathsf{ku}} \Leftrightarrow A_\beta$, hence we conclude $\vdash_{\mathsf{CL}} A_{\mathsf{ku}} \Leftrightarrow A$. □

The counter-example of Section 4.3.1 precisely shows that the Kuroda-equivalence does not necessarily hold, taking $t := P$, $n := 1$ and $u_1 := B$. To have the characterization property, we could impose the Kuroda-equivalence as a principle. However, such a condition is difficult to enforce, as it not only applies to the logical connectives and quantifiers, but also to any variable or constant representing a predicate.

4.3.3 Extensionality Conditions

We now investigate the characterization property in the setting of extensionality. When assuming functional extensionality and propositional extensionality, the formulas A and A^{ku} are equal, and therefore classically equivalent.

Lemma 4.14. *For any term t, we have $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} t_{\mathsf{ku}} = t$.*

Proof. We proceed by induction on the term t.

— Variable and constant: Using Eq-I, we directly derive $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} x_{\mathsf{ku}} = x$ and $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} c_{\mathsf{ku}} = c$ for $c \neq \forall$. We prove $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} \forall_{\mathsf{ku}} = \forall$ with

$$\dfrac{\dfrac{}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} \forall\ q = \forall\ q}\ \text{Eq-I} \qquad \dfrac{\dfrac{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} q\ y \Leftrightarrow \neg\neg(q\ y)}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} q\ y = \neg\neg(q\ y)}\ \text{PropExt}}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} q = (\lambda x.\ \neg\neg(q\ x))}\ \text{FunExt and Conv}}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} \forall\ (\lambda x.\ \neg\neg(q\ x)) = \forall\ q}\ \text{Eq-E and Conv}}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} (\lambda p.\ \forall\ (\lambda x.\ \neg\neg(p\ x))) = \forall}\ \text{FunExt and Conv}$$

as we have $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} q\ y \Leftrightarrow \neg\neg(q\ y)$ in classical logic.

— Application: We have $\vdash_{\mathsf{CL}}^{\mathfrak{efp}} t_{\mathsf{ku}}\ u_{\mathsf{ku}} = t_{\mathsf{ku}}\ u_{\mathsf{ku}}$ using Eq-I. Using successively Eq-E with the induction hypotheses $\vdash_{\mathsf{CL}}^{\mathfrak{efp}} t_{\mathsf{ku}} = t$ and $\vdash_{\mathsf{CL}}^{\mathfrak{efp}} u_{\mathsf{ku}} = u$, we obtain $\vdash_{\mathsf{CL}}^{\mathfrak{efp}} t_{\mathsf{ku}}\ u_{\mathsf{ku}} = t\ u$.

— Abstraction: By induction hypothesis, we have $t_{\mathsf{ku}} = t$. Using FunExt and Conv, we derive $\vdash_{\mathsf{CL}}^{\mathfrak{efp}} (\lambda x.\ t)_{\mathsf{ku}} = \lambda x.\ t$. □

Theorem 4.15: Characterization Property

For any higher-order formula A, we have $\vdash_{\mathsf{CL}}^{\mathfrak{efp}} A^{\mathsf{ku}} \Leftrightarrow A$.

Proof. By Lemma 4.14, $\vdash_{\mathsf{CL}}^{\mathfrak{efp}} A_{\mathsf{ku}} = A$. Using Eq-E on the tautology $A \Leftrightarrow A$, we get $\vdash_{\mathsf{CL}}^{\mathfrak{efp}} A_{\mathsf{ku}} \Leftrightarrow A$. We conclude $\vdash_{\mathsf{CL}}^{\mathfrak{efp}} A^{\mathsf{ku}} \Leftrightarrow A$ as we are in classical logic. □

Note that Lemma 4.14 implies the Kuroda-equivalence, therefore providing an alternative proof of Theorem 4.15 using Theorem 4.13.

4.4 Reverse Translation Property

In this section, we investigate the reverse translation property: if $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} A^{\mathsf{ku}}$ then $\Gamma \vdash_{\mathsf{CL}} A$. While the characterization property does not generally hold in higher-order logic, we show that the reverse translation property—a weaker property—does not hold either.

As in Section 4.2, we state the reverse translation property with Γ_{ku} instead of Γ^{ku}. Again, all the results can be simply adapted to have the version with Γ^{ku}.

4.4.1 Counter-Example

To prove that the reverse translation property does not hold in higher-order logic, we exhibit some Γ and A such that $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} A^{\mathsf{ku}}$ holds and $\Gamma \vdash_{\mathsf{CL}} A$ does not. Consider a constant R of type $o \to o \to o$ and two predicates P and P' of type $\tau \to o$. We take

$$\begin{array}{lcl} \Gamma & := & \forall q. R\ (q\ (\lambda x.\ \neg\neg(P\ x)))\ (q\ (\lambda x.\ \neg\neg(P'\ x))) \\ A & := & R\ (\forall\ (\lambda x.\ P\ x))\ (\forall\ (\lambda x.\ P'\ x)) \end{array}$$

and we therefore have

$$\begin{array}{lcl} \Gamma_{\mathsf{ku}} & \equiv_\beta & \forall q. \neg\neg(R\ (q\ (\lambda x.\ \neg\neg(P\ x)))\ (q\ (\lambda x.\ \neg\neg(P'\ x)))) \\ A^{\mathsf{ku}} & \equiv_\beta & \neg\neg(R\ (\forall\ (\lambda x.\ \neg\neg(P\ x)))\ (\forall\ (\lambda x.\ \neg\neg(P'\ x)))) \end{array}$$

In intuitionistic logic, we derive $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} A^{\mathsf{ku}}$ using All-E with $q := \forall$.

In classical logic, however, we cannot derive $\Gamma \vdash_{\mathsf{CL}} A$. Indeed, we cannot apply All-E with $q := \forall$ anymore: the extra double negations inside the arguments of R cannot be removed without further assumptions. We cannot apply All-E with $\lambda y.\ \forall\ (\lambda x.\ P\ x)$ either, as we would prove the formula $R\ (\forall\ (\lambda x.\ P\ x))\ (\forall\ (\lambda x.\ P\ x))$ instead of $R\ (\forall\ (\lambda x.\ P\ x))\ (\forall\ (\lambda x.\ P'\ x))$. As well as for the characterization property, it is the absence of extensionality that allows us to build a counter-example.

4.4.2 Sufficient Conditions

Since the characterization property implies the reverse translation property, the sufficient conditions investigated in Section 4.3 apply. The reverse translation property holds under the Kuroda-equivalence.

Theorem 4.16

If the Kuroda-equivalence holds, then the reverse translation property holds.

Proof. Suppose $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} A^{\mathsf{ku}}$. We directly have $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{CL}} A^{\mathsf{ku}}$, and we get $\Gamma^{\mathsf{ku}} \vdash_{\mathsf{CL}} A^{\mathsf{ku}}$ using the double-negation elimination. We use the characterization property, that holds under Kuroda-equivalence by Theorem 4.13. □

The reverse translation property also holds under functional extensionality and propositional extensionality.

Theorem 4.17: Reverse Translation Property

Let A be a formula and Γ be a context in higher-order logic.

1. If $\Gamma_{\mathsf{ku}} \vdash_{\mathsf{IL}} A^{\mathsf{ku}}$ then $\Gamma \vdash^{\mathfrak{efp}}_{\mathsf{CL}} A$.
2. For $* \in \{\mathfrak{e}, \mathfrak{ep}\}$, if $\Gamma_{\mathsf{ku}} \vdash^{*}_{\mathsf{IL}} A^{\mathsf{ku}}$ then $\Gamma \vdash^{\mathfrak{efp}}_{\mathsf{CL}} A$.
3. For $* \in \{\mathfrak{ef}, \mathfrak{efp}\}$, if $\Delta^{\mathfrak{ef}}, \Gamma_{\mathsf{ku}} \vdash^{*}_{\mathsf{IL}} A^{\mathsf{ku}}$ then $\Gamma \vdash^{\mathfrak{efp}}_{\mathsf{CL}} A$.

Proof. For the first and second item, we directly have $\Gamma_{\mathsf{ku}} \vdash^{\mathfrak{efp}}_{\mathsf{CL}} A^{\mathsf{ku}}$. We get $\Gamma^{\mathsf{ku}} \vdash^{\mathfrak{efp}}_{\mathsf{CL}} A^{\mathsf{ku}}$ using the double-negation elimination, and then we derive $\Gamma \vdash^{\mathfrak{efp}}_{\mathsf{CL}} A$ using Theorem 4.15. For the third item, we similarly get $\Delta^{\mathfrak{ef}}, \Gamma \vdash^{\mathfrak{efp}}_{\mathsf{CL}} A$, and we conclude using the fact that $\Delta^{\mathfrak{ef}}$ is derivable from functional extensionality in classical logic. □

4.5 Kolmogorov's Translation for Higher-Order Logic

We now extend Kolmogorov's translation to higher-order logic. To do so, we resort to a reformulation of Kolmogorov's translation by Dowek and Werner [DW03]. This reformulation generates the same result as the original translation, but produces it through a two-step process that is very similar to Kuroda's translation. The key feature is the treatment of atomic variables.

4.5.1 Kolmogorov's Translation à la Dowek-Werner

Dowek and Werner [DW03] defined a light version of Kolmogorov's translation:

$$\begin{array}{lcll}
(A \Rightarrow B)_{\mathsf{ko}} & := & \neg\neg A_{\mathsf{ko}} \Rightarrow \neg\neg B_{\mathsf{ko}} & \\
(A \wedge B)_{\mathsf{ko}} & := & \neg\neg A_{\mathsf{ko}} \wedge \neg\neg B_{\mathsf{ko}} & \\
(A \vee B)_{\mathsf{ko}} & := & \neg\neg A_{\mathsf{ko}} \vee \neg\neg B_{\mathsf{ko}} & \\
A_{\mathsf{ko}} & := & A & \text{if } A \text{ atomic or } \bot \\
(\forall x.A)_{\mathsf{ko}} & := & \forall x.\neg\neg A_{\mathsf{ko}} & \\
(\exists x.A)_{\mathsf{ko}} & := & \exists x.\neg\neg A_{\mathsf{ko}} &
\end{array}$$

Kolmogorov's translation can be formulated in terms of the light version, with $A^{\mathsf{ko}} = \neg\neg A_{\mathsf{ko}}$. We refer to this formulation of Kolmogorov's translation as the Kolmogorov-Dowek-Werner translation, abbreviated KDW translation. Both translations generate the same result, but the KDW translation produces it using a two-step process similar to Kuroda's translation. In particular, the KDW translation does *not* insert double negations in front of atomic formulas. This is key to extend Kuroda's translation to higher-order logic so that it preserves β-conversion [BR14]. It follows that the KDW translation can be naturally extended to higher-order logic.

Definition 4.18

The KDW translation of a higher-order formula A is given by $A^{\mathsf{ko}} := \neg\neg A_{\mathsf{ko}}$, where A_{ko} is inductively defined by:

$$\begin{array}{lcll}
x_{\mathsf{ko}} & := & x & \\
c_{\mathsf{ko}} & := & \begin{cases} \lambda p.\ \lambda q.\ c\ (\neg\neg p)\ (\neg\neg q) & \text{if } c \in \{\Rightarrow, \wedge, \vee\} \\ \lambda p.\ c\ (\lambda x.\ \neg\neg(p\ x)) & \text{if } c \in \{\forall, \exists\} \\ c & \text{otherwise} \end{cases} & \\
(\lambda x.\ t)_{\mathsf{ko}} & := & \lambda x.\ t_{\mathsf{ko}} & \\
(t\ u)_{\mathsf{ko}} & := & t_{\mathsf{ko}}\ u_{\mathsf{ko}} &
\end{array}$$

We have the following results regarding substitution and conversion, which are the counterparts of Propositions 4.2 and 4.4 and Corollaries 4.3, 4.5 and 4.6.

Proposition 4.19

Let t and u be terms, and A and B be higher-order formulas.

1. $(t[x \leftarrow u])_{\mathsf{ko}} = t_{\mathsf{ko}}[x \leftarrow u_{\mathsf{ko}}]$.
2. $(A[x \leftarrow u])^{\mathsf{ko}} = A^{\mathsf{ko}}[x \leftarrow u_{\mathsf{ko}}]$.
3. If $t \hookrightarrow_{\beta(\eta)} u$ then $t_{\mathsf{ko}} \hookrightarrow_{\beta(\eta)} u_{\mathsf{ko}}$.
4. If $t \equiv_{\beta(\eta)} u$ then $t_{\mathsf{ko}} \equiv_{\beta(\eta)} u_{\mathsf{ko}}$.
5. If $A \equiv_{\beta(\eta)} B$ then $A_{\mathsf{ko}} \equiv_{\beta(\eta)} B_{\mathsf{ko}}$ and $A^{\mathsf{ko}} \equiv_{\beta(\eta)} B^{\mathsf{ko}}$.

Proof. As their counterparts for Kuroda's translation (see Section 4.1). The first item is proved by induction on t. The second item is a corollary of the first item. The third item is

proved by induction on $t \hookrightarrow_{\beta(\eta)} u$ and relies on the first item. The fourth item is a corollary of the third item. The fifth item is a corollary of the fourth item. □

4.5.2 Soundness Property

There are two technicalities when proving the soundness property for the KDW translation. First, we assume a weak form of functional extensionality in the presence of FunExt, as for Kuroda's translation. Second, we use Proposition 3.7(4) in the presence of PropExt, hence only translating classical logic into intuitionistic logic.

Theorem 4.20: Soundness

Let A be a formula and Γ be a context in higher-order logic.

1. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} A^{\mathsf{ko}}$.
2. If $\Gamma \vdash^{\mathfrak{c}}_{\mathsf{CL}} A$ then $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{c}}_{\mathsf{ML}} A^{\mathsf{ko}}$.
3. If $\Gamma \vdash^{\mathfrak{cf}}_{\mathsf{CL}} A$ then $\Delta^{\mathfrak{cf}}, \Gamma_{\mathsf{ko}} \vdash^{\mathfrak{cf}}_{\mathsf{ML}} A^{\mathsf{ko}}$.
4. If $\Gamma \vdash^{\mathfrak{cp}}_{\mathsf{CL}} A$ then $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{cp}}_{\mathsf{IL}} A^{\mathsf{ko}}$.
5. If $\Gamma \vdash^{\mathfrak{cfp}}_{\mathsf{CL}} A$ then $\Delta^{\mathfrak{cf}}, \Gamma_{\mathsf{ko}} \vdash^{\mathfrak{cfp}}_{\mathsf{IL}} A^{\mathsf{ko}}$.

Proof. The first item is proved by induction on the derivation. We only show the cases that are specific to higher-order logic. The remaining cases are the same as for first-order logic.

— All-I: By induction hypothesis, we have $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\neg(P_{\mathsf{ko}}\ x)$. Using All-I and Conv, we derive $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \forall x.\neg\neg(P_{\mathsf{ko}}\ x)$. We conclude using Proposition 3.7(1).

— All-E: By induction hypothesis, we have $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\neg\forall x.\neg\neg(P_{\mathsf{ko}}\ x)$. Using Proposition 3.36(1), we get $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \forall x.\neg\neg\neg\neg(P_{\mathsf{ko}}\ x)$. Using All-E, we obtain $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\neg\neg\neg(P_{\mathsf{ko}}\ t_{\mathsf{ko}})$. We get $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\neg(P_{\mathsf{ko}}\ t_{\mathsf{ko}})$ using Proposition 3.7(2).

— Ex-I: By induction hypothesis, we have $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\neg(P_{\mathsf{ko}}\ t_{\mathsf{ko}})$. Using Conv and Ex-I, we derive $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \exists x.\neg\neg(P_{\mathsf{ko}}\ x)$. We get $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\neg\exists x.\neg\neg(P_{\mathsf{ko}}\ x)$ using Proposition 3.7(1).

— Ex-E: By induction hypotheses, we have $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\neg\exists x.\neg\neg(P_{\mathsf{ko}}\ x)$ and $\Gamma_{\mathsf{ko}}, P_{\mathsf{ko}}\ x \vdash_{\mathsf{ML}} \neg\neg A_{\mathsf{ko}}$. We want to prove $\Gamma_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\neg A_{\mathsf{ko}}$. We use Not-I and then Not-E with $\neg\forall x.\neg\neg\neg(P_{\mathsf{ko}}\ x)$. The first subgoal $\Gamma_{\mathsf{ko}}, \neg A_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\forall x.\neg\neg\neg(P_{\mathsf{ko}}\ x)$ derives from Proposition 3.36(2), weakening and the first induction hypothesis. The second subgoal $\Gamma_{\mathsf{ko}}, \neg A_{\mathsf{ko}} \vdash_{\mathsf{ML}} \forall x.\neg\neg\neg(P_{\mathsf{ko}}\ x)$ has the following proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Hypothesis}}{\Gamma_{\mathsf{ko}}, \neg A_{\mathsf{ko}}, P_{\mathsf{ko}}\ x \vdash_{\mathsf{ML}} \neg\neg A_{\mathsf{ko}}}\ \text{Weakening} \qquad \dfrac{}{\Gamma_{\mathsf{ko}}, \neg A_{\mathsf{ko}}, P_{\mathsf{ko}}\ x \vdash_{\mathsf{ML}} \neg A_{\mathsf{ko}}}\ \text{Ax}}{\Gamma_{\mathsf{ko}}, \neg A_{\mathsf{ko}}, P_{\mathsf{ko}}\ x \vdash_{\mathsf{ML}} \bot}\ \text{Not-E}}{\Gamma_{\mathsf{ko}}, \neg A_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg(P_{\mathsf{ko}}\ x)}\ \text{Not-I}}{\Gamma_{\mathsf{ko}}, \neg A_{\mathsf{ko}} \vdash_{\mathsf{ML}} \neg\neg\neg(P_{\mathsf{ko}}\ x)}\ \text{Proposition 3.7(2)}}{\Gamma_{\mathsf{ko}}, \neg A_{\mathsf{ko}} \vdash_{\mathsf{ML}} \forall x.\neg\neg\neg(P_{\mathsf{ko}}\ x)}\ \text{All-I}$$

— Conv derives from the induction hypothesis and Proposition 4.19.

For the remaining items, we reuse the cases of the first item with the additional cases:

— Eq-I derives from Eq-I and Proposition 3.7(1).

— Eq-E: We have $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{e}}_{\mathsf{ML}} \neg\neg(P_{\mathsf{ko}}\ u_{\mathsf{ko}})$ and $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{e}}_{\mathsf{ML}} \neg\neg(u_{\mathsf{ko}} = v_{\mathsf{ko}})$ by induction hypotheses. We directly have $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{e}}_{\mathsf{ML}} P_{\mathsf{ko}}\ u_{\mathsf{ko}} \Rightarrow (u_{\mathsf{ko}} = v_{\mathsf{ko}}) \Rightarrow P_{\mathsf{ko}}\ v_{\mathsf{ko}}$ using Eq-E. Therefore we get $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{e}}_{\mathsf{ML}} \neg\neg(P_{\mathsf{ko}}\ u_{\mathsf{ko}}) \Rightarrow \neg\neg(u_{\mathsf{ko}} = v_{\mathsf{ko}}) \Rightarrow \neg\neg(P_{\mathsf{ko}}\ v_{\mathsf{ko}})$ using Proposition 3.7(1) and Proposition 3.7(3). We conclude $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{e}}_{\mathsf{ML}} \neg\neg(P_{\mathsf{ko}}\ v_{\mathsf{ko}})$.

— FunExt: We directly use $\Delta^{\mathfrak{ef}}$ on the induction hypothesis.

— PropExt: We have $\vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(\neg\neg(\neg\neg A_{\mathsf{ko}} \Rightarrow \neg\neg B_{\mathsf{ko}}) \wedge \neg\neg(\neg\neg B_{\mathsf{ko}} \Rightarrow \neg\neg A_{\mathsf{ko}}))$ by induction hypothesis. Using Proposition 3.7(5), Proposition 3.7(3) and Proposition 3.7(2), we get $\vdash^{\mathfrak{ep}}_{\mathsf{IL}} ((\neg\neg A_{\mathsf{ko}} \Rightarrow \neg\neg B_{\mathsf{ko}}) \wedge (\neg\neg B_{\mathsf{ko}} \Rightarrow \neg\neg A_{\mathsf{ko}}))$. We derive $\vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(A_{\mathsf{ko}} \Leftrightarrow B_{\mathsf{ko}})$ using Proposition 3.7(4) and Proposition 3.7(5). Using PropExt, we also have $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} (A_{\mathsf{ko}} \Leftrightarrow B_{\mathsf{ko}}) \Rightarrow (A_{\mathsf{ko}} = B_{\mathsf{ko}})$. Using by Proposition 3.7(1) and Proposition 3.7(3), we get $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(A_{\mathsf{ko}} \Leftrightarrow B_{\mathsf{ko}}) \Rightarrow \neg\neg(A_{\mathsf{ko}} = B_{\mathsf{ko}})$. We conclude $\Gamma_{\mathsf{ko}} \vdash^{\mathfrak{ep}}_{\mathsf{IL}} \neg\neg(A_{\mathsf{ko}} = B_{\mathsf{ko}})$. □

Compared to Kuroda's translation, the KDW translation translates classical logic (without propositional extensionality) into minimal logic, and not only into intuitionistic logic. As noted by Ferreira and Oliva [FO12b], Kuroda's translation can be modified to translate classical logic into minimal logic, by additionally inserting double negations on the right-hand side of implications.

4.5.3 Characterization Property

As for Kuroda's translation, the characterization property for the KDW translation does not hold due to the intensional nature of higher-order logic, but does hold when we assume functional extensionality and propositional extensionality.

Theorem 4.21: Characterization

For any higher-order formula A, we have $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} A^{\mathsf{ko}} \Leftrightarrow A$.

As in Theorem 4.15, the characterization property relies on the following lemma.

Lemma 4.22. *For any term t, we have $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} t_{\mathsf{ko}} = t$.*

Proof. We adapt the proof of Lemma 4.14. The quantifier $\exists$ is treated as $\forall$. For the connectives $c \in \{\Rightarrow, \wedge, \vee\}$, we prove $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} c_{\mathsf{ko}} = c$ with

$$\dfrac{\dfrac{\dfrac{}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} c\ p\ q = c\ p\ q}\ \text{Eq-I} \quad \dfrac{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} p \Leftrightarrow \neg\neg p}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} p = \neg\neg p}\ \text{PropExt}}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} c\ (\neg\neg p)\ q = c\ p\ q}\ \text{Eq-E and Conv} \quad \dfrac{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} q \Leftrightarrow \neg\neg q}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} q = \neg\neg q}\ \text{PropExt}}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} (c\ (\neg\neg p)\ (\neg\neg q)) = c\ p\ q}\ \text{Eq-E and Conv}}{\vdash^{\mathfrak{efp}}_{\mathsf{CL}} (\lambda p.\ \lambda q.\ c\ (\neg\neg p)\ (\neg\neg q)) = c}\ \text{FunExt} \times 2 \text{ and Conv}$$

as we have $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} p \Leftrightarrow \neg\neg p$ and $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} q \Leftrightarrow \neg\neg q$ in classical logic. □

4.6 Conclusion

Brown and Rizkallah [BR14] extended Kuroda's translation to higher-order logic, showing that the soundness property holds, except in the presence of functional extensionality. We have discussed several conditions under which the soundness property holds in the presence of functional extensionality, including the principle of double-negation shift.

Brown and Rizkallah neither examined the characterization property nor the reverse translation property. We have shown that both properties do not hold in higher-order logic, although they are straightforward in first-order logic. We have identified conditions under which such properties hold. In particular, it is sufficient to assume functional extensionality and propositional extensionality.

Brown and Rizkallah [BR14] showed that Kolmogorov's translation and the Gödel-Gentzen translation does not *naturally* extend to higher-order logic, unlike Kuroda's translation. We have shown that Kolmogorov's translation can nevertheless be extended to higher-order logic using a reformulation by Dowek and Werner [DW03]. The Kolmogorov-Dowek-Werner translation allows us to translate higher-order classical logic directly into minimal logic, while Kuroda's translation stops at intuitionistic logic. We present in Table 4.1 an overview of the soundness property for Kuroda's translation and the Kolmogorov-Dowek-Werner translation.

Both Kuroda's translation and the Kolmogorov-Dowek-Werner translation use a two-step process, resulting in a special treatment of variables which is key to preserve β-conversion. More generally, it is possible to define many two-step translations, that insert at least the double negations of Kuroda's translation and at most those of the Kolmogorov-Dowek-Werner translation. In particular, all the double negations inserted by the Kolmogorov-Dowek-Werner translation are not necessary to translate classical logic into minimal logic: it suffices to modify Kuroda's translation so that it also inserts double negations on the right-hand side of implications [FO12b].

Contrary to Kolmogorov's and Kuroda's translations, the Gödel-Gentzen translation inserts double negations in a way that cannot be reformulated using a two-step process. Future work would be to investigate whether it is possible or not to find an elaborated way to extend the Gödel-Gentzen translation to higher-order logic.

We have seen that the role of both functional extensionality and propositional extensionality is predominant when extending double-negation translations to higher-order logic. One has to be careful when using both principles in an intuitionistic context: the Diaconescu-Goodman-Myhill theorem [Dia75, GM78] states that, together with the axiom of choice, they entail the principle of excluded middle.

Translation	Soundness property						Theorems
	Without equality	With $\mathfrak{e}$	With $\mathfrak{ef}$		With $\mathfrak{ep}$	With $\mathfrak{efp}$	
Kuroda	$\mathrm{CL} \to \mathrm{IL}$	$\mathrm{CL} \to \mathrm{IL}$	$\mathrm{CL} \to \Delta^{\mathfrak{ef}}, \mathrm{IL}$		$\mathrm{CL} \to \mathrm{IL}$	$\mathrm{CL} \to \Delta^{\mathfrak{ef}}, \mathrm{IL}$	4.7 and 4.9
Kolmogorov-Dowek-Werner	$\mathrm{CL} \to \mathrm{ML}$	$\mathrm{CL} \to \mathrm{ML}$	$\mathrm{CL} \to \Delta^{\mathfrak{ef}}, \mathrm{ML}$		$\mathrm{CL} \to \mathrm{IL}$	$\mathrm{CL} \to \Delta^{\mathfrak{ef}}, \mathrm{IL}$	4.20

Table 4.1: Summary table of the soundness property for the different translations of Chapter 4.

Chapter 5

Monad Translations

Summary

Several translations generalize double-negation translations by inserting monad operators instead of double negations. They eliminate particular axioms, for instance the principle of excluded middle or the principle of explosion, and therefore can be used to embed classical logic into intuitionistic logic or intuitionistic logic into minimal logic. Such translations have been defined for first-order logic.

In this chapter, we define a translation, parameterized by monad operators, for higher-order logic. We apply this translation to embed higher-order classical (respectively intuitionistic) logic into higher-order intuitionistic (respectively minimal) logic. By adapting Friedman's trick, we show that coherent formulas correspond to a constructive fragment of higher-order classical logic, meaning that we can transform classical proofs into intuitionistic proofs without modifying the proven statements. We extend Barr's theorem to higher-order coherent formulas.

This chapter is adapted from [Tra25].

So far, we have only focused on the double-negation translations, as they are the most well-known and intuitive translations that embed classical logic into intuitionistic logic or minimal logic. But other translations exist. The Prawitz-Malmnäs translation [PM68] embeds intuitionistic logic into minimal logic. It is inductively defined like Kolmogorov's translation, but it applies the operator $TA := A \vee \bot$ instead of $\neg\neg A$. Moreover, the double-negation translations can be parameterized by a formula R, using Friedman's trick [Fri78]. Instead of applying $\neg\neg A := (A \Rightarrow \bot) \Rightarrow \bot$, the parameterized translations apply the operator $TA := (A \Rightarrow R) \Rightarrow R$ in formulas and translate classical logic into intuitionistic logic.

So as to generalize even more the double-negation translations, it is possible to resort to *monad operators* instead of double negations. Monad operators are unary connectives T that satisfy $A \Rightarrow TA$ and $(A \Rightarrow TB) \Rightarrow (TA \Rightarrow TB)$ for any formulas A and B. They are called lax modalities in [Acz01], nuclei in [vdB19] and strong monads in [EO12], and they originate from propositional lax logic [FM97], topology and category theory [Mac71]. Several monad translations have been defined [Acz01, EO12, vdB19], and they generalize the Prawitz-Malmnäs translation, the double-negation translations

and the parameterized double-negation translations. These translations eliminate particular axioms, depending on the chosen monad operator, for example the principle of excluded middle or the principle of explosion. Such generic translations are therefore relevant to embed classical logic into intuitionistic logic or intuitionistic logic into minimal logic. As the double-negation translations, their generalizations with monad operators have only been defined for *first-order* logic.

Contributions. In this chapter, we define a Kuroda-style monad translation for higher-order logic. Depending on the choice of the monad operator, this translation either embeds higher-order classical logic into higher-order intuitionistic logic or higher-order intuitionistic logic into higher-order minimal logic.

Like the extension of double-negation translations to higher-order logic studied in Chapter 4, the characterization property holds under functional extensionality and propositional extensionality. Moreover, we underscore conditions under which the soundness property holds in the presence of functional extensionality and propositional extensionality.

We refine the translation for factorizable monad operators, that are monad operators that satisfy $(TA \Rightarrow TB) \Rightarrow T(A \Rightarrow B)$. The factorizable monad translation directly abstracts Kuroda's translation, and it introduces less operators in the formulas than the standard monad translation. It can be used to embed higher-order classical logic into higher-order intuitionistic logic.

When considering higher-order coherent formulas, the monad translation can be used to show that classical provability entails intuitionistic provability and that intuitionistic provability entails minimal provability. In particular, we are able to constructivize any classical proof of a statement that only involves higher-order coherent formulas. We extend Barr's theorem to higher-order coherent formulas.

Outline. We recall in Section 5.1 the necessary preliminaries about monad operators. The monad translation for higher-order logic is defined in Section 5.2. In Section 5.3, we show that this translation satisfies the soundness and characterization properties, and we examine its behavior in the presence of equality. We illustrate how the monad translation can be used with different monad operators to define embeddings between higher-order logics in Section 5.4, and we study the particular case of higher-order coherent formulas in Section 5.5. In Section 5.6, we refine the translation for factorizable monad operators.

5.1 Monad Operators

We say that T is a unary connective when it is a closed term of type $o \to o$.

Definition 5.1: Monad Operator

A unary connective T is a monad operator of L when the judgments (unit) and (bind) hold for any formulas A and B.

$$\vdash_{\mathsf{L}} A \Rightarrow TA \qquad \text{(unit)}$$
$$\vdash_{\mathsf{L}} (A \Rightarrow TB) \Rightarrow (TA \Rightarrow TB) \qquad \text{(bind)}$$

We use the terminology "monad operator" to avoid any confusion with monads in the sense of category theory. Monad operators are called lax modalities in [Acz01] and nuclei in [vdB19]. This definition of monad operators is equivalent to the axiomatization of strong monads in [EO12]. Following the Curry-Howard correspondence, monad operators coincide with the notion of monad in programming languages [Mog91], for instance in Haskell.

Example 5.2. *There are many monad operators of minimal logic:*

- $TA := A \vee \bot$,
- $T_R A := (A \Rightarrow R) \Rightarrow R$, *with parameter* R, *which corresponds to the continuation monad,*
- $TA := \neg\neg A$, *which is a special case of the previous one with* $R := \bot$,
- $T_R A := (A \Rightarrow R) \Rightarrow A$, *with parameter* R, *which corresponds to the Peirce monad [EO12].*

We recall some properties about monad operators that will be used in the rest of the chapter. They generalize Propositions 3.7 and 3.36 to any monad operator.

Proposition 5.3

Let T be a monad operator of L, A and B be formulas, and P be a predicate.

1. $\vdash_{\mathsf{L}} TTA \Leftrightarrow TA$
2. $\vdash_{\mathsf{L}} (A \Rightarrow B) \Rightarrow (TA \Rightarrow TB)$
3. $\vdash_{\mathsf{L}} T(A \wedge B) \Leftrightarrow (TA \wedge TB)$
4. $\vdash_{\mathsf{L}} T(A \Rightarrow B) \Rightarrow (TA \Rightarrow TB)$
5. $\vdash_{\mathsf{L}} T(A \Rightarrow TB) \Leftrightarrow (TA \Rightarrow TB)$
6. $\vdash_{\mathsf{L}} T(A \vee B) \Leftrightarrow T(TA \vee TB)$
7. $\vdash_{\mathsf{L}} T(\forall x.T(P\ x)) \Leftrightarrow \forall x.T(P\ x)$
8. $\vdash_{\mathsf{L}} T(\exists x.T(P\ x)) \Leftrightarrow T(\exists\ P)$

Proof. Let us detail each case.

- **Item 1 ($\Rightarrow$)**: Using (bind) we have $\vdash_{\mathsf{L}} (TA \Rightarrow TA) \Rightarrow (TTA \Rightarrow TA)$. We therefore derive $\vdash_{\mathsf{L}} TTA \Rightarrow TA$.
- **Item 1 ($\Leftarrow$)**: We use (unit) on TA.

— **Item 2**: Using (unit) on B we have $\vdash_{\mathsf{L}} (A \Rightarrow B) \Rightarrow (A \Rightarrow TB)$, and using (bind) $\vdash_{\mathsf{L}} (A \Rightarrow TB) \Rightarrow (TA \Rightarrow TB)$. Thus $\vdash_{\mathsf{L}} (A \Rightarrow B) \Rightarrow (TA \Rightarrow TB)$.

— **Item 3 (⇒)**: We have $\vdash_{\mathsf{L}} (A \wedge B) \Rightarrow A$ and $\vdash_{\mathsf{L}} (A \wedge B) \Rightarrow B$. Using Item 2 we easily derive $\vdash_{\mathsf{L}} T(A \wedge B) \Rightarrow (TA \wedge TB)$.

— **Item 3 (⇐)**: Using (bind), we directly have $\vdash_{\mathsf{L}} (A \Rightarrow T(A \wedge B)) \Rightarrow TA \Rightarrow T(A \wedge B)$ and $\vdash_{\mathsf{L}} (B \Rightarrow T(A \wedge B)) \Rightarrow TB \Rightarrow T(A \wedge B)$. Using these two facts and (unit) on $A \wedge B$, we easily derive $\vdash_{\mathsf{L}} TA \Rightarrow TB \Rightarrow T(A \wedge B)$. We conclude $\vdash_{\mathsf{L}} (TA \wedge TB) \Rightarrow T(A \wedge B)$.

— **Item 4**: We know that $\vdash_{\mathsf{L}} ((A \Rightarrow B) \wedge A) \Rightarrow B$. We derive $\vdash_{\mathsf{L}} T((A \Rightarrow B) \wedge A) \Rightarrow TB$ using Item 2. By Item 3, we get $\vdash_{\mathsf{L}} (T(A \Rightarrow B) \wedge TA) \Rightarrow TB$, therefore we conclude $\vdash_{\mathsf{L}} T(A \Rightarrow B) \Rightarrow (TA \Rightarrow TB)$.

— **Item 5 (⇒)**: We have $\vdash_{\mathsf{L}} T(A \Rightarrow TB) \Rightarrow (TA \Rightarrow TTB)$ using Item 4. We conclude $\vdash_{\mathsf{L}} T(A \Rightarrow TB) \Rightarrow (TA \Rightarrow TB)$ using Item 1.

— **Item 5 (⇐)**: We get $\vdash_{\mathsf{L}} (TA \Rightarrow TB) \Rightarrow (A \Rightarrow TB)$ and $\vdash_{\mathsf{L}} (A \Rightarrow TB) \Rightarrow T(A \Rightarrow TB)$ using (unit). It follows $\vdash_{\mathsf{L}} (TA \Rightarrow TB) \Rightarrow T(A \Rightarrow TB)$.

— **Item 6 (⇒)**: Using (unit) on A and Or-IL, we have $\vdash_{\mathsf{L}} A \Rightarrow (TA \vee TB)$. Similarly, we have $\vdash_{\mathsf{L}} B \Rightarrow (TA \vee TB)$. Therefore we have $\vdash_{\mathsf{L}} (A \vee B) \Rightarrow (TA \vee TB)$. We derive $\vdash_{\mathsf{L}} T(A \vee B) \Rightarrow T(TA \vee TB)$ by Item 2.

— **Item 6 (⇐)**: Using (bind), we get $\vdash_{\mathsf{L}} (A \Rightarrow T(A \vee B)) \Rightarrow (TA \Rightarrow T(A \vee B))$ and $\vdash_{\mathsf{L}} (B \Rightarrow T(A \vee B)) \Rightarrow (TB \Rightarrow T(A \vee B))$. Using (unit), we derive $\vdash_{\mathsf{L}} TA \Rightarrow T(A \vee B)$ and $\vdash_{\mathsf{L}} TB \Rightarrow T(A \vee B)$. It follows $\vdash_{\mathsf{L}} (TA \vee TB) \Rightarrow T(A \vee B)$. We conclude using (bind).

— **Item 7 (⇒)**: We have $\vdash_{\mathsf{L}} (\forall x.T(P\ x)) \Rightarrow T(P\ y)$ for a fresh variable y. Using (bind), we get $\vdash_{\mathsf{L}} T(\forall x.T(P\ x)) \Rightarrow T(P\ y)$. We derive $T(\forall x.T(P\ x)) \vdash_{\mathsf{L}} T(P\ y)$ and therefore $T(\forall x.T(P\ x)) \vdash_{\mathsf{L}} \forall x.T(P\ x)$. We conclude $\vdash_{\mathsf{L}} T(\forall x.T(P\ x)) \Rightarrow \forall x.T(P\ x)$.

— **Item 7 (⇐)**: We use (unit) on $\forall x.T(P\ x)$.

— **Item 8 (⇒)**: By Imp-I and Ex-I, we have $\vdash_{\mathsf{L}} (P\ x) \Rightarrow \exists\ P$. We get $\vdash_{\mathsf{L}} T(P\ x) \Rightarrow T(\exists\ P)$ using Item 2. Using weakening and Imp-E, we derive $\exists x.T(P\ x), T(P\ x) \vdash_{\mathsf{L}} T(\exists\ P)$. Using Ex-E, we get $\exists x.T(P\ x) \vdash_{\mathsf{L}} T(\exists\ P)$. Using Imp-I and (bind), we conclude $\vdash_{\mathsf{L}} T(\exists x.T(P\ x)) \Rightarrow T(\exists\ P)$.

— **Item 8 (⇐)**: For any variable x, we have $\vdash_{\mathsf{L}} (P\ x) \Rightarrow T(P\ x)$ using (unit). Therefore $\vdash_{\mathsf{L}} (\exists\ P) \Rightarrow \exists x.T(P\ x)$. Using Item 2, we conclude $\vdash_{\mathsf{L}} T(\exists\ P) \Rightarrow T(\exists x.T(P\ x))$. □

5.2 Monad Translation for Higher-Order Logic

Several translations [Acz01, EO12, vdB19] generalize the standard double-negation translations [Kol25, Gö33, Gen36, Kur51] to monad operators. In particular, van den Berg [vdB19] defined a Kuroda-style monad translation for first-order logic. It corresponds to Kuroda's translation, in which we insert monad operators instead of double negations. It also introduces additional monad operators on the right-hand side

of implications, and such a modification is necessary to generalize Kuroda's translation to *any* monad operator. We will see in Section 5.6 that this constraint can be removed for *factorizable* monad operators.

We define here a Kuroda-style monad translation for higher-order logic, taking advantage of the ideas developed in Chapter 4 and in [vdB19].

Definition 5.4: Monad Translation for Higher-Order Logic

Let A be a formula in higher-order logic and T be a monad operator. Its monad translation is $A^T := TA_T$, where $t \mapsto t_T$ is inductively defined by:

$$\begin{array}{lcl} x_T & := & x \\ c_T & := & \begin{cases} \lambda p.\ \forall x. T(p\ x) & \text{if } c = \forall \\ \lambda p.\ \lambda q.\ p \Rightarrow Tq & \text{if } c = \Rightarrow \\ c & \text{otherwise} \end{cases} \\ (\lambda x.\ t)_T & := & \lambda x.\ t_T \\ (t\ u)_T & := & t_T\ u_T \end{array}$$

In first-order logic, we have $(A[z \leftarrow w])^T = A^T[z \leftarrow w]$. As we are in higher-order logic, we have $(A[z \leftarrow w])^T = A^T[z \leftarrow w_T]$.

Proposition 5.5

For any term t, we have $(t[z \leftarrow w])_T = t_T[z \leftarrow w_T]$.

Proof. By induction on the term t. The proof is the same as the proof of Proposition 4.2, with subscript T instead of ku. □

Corollary 5.6. *For any higher-order formula A, we have $(A[z \leftarrow w])^T = A^T[z \leftarrow w_T]$.*

Proof. We have $(A[z \leftarrow w])_T = A_T[z \leftarrow w_T]$ by Proposition 5.5. Therefore we derive $(A[z \leftarrow w])^T = T(A[z \leftarrow w])_T = T(A_T[z \leftarrow w_T]) = (TA_T)[z \leftarrow w_T] = A^T[z \leftarrow w_T]$. □

In higher-order logic, terms are considered modulo $\beta(\eta)$-convertibility. It follows that the translation must preserve $\beta(\eta)$-convertibility for it to preserve provability.

Proposition 5.7

For any terms t and u, if $t \hookrightarrow_{\beta(\eta)} u$ then $t_T \hookrightarrow_{\beta(\eta)} u_T$.

Proof. By induction on $t \hookrightarrow_{\beta(\eta)} u$.

— We have $((\lambda x.\ t)\ u)_T = (\lambda x.\ t_T)\ u_T \hookrightarrow_\beta t_T[x \leftarrow u_T]$. Using Proposition 5.5, we get $((\lambda x.\ t)\ u)_T \hookrightarrow_\beta (t[x \leftarrow u])_T$.

— Suppose that x is not free in t. We have $(\lambda x.\ t\ x)_T = \lambda x.\ t_T\ x \hookrightarrow_{\beta\eta} t_T$.

— The other cases directly follow from the induction hypotheses. □

Corollary 5.8. *For any terms t and u, if $t \equiv_{\beta(\eta)} u$ then $t_T \equiv_{\beta(\eta)} u_T$.*

Proof. Directly follows from Proposition 5.7, as $\equiv_{\beta(\eta)}$ is the reflexive, symmetric and transitive closure of $\hookrightarrow_{\beta(\eta)}$. □

Corollary 5.9. *For any higher-order formulas A and B, if $A \equiv_{\beta(\eta)} B$ then $A_T \equiv_{\beta(\eta)} B_T$ and $A^T \equiv_{\beta(\eta)} B^T$.*

Proof. We have $A_T \equiv_{\beta(\eta)} B_T$ by Corollary 5.8, and therefore $TA_T \equiv_{\beta(\eta)} TB_T$. □

5.3 Monad Embedding

The monad translation generalizes double-negation translations with monad operators instead of double negations. Double-negation translations eliminate the use of the principle of excluded middle. Likewise, the monad translation eliminates the use of a particular formula [Acz01, EO12], called T-elimination.

$$TA \Rightarrow A \qquad (T\text{-elimination})$$

More formally, we extend the logic L with the inference rule T-Elim, and we write $\Gamma \vdash_{\mathsf{L+T}} A$ when $\Gamma \vdash A$ is derivable using the rules of L and T-Elim.

$$\frac{}{\Gamma \vdash TA \Rightarrow A}\ \text{T-Elim}$$

The monad translation eliminates any use of the T-Elim inference rule—this is the soundness property. The characterization property states that any formula and its translation are equivalent assuming T-Elim.

When extending the double-negation translation to higher-order logic in Chapter 4, the characterization property does not trivially hold, but does hold when we consider functional extensionality and propositional extensionality. The same reasoning applies when extending the monad translation to higher-order logic.

Theorem 5.10: Monad Embedding

Let T be a monad operator of L such that we have $\vdash_{\mathsf{L}} (TA)_T \Leftrightarrow TA_T$ for any formula A. Let A be a formula and Γ a context in higher-order logic.

1. If $\Gamma \vdash_{\mathsf{L+T}} A$ then $\Gamma_T \vdash_{\mathsf{L}} A^T$.
2. $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} A^T \Leftrightarrow A$.

Remark 5.11. *As for Kuroda's translation, the version of the soundness property with Γ^T instead of Γ_T is provable as well.*

Remark 5.12. *The condition $\vdash_{\mathsf{L}} (TA)_T \Leftrightarrow TA_T$ is useful when proving the T-Elim case of the soundness property.*

Proof of Theorem 5.10(1). We proceed by induction on the derivation. Most of the cases are direct applications of Proposition 5.3. We only show the most interesting cases.

— Or-E: By induction hypotheses, we have $\Gamma_T \vdash_{\mathsf{L}} T(A_T \vee B_T)$ and $\Gamma_T, A_T \vdash_{\mathsf{L}} TC_T$ and $\Gamma_T, B_T \vdash_{\mathsf{L}} TC_T$. To prove $\Gamma_T \vdash_{\mathsf{L}} TC_T$, it suffices to have $\Gamma_T \vdash_{\mathsf{L}} T(A_T \vee B_T) \Rightarrow TC_T$.

Using the second and third hypotheses, we derive $\Gamma_T \vdash_{\mathsf{L}} (A_T \vee B_T) \Rightarrow TC_T$. Using (bind) we get $\Gamma_T \vdash_{\mathsf{L}} T(A_T \vee B_T) \Rightarrow TC_T$, and we conclude the case.

— All-I: By induction hypothesis, we have $\Gamma_T \vdash_{\mathsf{L}} T(P_T\ x)$. We derive $\Gamma_T \vdash_{\mathsf{L}} \forall x.T(P_T\ x)$, that is $\Gamma_T \vdash_{\mathsf{L}} (\forall\ P)_T$. We conclude using (unit).

— All-E: By induction hypothesis, we have $\Gamma_T \vdash_{\mathsf{L}} T(\forall x.T(P_T\ x))$. Using Proposition 5.3(7), we derive $\Gamma_T \vdash_{\mathsf{L}} \forall x.T(P_T\ x)$. We get $\Gamma_T \vdash_{\mathsf{L}} T(P_T\ t_T)$ using All-E.

— Ex-I: By induction hypothesis, we have $\Gamma_T \vdash_{\mathsf{L}} T(P_T\ t_T)$. We get $\Gamma_T \vdash_{\mathsf{L}} \exists x.T(P_T\ x)$ using Ex-I. We conclude using (unit) and Proposition 5.3(8).

— Ex-E: By induction hypotheses, we have $\Gamma_T \vdash_{\mathsf{L}} T(\exists\ P_T)$ and $\Gamma_T, P_T\ x \vdash_{\mathsf{L}} TA_T$. To conclude the proof, it suffices to have $\Gamma_T \vdash_{\mathsf{L}} T(\exists\ P_T) \Rightarrow TA_T$.

We have $\Gamma_T, \exists\ P_T \vdash_{\mathsf{L}} \exists\ P_T$ (by Ax) and $\Gamma_T, \exists\ P_T, P_T\ x \vdash_{\mathsf{L}} TA_T$ (by weakening on the second hypothesis). Using Ex-E, we derive $\Gamma_T, \exists\ P_T \vdash_{\mathsf{L}} TA_T$ and therefore $\Gamma_T \vdash_{\mathsf{L}} (\exists\ P_T) \Rightarrow TA_T$. Using (bind), we obtain $\Gamma_T \vdash_{\mathsf{L}} T(\exists\ P_T) \Rightarrow TA_T$, and we conclude the case.

— Bot-E: By induction hypothesis, we have $\Gamma_T \vdash_{\mathsf{IL}} T\bot$. As we are in intuitionistic logic, we have $\Gamma_T \vdash_{\mathsf{IL}} \bot \Rightarrow A_T$. We deduce $\Gamma_T \vdash_{\mathsf{IL}} T\bot \Rightarrow TA_T$ using Proposition 5.3(2). We conclude $\Gamma_T \vdash_{\mathsf{IL}} TA_T$.

— PEM: As we are in classical logic, we have $\Gamma_T \vdash_{\mathsf{CL}} A_T \vee (A_T \Rightarrow \bot)$. Using (unit), we derive $\Gamma_T \vdash_{\mathsf{CL}} A_T \vee (A_T \Rightarrow T\bot)$, and then we conclude $\Gamma_T \vdash_{\mathsf{CL}} T(A_T \vee (A_T \Rightarrow T\bot))$.

— T-Elim: We have $(TA \Rightarrow A)_T = ((TA)_T \Rightarrow TA_T)$. We know $\vdash_{\mathsf{L}} (TA)_T \Leftrightarrow TA_T$, so we derive $\Gamma_T \vdash_{\mathsf{L}} (TA \Rightarrow A)_T$. We conclude $\Gamma_T \vdash_{\mathsf{L}} T(TA \Rightarrow A)_T$ using (unit).

— Conv derives from the induction hypothesis and Corollary 5.9. □

Before proving the second item of Theorem 5.10, we show the following lemma, stating that any term and its translation are equal, assuming T-Elim, functional extensionality and propositional extensionality.

Lemma 5.13. *For any term t, we have $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} t_T = t$.*

Proof. By induction on t.

— We directly have $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} x_T = x$ and $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} c_T = c$ for $c \notin \{\forall, \Rightarrow\}$. For $c \in \{\forall, \Rightarrow\}$, we derive $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} c_T = c$ from PropExt, FunExt, T-Elim and (unit), adapting the proofs of Lemmas 4.14 and 4.22.

— We have $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} (t\ u)_T = t\ u$ using the induction hypotheses and Eq-E.

— We derive $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} (\lambda x.\ t)_T = \lambda x.\ t$ using the induction hypothesis and FunExt. □

Proof of Theorem 5.10(2). We derive $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} A_T = A$ using Lemma 5.13, and therefore $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} A_T \Leftrightarrow A$. We conclude $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} A^T \Leftrightarrow A$ using (unit) and T-Elim. □

We proved that, without the symbols and inference rules for equality, the monad translation satisfies the soundness property. As for Kuroda's translation, the proof of the soundness property does not directly extend when we consider equality and extensionality. We consider axioms stating weak forms of functional extensionality and propositional extensionality with monad operators.

$$\forall f \forall g.\ (\forall x. T(f\ x = g\ x)) \Rightarrow T(f = g) \qquad (\Delta_T^{\mathfrak{ef}})$$

$$\forall A \forall B.\ (TA \Leftrightarrow TB) \Rightarrow T(A = B) \qquad (\Delta_T^{\mathfrak{ep}})$$

We write $\Delta_T^{\mathfrak{efp}}$ for the context $\Delta_T^{\mathfrak{ef}}, \Delta_T^{\mathfrak{ep}}$. When proving the soundness property in the presence of equality, the first formula $\Delta_T^{\mathfrak{ef}}$ is useful for the FunExt case and the second formula $\Delta_T^{\mathfrak{ep}}$ is useful for the PropExt case.

Remark 5.14. *$\Delta_T^{\mathfrak{ef}}$ directly generalizes $\Delta^{\mathfrak{ef}}$. Instead of $\Delta_T^{\mathfrak{ef}}$, we could also consider the generalizations of* $\mathsf{DNE}^{\mathfrak{e}}$*:*

$$\forall x \forall y.\ T(x = y) \Rightarrow x = y$$

However, it entails the T-elimination in the presence of propositional extensionality. This can be proved by adapting the arguments of Remark 4.11.

Theorem 5.15: Soundness Property for Equality

Let T be a monad operator of L such that we have $\vdash_{\mathsf{L}} (TA)_T \Leftrightarrow TA_T$ for any formula A. Let A be a formula and Γ be a context in higher-order logic.

1. If $\Gamma \vdash^{\mathfrak{e}}_{\mathsf{L}+\mathsf{T}} A$ then $\Gamma_T \vdash^{\mathfrak{e}}_{\mathsf{L}} A^T$.
2. For $* \in \{\mathfrak{ef}, \mathfrak{ep}, \mathfrak{efp}\}$, if $\Gamma \vdash^{*}_{\mathsf{L}+\mathsf{T}} A$ then $\Delta^*_T, \Gamma_T \vdash^{*}_{\mathsf{L}} A^T$.

Proof. For the first item, we complete the proof of Theorem 5.10 with the following cases:

— Eq-I: We directly use Eq-I on the induction hypothesis and then (unit).

— Eq-E: By induction hypotheses, we have $\Gamma_T \vdash^{\mathfrak{e}}_{\mathsf{L}} T(P_T\ u_T)$ and $\Gamma_T \vdash^{\mathfrak{e}}_{\mathsf{L}} T(u_T = v_T)$. Using Eq-E, we derive $\vdash^{\mathfrak{e}}_{\mathsf{L}} (P_T\ u_T) \Rightarrow (u_T = v_T) \Rightarrow (P_T\ v_T)$. Using Proposition 5.3(2) and Proposition 5.3(4), we get $\vdash^{\mathfrak{e}}_{\mathsf{L}} T(P_T\ u_T) \Rightarrow T(u_T = v_T) \Rightarrow T(P_T\ v_T)$. We conclude $\Gamma_T \vdash^{\mathfrak{e}}_{\mathsf{L}} T(P_T\ v_T)$.

For the second item, most cases are those of the first item that are reused with Δ^*_T in the context. The remaining cases are:

— FunExt: We directly use $\Delta_T^{\mathfrak{ef}}$ on the induction hypothesis.

— PropExt: By induction hypothesis, we have $\Delta^*_T, \Gamma_T \vdash^{*}_{\mathsf{L}} T((A_T \Rightarrow TB_T) \wedge (B_T \Rightarrow TA_T))$. Using Proposition 5.3(3) and Proposition 5.3(5), we get $\Delta^*_T, \Gamma_T \vdash^{*}_{\mathsf{L}} TA_T \Leftrightarrow TB_T$. We conclude using $\Delta_T^{\mathfrak{ep}}$. □

When extending Kuroda's translation to higher-order logic in Chapter 4, we only considered the formula $\Delta^{\mathfrak{ef}}$. The generalization to any monad operator requires us to additionally consider the formula $\Delta_T^{\mathfrak{ep}}$. In Section 5.6, we will see that this additional assumption can be dropped when considering factorizable monad operators.

5.4 Embeddings between Logics

Such monad translation can be directly applied with particular monad operators to embed classical logic into intuitionistic logic or intuitionistic logic into minimal logic.

There are many classical laws, such as the principle of excluded middle, the double-negation elimination, Peirce's law, or Clavius's law. All these laws are equivalent in intuitionistic logic (when quantifying on A and B).

$$\neg\neg A \Rightarrow A \qquad \text{(double-negation elimination)}$$
$$(\neg A \Rightarrow A) \Rightarrow A \qquad \text{(Clavius's law)}$$
$$((A \Rightarrow B) \Rightarrow A) \Rightarrow A \qquad \text{(Peirce's law)}$$

The T-elimination principle corresponds to the double-negation elimination when we choose $TA := (A \Rightarrow \bot) \Rightarrow \bot$ and to Clavius's law when we choose $TA := (A \Rightarrow \bot) \Rightarrow A$. For these monad operators, the T-Elim inference rule is therefore equivalent to PEM, and eliminating any use of T-Elim in the derivations means transforming classical provability into intuitionistic provability.

Theorem 5.16: Embeddings of Classical Logic into Intuitionistic Logic

Let A be a formula and Γ be a context in higher-order logic.

1. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_T \vdash_{\mathsf{IL}} A^T$ with $TA := (A \Rightarrow \bot) \Rightarrow \bot$.
2. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_T \vdash_{\mathsf{IL}} A^T$ with $TA := (A \Rightarrow \bot) \Rightarrow A$.

Moreover, for each of the above, we have $\vdash^{\mathfrak{efp}}_{\mathsf{CL}} A^T \Leftrightarrow A$.

Proof. We apply Theorem 5.10 with $\mathsf{L} := \mathsf{IL}$. T-Elim is equivalent to PEM, so $\mathsf{IL} + \mathsf{T}$ corresponds to CL. We only have to check the hypothesis $\vdash_{\mathsf{IL}} (TA)_T \Leftrightarrow TA_T$.

For the first item, $(TA)_T = (A_T \Rightarrow ((\bot \Rightarrow \bot) \Rightarrow \bot)) \Rightarrow ((\bot \Rightarrow \bot) \Rightarrow \bot)$ and $TA_T = (A_T \Rightarrow \bot) \Rightarrow \bot$. We easily have $\vdash_{\mathsf{IL}} (TA)_T \Leftrightarrow TA_T$.

For the second item, $(TA)_T = (A_T \Rightarrow ((\bot \Rightarrow \bot) \Rightarrow \bot)) \Rightarrow ((A_T \Rightarrow \bot) \Rightarrow A_T)$ and $TA_T = (A_T \Rightarrow \bot) \Rightarrow A_T$. We easily have $\vdash_{\mathsf{IL}} (TA)_T \Leftrightarrow TA_T$. □

Remark 5.17. *As remarked by Escardó and Oliva [EO12], the monad $T_R A := (A \Rightarrow R) \Rightarrow A$ can be used with $R \neq \bot$ to eliminate instances of Peirce's law. However, this only works for particular formulas, as the assumption $\vdash_{\mathsf{IL}} (T_R A)_{T_R} \Leftrightarrow T_R A_{T_R}$ of Theorem 5.10 does not necessarily hold for any R.*

When $TA := A \vee \bot$, the T-elimination principle is equivalent to the principle of explosion, and the T-Elim inference rule is therefore equivalent to the Bot-E inference rule. In that case, eliminating any use of T-Elim in the derivations means transforming intuitionistic provability into minimal provability.

Theorem 5.18: Embedding of Intuitionistic Logic into Minimal Logic

Let A be a formula and Γ be a context in higher-order logic. If $\Gamma \vdash_{\mathsf{IL}} A$ then $\Gamma_T \vdash_{\mathsf{ML}} A^T$ with $TA := A \vee \bot$. Moreover, we have $\vdash^{\mathfrak{efp}}_{\mathsf{IL}} A^T \Leftrightarrow A$.

Proof. We apply Theorem 5.10 with L := ML. T-Elim is equivalent to Bot-E, so ML + T corresponds to IL. The hypothesis $\vdash_{\mathsf{IL}} (TA)_T \Leftrightarrow TA_T$, is directly satisfied, as we have $(TA)_T = A_T \vee \bot$ and $TA_T = A_T \vee \bot$. □

This embedding generalizes the Prawitz-Malmnäs translation [PM68] to higher-order logic, but also refines it as we do not apply $TA := A \vee \bot$ on every subformula.

5.5 The Fragment of Higher-Order Coherent Formulas

Coherent formulas are formulas that can only be built using conjunctions, disjunctions and existential quantifiers. Because monad operators are only inserted after universal quantifiers and implications, it follows that $A_T = A$ for any coherent formula A. We lift to higher-order logic several results for coherent formulas that rely on this observation.

5.5.1 Constructivization of the Fragment

A first trick [CdJV16, vdB19] allows us to derive minimal provability from intuitionistic provability when the context is empty and when the formula is coherent.

Theorem 5.19

Let A be higher-order coherent formula. If $\vdash_{\mathsf{IL}} A$ then $\vdash_{\mathsf{ML}} A$.

Proof. Suppose $\vdash_{\mathsf{IL}} A$. We apply Theorem 5.10 with $TA := A \vee \bot$, and we derive $\vdash_{\mathsf{ML}} A_T \vee \bot$, that is $\vdash_{\mathsf{ML}} A \vee \bot$ since A is coherent. We know that minimal logic has the disjunction property, and that we cannot prove $\bot$ (as the context is empty). Hence, we conclude $\vdash_{\mathsf{ML}} A$. □

A second trick [vdB19] allows us to derive intuitionistic provability from classical provability when the formulas involved are coherent.

Theorem 5.20

Let Γ, A be a sequence of higher-order coherent formulas. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma \vdash_{\mathsf{IL}} A$.

To prove this result, we adapt to higher-order logic Friedman's translation [Fri78]. As an intermediate lemma, we prove that, for higher-order logic, Friedman's translation transforms classical proofs into intuitionistic proofs. Note that we cannot directly apply Theorem 5.10, as the T-elimination principle does not correspond to a classical law when $TA := (A \Rightarrow R) \Rightarrow R$.

Lemma 5.21. *Let $TA := (A \Rightarrow R) \Rightarrow R$ with parameter R. Let A be a formula and Γ be a context in higher-order logic. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_T \vdash_{\mathsf{IL}} A^T$.*

Proof. We proceed by induction on the derivation. All the cases are similar to the cases of Theorem 5.10, except that we do not need to consider the T-Elim case anymore, and that we change the PEM case. We prove $\Gamma_T \vdash_{\mathsf{IL}} T(A \vee \neg A)_T$ using

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_T, B \Rightarrow R, A_T, \bot \Rightarrow R \vdash_{\mathsf{IL}} A_T}\,\text{Ax}}{\Gamma_T, B \Rightarrow R, A_T, \bot \Rightarrow R \vdash_{\mathsf{IL}} B}\,\text{Or-IL}}{\Gamma_T, B \Rightarrow R, A_T, \bot \Rightarrow R \vdash_{\mathsf{IL}} R}\,\text{Imp-E with } B \Rightarrow R}{\Gamma_T, B \Rightarrow R \vdash_{\mathsf{IL}} A_T \Rightarrow ((\bot \Rightarrow R) \Rightarrow R)}\,\text{Imp-I} \times 2}{\Gamma_T, B \Rightarrow R \vdash_{\mathsf{IL}} B}\,\text{Or-IR} \qquad \dfrac{\Gamma_T, B \Rightarrow R \vdash_{\mathsf{IL}} B}{\Gamma_T, B \Rightarrow R \vdash_{\mathsf{IL}} R}\,\text{Imp-E with } B \Rightarrow R}{\Gamma_T \vdash_{\mathsf{IL}} \big(\big(A_T \vee \big(A_T \Rightarrow ((\bot \Rightarrow R) \Rightarrow R)\big)\big) \Rightarrow R\big) \Rightarrow R}\,\text{Imp-I}$$

where B is an abbreviation for the formula $A_T \vee \big(A_T \Rightarrow ((\bot \Rightarrow R) \Rightarrow R)\big)$. □

Proof of Theorem 5.20. Suppose $\Gamma \vdash_{\mathsf{CL}} A$. Without loss of generality, we assume that Γ and A have no free variables, as we can replace them by fresh constants. We apply Lemma 5.21, and we get $\Gamma_T \vdash_{\mathsf{IL}} (A_T \Rightarrow R) \Rightarrow R$, that is $\Gamma \vdash_{\mathsf{IL}} (A \Rightarrow R) \Rightarrow R$ since Γ and A are coherent. Choosing $R := A$, we derive $\Gamma \vdash_{\mathsf{IL}} A$. □

In other words, coherent formulas correspond to a constructive fragment of higher-order classical logic. Constructive fragments of classical logic have been studied for propositional logic [Gli28, Gö33] and first-order logic [PH15]. This is the first time, to our knowledge, that such results are proved for higher-order logic.

5.5.2 Barr's Theorem for Higher-Order Coherent Theories

Coherent implications are formulas of the form $\forall \vec{x}.A$ or $\forall \vec{x}.(A \Rightarrow B)$, where A and B are coherent formulas and $\vec{x}$ is a finite sequence of variables (possibly empty). A theory $\mathbb{T}$ is coherent when all its axioms are coherent implications. Barr's theorem [Bar74] states[1] that, in first-order logic, any coherent implication that is classically provable in a coherent theory is also intuitionistically provable in the same theory.

Remark 5.22. *Note that the theorem refers to first-order logic and not just coherent logic, because the principle of excluded middle cannot be expressed by a coherent formula or a coherent implication.*

Barr's theorem admits different proofs [Pal02, Neg03]. We extend Barr's theorem to higher-order coherent theories, by adapting Palmgren's proof [Pal02], that relies on Friedman's translation.

We have seen that coherent formulas are unchanged by the translation. Coherent implications imply their translation. In the rest of this section, we only consider coherent implications of the form $\forall \vec{x}.(A \Rightarrow B)$, as the case $\forall \vec{x}.A$ is easier.

Lemma 5.23. *Let A and B be higher-order coherent formulas, and $\vec{x}$ be a finite sequence of variables (possibly empty). We have $\vdash_{\mathsf{IL}} (\forall \vec{x}.(A \Rightarrow B)) \Rightarrow (\forall \vec{x}.(A \Rightarrow B))_T$.*

Proof. Since $(A \Rightarrow B)_T = A \Rightarrow TB$, we directly have $\vdash_{\mathsf{IL}} (A \Rightarrow B) \Rightarrow (A \Rightarrow B)_T$ using (unit) on B. If $\vec{x}$ is empty, this achieves the case. Suppose that $\vec{x}$ is the sequence $x_1, \ldots, x_n$. We derive $\vdash_{\mathsf{IL}} (A \Rightarrow B) \Rightarrow T(A \Rightarrow B)_T$ using (unit) on $(A \Rightarrow B)_T$. Using All-I, we derive

[1] Originating from topos theory, Barr's theorem is stated for geometric formulas, an extension of coherent formulas with infinite disjunctions. Barr's theorem also claims to remove the axiom of choice. Such an assertion must be considered with care: as shown by Rathjen [Rat16], adding the axiom of choice *externally* preserves conservativity, but adding it *internally* does not.

$\vdash_{\mathsf{IL}} \forall x_n.((A \Rightarrow B) \Rightarrow T(A \Rightarrow B)_T)$. Then we use the distributivity of $\forall$ over $\Rightarrow$ to derive $\vdash_{\mathsf{IL}} (\forall x_n.(A \Rightarrow B)) \Rightarrow (\forall x_n.T(A \Rightarrow B)_T)$, that is $\vdash_{\mathsf{IL}} (\forall x_n.(A \Rightarrow B)) \Rightarrow (\forall x_n.(A \Rightarrow B))_T$. We iterate this procedure to get $\vdash_{\mathsf{IL}} (\forall \vec{x}.(A \Rightarrow B)) \Rightarrow (\forall \vec{x}.(A \Rightarrow B))_T$. □

Theorem 5.24

Let $\mathbb{T}$ be a higher-order coherent theory, A and B be higher-order coherent formulas, and $\vec{x}$ be a finite sequence of variables (possibly empty). If $\vdash_{\mathsf{CL}} \forall \vec{x}.(A \Rightarrow B)$ in $\mathbb{T}$ then $\vdash_{\mathsf{IL}} \forall \vec{x}.(A \Rightarrow B)$ in $\mathbb{T}$.

Proof. Suppose that $\vdash_{\mathsf{CL}} \forall \vec{x}.(A \Rightarrow B)$ in $\mathbb{T}$. Then there exists some coherent implications Θ of $\mathbb{T}$ such that $\Theta \vdash_{\mathsf{CL}} \forall \vec{x}.(A \Rightarrow B)$. Using All-E, we instantiate the bound variables $\vec{x}$ by fresh variables also named $\vec{x}$ (without loss of generality) and obtain $\Theta \vdash_{\mathsf{CL}} A \Rightarrow B$. Using Weak and Imp-E, we get $\Theta, A \vdash_{\mathsf{CL}} B$. Applying Lemma 5.21, we derive $\Theta_T, A_T \vdash_{\mathsf{IL}} B^T$, that is $\Theta_T, A \vdash_{\mathsf{IL}} (B \Rightarrow R) \Rightarrow R$. By Lemma 5.23, we derive $\Theta, A \vdash_{\mathsf{IL}} (B \Rightarrow R) \Rightarrow R$. Choosing $R := B$, we derive $\Theta, A \vdash_{\mathsf{IL}} B$. Using Imp-I and All-I, we get $\Theta \vdash_{\mathsf{IL}} \forall \vec{x}.(A \Rightarrow B)$. We conclude that $\vdash_{\mathsf{IL}} \forall \vec{x}.(A \Rightarrow B)$ in $\mathbb{T}$. □

5.6 Refinement for Factorizable Monad Operators

So far, we have generalized Kuroda's translation to *any* monad operator, at the expense of the insertion of additional monad operators on the right-hand side of implications. In this section, we refine the translation for *factorizable* monad operators.

Definition 5.25: Factorizable Monad Operator

A monad operator T of L is factorizable when it satisfies the (factorization) judgment for any propositions A and B.

$$\vdash_{\mathsf{L}} (TA \Rightarrow TB) \Rightarrow T(A \Rightarrow B) \qquad \text{(factorization)}$$

For instance, (factorization) holds for the monad $T_R A := (A \Rightarrow R) \Rightarrow A$ in minimal logic and for the monad $TA := (A \Rightarrow \bot) \Rightarrow \bot$ in intuitionistic logic.

5.6.1 Factorizable Monad Translation

For factorizable monad operators, we do not need to insert a monad operator in the implication case of the translation [vdB19]. In that sense, the factorizable monad translation is a direct abstraction of Kuroda's translation, in which double negations have been replaced by factorizable monad operators.

Definition 5.26: Factorizable Monad Translation for Higher-Order Logic

Let A be a formula in higher-order logic and T be a factorizable monad operator. Its factorizable monad translation is $A^{\widetilde{T}} := TA_{\widetilde{T}}$, where $t \mapsto t_{\widetilde{T}}$ is inductively defined by:

$$\begin{array}{lcll} x_{\widetilde{T}} & := & x & \\ c_{\widetilde{T}} & := & \begin{cases} \lambda p.\, \forall x. T(p\ x) & \text{if } c = \forall \\ c & \text{otherwise} \end{cases} & \\ (\lambda x.\ t)_{\widetilde{T}} & := & \lambda x.\ t_{\widetilde{T}} & \\ (t\ u)_{\widetilde{T}} & := & t_{\widetilde{T}}\ u_{\widetilde{T}} & \end{array}$$

The translations $t \mapsto t_{\widetilde{T}}$ and $A \mapsto A^{\widetilde{T}}$ satisfy the same properties as $t \mapsto t_T$ and $A \mapsto A^T$ concerning substitution and convertibility.

Proposition 5.27

Let t and u be terms, and A and B be higher-order formulas.

1. $(t[z \leftarrow w])_{\widetilde{T}} = t_{\widetilde{T}}[z \leftarrow w_{\widetilde{T}}]$.
2. $(A[z \leftarrow w])^{\widetilde{T}} = A^{\widetilde{T}}[z \leftarrow w_{\widetilde{T}}]$.
3. If $t \hookrightarrow_{\beta(\eta)} u$ then $t_{\widetilde{T}} \hookrightarrow_{\beta(\eta)} u_{\widetilde{T}}$.
4. If $t \equiv_{\beta(\eta)} u$ then $t_{\widetilde{T}} \equiv_{\beta(\eta)} u_{\widetilde{T}}$.
5. If $A \equiv_{\beta(\eta)} B$ then $A_{\widetilde{T}} \equiv_{\beta(\eta)} B_{\widetilde{T}}$ and $A^{\widetilde{T}} \equiv_{\beta(\eta)} B^{\widetilde{T}}$.

Proof. We adapt the proofs of Proposition 5.5, Corollary 5.6, Proposition 5.7, Corollary 5.8 and Corollary 5.9. □

5.6.2 Factorizable Monad Embedding

The factorizable monad translation satisfies the soundness and characterization properties. It only applies to factorizable monad operators, but it simplifies the monad translation, as it introduces fewer monad operators while satisfying the same result.

Theorem 5.28: Factorizable Monad Embedding

Let T be a factorizable monad operator of L such that $\vdash_{\mathsf{L}} (TA)_{\widetilde{T}} \Leftrightarrow TA_{\widetilde{T}}$ for any formula A. Let A be a formula and Γ be a context in higher-order logic.

1. If $\Gamma \vdash_{\mathsf{L}+\mathsf{T}} A$ then $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} A^{\widetilde{T}}$.
2. $\vdash^{\mathfrak{cfp}}_{\mathsf{L}+\mathsf{T}} A^{\widetilde{T}} \Leftrightarrow A$.

Proof. We prove the first item by induction on the derivation. Most of the cases are the same as for Theorem 5.10. Conv derives from the induction hypothesis and Proposition 5.27. We only show the proof of the cases that differ.

— Imp-I: By induction hypothesis, we have $\Gamma_{\widetilde{T}}, A_{\widetilde{T}} \vdash_{\mathsf{L}} TB_{\widetilde{T}}$. We get $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} A_{\widetilde{T}} \Rightarrow TB_{\widetilde{T}}$, and we derive $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} TA_{\widetilde{T}} \Rightarrow TB_{\widetilde{T}}$ using (bind). We conclude $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} T(A_{\widetilde{T}} \Rightarrow B_{\widetilde{T}})$ using (factorization).

— Imp-E: By induction hypotheses, we have $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} T(A_{\widetilde{T}} \Rightarrow B_{\widetilde{T}})$ and $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} TA_{\widetilde{T}}$. We get $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} TA_{\widetilde{T}} \Rightarrow TB_{\widetilde{T}}$ using Proposition 5.3(4). We conclude $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} TB_{\widetilde{T}}$ using Imp-E.

— T-Elim: We directly have $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} TTA_{\widetilde{T}} \Rightarrow TA_{\widetilde{T}}$ using Proposition 5.3(1). We derive $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} T(TA_{\widetilde{T}} \Rightarrow A_{\widetilde{T}})$ using (factorization). Since we have $\vdash_{\mathsf{L}} (TA)_{\widetilde{T}} \Leftrightarrow TA_{\widetilde{T}}$, we conclude $\Gamma_{\widetilde{T}} \vdash_{\mathsf{L}} T(TA \Rightarrow A)_{\widetilde{T}}$.

For the second item, we show $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} t_{\widetilde{T}} = t$ by adapting the proof of Lemma 5.13. It follows that $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} A_{\widetilde{T}} = A$, and therefore $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} A_{\widetilde{T}} \Leftrightarrow A$. We conclude $\vdash^{\mathfrak{efp}}_{\mathsf{L+T}} A^{\widetilde{T}} \Leftrightarrow A$ using (unit) and T-Elim. □

In the presence of equality, the soundness property for the factorizable monad translation is simplified compared to the one for the monad translation. We can drop the formula $\Delta^{\mathfrak{ep}}_T$ of Theorem 5.15, as the PropExt case can now be proved without any additional hypothesis.

Theorem 5.29: Soundness Property for Equality

Let T be a factorizable monad operator of L such that $\vdash_{\mathsf{L}} (TA)_{\widetilde{T}} \Leftrightarrow TA_{\widetilde{T}}$ for any formula A. Let Γ, A be higher-order formulas.

1. For $* \in \{\mathfrak{e}, \mathfrak{ep}\}$, if $\Gamma \vdash^{*}_{\mathsf{L+T}} A$ then $\Gamma_{\widetilde{T}} \vdash^{*}_{\mathsf{L}} A^{\widetilde{T}}$.
2. For $* \in \{\mathfrak{ef}, \mathfrak{efp}\}$, if $\Gamma \vdash^{*}_{\mathsf{L+T}} A$ then $\Delta^{\mathfrak{ef}}_T, \Gamma_{\widetilde{T}} \vdash^{*}_{\mathsf{L}} A^{\widetilde{T}}$.

Proof. For the first item, we complete the proof of Theorem 5.28 with the following cases:

— Eq-I and Eq-E: The proofs are the same as for Theorem 5.15.

— PropExt: By induction hypothesis, we have $\Gamma_{\widetilde{T}} \vdash^{*}_{\mathsf{L}} T((A_{\widetilde{T}} \Rightarrow B_{\widetilde{T}}) \wedge (B_{\widetilde{T}} \Rightarrow A_{\widetilde{T}}))$. We derive $\Gamma_{\widetilde{T}} \vdash^{*}_{\mathsf{L}} T(A_{\widetilde{T}} \Rightarrow B_{\widetilde{T}})$ and $\Gamma_{\widetilde{T}} \vdash^{*}_{\mathsf{L}} T(B_{\widetilde{T}} \Rightarrow A_{\widetilde{T}})$ using Proposition 5.3(3), And-EL and And-ER. We have $\vdash^{*}_{\mathsf{L}} (A_{\widetilde{T}} \Rightarrow B_{\widetilde{T}}) \Rightarrow (B_{\widetilde{T}} \Rightarrow A_{\widetilde{T}}) \Rightarrow (A_{\widetilde{T}} = B_{\widetilde{T}})$ using PropExt, and we get $\vdash^{*}_{\mathsf{L}} T(A_{\widetilde{T}} \Rightarrow B_{\widetilde{T}}) \Rightarrow T(B_{\widetilde{T}} \Rightarrow A_{\widetilde{T}}) \Rightarrow T(A_{\widetilde{T}} = B_{\widetilde{T}})$ using Proposition 5.3(2) and Proposition 5.3(4). We conclude $\Gamma_{\widetilde{T}} \vdash^{*}_{\mathsf{L}} T(A_{\widetilde{T}} = B_{\widetilde{T}})$.

For the second item, we reuse most cases of the first item using $\Delta^{\mathfrak{ef}}_T$ in the context. The case FunExt corresponds to the one of Theorem 5.15, with $\Delta^{\mathfrak{ef}}_T$ instead of Δ^{*}_T. □

Unlike the monad translation of Section 5.2, the factorizable monad translation allows us to fully recover the specific behavior of the extension of Kuroda's double-negation translation to higher-order logic. Specifically, monad operators are not inserted on the right-hand side of implications, and the extra assumption $\Delta^{\mathfrak{ep}}_T$ is not needed for the soundness property in the presence of propositional equality.

5.6.3 Additional Embeddings between Logics

The embeddings of classical logic into intuitionistic logic of Theorem 5.16 are also valid with the factorizable monad translation. These embeddings introduce fewer monad operators than the monad translation.

Theorem 5.30: Embeddings of Classical Logic into Intuitionistic Logic

Let A be a formula and Γ be a context in higher-order logic.

1. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_{\widetilde{T}} \vdash_{\mathsf{IL}} A^{\widetilde{T}}$ with $TA := (A \Rightarrow \bot) \Rightarrow \bot$.
2. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_{\widetilde{T}} \vdash_{\mathsf{IL}} A^{\widetilde{T}}$ with $TA := (A \Rightarrow \bot) \Rightarrow A$.

Moreover, for each of the above, we have $\vdash^{\mathsf{cfp}}_{\mathsf{CL}} A^{\widetilde{T}} \Leftrightarrow A$.

Proof. The proof is the one of Theorem 5.16, but using Theorem 5.28 instead of Theorem 5.10. □

Remark 5.31. *Kuroda's translation embeds classical logic into intuitionistic logic, but not into minimal logic. Indeed, it requires the* (factorization) *property for the double-negation monad operator $TA := (A \Rightarrow \bot) \Rightarrow \bot$, and such a property is satisfied in intuitionistic logic, but not in minimal logic. As noted by Ferreira and Oliva [FO12b], the modified Kuroda's translation, in which double negations are also inserted on the right-hand side of implications, can also be shown to embed classical logic into minimal logic. This is because the modified translation does not require the* (factorization) *property.*

5.7 Conclusion

We have defined two monad embeddings for higher-order logic: the monad translation and the factorizable monad translation. The former broadens the scope of Kuroda's translation to any monad operator, but it additionally introduces monad operators on the right-hand side of implications as a trade-off. The latter directly generalizes Kuroda's translation to factorizable monad operators. For both translations, the soundness property holds. As we are in higher-order logic, the characterization property holds when we assume both functional extensionality and propositional extensionality. We have given conditions so that the soundness property holds in the presence of functional extensionality and propositional extensionality.

These results generalize to monad operators the approach taken for extending Kuroda's translation to higher-order logic seen in Chapter 4, and they extend to higher-order logic the approach taken for generalizing the double-negation translations to monad operators [Acz01, EO12, vdB19]. The monad translation and the factorizable monad translation entail various embeddings—depending on the monad operator and on the translation—of higher-order classical logic into higher-order intuitionistic logic, and of higher-order intuitionistic logic into higher-order minimal logic. We present in Table 5.1 an overview of the soundness property for the different translations of Chapters 4 and 5.

For higher-order coherent formulas, we have shown that classical provability entails intuitionistic provability, and that intuitionistic provability entails minimal provability. Coherent formulas therefore correspond to a constructive fragment of higher-order classical logic: for this fragment, we have provided an algorithm that transforms any classical proof into a minimal proof, without modifying the proven statement.

The original Barr's theorem is stated for geometric formulas, an extension of coherent formulas with infinite disjunctions. Barr's theorem admits proof-theoretical demonstrations for the first-order coherent case [Pal02, Neg03] and the first-order geometric case [Rat16, Neg21]. In this chapter, we gave a proof of the higher-order coherent case, by extending Palmgren's idea [Pal02]. In the next chapter, we propose to extend Barr's theorem to the higher-order geometric case. To do so, we define a higher-order infinitary logic.

Translation	Soundness property					Theorems
	Without equality	With $\mathfrak{e}$	With $\mathfrak{ef}$	With $\mathfrak{ep}$	With $\mathfrak{efp}$	
Kuroda	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}, \mathsf{IL}$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}, \mathsf{IL}$	4.7, 4.9
Kolmogorov-Dowek-Werner	$\mathsf{CL} \to \mathsf{ML}$	$\mathsf{CL} \to \mathsf{ML}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}, \mathsf{ML}$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}, \mathsf{IL}$	4.20
Monad	$\mathsf{L} + \mathsf{T} \to \mathsf{L}$	$\mathsf{L} + \mathsf{T} \to \mathsf{L}$	$\mathsf{L} + \mathsf{T} \to \Delta^{\mathfrak{ef}}_T, \mathsf{L}$	$\mathsf{L} + \mathsf{T} \to \Delta^{\mathfrak{ep}}_T, \mathsf{L}$	$\mathsf{L} + \mathsf{T} \to \Delta^{\mathfrak{ef}}_T, \Delta^{\mathfrak{ep}}_T, \mathsf{L}$	5.10, 5.15
with $TA := \neg\neg A$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}_T, \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ep}}_T, \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}_T, \Delta^{\mathfrak{ep}}_T, \mathsf{IL}$	5.15, 5.16
with $TA := (A \Rightarrow \bot) \Rightarrow A$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}_T, \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ep}}_T, \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}_T, \Delta^{\mathfrak{ep}}_T, \mathsf{IL}$	5.15, 5.16
with $TA := A \vee \bot$	$\mathsf{IL} \to \mathsf{ML}$	$\mathsf{IL} \to \mathsf{ML}$	$\mathsf{IL} \to \Delta^{\mathfrak{ef}}_T, \mathsf{ML}$	$\mathsf{IL} \to \Delta^{\mathfrak{ep}}_T, \mathsf{ML}$	$\mathsf{IL} \to \Delta^{\mathfrak{ef}}_T, \Delta^{\mathfrak{ep}}_T, \mathsf{ML}$	5.15, 5.18
Factorizable monad	$\mathsf{L} + \mathsf{T} \to \mathsf{L}$	$\mathsf{L} + \mathsf{T} \to \mathsf{L}$	$\mathsf{L} + \mathsf{T} \to \Delta^{\mathfrak{ef}}_T, \mathsf{L}$	$\mathsf{L} + \mathsf{T} \to \mathsf{L}$	$\mathsf{L} + \mathsf{T} \to \Delta^{\mathfrak{ef}}_T, \mathsf{L}$	5.28, 5.29
with $TA := \neg\neg A$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}_T, \mathsf{IL}$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}_T, \mathsf{IL}$	5.29, 5.30
with $TA := (A \Rightarrow \bot) \Rightarrow A$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}_T, \mathsf{IL}$	$\mathsf{CL} \to \mathsf{IL}$	$\mathsf{CL} \to \Delta^{\mathfrak{ef}}_T, \mathsf{IL}$	5.29, 5.30

Table 5.1: Summary table of the soundness property for the different translations of Chapters 4 and 5.

Chapter 6

Investigations on Higher-Order Infinitary Logic

Summary

Higher-order logic and infinitary logic are two extensions of first-order logic that allow greater expressivity. Both features have not been investigated together yet. In this chapter, we define a higher-order infinitary logic, based on an extension of simple type theory. The resulting logic features higher-order quantifiers, infinite conjunctions and infinite disjunctions. We establish results at both the syntactic level and the semantic level. We introduce a sound notion of model, and we show a strong version of completeness that entails the cut-elimination theorem for natural deduction. Moreover, we extend Barr's theorem to the higher-order geometric case.

This chapter is adapted from [THA26].

Infinitary logic is an extension of first-order logic that allows infinite long statements, or proofs with an infinite number of branches. This is typically achieved by featuring infinite sequences of conjunctions, disjunctions or quantifiers. Such an expressive power is useful when axiomatizing various mathematical concepts, but also when formalizing problems from graph theory or epistemic logic [Rat16, Neg21]. Infinitary logic was first coined by Zermelo [Tay10] for studying the foundations of set theory. It was formally established by Karp [Kar64] and its properties were investigated in [Bar69, Kei71]. Among those results is the cut-elimination theorem for infinitary logics [LE65, Tai68, Fef68, Tak75, Neg21].

Geometric formulas are particular infinitary formulas, built using finite conjunctions, infinite disjunctions and existential quantifiers. This is sufficient to formalize numerous mathematical structures, like groups, rings and fields [Rat16, LM24]. Geometric formulas are an extension of coherent formulas, that we investigated in Section 5.5. In particular, we proved Barr's theorem for the higher-order coherent case, while the original Barr's theorem is stated for the first-order geometric case.

Both infinitary and higher-order features have not been investigated together yet.

Contributions. In this chapter, our aim is to define and study a higher-order infinitary logic, and to extend Barr's theorem to higher-order geometric formulas.

First, we extend simple type theory with a type, a constructor, and destructors for families of propositions indexed by an arbitrary countable set. We show that, in this setting, the β-reduction neither satisfies strong normalization nor confluence, but every well-typed term has a head normal form.

Second, we define a higher-order infinitary logic, along with cut-free natural deduction inference rules. We define a sound notion of model. We prove a completeness theorem with respect to the cut-free inference rules, allowing us to derive the cut-elimination theorem. Our results apply for both the intuitionistic variant and the classical variant of this logic.

Third, we show that Barr's theorem can be extended to higher-order infinitary logic, allowing us to constructivize classical proofs of higher-order geometric implications.

Outline. We extend simple type theory with families of propositions in Section 6.1. We define higher-order infinitary logic and its natural deduction system in Section 6.2. We study its semantics and prove the cut-elimination theorem in Section 6.3. We prove Barr's theorem in Section 6.4.

6.1 An Extension of Simple Type Theory

We want to model a higher-order infinitary logic, that is to have an infinite conjunction and an infinite disjunction. In simple type theory, we cannot define function symbols taking infinitely many arguments. That is why we extend simple type theory with a new type for families of propositions indexed by an arbitrary countable set, along with a constructor and destructors.

Definition 6.1

Types and terms are defined inductively:

$$\begin{array}{llll} \text{Types} & \tau, \sigma & ::= & \iota \mid o \mid o_{\mathcal{I}} \mid \tau \to \sigma \\ \text{Terms} & t, u, A_i, F & ::= & x \mid c \mid t\ u \mid \lambda x.\ t \mid \{i \mapsto A_i\}_{\mathcal{I}} \mid \pi_j\ F \end{array}$$

where ι is the type of individuals, o is the type of propositions, $o_{\mathcal{I}}$ is the type of families of propositions indexed by a countable set $\mathcal{I}$, x is a variable, c is a constant, $\{i \mapsto A_i\}_{\mathcal{I}}$ is the family of propositions $(A_i)_{i \in \mathcal{I}}$, and $\pi_j\ F$ is the projection on the j-th component of F.

Remark 6.2. *It is also possible to define a type theory with multiple base types $\iota_1, \ldots, \iota_n$ and multiple types of families of propositions $o_{\mathcal{I}_1}, \ldots, o_{\mathcal{I}_m}$.*

Free variables, substitutions, typing contexts and typing judgments are defined as for simple type theory (see Section 3.3.1). The typing rules of our extension of simple type theory are presented in Figure 6.1. Note that there is one Proj rule for each $j \in \mathcal{I}$.

We write Λ_τ for the set of terms of type τ. As Negri did for infinitary formulas [Neg21], we restrict terms so that they do not have infinitely many free variables. This simplifies the treatment of free variables in the All-I and Ex-E rules.

$$\frac{c : \tau \in \Sigma}{\Gamma \vdash c : \tau}\ \text{Const} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \text{Var} \qquad \frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x.\ t : \sigma \to \tau}\ \text{Abs}$$

$$\frac{\Gamma \vdash t : \sigma \to \tau \qquad \Gamma \vdash u : \sigma}{\Gamma \vdash t\ u : \tau}\ \text{App} \qquad \frac{\Gamma \vdash A_i : o \text{ for every } i \in \mathcal{I}}{\Gamma \vdash \{i \mapsto A_i\}_{\mathcal{I}} : o_{\mathcal{I}}}\ \text{Fam}$$

$$\frac{\Gamma \vdash F : o_{\mathcal{I}}}{\Gamma \vdash \pi_j\ F : o}\ \text{Proj}$$

Figure 6.1: Typing rules of the extension of simple type theory.

Remark 6.3. *Due to the typing rules of Figure 6.1, well-typed families $\{i \mapsto A_i\}_{\mathcal{I}}$ are necessarily of type $o_{\mathcal{I}}$, and well-typed projections $\pi_j\ F$ are necessarily of type o.*

Computation is introduced in this type theory through the usual β-reduction rule and η-contraction rule, extended with rules characterizing the projections.

Definition 6.4

We consider the following rules:

- the β-reduction rule $(\lambda x.\ t)\ u \hookrightarrow t[x \leftarrow u]$,
- the β-like projection rules $\pi_j\ \{i \mapsto A_i\}_{\mathcal{I}} \hookrightarrow A_j$,
- the η-contraction rule $\lambda x.\ t\ x \hookrightarrow t$, where x is not free in t,
- the η-like rule $\{i \mapsto \pi_i\ F\}_{\mathcal{I}} \hookrightarrow F$.

The reduction relation $\hookrightarrow$ is the smallest relation that contains these rules and such that:

- if $t \hookrightarrow t'$, then $t\ u \hookrightarrow t'\ u$ and $u\ t \hookrightarrow u\ t'$ and $\lambda x.\ t \hookrightarrow \lambda x.\ t'$.
- if $A_i \hookrightarrow B_i$ for every $i \in \mathcal{J}$, where $\mathcal{J} \neq \emptyset$ and $\mathcal{J} \subseteq I$, and $A_i = B_i$ otherwise, then $\{i \mapsto A_i\}_{\mathcal{I}} \hookrightarrow \{i \mapsto B_i\}_{\mathcal{I}}$,
- if $F \hookrightarrow F'$ then $\pi_j\ F \hookrightarrow \pi_j\ F'$.

The relation $\equiv$ is the reflexive, symmetric and transitive closure of $\hookrightarrow$.

The $\beta\pi$-reduction $\hookrightarrow_{\beta\pi}$ and the $\beta\pi$-conversion $\equiv_{\beta\pi}$ are defined like $\hookrightarrow$ and $\equiv$, but only considering the β-reduction rule and the β-like projection rules.

Properties. We have extended simple type theory with families of propositions. We have to check whether the intuition and expected results are confirmed.

Lemma 6.5 (Substitution)**.** *If* $\Gamma, x : \sigma, \Delta \vdash t : \tau$ *and* $\Gamma, \Delta \vdash u : \sigma$*, then* $\Gamma, \Delta \vdash t[x \leftarrow u] : \tau$*.*

Proof. By induction on $\Gamma, x : \sigma, \Delta \vdash t : \tau$. We only show the cases specific to families of propositions.

— Fam: Suppose that $\Gamma, x : \sigma, \Delta \vdash A_i : o$ for every $i \in \mathcal{I}$. By induction hypotheses, we have $\Gamma, \Delta \vdash A_i[x \leftarrow u] : o$ for every $i \in \mathcal{I}$. We derive $\Gamma, \Delta \vdash \{i \mapsto A_i[x \leftarrow u]\}_{\mathcal{I}} : o_{\mathcal{I}}$ using Fam.

— Proj: Suppose that $\Gamma, x : \sigma, \Delta \vdash F : o_{\mathcal{I}}$. We have $\Gamma, \Delta \vdash F[x \leftarrow u] : o_{\mathcal{I}}$ by induction hypothesis. Using Proj, we derive $\Gamma, \Delta \vdash \pi_j\ F[x \leftarrow u] : o$. □

Lemma 6.6 (Subject Reduction)**.** *If* $\Gamma \vdash t : \tau$ *and* $t \hookrightarrow u$*, then* $\Gamma \vdash u : \tau$*.*

Proof. By induction on $t \hookrightarrow u$.

— Suppose that $(\lambda x.\ t')\ v \hookrightarrow t'[x \leftarrow v]$. We necessarily have $\Gamma, x : \sigma \vdash t' : \tau$ and $\Gamma \vdash v : \sigma$. By Lemma 6.5, we derive that $\Gamma \vdash t'[x \leftarrow v] : \tau$.

— Suppose that $\pi_j\ \{i \mapsto A_i\}_{\mathcal{I}} \hookrightarrow A_j$. We necessarily have $\Gamma \vdash \pi_j\ \{i \mapsto A_i\}_{\mathcal{I}} : o$, and thus $\Gamma \vdash A_j : o$.

— Suppose that $\{i \mapsto \pi_i\ F\}_{\mathcal{I}} \hookrightarrow F$. We necessarily have $\Gamma \vdash \{i \mapsto \pi_i\ F\}_{\mathcal{I}} : o_{\mathcal{I}}$, and thus $\Gamma \vdash \pi_i\ F : o$ and $\Gamma \vdash F : o_{\mathcal{I}}$.

— Suppose that $\lambda x.\ t\ \ x \hookrightarrow t$, where x is not free in t. We necessarily have $\Gamma \vdash \lambda x.\ t\ x : \sigma \to \tau$, and thus $\Gamma \vdash t : \sigma \to \tau$.

— Suppose that $A_i \hookrightarrow B_i$ for every $i \in \mathcal{J}$ (where $\mathcal{J} \neq \emptyset$ and $\mathcal{J} \subseteq I$) and $A_i = B_i$ otherwise. We necessarily have $\Gamma \vdash \{i \mapsto A_i\}_{\mathcal{I}} : o_{\mathcal{I}}$, hence $\Gamma \vdash A_i : o$. Therefore $\Gamma \vdash B_i : o$ (by induction hypothesis when $i \in \mathcal{J}$ and by definition otherwise). It follows that $\Gamma \vdash \{i \mapsto B_i\}_{\mathcal{I}} : o_{\mathcal{I}}$.

— The other cases are proved similarly. □

Lemma 6.7. *The* β*-reduction and the* $\beta\pi$*-reduction relations are neither terminating nor confluent.*

Proof. For every $n \in \mathbb{N}^*$, let A_n be $P\ x \Rightarrow \ldots \Rightarrow P\ x$, where $P\ x$ occurs n times. Consider the term $(\lambda x.\ \{n \mapsto A_n\}_{\mathbb{N}^*})\ ((\lambda z.\ t)\ u)$.

— On the one hand, this term reduces to $(\lambda x.\ \{n \mapsto A_n\}_{\mathbb{N}^*})\ t[z \leftarrow u]$ and therefore to $\{n \mapsto A_n[x \leftarrow t[z \leftarrow u]]\}_{\mathbb{N}^*}$.

— On the other hand, this term reduces to $\{n \mapsto A_n[x \leftarrow (\lambda z.\ t)\ u]\}_{\mathbb{N}^*}$.

Each term $A_n[x \leftarrow (\lambda z.\ t)\ u]$ reduces to $A_n[x \leftarrow t[z \leftarrow u]]$ in n steps. But the family $\{n \mapsto A_n[x \leftarrow (\lambda z.\ t)\ u]\}_{\mathbb{N}^*}$ does not reduce to $\{n \mapsto A_n[x \leftarrow t[z \leftarrow u]]\}_{\mathbb{N}^*}$ in a finite number of steps, as there will always be a proposition $A_n[x \leftarrow (\lambda z.\ t)\ u]$ that has not yet reached $A_n[x \leftarrow t[z \leftarrow u]]$. This is a counter-example of both strong normalization and confluence for the β-reduction relation. It also applies for the $\beta\pi$-reduction relation, using $\pi_m\ \{n \mapsto B_n\}_{\mathbb{N}}$ instead of $((\lambda z.\ t)\ u)$ and B_m instead of $t[z \leftarrow u]$. □

We shall see, however, that well-typed terms have $\beta\pi$-head-normal forms.

Definition 6.8: $\beta\pi$-Head Normal Forms

$\beta\pi$-head normal forms are inductively defined by:

- an application $t\ u_1\ \ldots\ u_n$ where t is a variable or a constant and possibly $n = 0$,
- a projection $\pi_j\ (t\ u_1\ \ldots\ u_n)$ where t is a variable or a constant and possibly $n = 0$,
- an abstraction $\lambda x.\ t$ where t is a $\beta\pi$-head normal form,
- a family of propositions $\{i \mapsto A_i\}_{\mathcal{I}}$.

A term has a $\beta\pi$-head normal form when it is a $\beta\pi$-head normal form or when it $\beta\pi$-reduces to a $\beta\pi$-head normal form.

Remark 6.9. *In the case for families of propositions $\{i \mapsto A_i\}_{\mathcal{I}}$, we do not require every A_i to be a $\beta\pi$-head normal form. Otherwise, we could create a counter-example in which each A_n reaches a $\beta\pi$-head normal form in n steps, so it would be impossible for the family of propositions $\{n \mapsto A_n\}_{\mathbb{N}}$ to reach a $\beta\pi$-head normal form in a finite number of steps.*

We show that every well-typed term has a $\beta\pi$-head normal form. To do so, we adapt the well-known reducibility technique [Tai67, Gir72].

Definition 6.10

Let τ be a type. The set of terms R_τ is defined by induction:

- When $\tau \in \{\iota, o\}$, a term t belongs to R_τ if and only if it is of type τ and has a $\beta\pi$-head normal form.
- When $\tau = o_{\mathcal{I}}$, a term t belongs to $R_{o_{\mathcal{I}}}$ if and only if it is of type $o_{\mathcal{I}}$, it has a $\beta\pi$-head normal form, and for every $i \in \mathcal{I}$, we have $(\pi_i\ t) \in R_o$.
- When $\tau = \tau_1 \to \tau_2$, a term t belongs to R_τ if and only if it is of type $\tau_1 \to \tau_2$, it has a $\beta\pi$-head normal form, and for every term $u \in R_{\tau_1}$, we have $(t\ u) \in R_{\tau_2}$.

By definition, if $t \in R_\tau$ then t is of type τ and has a $\beta\pi$-head normal form.

Lemma 6.11. *Let t, u be terms of type τ such that $t \hookrightarrow_{\beta\pi} u$. If $u \in R_\tau$ then $t \in R_\tau$.*

Proof. By induction on the type τ. As $u \in R_\tau$, we know that u has a $\beta\pi$-head normal form, and thus t also has a $\beta\pi$-head normal form.

- If $\tau \in \{\iota, o\}$, we have $t \in R_\tau$ by definition.
- If $\tau = o_{\mathcal{I}}$, then for every $i \in \mathcal{I}$ we have $(\pi_i\ u) \in R_o$. Since $\pi_i\ t \hookrightarrow_{\beta\pi} \pi_i\ u$, we get $(\pi_i\ t) \in R_o$ by induction hypothesis. Therefore $t \in R_{o_{\mathcal{I}}}$.
- If $\tau = \tau_1 \to \tau_2$, for every $v \in R_{\tau_1}$, we have $(u\ v) \in R_{\tau_2}$. Since $t\ v \hookrightarrow_{\beta\pi} u\ v$, we get $(t\ v) \in R_{\tau_2}$ by induction hypothesis. Therefore $t \in R_{\tau_1 \to \tau_2}$. □

Lemma 6.12. *Let $t\ u_1\ \dots\ u_n$ be a term of type τ, with t a variable or a constant and possibly $n = 0$. Then $(t\ u_1\ \dots\ u_n) \in R_\tau$.*

Proof. By induction on the type τ. The term $t\ u_1\ \dots\ u_n$ is directly a $\beta\pi$-head normal form of type τ.

— If $\tau \in \{\iota, o\}$, then $(t\ u_1\ \dots\ u_n) \in R_\tau$ by definition.

— If $\tau = o_\mathcal{I}$, then for every $i \in \mathcal{I}$ we have $\pi_i\ (t\ u_1\ \dots\ u_n) \in R_o$ (as it is directly a $\beta\pi$-head normal form of type o), and thus $(t\ u_1\ \dots\ u_n) \in R_{o_\mathcal{I}}$.

— If $\tau = \tau_1 \to \tau_2$, then for every $v \in R_{\tau_1}$ we have $(t\ u_1\ \dots\ u_n\ v) \in R_{\tau_2}$ (by induction hypothesis), and thus $(t\ u_1\ \dots\ u_n) \in R_{\tau_1 \to \tau_2}$. □

Proposition 6.13

Let t be a term of type τ and θ be a substitution mapping each variable of type τ' to an element of $R_{\tau'}$. We have $t\theta \in R_\tau$.

Proof. We proceed by induction on t.

— If t is a variable, then we directly have $t\theta \in R_\tau$.

— If t is a constant, then $t\theta = t$, and we have $t \in R_\tau$ using Lemma 6.12.

— If t is an application $(u\ v)$, then u has type $\sigma \to \tau$ and v has type σ. By induction hypotheses, $u\theta \in R_{\sigma\to\tau}$ and $v\theta \in R_\sigma$. By definition of $R_{\sigma\to\tau}$, we have $t\theta = (u\theta\ v\theta) \in R_\tau$.

— If t is an abstraction $\lambda x.\ t'$, then $\tau = \tau_1 \to \tau_2$ and t' has type τ_2. We want to show that $t\theta \in R_\tau$, that is $(\lambda x.\ t'\theta) \in R_{\tau_1\to\tau_2}$. By induction hypothesis, $t'\theta \in R_{\tau_2}$. It follows that $t'\theta$ has a $\beta\pi$-head normal form and therefore that $(\lambda x.\ t'\theta)$ has a $\beta\pi$-head normal form. Let $u \in R_{\tau_1}$. We have $(\lambda x.\ t'\theta)\ u \hookrightarrow_{\beta\pi} t'[\theta, x \leftarrow u]$. We have $t'[\theta, x \leftarrow u] \in R_{\tau_2}$ by induction hypothesis. Using Lemma 6.11, we get $((\lambda x.\ t'\theta)\ u) \in R_{\tau_2}$. Therefore $(\lambda x.\ t'\theta) \in R_{\tau_1\to\tau_2}$.

— If t is a projection $\pi_j\ F$, then $\tau = o$ and F has type $o_\mathcal{I}$. By induction hypothesis, $F\theta \in R_{o_\mathcal{I}}$. It follows that $(\pi_j\ F\theta) \in R_o$, that is $(\pi_j\ F)\theta \in R_o$.

— Suppose that t is a family of propositions $\{i \mapsto A_i\}_\mathcal{I}$. By definition, $\{i \mapsto A_i\theta\}_\mathcal{I}$ is a $\beta\pi$-head normal form. By induction hypothesis, we have $A_j\theta \in R_o$ for every $j \in \mathcal{I}$. We have $\pi_j\ \{i \mapsto A_i\theta\}_\mathcal{I} \hookrightarrow_{\beta\pi} A_j\theta$. Using Lemma 6.11, we get $\pi_j\ \{i \mapsto A_i\theta\}_\mathcal{I} \in R_o$. Therefore $\{i \mapsto A_i\theta\}_\mathcal{I} \in R_{o_\mathcal{I}}$. □

Theorem 6.14

Every well-typed term has a $\beta\pi$-head normal form.

Proof. Let t be a term of type τ. We take the substitution θ that maps each variable x of type τ' to itself. We have $x \in R_{\tau'}$ by Lemma 6.12. We apply Proposition 6.13 and we get $t\theta = t \in R_\tau$. By definition of R_τ, t has a $\beta\pi$-head normal form. □

Remark 6.15. *Weak head normal forms can be defined like head normal forms, except that the condition on t in the abstraction case is dropped. From Theorem 6.14, we directly get that every well-typed term has a weak head normal form.*

6.2 Higher-Order Infinitary Logic

Syntax. The logical constants defining the connectives and quantifiers are the same as in higher-order logic (see Section 3.3.2), except that we replace the finite conjunction $\wedge$ and the finite disjunction $\vee$ of type $o \to o \to o$ by the infinite conjunction $\bigwedge_{\mathcal{I}}$ and the infinite disjunction $\bigvee_{\mathcal{I}}$ of type $o_{\mathcal{I}} \to o$.

Example 6.16. *Suppose that we want to model the interactions between researchers, represented by terms of type ι. We have several binary relations of type $\iota \to \iota \to o$, for example* coauthors, colleagues *or* friends*. We want to know whether two researchers x and y are related, no matter the number of intermediate researchers between them or the nature of the relations involved. Intuitively, we check whether x and y are related in n steps, and we do this until we have found such a $n \in \mathbb{N}^*$. In our logic, we easily model this problem with*

$$(\exists' R.\ R\ x\ y) \vee \bigvee\nolimits_{\mathbb{N}\setminus\{0,1\}} \{n \mapsto \exists z_1 \ldots \exists z_{n-1}. \exists' R_1 \ldots \exists' R_n.\ R_1\ x\ z_1 \wedge \ldots \wedge R_n\ z_{n-1}\ y\}_{\mathbb{N}\setminus\{0,1\}}$$

where $\exists' R$ quantifies over relations R that are coauthors, colleagues *or* friends*—this can be formalized using $\exists$ and an equality predicate. This formula features an infinite disjunction and higher-order quantifiers.*

When taking $y =$ Erdos*, this problem generalizes Erdős number to any kind of relations between researchers. We exactly obtain Erdős number when the relations R_i are* coauthors*.*

Natural deduction. A context Γ is a finite sequence of formulas. We write $\Gamma \vdash A$ when A is derivable from Γ. We write $\mathsf{fv}(t_1, \ldots, t_n)$ for the set of free variables that occur in the terms $t_1, \ldots, t_n$. Remember that we always assume that terms have a finite number of free variables.

The natural deduction rules for higher-order infinitary logic are those of higher-order logic (Figure 3.1 and Figure 3.4), that are modified in two ways.

First, the Conv rule is stated with the conversion $\equiv$ and not only $\equiv_\beta$. Note that the Conv rule implicitly requires that B is of type o.

Second, the natural deduction rules for the infinite conjunction and disjunction generalize the usual natural deduction rules for the finite conjunction and disjunction:

$$\frac{\Gamma \vdash \pi_i\ F \text{ for every } i \in \mathcal{I}}{\Gamma \vdash \bigwedge_{\mathcal{I}} F}\ \text{AndInf-I} \qquad \frac{\Gamma \vdash \bigwedge_{\mathcal{I}} F}{\Gamma \vdash \pi_j\ F}\ \text{AndInf-E}$$

$$\frac{\Gamma \vdash \pi_j\ F}{\Gamma \vdash \bigvee_{\mathcal{I}} F}\ \text{OrInf-I} \qquad \frac{\Gamma \vdash \bigvee_{\mathcal{I}} F \qquad \Gamma, \pi_i\ F \vdash C \text{ for every } i \in \mathcal{I}}{\Gamma \vdash C}\ \text{OrInf-E}$$

There is one rule AndInf-E and one rule OrInf-I for each $j \in \mathcal{I}$. Note that, due to the OrInf-E and AndInf-I inference rules, derivation trees possibly have an infinite width, but each branch has a finite height.

The formula $A \wedge B$ (respectively $A \vee B$) is defined by $\bigwedge_{\{0,1\}} \{i \mapsto A_i\}_{\{0,1\}}$ (respectively $\bigvee_{\{0,1\}} \{i \mapsto A_i\}_{\{0,1\}}$), where $A_0 = A$ and $A_1 = B$. The inference rules for the finite conjunction (And-I, And-EL, And-ER) and disjunction (Or-IL, Or-IR, Or-E) are particular cases of those for the infinite conjunction and disjunction.

Lemma 6.17. *The weakening rule, contraction rule and exchange rule are admissible.*

$$\frac{\Gamma, \Delta \vdash B}{\Gamma, A, \Delta \vdash B}\ \text{Weak} \qquad \frac{\Gamma, A, A, \Delta \vdash B}{\Gamma, A, \Delta \vdash B}\ \text{Contr} \qquad \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}\ \text{Exch}$$

Proof. By induction on the derivation. □

Example 6.18 (Natural numbers)**.** *Let us define 0 of type ι, and the successor function* succ *of type $\iota \to \iota$. We also set an infix equality symbol $=$ of type $\iota \to \iota \to o$, that satisfies the reflexivity axiom $\forall x.\ x = x$ and Leibniz's law $\forall x. \forall y. \forall P.\ (x = y \wedge P\ x) \Rightarrow P\ y$. We encode the induction principle using a higher-order quantification on any predicate P.*

$$\forall P.\ \Big(\big(P\ 0 \wedge \forall x.\ (P\ x \Rightarrow P\ (\mathsf{succ}\ x))\big) \Rightarrow \forall x.\ P\ x\Big) \qquad \text{(NatInd)}$$

We also encode an infinitary formula, stating that natural numbers are iterated successors of zero, where $\mathsf{succ}^n\ 0$ *means that the successor* succ *has been applied n times to 0.*

$$\forall x.\ \bigvee\nolimits_{\mathbb{N}} \{n \mapsto x = \mathsf{succ}^n\ 0\}_{\mathbb{N}} \qquad \text{(ItNat)}$$

In higher-order infinitary logic, these two principles are equivalent.

— *To prove that* NatInd *implies* ItNat*, we use the induction principle* NatInd *with the predicate $\lambda x.\ \bigvee_{\mathbb{N}} \{n \mapsto x = \mathsf{succ}^n\ 0\}_{\mathbb{N}}$. We prove the base case with* OrInf-I *and $n = 0$. We prove the induction step with* OrInf-E*, as once we assume that $x = \mathsf{succ}^n\ 0$ for some $n \in \mathbb{N}$, it is easy to show that $\mathsf{succ}\ x = \mathsf{succ}^{n+1}\ 0$, and thus $\bigvee_{\mathbb{N}} \{n \mapsto \mathsf{succ}\ x = \mathsf{succ}^n\ 0\}_{\mathbb{N}}$.*

— *To prove that* ItNat *implies* NatInd*, we use* OrInf-E *with the infinite disjunction* ItNat*. For every $n \in \mathbb{N}$, we have to prove $P\ x$ assuming $P\ 0 \wedge \forall x.\ (P\ x \Rightarrow P\ (\mathsf{succ}\ x))$ and $x = \mathsf{succ}^n\ 0$. Using Leibniz's law, it remains to show $P\ (\mathsf{succ}^n\ 0)$. To do so, we apply n times the induction step from $P\ 0$.*

We can therefore conveniently use either of the two principles.

Cuts. Unlike in sequent calculus, cuts in natural deduction may be defined with various degrees of looseness. The restricted notion of cut is that of an introduction rule immediately followed by its corresponding elimination rule.

Example 6.19 (Cut)**.** *The following proof*

$$\dfrac{\dfrac{\dfrac{}{A, B \vdash A}\ \textit{Ax} \quad \dfrac{}{A, B \vdash B}\ \textit{Ax}}{A, B \vdash A \wedge B}\ \textit{And-I}}{A, B \vdash A}\ \textit{And-EL}$$

contains a cut, as the And-I is immediately followed by And-EL.

In natural deduction, there is also the notion of *commutative cuts*, that separate the introduction and elimination rules of a single connective with a series of Or-E, OrInf-E and Ex-E rules.

Example 6.20 (Commutative Cut)**.** *The following proof, where* $\Gamma = A, B, C \vee D$*,*

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash C \vee D}\,\textit{Ax} \quad \dfrac{\dfrac{}{\Gamma, C \vdash A}\,\textit{Ax} \quad \dfrac{}{\Gamma, C \vdash B}\,\textit{Ax}}{\Gamma, C \vdash A \wedge B}\,\textit{And-I} \quad \dfrac{\dfrac{}{\Gamma, D \vdash A}\,\textit{Ax} \quad \dfrac{}{\Gamma, D \vdash B}\,\textit{Ax}}{\Gamma, D \vdash A \wedge B}\,\textit{And-I}}{\Gamma \vdash A \wedge B}\,\textit{Or-E}}{\Gamma \vdash A}\,\textit{And-EL}$$

contains a commutative cut, as the rules And-I and And-EL are separated by Or-E.

Cut-free natural deduction. We define a cut-free natural deduction system following [DH07, DH12, GH15]. We introduce two restrictions of the turnstile $\vdash$, to respectively account for cut-free derivations ($\vdash^{\uparrow}$), and neutral cut-free derivations ($\vdash^{\downarrow}$). The cut-free natural deduction inference rules are given in Figure 6.2, where $\vdash^{*}$ is used to denote either $\vdash^{\downarrow}$ or $\vdash^{\uparrow}$. The Coerce rule turns neutral cut-free proofs into cut-free proofs. The natural deduction inference rules (with cuts) is obtained by ignoring the adornment of the turnstile—the Coerce rule becomes vacuously admissible with the plain turnstile relation.

The rules of Figure 6.2 precludes the traditional cuts. For instance, the proof of Example 6.19, that contains a cut, cannot be expressed because the conclusion of And-I is $A, B \vdash^{\uparrow} A \wedge B$ while the premise of And-EL is $A, B \vdash^{\downarrow} A \wedge B$.

The rules of Figure 6.2 additionally precludes the so-called commutative cuts. This is because we have restricted the conclusion sequent of Or-E, OrInf-E and Ex-E, plus the Bot-E rule, to be cut-free ($\vdash^{\uparrow}$), rather than letting it being neutral ($\vdash^{\downarrow}$), as it would be the case for lighter cut-free proof definitions. For instance, the proof of Example 6.20, that contains a commutative cut, cannot be expressed with the cut-free rules, because the conclusion of Or-E has to be $\Gamma \vdash^{\uparrow} A \wedge B$ while the premise of And-EL has to be $\Gamma \vdash^{\downarrow} A \wedge B$.

For both the natural deduction system and the cut-free natural deduction system, we consider either classical logic CL or intuitionistic logic IL—in which we do not have the PEM rule. We write $\vdash_{\mathsf{L}}$ when we consider the inference rules of $\mathsf{L} \in \{\mathsf{CL}, \mathsf{IL}\}$.

Lemma 6.21. *The weakening rule, contraction rule and exchange rule are admissible for* $\vdash^{\downarrow}$ *and* $\vdash^{\uparrow}$*.*

$$\dfrac{\Gamma, \Delta \vdash^{*} B}{\Gamma, A, \Delta \vdash^{*} B}\,\textsf{Weak} \qquad \dfrac{\Gamma, A, A, \Delta \vdash^{*} B}{\Gamma, A, \Delta \vdash^{*} B}\,\textsf{Contr} \qquad \dfrac{\Gamma, A, B, \Delta \vdash^{*} C}{\Gamma, B, A, \Delta \vdash^{*} C}\,\textsf{Exch}$$

Proof. By mutual induction on the derivation. □

Lemma 6.22 (Axiom Replacement)**.** *The following rule is admissible:*

$$\dfrac{\Gamma \vdash^{\downarrow} A \qquad \Delta, A, \Sigma \vdash^{*} B}{\Delta, \Gamma, \Sigma \vdash^{*} B}\,\textsf{Ax-R}$$

$$\frac{}{\Gamma, A, \Delta \vdash^{\downarrow} A}\ \text{Ax} \qquad \frac{\Gamma \vdash^{*} A \qquad A \equiv B}{\Gamma \vdash^{*} B}\ \text{Conv} \qquad \frac{\Gamma \vdash^{\downarrow} A}{\Gamma \vdash^{\uparrow} A}\ \text{Coerce}$$

$$\frac{\Gamma, A \vdash^{\uparrow} B}{\Gamma \vdash^{\uparrow} A \Rightarrow B}\ \text{Imp-I} \qquad \frac{\Gamma \vdash^{\downarrow} A \Rightarrow B \qquad \Gamma \vdash^{\uparrow} A}{\Gamma \vdash^{\downarrow} B}\ \text{Imp-E}$$

$$\frac{\Gamma \vdash^{\uparrow} \pi_i\ F \text{ for every } i \in \mathcal{I}}{\Gamma \vdash^{\uparrow} \bigwedge_{\mathcal{I}} F}\ \text{AndInf-I} \qquad \frac{\Gamma \vdash^{\downarrow} \bigwedge_{\mathcal{I}} F}{\Gamma \vdash^{\downarrow} \pi_j\ F}\ \text{AndInf-E}$$

$$\frac{\Gamma \vdash^{\uparrow} \pi_j\ F}{\Gamma \vdash^{\uparrow} \bigvee_{\mathcal{I}} F}\ \text{OrInf-I} \qquad \frac{\Gamma \vdash^{\downarrow} \bigvee_{\mathcal{I}} F \qquad \Gamma, \pi_i\ F \vdash^{\uparrow} C \text{ for every } i \in \mathcal{I}}{\Gamma \vdash^{\uparrow} C}\ \text{OrInf-E}$$

$$\frac{\Gamma \vdash^{\uparrow} P\ x \qquad x \notin \mathsf{fv}(\Gamma, P)}{\Gamma \vdash^{\uparrow} \forall\ P}\ \text{All-I} \qquad \frac{\Gamma \vdash^{\downarrow} \forall\ P}{\Gamma \vdash^{\downarrow} P\ t}\ \text{All-E}$$

$$\frac{\Gamma \vdash^{\uparrow} P\ t}{\Gamma \vdash^{\uparrow} \exists\ P}\ \text{Ex-I} \qquad \frac{\Gamma \vdash^{\downarrow} \exists\ P \qquad \Gamma, P\ x \vdash^{\uparrow} A \qquad x \notin \mathsf{fv}(\Gamma, P, A)}{\Gamma \vdash^{\uparrow} A}\ \text{Ex-E}$$

$$\frac{\Gamma \vdash^{\downarrow} \bot}{\Gamma \vdash^{\uparrow} A}\ \text{Bot-E} \qquad \frac{}{\Gamma \vdash^{\downarrow} A \vee \neg A}\ \text{PEM}$$

Figure 6.2: Cut-free natural deduction rules of higher-order infinitary logic.

Proof. By induction on the derivation of $\Delta, A, \Sigma \vdash^{*} B$. Suppose $\Gamma \vdash^{\downarrow} A$. We only show a few cases.

— AndInf-I: Suppose $\Delta, A, \Sigma \vdash^{\uparrow} \pi_i\ F$ for every $i \in \mathcal{I}$. By induction hypotheses, we get $\Delta, \Gamma, \Sigma \vdash^{\uparrow} \pi_i\ F$ for every $i \in \mathcal{I}$. Using AndInf-I, we derive $\Delta, \Gamma, \Sigma \vdash^{\uparrow} \bigwedge_{\mathcal{I}} F$.

— OrInf-E: Suppose $\Delta, A, \Sigma \vdash^{\downarrow} \bigvee_{\mathcal{I}} F$ and $\Delta, A, \Sigma, \pi_i\ F \vdash^{\uparrow} C$ for every $i \in \mathcal{I}$. By induction hypotheses, we get $\Delta, \Gamma, \Sigma \vdash^{\downarrow} \bigvee_{\mathcal{I}} F$ and $\Delta, \Gamma, \Sigma, \pi_i\ F \vdash^{\uparrow} C$ for every $i \in \mathcal{I}$. Using OrInf-E, we derive $\Delta, \Gamma, \Sigma \vdash^{\uparrow} C$. □

6.3 Soundness, Strong Completeness and Cut Elimination

In this section, we study the semantics of our higher-order infinitary logic. To do so, it is important to take into account its *non-extensional* nature. To illustrate this feature, let P be a predicate of type $o \to o$. Without any further information on P, it is impossible to show $P\ \bot \vdash P\ (\bot \wedge \bot)$, although $\bot$ and $\bot \wedge \bot$ are logically equivalent. Therefore, the term $\bot$ must receive a different interpretation than $\bot \wedge \bot$, although we have $\bot_{\mathcal{H}} = \bot_{\mathcal{H}} \wedge_{\mathcal{H}} \bot_{\mathcal{H}}$ in any Heyting algebra $\mathcal{H}$. This imposes a separation of the domain of

interpretation of formulas M_o from the Heyting algebra $\mathcal{H}$. To do so, we resort to *Heyting applicative structure*, and we define a sound notion of model following [DL05, HL08a, HL08b].

Proving completeness amounts to exhibit a particular Heyting algebra, based on provability. Instead of building the standard Lindenbaum algebra for the natural deduction with cuts, we construct a cut-free Heyting algebra following [HL08a, HL08b], that is based on *Schütte's semi-valuations* [Sch60, Smu68]. We use this Heyting algebra to build a specific model, based on the *V-complex technique* [Pra68, Tak67, And71], ensuring $\mathsf{M}_o \neq \mathcal{H}$. This allows us to prove a strong version of completeness that entails the cut-elimination theorem for higher-order infinitary logic in natural deduction.

Note that we establish a notion of model, based in Heyting algebras, for the *intuitionistic* variant of higher-order infinitary logic. It also applies to the classical variant, provided that we restrict the Heyting algebras $\mathcal{H}$ of the models to be Boolean algebras. Moreover, the cut-free Heyting algebra we build is also a Boolean algebra, provided that we are in classical logic. Therefore, all the results and constructions of this section apply to both the intuitionistic variant and the classical variant.

We recall the preliminaries on Heyting algebras in Section 6.3.1. We define a sound notion of model in Section 6.3.2. We build a cut-free-provability Heyting algebra and a cut-free-provability model in Sections 6.3.3 and 6.3.4. We consider the classical variant of higher-order infinitary logic in Section 6.3.5.

6.3.1 Preliminaries on Heyting Algebras

We recall that a *Heyting algebra* is an ordered lattice $(\mathcal{H}, \leq, \wedge, \vee)$, where

— $\leq$ is a partial order on $\mathcal{H}$,

$$\begin{array}{ll} \text{for any } a, b, c \in \mathcal{H}, & a \leq a \\ & \text{if } a \leq b \text{ and } b \leq a \text{ then } a = b \\ & \text{if } a \leq b \text{ and } b \leq c \text{ then } a \leq c \end{array}$$

— $\wedge$ is a binary greatest lower bound operator, called meet,

$$\begin{array}{ll} \text{for any } a, b, c \in \mathcal{H}, & a \wedge b \leq a \\ & a \wedge b \leq b \\ & \text{if } c \leq a \text{ and } c \leq b \text{ then } c \leq a \wedge b \end{array}$$

— $\vee$ is a binary largest upper bound operator, called join,

$$\begin{array}{ll} \text{for any } a, b, c \in \mathcal{H}, & a \leq a \vee b \\ & b \leq a \vee b \\ & \text{if } a \leq c \text{ and } b \leq c \text{ then } a \vee b \leq c \end{array}$$

— In addition, $\mathcal{H}$ is equipped with a relative pseudo-complement operator $\rightarrow$, that verifies the *implication property*:

$$\text{for any } a, b, c \in \mathcal{H}, \quad a \wedge b \leq c \text{ if and only if } a \leq b \rightarrow c$$

A *complete Heyting algebra* is a Heyting algebra whose base lattice is complete, that is to say, that enjoys arbitrary meets ($\bigwedge$) and joins ($\bigvee$). The global maximum (empty meet) and minimum (empty joint) will be denoted $\top_{\mathcal{H}}$ and $\bot_{\mathcal{H}}$, respectively. The support $\mathcal{H}$ of the Heyting algebra will also denote the algebra itself. In any Heyting algebra, $\wedge$ and $\vee$ distribute over each other.

$$\text{for any } a,b,c \in \mathcal{H}, \quad \begin{array}{l} a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \\ a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \end{array}$$

Moreover, $\wedge$ distributes over $\bigvee$, whenever the latter exists.

A Heyting algebra is a *Boolean algebra* when $a \vee (a \to \bot_{\mathcal{H}}) = \top_{\mathcal{H}}$ for any $a \in \mathcal{H}$.

6.3.2 Models for Higher-Order Infinitary Logic

The non-extensional nature of our higher-order infinitary logic imposes a separation of the domain of interpretation of formulas M_o from the Heyting algebra $\mathcal{H}$. The latter contains truth values, while the former carries more information. Both are linked through a truth-value extraction function $\omega : \mathsf{M}_o \to \mathcal{H}$.

We adapt the notion of model introduced in [DL05, HL08a, HL08b], that is based on Heyting applicative structures.

Definition 6.23: Heyting Applicative Structure

A Heyting applicative structure is a quintuple $\langle \mathsf{M}, \mathsf{App}, \mathsf{Const}, \omega, \mathcal{H} \rangle$ composed of:

- a set $\mathsf{M}_\tau \neq \emptyset$ for each type τ, called domain,
- a function $\mathsf{App}_{\tau,\sigma} : \mathsf{M}_{\tau\to\sigma} \times \mathsf{M}_\tau \to \mathsf{M}_\sigma$ for each pair of types (τ, σ), and a function $\mathsf{App}_{o_\mathcal{I}} : \mathsf{M}_{o_\mathcal{I}} \times \mathcal{I} \to \mathsf{M}_o$
- an interpretation function Const_τ for each type τ, taking constants of type τ into M_τ,
- a complete Heyting algebra $\mathcal{H}$,
- a function $\omega : \mathsf{M}_o \to \mathcal{H}$,

such that the following conditions are satisfied, for every $m \in \mathsf{M}_{\tau\to o}$ and every $m' \in \mathsf{M}_{o_\mathcal{I}}$:

$$\begin{array}{rcl} \omega(\mathsf{Const}(\bot)) & = & \bot_{\mathcal{H}} \\ \omega(\mathsf{App}(\mathsf{App}(\mathsf{Const}(\Rightarrow), d_1), d_2)) & = & \omega(d_1) \to_{\mathcal{H}} \omega(d_2) \\ \omega(\mathsf{App}(\mathsf{Const}(\exists_\tau), m)) & = & \bigvee_{d \in \mathsf{M}_\tau} \omega(\mathsf{App}(m, d)) \\ \omega(\mathsf{App}(\mathsf{Const}(\forall_\tau), m)) & = & \bigwedge_{d \in \mathsf{M}_\tau} \omega(\mathsf{App}(m, d)) \\ \omega(\mathsf{App}(\mathsf{Const}(\bigwedge_\mathcal{I}), m')) & = & \bigwedge_{i \in \mathcal{I}} \omega(\mathsf{App}(m', i)) \\ \omega(\mathsf{App}(\mathsf{Const}(\bigvee_\mathcal{I}), m')) & = & \bigvee_{i \in \mathcal{I}} \omega(\mathsf{App}(m', i)) \end{array}$$

When interpreting formulas in a model, and as soon as we have complete Heyting algebras, there is not much difference between quantifiers and the infinitary connectives. In both cases, we use arbitrary meets and joins: indexed by $d \in \mathsf{M}_\tau$ for the quantifiers and indexed by $i \in \mathcal{I}$ for the infinitary connectives.

An assignment is a function φ with finite support, such that for any variable x of type τ of the support, $\varphi(x)$ is an element of M_τ. The assignment $\varphi[d/x]$ is identical to φ, save on x, where $\varphi[d/x](x) = d$.

Definition 6.24: Model

Let $\langle \mathsf{M}, \mathsf{App}, \mathsf{Const}, \omega, \mathcal{H}\rangle$ be a Heyting applicative structure. A model is a function $[\![\_]\!]\_$ that maps a term of type τ and an assignment (with a support containing $\mathsf{fv}(t)$) to M_τ, such that:

$$\begin{array}{rcll}
[\![c]\!]_\varphi & = & \mathsf{Const}(c) & \\
[\![x]\!]_\varphi & = & \varphi(x) & \\
[\![t\ u]\!]_\varphi & = & \mathsf{App}([\![t]\!]_\varphi, [\![u]\!]_\varphi) & \\
[\![\pi_j\ F]\!]_\varphi & = & \mathsf{App}([\![F]\!]_\varphi, j) & \\
\mathsf{App}([\![\lambda x_\tau.\ t_\sigma]\!]_\varphi, d) & = & [\![t]\!]_{\varphi[d/x]} & \text{for every } d \in \mathsf{M}_\tau \\
\mathsf{App}([\![\{i \mapsto A_i\}_\mathcal{I}]\!]_\varphi, j) & = & [\![A_j]\!]_\varphi & \text{for every } j \in \mathcal{I} \\
[\![t]\!]_\varphi & = & [\![u]\!]_\varphi & \text{if } t \equiv u
\end{array}$$

By model, we will often refer to the combination of the Heyting applicative structure, the model interpretation function above, and the assignment. With this notion, we can state the soundness theorem.

When $\Gamma = A_1, \ldots, A_n$, we write $\omega([\![\Gamma]\!]_\varphi)$ for $\omega([\![A_1]\!]_\varphi) \wedge \ldots \wedge \omega([\![A_n]\!]_\varphi)$.

Theorem 6.25: Soundness

If $\Gamma \vdash A$ then for any model $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![A]\!]_\varphi)$.

Proof. By induction on the derivation. We show the cases for the infinitary connectives and the case for PEM.

— AndInf-I: By induction hypotheses, we have $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![\pi_i\ F]\!]_\varphi)$ for every $i \in \mathcal{I}$. As we are in a complete Heyting algebra, we have $\omega([\![\Gamma]\!]_\varphi) \leq \bigwedge_{i\in\mathcal{I}} \omega([\![\pi_i\ F]\!]_\varphi)$, that is $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![\bigwedge_\mathcal{I} F]\!]_\varphi)$.

— AndInf-E: By induction hypothesis, we have $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![\bigwedge_\mathcal{I} F]\!]_\varphi)$. By definition, we have $\omega([\![\bigwedge_\mathcal{I} F]\!]_\varphi) = \omega(\mathsf{App}(\mathsf{Const}(\bigwedge_\mathcal{I}), [\![F]\!]_\varphi)) = \bigwedge_{i\in\mathcal{I}} \omega(\mathsf{App}([\![F]\!]_\varphi, i)) = \bigwedge_{i\in\mathcal{I}} \omega([\![\pi_i\ F]\!]_\varphi)$. Therefore $\omega([\![\Gamma]\!]_\varphi) \leq \bigwedge_{i\in\mathcal{I}} \omega([\![\pi_i\ F]\!]_\varphi) \leq \omega([\![\pi_j\ F]\!]_\varphi)$.

— OrInf-I: By induction hypothesis, we have $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![\pi_j\ F]\!]_\varphi)$. We have $\omega([\![\pi_j\ F]\!]_\varphi) \leq \bigvee_{i\in\mathcal{I}} \omega([\![\pi_i\ F]\!]_\varphi) = \omega([\![\bigvee_\mathcal{I} F]\!]_\varphi)$. Therefore $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![\bigvee_\mathcal{I} F]\!]_\varphi)$.

— OrInf-E: By induction hypotheses, we have $\omega([\![\Gamma]\!]_\varphi) \leq \bigvee_{i\in\mathcal{I}} \omega([\![\pi_i\ F]\!]_\varphi)$ and $\omega([\![\Gamma]\!]_\varphi) \wedge \omega([\![\pi_i\ F]\!]_\varphi) \leq \omega([\![C]\!]_\varphi)$ for any $i \in \mathcal{I}$. Using the implication property, we get $\omega([\![\pi_i\ F]\!]_\varphi) \leq \omega([\![\Gamma]\!]_\varphi) \to \omega([\![C]\!]_\varphi)$ for any $i \in \mathcal{I}$. Therefore we derive $\bigvee_{i\in\mathcal{I}} \omega([\![\pi_i\ F]\!]_\varphi) \leq \omega([\![\Gamma]\!]_\varphi) \to \omega([\![C]\!]_\varphi)$. We get $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![\Gamma]\!]_\varphi) \to \omega([\![C]\!]_\varphi)$, and thus $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![C]\!]_\varphi)$. □

6.3.3 The Cut-Free-Provability Heyting Algebra

We build a cut-free Heyting algebra following [HL08a, HL08b]. It is based on Schütte's semi-valuations [Sch60, Smu68], that have been first used to show cut admissibility for higher-order classical logic [Tak67, Pra68, And71]. Semi-valuations have been subsequently extended to higher-order intuitionistic logic [Mae91, DL05] and are nowadays a versatile tool, that can be fine-tuned a number of ways and fits many logics [HL08a, HL08b, Mae91, Oka02]. In this section, semi-valuations are embodied by the upper and lower values $\lceil \_ \rceil$ and $\lfloor \_ \rfloor$, that respectively represent the maximal and minimal constraints that our target model interpretation should respect. This is the essence of the model construction of Definition 6.32, that relies itself on Proposition 6.31.

Definition 6.26: Upper Value

The upper value of the judgment $\Gamma \vdash A$, denoted $\lceil \Gamma \vdash A \rceil$, is the set of contexts $\{\Delta \mid \Gamma, \Delta \vdash^{\uparrow} A \text{ is derivable}\}$. We write $\lceil A \rceil$ for $\lceil \vdash A \rceil$.

Definition 6.27: Cut-Free-Provability Algebra

Let $\mathcal{H}$ be the least set containing the $\lceil \Gamma \vdash A \rceil$ for any Γ, A, that is closed by arbitrary intersection. Let $\leq$ be $\subseteq$ and $\wedge$ be $\cap$. Let arbitrary meets and joins of any set of elements $S \subseteq \mathcal{H}$ be:

— $\bigwedge S = \bigcap S$, the intersection of the elements of S,

— $\bigvee S = \bigcap \{c \in \mathcal{H} \mid \forall s \in S, s \leq c\}$, the pseudo-union.

For any $a, b \in \mathcal{H}$, we let $a \to b = \bigvee \{c \in \mathcal{H} \mid a \wedge c \leq b\}$.

Note that for any $S \subseteq \mathcal{H}$, the set $\{c \in \mathcal{H} \mid \forall s \in S, s \leq c\}$ is not empty, as it contains at least the set of all contexts. So arbitrary meets and joins always exist. Moreover, the definition of arbitrary joins can be slightly simplified and proved to be equal to $\bigvee S = \bigcap \{\lceil \Gamma \vdash A \rceil \mid \forall s \in S, s \leq \lceil \Gamma \vdash A \rceil\}$. This is because any $c \in \mathcal{H}$ is of the form $\bigcap_{k \in K} \lceil \Gamma_k \vdash A_k \rceil$. In the sequel, we will use either of those characterizations.

Proposition 6.28

$\mathcal{H}$ is a complete Heyting algebra.

Proof. By construction, $\mathcal{H}$ is an ordered lattice that enjoys arbitrary joins and meets. We must check the implication property: $a \wedge b \leq c$ if and only if $a \leq b \to c$. For the direct implication, suppose $a \wedge b \leq c$. We have $a \subseteq a$ and $b \wedge a \leq c$. Therefore we derive $a \leq \bigvee \{d \in \mathcal{H} \mid b \wedge d \leq c\} = b \to c$.

Let us show the reverse implication. Let $\Gamma \in a \wedge b$ and suppose that $a \leq b \to c$. By definition of $\mathcal{H}$, we have $c = \bigcap_{k \in K} \lceil \Sigma_k \vdash C_k \rceil$. We want to show that $\Sigma_k, \Gamma \vdash^{\uparrow} C_k$ for each $k \in K$. We have $\Gamma \in a \leq b \to c = \bigvee \{d \in \mathcal{H} \mid b \wedge d \leq c\}$.

As an intermediate step, let us prove that $d \leq \lceil \Sigma_k, \Gamma \vdash C_k \rceil$ for any d such that $b \wedge d \leq c$. Let $\Delta \in d$. Upper values are provability-based, and therefore elements of $\mathcal{H}$

admit weakening, contraction and exchange. From $\Gamma \in b$ and $\Delta \in d$, we thus get $\Gamma, \Delta \in b$ and $\Gamma, \Delta \in d$, and therefore $\Gamma, \Delta \in b \wedge d \leq c$. This entails $\Gamma, \Delta \in \lceil \Sigma_k \vdash C_k \rceil$, i.e., $\Delta \in \lceil \Sigma_k, \Gamma \vdash C_k \rceil$. Thus $d \leq \lceil \Sigma_k, \Gamma \vdash C_k \rceil$ for any d such that $b \wedge d \leq c$.

Since $\Gamma \in \bigvee \{d \in \mathcal{H} \mid b \wedge d \leq c\}$, we derive $\Gamma \in \lceil \Sigma_k, \Gamma \vdash C_k \rceil$. We therefore get $\Sigma_k, \Gamma, \Gamma \vdash^{\uparrow} C_k$. By contraction, we derive $\Sigma_k, \Gamma \vdash^{\uparrow} C_k$. □

Definition 6.29: Lower Value

Let $s \in \mathcal{H}$ be a set of contexts. Its lower value is the least element of $\mathcal{H}$ that contains s, i.e., $\lfloor s \rfloor := \bigcap \{c \in \mathcal{H} \mid s \subseteq c\}$. We write $\lfloor A \rfloor$ for $\lfloor \{A\} \rfloor$.

Lower values can be shown to be equal to $\lfloor s \rfloor = \bigcap \{\lceil \Gamma \vdash A \rceil \in \mathcal{H} \mid s \subseteq \lceil \Gamma \vdash A \rceil\}$. The rationale behind $\lfloor A \rfloor$ is the following:

$\Gamma \in \lfloor A \rfloor$ if and only if $\Delta, A \vdash^{\uparrow} B$ entails $\Delta, \Gamma \vdash^{\uparrow} B$.

Lower values have strong links with the characterization of neutral proofs.

Lemma 6.30. *If $\Gamma \vdash^{\downarrow} A$, then $\Gamma \in \lfloor A \rfloor$.*

Proof. Suppose $\Gamma \vdash^{\downarrow} A$ and $\Delta, A \vdash^{\uparrow} B$. We derive $\Delta, \Gamma \vdash^{\uparrow} B$ using Lemma 6.22. Therefore we get $\Gamma \in \lfloor A \rfloor$. □

The following property characterizes the lower and upper values for the connectives and quantifiers. It will be of crucial importance, as identified by [Oka02, DL05]. Note that the inequalities for implication mix lower and upper values. More generally, for any formula A, we have the inequality $\lfloor A \rfloor \leq \lceil A \rceil$.

Proposition 6.31: Main Property

The lower and upper values respect the following constraints:

$$\begin{array}{rcl@{\qquad}rcl}
\lfloor \bot \rfloor & = & \bot_{\mathcal{H}} & & & \\
\lfloor A \Rightarrow B \rfloor & \leq & \lceil A \rceil \to \lfloor B \rfloor & \lfloor A \rfloor \to \lceil B \rceil & \leq & \lceil A \Rightarrow B \rceil \\
\lfloor \forall_\tau\ P \rfloor & \leq & \bigwedge_{t \in \Lambda_\tau} \lfloor P\ t \rfloor & \bigwedge_{t \in \Lambda_\tau} \lfloor P\ t \rfloor & \leq & \lceil \forall_\tau\ P \rceil \\
\lfloor \exists_\tau\ P \rfloor & \leq & \bigvee_{t \in \Lambda_\tau} \lfloor P\ t \rfloor & \bigvee_{t \in \Lambda_\tau} \lceil P\ t \rceil & \leq & \lceil \exists_\tau\ P \rceil \\
\lfloor \bigwedge_{\mathcal{I}} F \rfloor & \leq & \bigwedge_{i \in \mathcal{I}} \lfloor \pi_i\ F \rfloor & \bigwedge_{i \in \mathcal{I}} \lceil \pi_i\ F \rceil & \leq & \lceil \bigwedge_{\mathcal{I}} F \rceil \\
\lfloor \bigvee_{\mathcal{I}} F \rfloor & \leq & \bigvee_{i \in \mathcal{I}} \lfloor \pi_i\ F \rfloor & \bigvee_{i \in \mathcal{I}} \lceil \pi_i\ F \rceil & \leq & \lceil \bigvee_{\mathcal{I}} F \rceil
\end{array}$$

Proof. Each case involves the corresponding cut-free natural deduction rule. The cases for the infinitary connectives are the added value of our work, compared to [HL08a]. Those four cases also faithfully represent the cases for the quantifiers.

— Implication (lower): Let us show $\lfloor A \Rightarrow B \rfloor \wedge \lceil A \rceil \leq \lfloor B \rfloor$. Let $\Gamma \in \lfloor A \Rightarrow B \rfloor \wedge \lceil A \rceil$. Suppose $\Delta, B \vdash^{\uparrow} C$. Combining the proof

$$\dfrac{\dfrac{\dfrac{}{\Gamma, A \Rightarrow B \vdash^{\downarrow} A \Rightarrow B}\ \text{Ax} \qquad \dfrac{\Gamma \vdash^{\uparrow} A}{\Gamma, A \Rightarrow B \vdash^{\uparrow} A}\ \text{Weak}}{\Gamma, A \Rightarrow B \vdash^{\downarrow} B}\ \text{Imp-E} \qquad \Delta, B \vdash^{\uparrow} C}{\Delta, \Gamma, A \Rightarrow B \vdash^{\uparrow} C}\ \text{Ax-R}$$

and the definition of $\Gamma \in \lfloor A \Rightarrow B \rfloor$, we get $\Delta, \Gamma, \Gamma \vdash^{\uparrow} C$, and thus $\Delta, \Gamma \vdash^{\uparrow} C$ by contraction. Therefore $\Gamma \in \lfloor B \rfloor$.

— Implication (upper): Let $\Gamma \in d$ such that $d \leq \lfloor A \rfloor \to \lceil B \rceil$. We have $\Gamma, A \in \lfloor A \rfloor$ by weakening of $A \in \lfloor A \rfloor$ (obtained using Lemma 6.30), and $\Gamma, A \in d$ by weakening of $\Gamma \in d$. Hence $\Gamma, A \in d \wedge \lfloor A \rfloor$. Using the implication property, we have $d \wedge \lfloor A \rfloor \leq \lceil B \rceil$, and therefore $\Gamma, A \in \lceil B \rceil$. Using the proof

$$\dfrac{\Gamma, A \vdash^{\uparrow} B}{\Gamma \vdash^{\uparrow} A \Rightarrow B}\ \text{Imp-I}$$

we get $\Gamma \in \lceil A \Rightarrow B \rceil$.

— Infinite conjunction (lower): Let $\Gamma \in \lfloor \bigwedge_{\mathcal{I}} F \rfloor$ and $j \in \mathcal{I}$. Suppose $\Delta, \pi_j\ F \vdash^{\uparrow} B$. Combining the proof

$$\dfrac{\dfrac{\dfrac{}{\bigwedge_{\mathcal{I}} F \vdash^{\downarrow} \bigwedge_{\mathcal{I}} F}\ \text{Ax}}{\bigwedge_{\mathcal{I}} F \vdash^{\downarrow} \pi_j\ F}\ \text{AndInf-E} \qquad \Delta, \pi_j\ F \vdash^{\uparrow} B}{\Delta, \bigwedge_{\mathcal{I}} F \vdash^{\uparrow} B}\ \text{Ax-R}$$

and the definition of $\Gamma \in \lfloor \bigwedge_{\mathcal{I}} F \rfloor$, we get $\Delta, \Gamma \vdash^{\uparrow} B$. It follows that $\Gamma \in \lfloor \pi_j\ F \rfloor$ for every $j \in \mathcal{I}$, and thus $\lfloor \bigwedge_{\mathcal{I}} F \rfloor \leq \lfloor \pi_j\ F \rfloor$. Therefore $\lfloor \bigwedge_{\mathcal{I}} F \rfloor \leq \bigwedge_{i \in \mathcal{I}} \lfloor \pi_i\ F \rfloor$.

— Infinite conjunction (upper): Let $\Gamma \in \bigwedge_{i \in \mathcal{I}} \lceil \pi_i\ F \rceil$. Using the proof

$$\dfrac{\Gamma \vdash^{\uparrow} \pi_i\ F \text{ for every } i \in \mathcal{I}}{\Gamma \vdash^{\uparrow} \bigwedge_{\mathcal{I}} F}\ \text{AndInf-I}$$

we get $\Gamma \in \lceil \bigwedge_{\mathcal{I}} F \rceil$.

— Infinite disjunction (lower): Let $\Gamma \in \lfloor \bigvee_{\mathcal{I}} F \rfloor$. Let $\bigcap_{k \in K} \lceil \Delta_k \vdash B_k \rceil$ be an upper bound of all the $\lfloor \pi_i\ F \rfloor$. As $\pi_i\ F \in \lfloor \pi_i\ F \rfloor$, we have $\pi_i\ F \in \lceil \Delta_k \vdash B_k \rceil$ for any $k \in K$. Combining the proof

$$\dfrac{\dfrac{}{\Delta_k, \bigvee_{\mathcal{I}} F \vdash^{\downarrow} \bigvee_{\mathcal{I}} F}\ \text{Ax} \qquad \dfrac{\Delta_k, \pi_i\ F \vdash^{\uparrow} B_k \text{ for every } i \in \mathcal{I}}{\Delta_k, \bigvee_{\mathcal{I}} F, \pi_i\ F \vdash^{\uparrow} B_k \text{ for every } i \in \mathcal{I}}\ \text{Weakening}}{\Delta_k, \bigvee_{\mathcal{I}} F \vdash^{\uparrow} B_k}\ \text{OrInf-E}$$

and $\Gamma \in \lfloor \bigvee_{\mathcal{I}} F \rfloor$, we get $\Delta_k, \Gamma \vdash^{\uparrow} B_k$. It follows that $\Gamma \in \bigcap_{k \in K} \lceil \Delta_k \vdash B_k \rceil$, and therefore $\lfloor \bigvee_{\mathcal{I}} F \rfloor \leq \bigcap_{k \in K} \lceil \Delta_k \vdash B_k \rceil$. As $\bigvee_{i \in \mathcal{I}} \lfloor \pi_i\ F \rfloor$ is an upper bound of all the $\lfloor \pi_i\ F \rfloor$, we get $\lfloor \bigvee_{\mathcal{I}} F \rfloor \leq \bigvee_{i \in \mathcal{I}} \lfloor \pi_i\ F \rfloor$.

— Infinite disjunction (upper): Let $j \in \mathcal{I}$ and $\Gamma \in \lceil \pi_j\ F \rceil$. Using the proof

$$\dfrac{\Gamma \vdash^{\uparrow} \pi_j\ F}{\Gamma \vdash^{\uparrow} \bigvee_{\mathcal{I}} F}\ \text{OrInf-I}$$

we get $\Gamma \in \lceil \bigvee_{\mathcal{I}} F \rceil$. Therefore $\lceil \pi_j\ F \rceil \leq \lceil \bigvee_{\mathcal{I}} F \rceil$ for every $j \in \mathcal{I}$. It follows that $\bigvee_{i \in \mathcal{I}} \lceil \pi_i\ F \rceil \leq \lceil \bigvee_{\mathcal{I}} F \rceil$. □

6.3.4 The Cut-Free-Provability Model

To build the cut-free-provability model, we first define the domains M_τ. As our logic is intensional, it is important to have $\mathsf{M}_o \neq \mathcal{H}$. This calls for the so-called V-complex construction, defined by Takahashi and Prawitz to show cut admissibility of classical higher-order logic [Pra68, Tak67, And71]. The core of this construction is an entanglement of a syntactic and a semantic value.

Definition 6.32: V-Complexes

We define the domain M_τ by induction on τ, as sets of pairs of the form $\langle [t], \nu \rangle$, where t is a term of type τ and $[t]$ is its $\equiv$-equivalence class, and the following conditions hold on ν:

— for type ι, ν is a fixed constant, traditionally named ι,

— for type o, $\nu \in \mathcal{H}$ and $\lfloor t \rfloor \leq \nu \leq \lceil t \rceil$,

— for type $\tau \to \sigma$, ν is a function of type $\mathsf{M}_\tau \to \mathsf{M}_\sigma$ such that, for any $\langle [u], \nu_u \rangle \in \mathsf{M}_\tau$, there exists some ν' such that $\nu(\langle [u], \nu_u \rangle) = \langle [t\ u], \nu' \rangle \in \mathsf{M}_\sigma$,

— for type $o_\mathcal{I}$, ν is a function of type $\mathcal{I} \to \mathsf{M}_o$ such that, for any $i \in \mathcal{I}$, there exists some ν_i such that $\nu(i) = \langle [\pi_i\ t], \nu_i \rangle \in \mathsf{M}_o$.

We need to ensure that the domains M_τ are well-formed. We first check that the definition does not depend on the chosen representatives of the $\equiv$-equivalence classes:

— if $[t] = [t']$ then $\lfloor t \rfloor = \lfloor t' \rfloor$ and $\lceil t \rceil = \lceil t' \rceil$ (because of the Conv rule),

— if $t' \in [t]$ and $u' \in [u]$ then $t'\ u' \equiv t\ u$ and $[t'\ u'] = [t\ u]$,

— if $t' \in [t]$ then $[\pi_i\ t'] = [\pi_i\ t]$ for every $i \in \mathcal{I}$.

We also check the non-emptiness of these domains, thanks to the following lemma.

Lemma 6.33 (Canonical Representatives)**.** *For any term t of type τ, there exists $\rho(t)$ such that $\langle [t], \rho(t) \rangle \in \mathsf{M}_\tau$.*

Proof. The definition of $\rho(\_)$ is done by induction on the type τ. The verification that it meets the requirements is performed altogether and is immediate, given that all the manipulated structures are invariant under $\equiv$. We let $\rho(t_\iota) = \iota$, $\rho(t_o) = \lceil t_o \rceil$[1], $\rho(t_{\tau\to\sigma})$ be the function $\langle [u_\tau], \nu_u \rangle \mapsto \langle [t\ u], \rho(t\ u) \rangle$, and $\rho(t_{o_\mathcal{I}})$ be the function $i \mapsto \langle [\pi_i\ t], \rho(\pi_i\ t) \rangle$. □

We now define the interpretation of constants. We write $\|\_\|_1$ (respectively $\|\_\|_2$) for the projection of a member of M_τ on its first (respectively second) component. Following the definition of the domains, $\|\_\|_1$ returns an equivalence class. Since all our constructions are invariant under $\equiv$, it is sound, for any $d \in \mathsf{M}_\tau$, to define the term $\|d\|_r$ as *a* representative of the equivalence class $\|d\|_1$.

[1] The choice of $\lceil \_ \rceil$ is arbitrary and represents, in certain circumstances, a valuable degree of freedom.

Definition 6.34: Interpretation of Constants

We let:

$$\begin{array}{lcl}
\mathsf{Const}(\bot) & := & \langle [\bot], \bot_{\mathcal{H}} \rangle \\
\mathsf{Const}(\Rightarrow) & := & \langle [\Rightarrow], \langle [A], a \rangle \mapsto \langle [\Rightarrow A], \langle [B], b \rangle \mapsto \langle [A \Rightarrow B], a \to b \rangle \rangle \rangle \\
\mathsf{Const}(\forall_\tau) & := & \langle [\forall_\tau], \langle [P], p \rangle \mapsto \langle [\forall\ P], \bigwedge_{d \in \mathsf{M}_\tau} \|p(d)\|_2 \rangle \rangle \\
\mathsf{Const}(\exists_\tau) & := & \langle [\exists_\tau], \langle [P], p \rangle \mapsto \langle [\exists\ P], \bigvee_{d \in \mathsf{M}_\tau} \|p(d)\|_2 \rangle \rangle \\
\mathsf{Const}(\bigwedge_{\mathcal{I}}) & := & \langle [\bigwedge_{\mathcal{I}}], \langle [F], f \rangle \mapsto \langle [\bigwedge_{\mathcal{I}} F], \bigwedge_{i \in \mathcal{I}} \|f(i)\|_2 \rangle \rangle \\
\mathsf{Const}(\bigvee_{\mathcal{I}}) & := & \langle [\bigvee_{\mathcal{I}}], \langle [F], f \rangle \mapsto \langle [\bigvee_{\mathcal{I}} F], \bigvee_{i \in \mathcal{I}} \|f(i)\|_2 \rangle \rangle \\
\mathsf{Const}(c) & := & \langle [c], \rho(c) \rangle \text{ for non-logical constants}
\end{array}$$

Lemma 6.35. *The interpretation of constants are members of their respective domains.*

Proof. For non-logical constants, the claim holds by Lemma 6.33. For logical constants, proving the claim boils down to checking that the end truth value respects the conditions of being a member of M_o, that imposes the truth value to be bounded by the lower and upper values. For this, we use Proposition 6.31.

— Contradiction: We directly have $\lfloor \bot \rfloor = \bot_{\mathcal{H}}$.

— Implication: By definition, $\lfloor A \rfloor \leq a \leq \lceil A \rceil$ and $\lfloor B \rfloor \leq b \leq \lceil B \rceil$. Remember that $\to$ is contravariant in its left argument (if $c \leq c'$, then $c' \to d \leq c \to d$). Therefore we have:

$$\lfloor A \Rightarrow B \rfloor \leq \lceil A \rceil \to \lfloor B \rfloor \leq a \to b \leq \lfloor A \rfloor \to \lceil B \rceil \leq \lceil A \to B \rceil$$

— Quantifiers: By definition of $\langle [P], p \rangle \in \mathsf{M}_{\tau \to o}$, we have $p(d) \in \mathsf{M}_o$ for any $d \in \mathsf{M}_\tau$, that is to say $\lfloor P\ \|d\|_r \rfloor \leq \|p(d)\|_2 \leq \lceil P\ \|d\|_r \rceil$. By Lemma 6.33, all $t \in \Lambda_\tau$ are represented in M_τ by at least $\langle [t], \rho(t) \rangle$, and lower and upper values are stable by $\equiv$. Thus:

$$\begin{array}{lcl}
\lfloor \forall_\tau\ P \rfloor \leq \bigwedge_{t \in \Lambda_\tau} \lfloor P\ t \rfloor & = & \bigwedge_{d \in \mathsf{M}_\tau} \lfloor P\ \|d\|_r \rfloor \\
& \leq & \bigwedge_{d \in \mathsf{M}_\tau} \|p(d)\|_2 \\
& \leq & \bigwedge_{d \in \mathsf{M}_\tau} \lceil P\ \|d\|_r \rceil = \bigwedge_{t \in \Lambda_\tau} \lceil P\ t \rceil \leq \lceil \forall_\tau\ P \rceil \\
\lfloor \exists_\tau\ P \rfloor \leq \bigvee_{t \in \Lambda_\tau} \lfloor P\ t \rfloor & = & \bigvee_{d \in \mathsf{M}_\tau} \lfloor P\ \|d\|_r \rfloor \\
& \leq & \bigvee_{d \in \mathsf{M}_\tau} \|p(d)\|_2 \\
& \leq & \bigvee_{d \in \mathsf{M}_\tau} \lceil P\ \|d\|_r \rceil = \bigvee_{t \in \Lambda_\tau} \lceil P\ t \rceil \leq \lceil \exists_\tau\ P \rceil
\end{array}$$

— Infinitary connectives: By definition of $\langle [F], f \rangle \in \mathsf{M}_{o_{\mathcal{I}}}$, and for any $i \in \mathcal{I}$, we have $\lfloor \pi_i\ F \rfloor \leq \|f(i)\|_2 \leq \lceil \pi_i\ F \rceil$. Therefore:

$$\lfloor \bigvee_{\mathcal{I}} F \rfloor \leq \bigvee_{i \in \mathcal{I}} \lfloor \pi_i\ F \rfloor \leq \bigvee_{i \in \mathcal{I}} \|f(i)\|_2 \leq \bigvee_{i \in \mathcal{I}} \lceil \pi_i\ F \rceil \leq \lceil \bigvee_{\mathcal{I}} F \rceil$$

$$\lfloor \bigwedge_{\mathcal{I}} F \rfloor \leq \bigwedge_{i \in \mathcal{I}} \lfloor \pi_i\ F \rfloor \leq \bigwedge_{i \in \mathcal{I}} \|f(i)\|_2 \leq \bigwedge_{i \in \mathcal{I}} \lceil \pi_i\ F \rceil \leq \lceil \bigwedge_{\mathcal{I}} F \rceil$$

□

We build the Heyting applicative structure by defining ω as the second projection $\|\_\|_2$, and App so that $\mathsf{App}(m, d) = \|m\|_2(d)$ and $\mathsf{App}(m', i) = \|m'\|_2(i)$.

Definition 6.36

We let:

$$\begin{array}{lcl}
\omega(\langle [_], \nu \rangle) & := & \nu \\
\mathsf{App}(\langle [_], \nu \rangle, d) & := & \nu(d) \\
\mathsf{App}(\langle [_], \nu \rangle, i) & := & \nu(i)
\end{array}$$

Lemma 6.37. *$\langle \mathsf{M}, \mathsf{App}, \mathsf{Const}, \omega, \mathcal{H}\rangle$ is a Heyting applicative structure.*

Proof. The conditions for being a Heyting applicative structure are immediate using the definitions of Const, App and ω. We only highlight a few cases.

$$\begin{aligned}
\omega(\mathsf{App}(\mathsf{App}(\mathsf{Const}(\Rightarrow), d_1), d_2)) &= \omega(\langle [\|d_1\|_r \Rightarrow \|d_2\|_r], \|d_1\|_2 \to \|d_2\|_2\rangle)\\
&= \|d_1\|_2 \to \|d_2\|_2\\
&= \omega(d_1) \to \omega(d_2)
\end{aligned}$$

$$\begin{aligned}
\omega(\mathsf{App}(\mathsf{Const}(\exists_\tau), m)) &= \omega(\langle [\exists_\tau \|m\|_r], \textstyle\bigvee_{d\in\mathsf{M}_\tau} \|\|m\|_2(d)\|_2\rangle)\\
&= \textstyle\bigvee_{d\in\mathsf{M}_\tau} \|\|m\|_2(d)\|_2\\
&= \textstyle\bigvee_{d\in\mathsf{M}_\tau} \omega(\mathsf{App}(m, d))
\end{aligned}$$

$$\begin{aligned}
\omega(\mathsf{App}(\mathsf{Const}(\textstyle\bigwedge_{\mathcal{I}}), m')) &= \omega(\langle [\textstyle\bigwedge_{\mathcal{I}} \|m'\|_r], \textstyle\bigwedge_{i\in\mathcal{I}} \|\|m'\|_2(i)\|_2\rangle)\\
&= \textstyle\bigwedge_{i\in\mathcal{I}} \|\|m'\|_2(i)\|_2\\
&= \textstyle\bigwedge_{i\in I} \omega(\mathsf{App}(m', i))
\end{aligned}$$

□

For any assignment φ, we let $\|\varphi\|_r$ be any substitution that maps the variables x of the support of φ to $\|\varphi(x)\|_r$. Note that $\|\varphi\|_r$ is not uniquely determined because each $\equiv$-equivalence class has many representatives.

Proposition 6.38

There exists a unique function $[\![\_]\!]\_$ that satisfies all the conditions of Definition 6.24, except the last one, and such that $\|[\![t]\!]_\varphi\|_1 = [t\|\varphi\|_r]$ for any term t with variables in the support of φ.

Proof. We build $[\![\_]\!]\_$ by induction on t as the unique function that satisfies the conditions of Definition 6.24, except the last one, and such that $\|[\![t]\!]_\varphi\|_1 = [t\|\varphi\|_r]$. The cases for constants and variables are immediate.

— Application: By induction hypotheses, $[\![t]\!]_\varphi \in \mathsf{M}_{\tau\to\sigma}$ and $[\![u]\!]_\varphi \in \mathsf{M}_\tau$ are uniquely defined, and we have $\|[\![t]\!]_\varphi\|_1 = [t\|\varphi\|_r]$ and $\|[\![u]\!]_\varphi\|_1 = [u\|\varphi\|_r]$. As required by Definition 6.24, we set $[\![t\ u]\!]_\varphi := \mathsf{App}([\![t]\!]_\varphi, [\![u]\!]_\varphi) \in \mathsf{M}_\sigma$. By definition of App and $\mathsf{M}_{\tau\to\sigma}$, we have $\|[\![t\ u]\!]_\varphi\|_1 = \|\mathsf{App}([\![t]\!]_\varphi, [\![u]\!]_\varphi)\|_1 = [t\|\varphi\|_r\ u\|\varphi\|_r] = [(t\ u)\|\varphi\|_r]$.

— Abstraction: By definition of $\mathsf{M}_{\tau\to\sigma}$, $[\![\lambda x.\ t]\!]_\varphi$ should be of the form $\langle [w], \langle [u], \nu_u\rangle \mapsto \langle [w\ u], \nu_{uw}\rangle\rangle$, with $\langle [w\ u], \nu_{uw}\rangle \in \mathsf{M}_\sigma$ for every $\langle [u], \nu_u\rangle \in \mathsf{M}_\tau$. The condition $\|[\![\lambda x.\ t]\!]_\varphi\|_1 = [(\lambda x.\ t)\|\varphi\|_r]$ imposes

$$[w] = [(\lambda x.\ t)\|\varphi\|_r] = [\lambda x.\ t\|\varphi\|_r]$$

For every $d = \langle [u], \nu_u\rangle \in \mathsf{M}_\tau$, Definition 6.24 requires $\mathsf{App}([\![\lambda x.\ t]\!]_\varphi, d) = [\![t]\!]_{\varphi[d/x]}$, that is $\langle [w\ u], \nu_{uw}\rangle = [\![t]\!]_{\varphi[d/x]}$. This imposes $\nu_{uw} := \|[\![t]\!]_{\varphi[d/x]}\|_2$ and is enforced because

$$[w\ u] = [(\lambda x.\ t\|\varphi\|_r)\ u] = [t\|\varphi\|_r[x \leftarrow u]] = [t\|\varphi[d/x]\|_r] = \|[\![t]\!]_{\varphi[d/x]}\|_1$$

using the induction hypothesis. As $\langle [w\ u], \nu_{uw}\rangle = [\![t]\!]_{\varphi[d/x]}$, we directly have $\langle [w\ u], \nu_{uw}\rangle \in \mathsf{M}_\sigma$.

— Projection: By induction hypothesis, $\|[\![F]\!]_\varphi\|_1 = [F\|\varphi\|_r]$. By definition of $\mathsf{M}_{o_\mathcal{I}}$, it follows that $[\![F]\!]_\varphi = \langle [F\|\varphi\|_r], i \mapsto \langle [\pi_i\ F\|\varphi\|_r], \nu_i\rangle\rangle$, with $\langle [\pi_i\ F\|\varphi\|_r], \nu_i\rangle \in \mathsf{M}_o$. As required by Definition 6.24, we set $[\![\pi_j\ F]\!]_\varphi := \mathsf{App}([\![F]\!]_\varphi, j) = \langle [\pi_j\ F\|\varphi\|_r], \nu_j\rangle$. Therefore we have $[\![\pi_j\ F]\!]_\varphi \in \mathsf{M}_o$ and $\|[\![\pi_j\ F]\!]_\varphi\|_1 = [\pi_j\ F\|\varphi\|_r] = [(\pi_j\ F)\|\varphi\|_r]$.

— Family: By definition of $\mathsf{M}_{o_\mathcal{I}}$, $[\![\{i \mapsto A_i\}_\mathcal{I}]\!]_\varphi$ should be of the form $\langle [t], i \mapsto \langle [\pi_i\ t], \nu_i\rangle\rangle$, with $\langle [\pi_i\ t], \nu_i\rangle \in \mathsf{M}_o$ for every $i \in \mathcal{I}$. The condition $\|[\![\{i \mapsto A_i\}_\mathcal{I}]\!]_\varphi\|_1 = [\{i \mapsto A_i\}_\mathcal{I}\|\varphi\|_r]$ imposes

$$[t] = [\{i \mapsto A_i\}_\mathcal{I}\|\varphi\|_r] = [\{i \mapsto A_i\|\varphi\|_r\}_\mathcal{I}]$$

For every $j \in \mathcal{I}$, Definition 6.24 requires $\mathsf{App}([\![\{i \mapsto A_i\}_\mathcal{I}]\!]_\varphi, j) = [\![A_j]\!]_\varphi$, that is $\langle [\pi_j\ t], \nu_j\rangle = [\![A_j]\!]_\varphi$. This imposes $\nu_j := \|[\![A_j]\!]_\varphi\|_2$ and is enforced because

$$[\pi_j\ t] = [\pi_j\ \{i \mapsto A_i\|\varphi\|_r\}_\mathcal{I}] = [A_j\|\varphi\|_r] = \|[\![A_j]\!]_\varphi\|_1$$

using the induction hypothesis. □

Lemma 6.39 (Substitution)**.** *We have* $[\![t[x \leftarrow u]]\!]_\varphi = [\![t]\!]_{\varphi[x/[\![u]\!]_\varphi]}$.

Proof. By induction on t. The cases for constants and variables are immediate. The case for application follows from the induction hypotheses. The case for abstraction is similar to the case for family of propositions.

— Projection: We have $[\![(\pi_j\ F)[x \leftarrow u]]\!]_\varphi = [\![\pi_j\ F[x \leftarrow u]]\!]_\varphi = \mathsf{App}([\![F[x \leftarrow u]]\!]_\varphi, j)$. By induction hypothesis, we have $[\![F[x \leftarrow u]]\!]_\varphi = [\![F]\!]_{\varphi[x/[\![u]\!]_\varphi]}$. It follows that $[\![(\pi_j\ F)[x \leftarrow u]]\!]_\varphi = \mathsf{App}([\![F]\!]_{\varphi[x/[\![u]\!]_\varphi]}, j) = [\![\pi_j\ F]\!]_{\varphi[x/[\![u]\!]_\varphi]}$.

— Family: By definition, we have $\mathsf{App}([\![\{i \mapsto A_i[x \leftarrow u]\}_\mathcal{I}]\!]_\varphi, j) = [\![A_j[x \leftarrow u]]\!]_\varphi$ and $\mathsf{App}([\![\{i \mapsto A_i\}_\mathcal{I}]\!]_{\varphi[[\![u]\!]_\varphi/x]}, j) = [\![A_j]\!]_{\varphi[[\![u]\!]_\varphi/x]}$. Using the induction hypothesis, it follows that $\mathsf{App}([\![\{i \mapsto A_i[x \leftarrow u]\}_\mathcal{I}]\!]_\varphi, j) = \mathsf{App}([\![\{i \mapsto A_i\}_\mathcal{I}]\!]_{\varphi[[\![u]\!]_\varphi/x]}, j)$. This entails $\|[\![\{i \mapsto A_i[x \leftarrow u]\}_\mathcal{I}]\!]_\varphi\|_2 = \|[\![\{i \mapsto A_i\}_\mathcal{I}]\!]_{\varphi[[\![u]\!]_\varphi/x]}\|_2$ by definition of App and $\mathsf{M}_{o_\mathcal{I}}$. Moreover, by Proposition 6.38, we have $\|[\![\{i \mapsto A_i[x \leftarrow u]\}_\mathcal{I}]\!]_\varphi\|_1 = [\{i \mapsto A_i[x \leftarrow u]\}_\mathcal{I}\|\varphi\|_r] = [\{i \mapsto A_i\}_\mathcal{I}\|\varphi[[\![u]\!]_\varphi/x]\|_r] = \|[\![\{i \mapsto A_i\}_\mathcal{I}]\!]_{\varphi[[\![u]\!]_\varphi/x]}\|_1$. We conclude $[\![\{i \mapsto A_i[x \leftarrow u]\}_\mathcal{I}]\!]_\varphi = [\![\{i \mapsto A_i\}_\mathcal{I}]\!]_{\varphi[[\![u]\!]_\varphi/x]}$. □

Proposition 6.40: Model

$[\![\_]\!]\_$ is a model.

Proof. It only remains to show that $[\![t]\!]_\varphi = [\![u]\!]_\varphi$ when $t \equiv u$. We show it by induction on the derivation of $t \equiv u$.

— We have $[\![(\lambda x.\ t)\ u]\!]_\varphi = \mathsf{App}([\![\lambda x.\ t]\!]_\varphi, [\![u]\!]_\varphi) = [\![t]\!]_{\varphi[x/[\![u]\!]_\varphi]}$. Using Lemma 6.39, we derive $[\![(\lambda x.\ t)\ u]\!]_\varphi = [\![t[x \leftarrow u]]\!]_\varphi$.

— We directly have $[\![\pi_j\ \{i \mapsto A_i\}_\mathcal{I}]\!]_\varphi = \mathsf{App}([\![\{i \mapsto A_i\}_\mathcal{I}]\!]_\varphi, j) = [\![A_j]\!]_\varphi$.

— We have $\|[\![\lambda x.\ t\ x]\!]_\varphi\|_1 = [(\lambda x.\ t\ x)\|\varphi\|_r] = [\lambda x.\ t\|\varphi\|_r\ x] = [t\|\varphi\|_r] = \|[\![t]\!]_\varphi\|_1$ by Proposition 6.38. Moreover, for any $d \in \mathsf{M}_\tau$, we have

$$\mathsf{App}([\![\lambda x.\ t\ x]\!]_\varphi, d) = [\![t\ x]\!]_{\varphi[d/x]} = \mathsf{App}([\![t]\!]_{\varphi[d/x]}, [\![x]\!]_{\varphi[d/x]}) = \mathsf{App}([\![t]\!]_\varphi, d)$$

By definition of App and $\mathsf{M}_{\tau\to\sigma}$, this entails $\|[\![\lambda x.\ t\ x]\!]_\varphi\|_2 = \|[\![t]\!]_\varphi\|_2$. Therefore we conclude $[\![\lambda x.\ t\ x]\!]_\varphi = [\![t]\!]_\varphi$.

— We have $\|\{i \mapsto \pi_i \ F\}_{\mathcal{I}}\|_1 = [\{i \mapsto \pi_i \ F\}_{\mathcal{I}} \|\varphi\|_r] = [F\|\varphi\|_r] = \|[\![F]\!]_\varphi\|_1$ by Proposition 6.38. Moreover, for any $j \in \mathcal{I}$, we have

$$\mathsf{App}([\![\{i \mapsto \pi_i \ F\}_{\mathcal{I}}]\!]_\varphi, j) = [\![\pi_j \ F]\!]_\varphi = \mathsf{App}([\![F]\!]_\varphi, j)$$

By definition of App and $\mathsf{M}_{o_{\mathcal{I}}}$, this entails $\|[\![\{i \mapsto \pi_i \ F\}_{\mathcal{I}}]\!]_\varphi\|_2 = \|[\![F]\!]_\varphi\|_2$. Therefore we conclude $[\![\{i \mapsto \pi_i \ F\}_{\mathcal{I}}]\!]_\varphi = [\![F]\!]_\varphi$.

— The other cases follow from the induction hypotheses. □

We use this specific model to show the strong completeness result.

Theorem 6.41: Strong Completeness

If $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![A]\!]_\varphi)$ for any model, then $\Gamma \vdash^{\uparrow} A$.

Proof. We use the interpretation in the Heyting applicative structure above with the assignment $\varphi = [x \leftarrow \langle [x], \rho(x) \rangle]$. Let $\Gamma = A_1, \ldots, A_n$. We know that $[\![A_i]\!]_\varphi \in \mathsf{M}_o$. Its first member is $\|[\![A_i]\!]_\varphi\|_1 = [A_i]$ by Proposition 6.38 and by construction of φ. Its second member is $\omega([\![A_i]\!]_\varphi)$. Therefore $\lfloor A_i \rfloor \leq \omega([\![A_i]\!]_\varphi)$. As $\Gamma \in \lfloor A_i \rfloor$, we have $\Gamma \in \omega([\![A_i]\!]_\varphi)$. Thus $\Gamma \in \omega([\![\Gamma]\!]_\varphi)$. Similarly, $[\![A]\!]_\varphi \in \mathsf{M}_o$, so $\omega([\![A]\!]_\varphi) \leq \lceil A \rceil$. By hypothesis, we derive $\Gamma \in \lceil A \rceil$. We conclude $\Gamma \vdash^{\uparrow} A$. □

The cut-elimination theorem is now easy to derive.

Theorem 6.42: Cut Elimination

If $\Gamma \vdash A$ then $\Gamma \vdash^{\uparrow} A$.

Proof. Immediate combination of Theorems 6.25 and 6.41. □

Remark 6.43. *This cut-elimination result for natural deduction could be transferred to sequent calculus, using a translation from cut-free natural deduction to cut-free sequent calculus [DW03].*

6.3.5 Models for the Classical Variant

The definitions, constructions and results developed in Section 6.3 for the intuitionistic variant of higher-order infinitary logic also apply to the classical variant.

The notion of model we have established for the intuitionistic variant, based on Heyting algebras, also applies to the classical variant, provided that we restrict the definition to Boolean algebras. We recall that a Heyting algebra is a Boolean algebra when $a \vee (a \to \bot_{\mathcal{H}}) = \top_{\mathcal{H}}$ for any $a \in \mathcal{H}$.

Soundness. The soundness result (Theorem 6.25) hold for the classical variant of higher-order infinitary logic. We modify the proof by adding the PEM case. We have $\omega([\![A \vee \neg A]\!]_\varphi) = \omega([\![A]\!]_\varphi) \vee \omega([\![\neg A]\!]_\varphi) = \omega([\![A]\!]_\varphi) \vee (\omega([\![A]\!]_\varphi) \to \bot_{\mathcal{H}})$. As $\mathcal{H}$ is Boolean, we have $\omega([\![A]\!]_\varphi) \vee (\omega([\![A]\!]_\varphi) \to \bot_{\mathcal{H}}) = \top_{\mathcal{H}}$, and therefore $\omega([\![\Gamma]\!]_\varphi) \leq \omega([\![A \vee \neg A]\!]_\varphi)$.

Strong Completeness and Cut Elimination. The cut-free-provability Heyting algebra we have built in Definition 6.27 is also a Boolean algebra, provided that the provability is classical, i.e., provided that we are in classical logic. To show this, it is key to characterize $a \to \bot_{\mathcal{H}}$ following [Mae91].

Proposition 6.44

For any $a \in \mathcal{H}$, we have $a \to \bot_{\mathcal{H}} = \{\Gamma \mid a \leq \lceil \Gamma \vdash \bot \rceil\}$.

The proof relies on the following intermediate lemma, which states that the Bot-E rule is admissible when using $\vdash^{\uparrow}$ in the premise and conclusion.

Lemma 6.45. *If $\Gamma \vdash^{\uparrow} \bot$, then $\Gamma \vdash^{\uparrow} A$ for any A.*

Proof. We prove a stronger result: if $\Gamma \vdash^{\uparrow} B$ with $B \equiv \bot$, then $\Gamma \vdash^{\uparrow} A$ for any A. We proceed by induction on the derivation.

— Conv: Suppose $\Gamma \vdash^{\uparrow} C$ with $C \equiv B$. As $B \equiv \bot$, we have $C \equiv \bot$. By induction hypothesis, we have $\Gamma \vdash^{\uparrow} A$.

— Coerce: Suppose $\Gamma \vdash^{\downarrow} B$ with $B \equiv \bot$. We have $\Gamma \vdash^{\downarrow} \bot$ using Conv. Using Bot-E, we get $\Gamma \vdash^{\uparrow} A$.

— OrInf-E: Suppose $\Gamma \vdash^{\downarrow} \bigvee_{\mathcal{I}} F$ and $\Gamma, \pi_i\ F \vdash^{\uparrow} C$ for every $i \in \mathcal{I}$, with $C \equiv \bot$. By induction hypotheses, we have $\Gamma, \pi_i\ F \vdash^{\uparrow} A$ for every $i \in \mathcal{I}$. Using OrInf-E, we derive $\Gamma \vdash^{\uparrow} A$.

— Ex-E: Suppose $\Gamma \vdash^{\downarrow} \exists\ P$ and $\Gamma, P\ x \vdash^{\uparrow} C$ with $C \equiv \bot$. By induction hypothesis, we have $\Gamma, P\ x \vdash^{\uparrow} A$. Using Ex-E, we derive $\Gamma \vdash^{\uparrow} A$.

— Bot-E: Suppose $\Gamma \vdash^{\downarrow} \bot$. Using Bot-E, we get $\Gamma \vdash^{\uparrow} A$.

— The cases Ax, Imp-E, AndInf-E, All-E and PEM do not apply as the conclusion of these rules uses $\vdash^{\downarrow}$. The cases Imp-I, AndInf-I, OrInf-I, All-I and Ex-I do not apply as the formula in the conclusion of these rules cannot be convertible to $\bot$. □

Proof of Proposition 6.44. Let us show $a \to \bot_{\mathcal{H}} \leq \{\Gamma \mid a \leq \lceil \Gamma \vdash \bot \rceil\}$. Let $\Gamma \in a \to \bot_{\mathcal{H}}$. Let $\Delta \in a$. Upper values are provability-based, and therefore elements of $\mathcal{H}$ admit weakening, contraction and exchange. It follows that $\Gamma, \Delta \in a \to \bot_{\mathcal{H}}$ and $\Gamma, \Delta \in a$. Therefore we get $\Gamma, \Delta \in a \to \bot_{\mathcal{H}} \wedge a$. We know that $a \to \bot_{\mathcal{H}} \leq a \to \bot_{\mathcal{H}}$, and thus $a \to \bot_{\mathcal{H}} \wedge a \leq \bot_{\mathcal{H}}$ by the implication property. We derive $\Gamma, \Delta \in \bot_{\mathcal{H}} = \lfloor \bot \rfloor \leq \lceil \bot \rceil$. Thus $\Delta \in \lceil \Gamma \vdash \bot \rceil$. We conclude $a \leq \lceil \Gamma \vdash \bot \rceil$ for any $\Gamma \in a \to \bot_{\mathcal{H}}$, and then $a \to \bot_{\mathcal{H}} \leq \{\Gamma \mid a \leq \lceil \Gamma \vdash \bot \rceil\}$.

Let us show $\{\Gamma \mid a \leq \lceil \Gamma \vdash \bot \rceil\} \leq a \to \bot_{\mathcal{H}}$. Using the implication property, this is equivalent to $\{\Gamma \mid a \leq \lceil \Gamma \vdash \bot \rceil\} \wedge a \leq \bot_{\mathcal{H}}$. Let $\Gamma \in a$ such that $a \leq \lceil \Gamma \vdash \bot \rceil$. Therefore $\Gamma, \Gamma \vdash^{\uparrow} \bot$, and thus $\Gamma \vdash^{\uparrow} \bot$ by contraction. Using weakening and Lemma 6.45, we derive $\Delta, \Gamma \vdash^{\uparrow} B$ for any B and Γ. It follows that $\Gamma \in \lfloor \bot \rfloor$, that is $\Gamma \in \bot_{\mathcal{H}}$. We conclude $\{\Gamma \mid a \leq \lceil \Gamma \vdash \bot \rceil\} \wedge a \leq \bot_{\mathcal{H}}$. □

Proposition 6.46

In classical logic, $\mathcal{H}$ is a Boolean algebra.

Before proving that $\mathcal{H}$ is Boolean, we show some intermediate lemmas.

Lemma 6.47 (Kleene's Inversion Lemma)**.** *If $\Gamma \vdash^* A \Rightarrow B$ then $\Gamma, A \vdash^* B$.*

Proof. For $\vdash^{\downarrow}$, we have $\Gamma, A \vdash^{\downarrow} A \Rightarrow B$ using Weak. We also have $\Gamma, A \vdash^{\uparrow} A$ using Ax and Coerce. Thus we derive $\Gamma, A \vdash^{\downarrow} B$ using Imp-E.

For $\vdash^{\uparrow}$, we prove a stronger result: if $\Gamma \vdash^{\uparrow} C$ with $C \equiv A \Rightarrow B$ then $\Gamma, A \vdash^{\uparrow} B$. We proceed by induction on the derivation of $\Gamma \vdash^{\uparrow} C$.

- Conv: Suppose $\Gamma \vdash^{\uparrow} D$ with $D \equiv C$. As $C \equiv A \Rightarrow B$, we have $D \equiv A \Rightarrow B$. By induction hypothesis, we have $\Gamma, A \vdash^{\uparrow} B$.
- Coerce: Suppose $\Gamma \vdash^{\downarrow} C$ with $C \equiv A \Rightarrow B$. Using Conv, we have $\Gamma \vdash^{\downarrow} A \Rightarrow B$. By Kleene's inversion lemma on $\vdash^{\downarrow}$, we get $\Gamma, A \vdash^{\downarrow} B$. Therefore we derive $\Gamma, A \vdash^{\uparrow} B$ using Coerce.
- Imp-I: We directly have $\Gamma, A \vdash^{\uparrow} B$.
- OrInf-E: Suppose $\Gamma \vdash^{\downarrow} \bigvee_{\mathcal{I}} F$ and $\Gamma, \pi_i\ F \vdash^{\uparrow} C$ for every $i \in \mathcal{I}$, with $C \equiv A \Rightarrow B$. By induction hypotheses and Exch, we have $\Gamma, A, \pi_i\ F \vdash^{\uparrow} B$ for every $i \in \mathcal{I}$. Using OrInf-E, we derive $\Gamma, A \vdash^{\uparrow} B$.
- Ex-E: Suppose $\Gamma \vdash^{\downarrow} \exists\ P$ and $\Gamma, P\ x \vdash^{\uparrow} C$, with $C \equiv A \Rightarrow B$. By induction hypothesis and Exch, we have $\Gamma, A, P\ x \vdash^{\uparrow} B$. Using Ex-E, we derive $\Gamma, A \vdash^{\uparrow} B$.
- Bot-E: Suppose $\Gamma \vdash^{\downarrow} \bot$. Using Weak and Bot-E, we get $\Gamma, A \vdash^{\uparrow} B$.
- The cases Ax, Imp-E, AndInf-E, All-E and PEM do not apply as the conclusion of these rules uses $\vdash^{\downarrow}$. The cases AndInf-I, OrInf-I, All-I and Ex-I do not apply as the formula in the conclusion of these rules cannot be convertible to $A \Rightarrow B$. □

Lemma 6.48. *If $\Gamma \vdash^{\uparrow} A$ then $\Gamma \vdash^{\uparrow} \neg\neg A$.*

Proof. Using Weak, we get $\Gamma, A \Rightarrow \bot \vdash^{\uparrow} A$. Using Imp-E and Ax, we derive $\Gamma, A \Rightarrow \bot \vdash^{\downarrow} \bot$. Using Coerce and Imp-I, we conclude $\Gamma \vdash^{\uparrow} \neg\neg A$. □

Proof of Proposition 6.46. We want to prove $a \vee (a \to \bot_{\mathcal{H}}) = \top_{\mathcal{H}}$. By definition of the Heyting algebra, we have $a \vee (a \to \bot_{\mathcal{H}}) = \bigcap_{k \in K} \lceil \Gamma_k \vdash A_k \rceil$ for some A_k and Γ_k. Let $\ell \in K$. We have $a \leq a \vee (a \to \bot_{\mathcal{H}}) = \bigcap_{k \in K} \lceil \Gamma_k \vdash A_k \rceil \leq \lceil \Gamma_\ell \vdash A_\ell \rceil$. Using Lemma 6.48, it follows that $a \leq \lceil \Gamma_\ell \vdash \neg\neg A_\ell \rceil$. Using Lemma 6.47, we derive $a \leq \lceil \Gamma_\ell, \neg A_\ell \vdash \bot \rceil$. Using Proposition 6.44, we derive $\Gamma_\ell, \neg A_\ell \in a \to \bot_{\mathcal{H}}$. Since $a \to \bot_{\mathcal{H}} \leq \bigcap_{k \in K} \lceil \Gamma_k \vdash A_k \rceil$, we have in particular $\Gamma_\ell, \neg A_\ell \in \lceil \Gamma_\ell \vdash A_\ell \rceil$, i.e., $\Gamma_\ell, \Gamma_\ell, \neg A_\ell \vdash^{\uparrow} A_\ell$. By contraction, we get $\Gamma_\ell, \neg A_\ell \vdash^{\uparrow} A_\ell$. Using Ax and Coerce, we also have $\Gamma_\ell, A_\ell \vdash^{\uparrow} A_\ell$. Using PEM on A_ℓ and Or-E, we derive $\Gamma_\ell \vdash^{\uparrow} A_\ell$. It follows that $\top_{\mathcal{H}} \leq \lceil \Gamma_\ell \vdash A_\ell \rceil$. We also have $\lceil \Gamma_\ell \vdash A_\ell \rceil \leq \top_{\mathcal{H}}$, and therefore $\lceil \Gamma_\ell \vdash A_\ell \rceil = \top_{\mathcal{H}}$ for any $\ell \in K$. We conclude $a \vee (a \to \bot_{\mathcal{H}}) = \bigcap_{k \in K} \lceil \Gamma_k \vdash A_k \rceil = \top_{\mathcal{H}}$. □

Remark 6.49. *Although we proved Lemmas 6.45, 6.47 and 6.48 and Proposition 6.44 in order to define models of the classical variant of higher-order infinitary logic, these results hold for both the intuitionistic variant and the classical variant. The only place where PEM is needed is in the proof of Proposition 6.46.*

The strong completeness result (Theorem 6.41) and the cut-elimination theorem (Theorem 6.42) hold for the classical variant of higher-order infinitary logic. The proofs are unchanged compared to the intuitionistic variant: the cut-free-provability model now refer to the cut-free-provability Boolean algebra.

6.4 Barr's Theorem for Higher-Order Geometric Theories

Geometric formulas are built using finite conjunctions, infinite disjunctions and existential quantifiers. Geometric implications are formulas of the form $\forall \vec{x}.A$ or $\forall \vec{x}.(A \Rightarrow B)$, where A and B are geometric formulas and $\vec{x}$ is a finite sequence of variables (possibly empty). A theory $\mathbb{T}$ is geometric when all its axioms are geometric implications. Barr's theorem states that, in infinitary logic, any geometric implication that is classically provable in a geometric theory is also intuitionistically provable in the same theory.

In this section, we extend Barr's theorem [Bar74] to higher-order geometric theories. As in Chapter 5 and in [Pal02, vdB19], we follow the approach based on Friedman's translation [Fri78]. Throughout this section, we consider a higher-order infinitary logic with infinite disjunctions, but with finite conjunctions. As we will explain in Remark 6.53, this is because, intuitively, infinite conjunctions behave similarly to universal quantifiers. Moreover, we only consider geometric implications of the form $\forall \vec{x}.(A \Rightarrow B)$, as the case $\forall \vec{x}.A$ is easier.

6.4.1 Monad Translation for Higher-Order Infinitary Logic

We adapt the monad translation defined in Chapter 5 so that it takes into account the constructor and destructors for families of propositions.

Definition 6.50: Monad Translation for Higher-Order Infinitary Logic

Let A be a formula in higher-order infinitary logic. Its monad translation is $A^T := TA_T$, where $t \mapsto t_T$ is inductively defined by:

$$
\begin{array}{lcl}
x_T & := & x \\
c_T & := & \begin{cases} \lambda p.\ \forall x.\ T(p\ x) & \text{if } c = \forall \\ \lambda p.\ \lambda q.\ p \Rightarrow Tq & \text{if } c = \Rightarrow \\ c & \text{otherwise} \end{cases} \\
(\lambda x.\ t)_T & := & \lambda x.\ t_T \\
(t\ u)_T & := & t_T\ u_T \\
(\{i \mapsto A_i\}_{\mathcal{I}})_T & := & \{i \mapsto (A_i)_T\}_{\mathcal{I}} \\
(\pi_j\ F)_T & := & \pi_j\ F_T
\end{array}
$$

As we are in higher-order logic, our translation must commute with substitution and preserve convertibility. Note that all the results of this section also hold if we only consider the β-reduction rule and the β-like projection rules.

Proposition 6.51

Let t and u be terms, and A and B be higher-order infinitary formulas.

1. $(t[z \leftarrow w])_T = t_T[z \leftarrow w_T]$.
2. $(A[z \leftarrow w])^T = A^T[z \leftarrow w_T]$.
3. If $t \hookrightarrow u$ then $t_T \hookrightarrow u_T$.
4. If $t \equiv u$ then $t_T \equiv u_T$.
5. If $A \equiv B$ then $A_T \equiv B_T$ and $A^T \equiv B^T$.

Proof. We adapt the proofs of Proposition 5.5, Corollary 5.6, Proposition 5.7, Corollary 5.8 and Corollary 5.9. □

6.4.2 Translation from Classical Logic to Intuitionistic Logic

We prove that the Kuroda-style Friedman's translation for higher-order logic with infinite disjunctions goes from classical logic to intuitionistic logic. Let us remark that Proposition 5.3 also holds in higher-order infinitary logic.

Theorem 6.52

Let $TA := (A \Rightarrow R) \Rightarrow R$ with parameter R. Let A be a formula and Γ be a context in higher-order infinitary logic. If $\Gamma \vdash_{\mathsf{CL}} A$ then $\Gamma_T \vdash_{\mathsf{IL}} A^T$.

Proof. We proceed by induction on the derivation. We only show the new cases, compared to Lemma 5.21.

— OrInf-I: By induction hypothesis, we have $\Gamma_T \vdash_{\mathsf{IL}} T(\pi_j\ F_T)$. Using Ax, we have $\Gamma_T, \pi_j\ F_T \vdash_{\mathsf{IL}} \pi_j\ F_T$. Using OrInf-I, we get $\Gamma_T, \pi_j\ F_T \vdash_{\mathsf{IL}} \bigvee_{\mathcal{I}} F_T$. Using Imp-I and Proposition 5.3(2), we derive $\Gamma_T \vdash_{\mathsf{IL}} T(\pi_j\ F_T) \Rightarrow T(\bigvee_{\mathcal{I}} F_T)$. We conclude using the induction hypothesis.

— OrInf-E: By induction hypotheses, we have $\Gamma_T \vdash_{\mathsf{IL}} T(\bigvee_{\mathcal{I}} F_T)$ and $\Gamma_T, \pi_i\ F_T \vdash_{\mathsf{IL}} TC_T$ for every $i \in \mathcal{I}$. By Ax, we have $\Gamma_T, \bigvee_{\mathcal{I}} F_T \vdash_{\mathsf{IL}} \bigvee_{\mathcal{I}} F_T$. By weakening of the induction hypotheses, we have $\Gamma_T, \bigvee_{\mathcal{I}} F_T, \pi_i\ F_T \vdash_{\mathsf{IL}} TC_T$ for every $i \in \mathcal{I}$. Therefore, we derive $\Gamma_T, \bigvee_{\mathcal{I}} F_T \vdash_{\mathsf{IL}} TC_T$ using OrInf-E. We get $\Gamma_T \vdash_{\mathsf{IL}} T(\bigvee_{\mathcal{I}} F_T) \Rightarrow TC_T$ using Imp-I and (bind). We conclude using Imp-E with the first induction hypothesis. □

Remark 6.53. *The And-I case crucially depends on* $\vdash_{\mathsf{IL}} (TA \wedge TB) \Rightarrow T(A \wedge B)$. *Similarly, the* AndInf-I *case would crucially depend on* $\vdash_{\mathsf{IL}} (\bigwedge_{\mathcal{I}} TF) \Rightarrow T(\bigvee_{\mathcal{I}} F)$. *However, the proof of* $\vdash_{\mathsf{IL}} (TA \wedge TB) \Rightarrow T(A \wedge B)$ *does not extend to a proof of* $\vdash_{\mathsf{IL}} (\bigwedge_{\mathcal{I}} TF) \Rightarrow T(\bigvee_{\mathcal{I}} F)$. *This is not surprising:* $(\bigwedge_{\mathcal{I}} TF) \Rightarrow T(\bigvee_{\mathcal{I}} F)$ *is very similar to* $(\forall x.TA) \Rightarrow T(\forall x.A)$, *which generalizes the double-negation shift and does not hold in intuitionistic logic. This explains why, in this section, we consider finite conjunctions and not infinite conjunctions.*

6.4.3 Constructivization for Geometric Theories

Geometric formulas do not contain implications or universal quantifiers, while the translation $t \mapsto t_T$ only inserts T after universal quantifiers and on the right-hand side of implications. It follows that for any geometric formula A, we directly have $A_T = A$. The same result does not hold for geometric implications $\forall \vec{x}.(A \Rightarrow B)$. But geometric implications imply their translation.

Lemma 6.54. *Let A and B be higher-order geometric formulas, and $\vec{x}$ be a finite sequence of variables (possibly empty). We have $\vdash_{\mathsf{IL}} (\forall \vec{x}.(A \Rightarrow B)) \Rightarrow (\forall \vec{x}.(A \Rightarrow B))_T$.*

Proof. Similar to Lemma 5.23. □

Theorem 6.55

Let $\mathbb{T}$ be a higher-order geometric theory, A and B be higher-order geometric formulas, and $\vec{x}$ be a finite sequence of variables (possibly empty). If $\vdash_{\mathsf{CL}} \forall \vec{x}.(A \Rightarrow B)$ in $\mathbb{T}$ then $\vdash_{\mathsf{IL}} \forall \vec{x}.(A \Rightarrow B)$ in $\mathbb{T}$.

Proof. Similar to Theorem 5.24, except that we refer to Lemma 6.54 instead of Lemma 5.23, and to Theorem 6.52 instead of Lemma 5.21. □

6.5 Conclusion

We have extended simple type theory with a type, a constructor and destructors for families of propositions indexed by an arbitrary countable set. In this setting, the β-reduction neither satisfies strong normalization nor confluence. However, every well-typed term has a head normal form.

We have defined a higher-order infinitary logic based on this type theory. It features higher-order quantifiers as well as conjunctions and disjunctions indexed by an arbitrary countable set. We have defined a cut-free natural deduction system that precludes traditional cuts and commutative cuts.

We have investigated the semantics of this higher-order infinitary logic, for both the intuitionistic variant and the classical variant. We have defined a sound notion of model based on Heyting and Boolean algebras, and we have shown a strong notion of completeness based on V-complexes. These constructions allow us to derive the cut-elimination theorem for natural deduction.

Besides the elimination of cuts from proofs, we also studied the elimination of the principle of excluded middle. We have extended Barr's theorem to higher-order geometric theories. In other words, any geometric implication that is classically provable in a geometric theory is also intuitionistically provable in the same theory.

Part II

TRANSLATIONS BETWEEN THEORIES WITH REWRITING

Chapter 7

Introduction of Part II

Logical Frameworks

The Automath project [dB80], launched in the late 1960s, aimed at developing a tool for formalizing mathematics. As explained by de Bruijn, the goal was to be independent of any particular logic:

"The Automath system is like a big restaurant that serves all sorts of food: vegetarian, kosher, or anything else the customer wants. The languages are not tied to any logical system: hardly any logic has been built in."

This idea led to the development of ***logical frameworks*** [Pfe01], that are meta-languages for formalizing deductive systems. Logical frameworks provide an environment with basic features, and users encode the language, axioms and inference rules. The definitions encoded by the users are called ***theories***.

Isabelle [Pau94] is a logical framework based on a minimal version of higher-order logic. It has been extended with additional features, such as the Sledgehammer tool [PS07], that calls external automated theorem provers, or Isabelle/Isar [Wen99], that allows human-readable proofs. Notable theories encoded in Isabelle include Zermelo-Fraenkel set theory, called Isabelle/ZF [Pau93], and higher-order logic, called Isabelle/HOL [NPW02]. Isabelle/HOL has many libraries, for example related to algebra, analysis or Hoare logic [Nip02].

The well-known Edinburgh Logical Framework [HHP93] is a logical framework based on dependently typed λ-calculus. It relies on the ***Curry-Howard correspondence***: judgments are represented as types, and proofs of a given judgment are represented as terms of the corresponding type. Proof checking is thus reduced to ***type checking***.

A variety of extensions of the Edinburgh Logical Framework has been developed, including with logic programming in Twelf [PS99], abstraction over contexts in Beluga [PD10], side conditions in LFSC [SOR+13], monadic side conditions in $\text{LLF}_{\mathcal{P}}$ [HLMS17], and user-definable features in MMT [Rab18]. Many different logics can be encoded in the Edinburgh Logical Framework, including first-order and higher-order logics [HHP93], modal logics [AHMP92, AHMP98], foundations of mathematics [IR11] and Hoare logic [AHMP92].

The $\lambda\Pi$-Calculus Modulo Rewriting

The $\lambda\Pi$-calculus modulo rewriting [CD07] extends the Edinburgh Logical Framework with user-defined ***rewrite rules***. The user can define certain notions through rewrite rules instead of axioms. For instance, the axioms

$$x + 0 = x$$
$$x + \mathsf{succ}\ y = \mathsf{succ}\ (x + y)$$

can be replaced by the rewrite rules

$$x + 0 \hookrightarrow x$$
$$x + \mathsf{succ}\ y \hookrightarrow \mathsf{succ}\ (x + y)$$

In appearance insignificant, this deeply changes the behavior of the framework. In particular, terms are now considered modulo the congruence relation $\equiv_{\beta\mathcal{R}}$, generated by the usual β-reduction rule and by the user-defined rewrite rules. It follows that derivations in the $\lambda\Pi$-calculus modulo rewriting are based not only on ***deduction***, but also on ***computation***. While deductive steps of the derivations are written by the user, the computational parts of the derivations are directly managed by the framework. For instance, instead of proving that $2 + 2 = 4$ using the axioms, we only prove that $4 = 4$, as the framework computes that $2 + 2 \equiv_{\beta\mathcal{R}} 4$ using the rewrite rules. This results in a more user-friendly logical framework, in which part of the work is discharged to the system.

The $\lambda\Pi$-calculus modulo rewriting has been implemented in the Dedukti proof language [Sai15, ABC+16] and the Lambdapi proof assistant [HB20]. Many different logic and formal systems can be encoded in the $\lambda\Pi$-calculus modulo rewriting and Dedukti, such as predicate logic, Simple Type Theory and the Calculus of Constructions [BDG+23].

Interoperability between Proof Systems

Interoperability between proof systems can be achieved by defining one-to-one translations. For example, translations have been defined from HOL and HOL Light to Isabelle/HOL [OS06], from Isabelle/HOL to HOL Light [McL06], from HOL Light to Rocq [KW10], or from Maude to Lean [RR25].

Alternatively, it is possible to use a logical framework as a middleware. To exchange a proof from a proof system A to a proof system B, we proceed in three steps:

1. We import the proof from A to the encoding of A in the logical framework,
2. We translate, within the logical framework, the proof from the encoding of A to the encoding of B,
3. We export the proof from the encoding of B to B.

The advantage of this technique is that it treats all the translations uniformly, using the logical framework as a common language.

The $\lambda\Pi$-calculus modulo rewriting and Dedukti have been designed to act as a bridge between proof systems. As a case in point, the logic of many proof assistants can be expressed inside the $\lambda\Pi$-calculus modulo rewriting [Thi20]. Dedukti was used to translate

the Matita arithmetic library, that includes Fermat's Little Theorem, to Rocq, Lean and PVS in [Thi20], and to Agda in [FB24]. The HOL Light standard library was exported to Rocq, using Dedukti [Bla24]. As a consequence, exchanging proofs between different theories encoded in the $\lambda\Pi$-calculus modulo rewriting strengthens the interoperability between proof systems.

Related work: Universal Proof Libraries

So far, we have addressed the large number of proof systems and proof libraries under the prism of interoperability. A complementary approach is to build a universal proof library, that aims at organizing, managing and retrieving all this mathematical knowledge.

The Formal Digital Library [ABC+04] is a prototype universal library, based on a data structure that allows representing definitions, theorems, proofs and even articles. It gathers knowledge coming from Nuprl and PVS.

The Logosphere project [SPK+] aimed at building such a universal proof library, using the Edinburgh Logical Framework. HOL and Nuprl were encoded in Twelf, and a translation was established from the HOL encoding to the Nurpl encoding [SS06]. The LATIN project [CHK+11] built an atlas of logic and logic translations, based on the Edinburgh Logical Framework and implemented in Twelf. This library puts a particular emphasis on the modularity between logics, that is their inheritance relationships. The LATIN project was revised from scratch using the MMT system [Rab18].

The Logipedia encyclopedia [DT20, Thi20] features theorems and proofs expressed in different proof systems, such as Rocq, Matita, Lean and PVS. This allows the user to compare the different formalizations of the theorem. The encyclopedia contains in particular an arithmetic library of proofs obtained from Matita and translated to other systems using Dedukti.

Contributions of Part II

In Part II, our goal is to develop techniques to exchange proofs between different theories of the $\lambda\Pi$-calculus modulo rewriting.

In **Chapter 8**, we present the preliminaries to address Part II. We recall the syntax of the $\lambda\Pi$-calculus modulo rewriting and of the Dedukti proof language. We highlight several examples of theories encoded in the $\lambda\Pi$-calculus modulo rewriting.

In **Chapter 9**, we focus on theories of the $\lambda\Pi$-calculus modulo rewriting that feature higher-order logic. We translate them from classical logic to intuitionistic logic, by extending Kuroda's double-negation translation to the $\lambda\Pi$-calculus modulo rewriting. To do so, we adapt the translation of Chapter 4 and take into account three additional features, namely dependent types, rewrite rules, and the Curry-Howard correspondence. We implement this translation in Dedukti.

In **Chapter 10**, we define different translation mechanisms for the $\lambda\Pi$-calculus modulo rewriting. In particular, we extend theory morphisms and logical relations from the Edinburgh Logic Framework to the $\lambda\Pi$-calculus modulo rewriting. These translations are

guaranteed to be correct, provided that the parameters supplied by the user are correct. These templates can be instantiated by the user to generate translations between theories of the $\lambda\Pi$-calculus modulo rewriting. We apply these translation templates to various theories, allowing us to analyze in practice the advantages and drawbacks of rewriting. We implement the translation templates in Dedukti.

Chapter 8

Preliminaries: A Logical Framework with Rewriting

Summary

In this chapter, we present the necessary preliminaries to apprehend Part II. We give an overview of $\lambda\Pi$-calculus modulo rewriting and Dedukti. We describe how rewriting can be leveraged to encode theories inside this logical framework.

The Edinburgh Logical Framework [HHP93], also known as LF or $\lambda\Pi$-calculus, extends simply typed λ-calculus (as described in Chapter 3) with *dependent types*. The $\lambda\Pi$-calculus modulo rewriting [CD07], abbreviated $\lambda\Pi/\mathcal{R}$, extends LF with user-defined *rewrite rules*.

Dependent types [Hof97] are types that depend on a value. In particular, LF and $\lambda\Pi/\mathcal{R}$ feature dependent function types $\Pi x : A.\ B$. The variable x may occur in B, thus introducing a dependency. Dependent function types $\Pi x : A.\ B$ generalize function types $A \to B$. In programming languages with dependent types, vectors indexed by their lengths are for example encoded using the type $\Pi n : \mathsf{nat}.\ \mathsf{vec}\ n$.

Rewrite rules [DJ91] are pairs of terms, written $\ell \hookrightarrow r$, where ℓ is the left-hand side and r is the right-hand side. They allow replacing, in terms, instances of ℓ with r. Rewrite rules $\ell \hookrightarrow r$ correspond to oriented equations $\ell = r$. This orientation is useful when defining algorithmic procedures that check equality between terms.

8.1 The $\lambda\Pi$-Calculus Modulo Rewriting

In this section, we present the syntax of $\lambda\Pi/\mathcal{R}$, its conversion and typing rules, and then give an overview of its main meta-properties.

8.1.1 Syntax

The terms of $\lambda\Pi/\mathcal{R}$ are divided into three levels: objects (denoted by M and N), types (denoted by A and B), and kinds (denoted by K).

Definition 8.1: Syntax of $\lambda\Pi/\mathcal{R}$

The syntax of $\lambda\Pi/\mathcal{R}$ is given by the following grammars:

Objects	$M, N ::= c \mid x \mid \lambda x : A.\ M \mid M\ N$
Types	$A, B ::= a \mid \Pi x : A.\ B \mid \lambda x : A.\ B \mid A\ M$
Kinds	$K ::= \mathsf{Type} \mid \Pi x : A.\ K$
Terms	$t, u, v, T ::= M \mid A \mid K \mid \mathsf{Kind}$

where c is an object constant, a is a type constant, and x is a variable.

Dependent types $\Pi x : A.\ B$ (respectively $\Pi x : A.\ K$) are written $A \to B$ (respectively $A \to K$) when x does not occur in B (respectively K). Applications $(t\ M)\ N$ are written $t\ M\ N$, and dependent types $\Pi x : A.\ (\Pi y : B.\ t)$ are written $\Pi x : A.\ \Pi y : B.\ t$. For simplicity, abstractions $\lambda x : A.\ \lambda y : A.\ t$ are written $\lambda x, y : A.\ t$, and dependent types $\Pi x : A.\ \Pi y : A.\ t$ are written $\Pi x, y : A.\ t$. In abstractions and dependent types, we may omit the type of bound variables when it is clear.

Substitutions θ associate terms to variables. We write $t\theta$ for the result of the capture-avoiding substitution θ in the term t (i.e., possibly renaming bound variables of t to avoid capturing free variables). Contexts Γ are used to specify the type of the free variables, and theories $\mathbb{T}$ are used to declare the constants and rewrite rules considered by the users. Substitutions, contexts and theories are finite sequences, and are written $\varnothing$ when empty.

Rewrite rules are pairs $M \hookrightarrow N$ or $A \hookrightarrow B$. We only consider rewrite rules $\ell \hookrightarrow r$ such that the free variables of r are also free in ℓ, and such that ℓ is *algebraic* but not simply a variable. A term is called algebraic when it is a variable or the application of a constant to algebraic arguments. These two conditions allow inferring the types of all free variables in a rewrite rule from their occurrences in ℓ. Therefore, we do not need to specify the context that binds the free variables of rewrite rules.

Definition 8.2

Substitutions, contexts and theories of $\lambda\Pi/\mathcal{R}$ are defined by:

Substitutions	$\theta ::= \varnothing \mid \theta, x \leftarrow N$
Contexts	$\Gamma ::= \varnothing \mid \Gamma, x : A$
Theories	$\mathbb{T} ::= \varnothing \mid \mathbb{T},\ c : A \mid \mathbb{T},\ a : K \mid \mathbb{T},\ M \hookrightarrow N \mid \mathbb{T},\ A \hookrightarrow B$

In the syntax of terms, abstractions and dependent types only abstract over variables x of type A, and not variable x of kind K. Similarly, contexts only declare variables x of type A, and not variable x of kind K.

We write $\mathsf{dom}(\Gamma)$ for the domain of Γ, and $\mathsf{dom}(\mathbb{T})$ for the domain of $\mathbb{T}$.

8.1.2 Conversion

The conversion $\equiv_{\beta\mathcal{R}}$, also called definitional equality, is generated by the usual β-reduction rule and by the user-defined rewrite rules. As rewrite rules are specified in the theory, it follows that every theory has a specific conversion relation.

Definition 8.3: Conversion $\equiv_{\beta\mathcal{R}}$

The relation $\hookrightarrow_{\beta\mathcal{R}}$ is the smallest relation such that:

— $(\lambda x : A.\ M)\ N \hookrightarrow_{\beta\mathcal{R}} M[x \leftarrow N]$ and $(\lambda x : A.\ B)\ N \hookrightarrow_{\beta\mathcal{R}} B[x \leftarrow N]$,

— If $\ell \hookrightarrow r \in \mathbb{T}$ then $\ell\theta \hookrightarrow_{\beta\mathcal{R}} r\theta$ for any substitution θ,

— If $M \hookrightarrow_{\beta\mathcal{R}} M'$ then

$$\begin{array}{rcl} M\ N & \hookrightarrow_{\beta\mathcal{R}} & M'\ N \\ N\ M & \hookrightarrow_{\beta\mathcal{R}} & N\ M' \end{array} \qquad \begin{array}{rcl} A\ M & \hookrightarrow_{\beta\mathcal{R}} & A\ M' \\ \lambda x : A.\ M & \hookrightarrow_{\beta\mathcal{R}} & \lambda x : A.\ M' \end{array}$$

— If $A \hookrightarrow_{\beta\mathcal{R}} A'$ then

$$\begin{array}{rcl} A\ M & \hookrightarrow_{\beta\mathcal{R}} & A'\ M \\ \lambda x : A.\ M & \hookrightarrow_{\beta\mathcal{R}} & \lambda x : A'.\ M \\ \lambda x : A.\ B & \hookrightarrow_{\beta\mathcal{R}} & \lambda x : A'.\ B \\ \lambda x : B.\ A & \hookrightarrow_{\beta\mathcal{R}} & \lambda x : B.\ A' \end{array} \qquad \begin{array}{rcl} \Pi x : A.\ B & \hookrightarrow_{\beta\mathcal{R}} & \Pi x : A'.\ B \\ \Pi x : B.\ A & \hookrightarrow_{\beta\mathcal{R}} & \Pi x : B.\ A' \\ \Pi x : A.\ K & \hookrightarrow_{\beta\mathcal{R}} & \Pi x : A'.\ K \end{array}$$

— If $K \hookrightarrow_{\beta\mathcal{R}} K'$ then $\Pi x : A.\ K \hookrightarrow_{\beta\mathcal{R}} \Pi x : A.\ K'$,

The conversion $\equiv_{\beta\mathcal{R}}$ is the reflexive, symmetric and transitive closure of $\hookrightarrow_{\beta\mathcal{R}}$.

Remark 8.4 (η-Conversion)**.** *Logical frameworks are often presented with η-conversion, as it is necessary for some meta-properties such as adequacy. However, the combination of rewriting and η-conversion is subtle and must be handled carefully [Klo80, Bla16]. That is why $\lambda\Pi/\mathcal{R}$ has been defined without η-conversion. Dedukti however provides a flag to turn η-conversion on or off.*

In order to retain compatibility with the existing literature on both $\lambda\Pi/\mathcal{R}$ (without η-conversion) and other logical frameworks (usually with η-conversion), we systematically develop our results for both variants. In particular, all our meta-theorems hold with and without η-conversion, unless mentioned otherwise.

Remark 8.5. *Combining the η-contraction rule with rewrite rules may break confluence. For example, consider the rewrite rule $f\ x \hookrightarrow c$, where f and c are constants. We have $\lambda x.\ f\ x \hookrightarrow f$ and $\lambda x.\ f\ x \hookrightarrow \lambda x.\ c$, but there is no common reduct of f and $\lambda x.\ c$. On the contrary, the η-expansion rule can be combined with many rewrite rules in various typed λ-calculi, so that confluence and strong normalization are preserved [DCK93, Dou93, DCK94, JG95, DCK96]. For this reason, we prefer to use the η-expansion rule in the rest of this work.*

Definition 8.6: Conversion $\equiv_{\beta\eta\mathcal{R}}$

The relation $\hookrightarrow_{\beta\eta\mathcal{R}}$ is defined like $\hookrightarrow_{\beta\mathcal{R}}$, with additional cases for η-expansion:

— $M \hookrightarrow_{\beta\eta\mathcal{R}} \lambda x : A.\ M\ x$, where x is not free in M,

— $B \hookrightarrow_{\beta\eta\mathcal{R}} \lambda x : A.\ B\ x$, where x is not free in B.

The conversion $\equiv_{\beta\eta\mathcal{R}}$ is the reflexive, symmetric and transitive closure of $\hookrightarrow_{\beta\eta\mathcal{R}}$.

We use the subscript $\beta(\eta)\mathcal{R}$ to indicate that either variant can be used if that variant is globally fixed.

Lemma 8.7. *The conversion $\equiv_{\beta(\eta)\mathcal{R}}$ is a congruence relation.*

Proof. Let us prove the following case: if $A \equiv_{\beta(\eta)\mathcal{R}} B$ and $M \equiv_{\beta(\eta)\mathcal{R}} N$, then $\lambda x : A.\ M \equiv_{\beta(\eta)\mathcal{R}} \lambda x : B.\ N$. We easily prove by induction on $A \equiv_{\beta(\eta)\mathcal{R}} B$ that $A \equiv_{\beta(\eta)\mathcal{R}} B$ entails $\lambda x : A.\ M \equiv_{\beta(\eta)\mathcal{R}} \lambda x : B.\ M$. Similarly, we prove by induction on $M \equiv_{\beta(\eta)\mathcal{R}} N$ that $M \equiv_{\beta(\eta)\mathcal{R}} N$ entails $\lambda x : B.\ M \equiv_{\beta(\eta)\mathcal{R}} \lambda x : B.\ N$. The combination of the two results achieves the case. The other cases are proved similarly. □

8.1.3 Typing Rules

We write $\vdash_{\mathbb{T}} \Gamma$ when the context Γ is well-formed, and $\Gamma \vdash_{\mathbb{T}} t : T$ when the term t has type T in the context Γ. For convenience, $\varnothing \vdash_{\mathbb{T}} t : T$ is simply written $\vdash_{\mathbb{T}} t : T$. The typing rules for $\lambda\Pi/\mathcal{R}$ are given in Figure 8.1.

Remark 8.8 (Untyped Conversion)**.** *Type systems, such as LF and $\lambda\Pi/\mathcal{R}$, can be presented with typed conversion (where $\Gamma \vdash M \equiv N : A$ means that M and N are convertible and that each term of the conversion chain is well-typed) or untyped conversion (where $M \equiv N$ only means that M and N are convertible).*

In $\lambda\Pi/\mathcal{R}$, due to the presence of rewriting, it is preferable to use the untyped variant [CD07, ABC+16, Sai15, BDG+23], as it allows an efficient implementation of the conversion relation. Indeed, unlike the typed conversion, the untyped conversion does not require to check that each term involved in the conversion chain is well-typed. However, to avoid accidentally converting well-typed terms into ill-typed terms, the rules Conv-Type and Conv-Kind check that the converted term is well-typed before using it.

The equivalence between the variant with typed conversion and the variant with untyped conversion has been investigated for Pure Type Systems with β-conversion and $\beta\eta$-conversion in [GW94, Ada06, Sil10, SH12]. The case of $\lambda\Pi/\mathcal{R}$, where the conversion additionally includes user-defined rewrite rules, remains to be investigated.

8.1.4 Well-Formed Theories

We write $\mathbb{T} \vdash$ when the theory $\mathbb{T}$ is well-formed. When declaring a constant $c : A$ (respectively $a : K$), we must ensure that A is a type (respectively K is a kind). When declaring rewrite rules, we ensure that they preserve typing.

Definition 8.9

Let $\mathbb{T}$ be a theory and ℓ, r be two terms. The rewrite rule $\ell \hookrightarrow r$ is *well-formed* in $\mathbb{T}$, written $\mathbb{T} \vdash \ell \hookrightarrow r$, when it preserves typing: if $\Gamma \vdash_{\mathbb{T}} \ell\theta : T$ for some substitution θ, then $\Gamma \vdash_{\mathbb{T}} r\theta : T$.

Contexts

$$\frac{}{\vdash_{\mathbb{T}} \varnothing}\ \text{Empty-Ctx} \qquad \frac{\vdash_{\mathbb{T}} \Gamma \qquad \Gamma \vdash_{\mathbb{T}} A : \mathsf{Type} \qquad x \notin \mathsf{dom}(\Gamma)}{\vdash_{\mathbb{T}} \Gamma, x : A}\ \text{Decl-Var}$$

Objects

$$\frac{\vdash_{\mathbb{T}} \Gamma \qquad c : A \in \mathbb{T}}{\Gamma \vdash_{\mathbb{T}} c : A}\ \text{Const-Obj} \qquad \frac{\vdash_{\mathbb{T}} \Gamma \qquad x : A \in \Gamma}{\Gamma \vdash_{\mathbb{T}} x : A}\ \text{Var}$$

$$\frac{\Gamma \vdash_{\mathbb{T}} A : \mathsf{Type} \qquad \Gamma, x : A \vdash_{\mathbb{T}} B : \mathsf{Type} \qquad \Gamma, x : A \vdash_{\mathbb{T}} M : B}{\Gamma \vdash_{\mathbb{T}} \lambda x : A.\ M : \Pi x : A.\ B}\ \text{Abs-Obj}$$

$$\frac{\Gamma \vdash_{\mathbb{T}} M : \Pi x : A.\ B \qquad \Gamma \vdash_{\mathbb{T}} N : A}{\Gamma \vdash_{\mathbb{T}} M\ N : B[x \leftarrow N]}\ \text{App-Obj}$$

$$\frac{\Gamma \vdash_{\mathbb{T}} M : A \qquad \Gamma \vdash_{\mathbb{T}} B : \mathsf{Type} \qquad A \equiv_{\beta(\eta)\mathcal{R}} B}{\Gamma \vdash_{\mathbb{T}} M : B}\ \text{Conv-Type}$$

Types

$$\frac{\vdash_{\mathbb{T}} \Gamma \qquad a : K \in \mathbb{T}}{\Gamma \vdash_{\mathbb{T}} a : K}\ \text{Const-Type}$$

$$\frac{\Gamma \vdash_{\mathbb{T}} A : \mathsf{Type} \qquad \Gamma, x : A \vdash_{\mathbb{T}} B : \mathsf{Type}}{\Gamma \vdash_{\mathbb{T}} \Pi x : A.\ B : \mathsf{Type}}\ \text{Prod-Type}$$

$$\frac{\Gamma \vdash_{\mathbb{T}} A : \mathsf{Type} \qquad \Gamma, x : A \vdash_{\mathbb{T}} K : \mathsf{Kind} \qquad \Gamma, x : A \vdash_{\mathbb{T}} B : K}{\Gamma \vdash_{\mathbb{T}} \lambda x : A.\ B : \Pi x : A.\ K}\ \text{Abs-Type}$$

$$\frac{\Gamma \vdash_{\mathbb{T}} A : \Pi x : B.\ K \qquad \Gamma \vdash_{\mathbb{T}} M : B}{\Gamma \vdash_{\mathbb{T}} A\ M : K[x \leftarrow M]}\ \text{App-Type}$$

$$\frac{\Gamma \vdash_{\mathbb{T}} A : K \qquad \Gamma \vdash_{\mathbb{T}} K' : \mathsf{Kind} \qquad K \equiv_{\beta(\eta)\mathcal{R}} K'}{\Gamma \vdash_{\mathbb{T}} A : K'}\ \text{Conv-Kind}$$

Kinds

$$\frac{\vdash_{\mathbb{T}} \Gamma}{\Gamma \vdash_{\mathbb{T}} \mathsf{Type} : \mathsf{Kind}}\ \text{Sort} \qquad \frac{\Gamma \vdash_{\mathbb{T}} A : \mathsf{Type} \qquad \Gamma, x : A \vdash_{\mathbb{T}} K : \mathsf{Kind}}{\Gamma \vdash_{\mathbb{T}} \Pi x : A.\ K : \mathsf{Kind}}\ \text{Prod-Kind}$$

Figure 8.1: Typing rules for $\lambda\Pi/\mathcal{R}$.

This condition may appear surprising: $\lambda\Pi/\mathcal{R}$ does not require, as we could intuitively expect, that the left-hand side ℓ and the right-hand side r are well-typed and have the same type. Instead, $\lambda\Pi/\mathcal{R}$ only requires that every rewrite rule preserves typing whenever a substitution instance of the left-hand side is well-typed. The most common situation where it helps to have this generality is when we linearize rewrite rules.

Example 8.10 (Left-Linear Rules)**.** *Consider an encoding of polymorphic lists, with the constants* cons *and* hd *that take the sort of the elements of the list as a first argument. The natural rewrite rule is* hd a (cons a h t) $\hookrightarrow h$. *However, so as to reuse existing confluence criteria on left-linear rules, it is preferable [Bla01, Sai15] to use the linearized version* hd a (cons a' h t) $\hookrightarrow h$. *Its left-hand side is ill-typed, but this rule is still type-preserving in the sense of Definition 8.9, because substitution instances of the left-hand side can only be well-typed if they substitute the same terms for a and a'.*

While this type-preservation condition makes it more difficult to check individual rules, the following lemma from [Sai15, see Theorem 2.6.19] states that the usual intuition is a sufficient criterion.

Lemma 8.11 (Typed Rewrite Rules)**.** *Let $\ell \hookrightarrow r$ be a rewrite rule. If Γ declares the variables of ℓ, and we have $\Gamma \vdash \ell : T$ and $\Gamma \vdash r : T$, then $\ell \hookrightarrow r$ preserves typing.*

Well-formedness rules for theories of $\lambda\Pi/\mathcal{R}$ are given in Figure 8.2.

Theories

$$\frac{}{\varnothing \vdash}\ \text{Empty-Thy} \qquad \frac{\mathbb{T} \vdash \quad \vdash_{\mathbb{T}} A : \mathsf{Type} \quad c \notin \mathsf{dom}(\mathbb{T})}{\mathbb{T}, c : A \vdash}\ \text{Decl-Obj}$$

$$\frac{\mathbb{T} \vdash \quad \vdash_{\mathbb{T}} K : \mathsf{Kind} \quad a \notin \mathsf{dom}(\mathbb{T})}{\mathbb{T}, a : K \vdash}\ \text{Decl-Type}$$

$$\frac{\mathbb{T} \vdash \quad \mathbb{T} \vdash M \hookrightarrow N}{\mathbb{T}, M \hookrightarrow N \vdash}\ \text{Rewr-Obj} \qquad \frac{\mathbb{T} \vdash \quad \mathbb{T} \vdash A \hookrightarrow B}{\mathbb{T}, A \hookrightarrow B \vdash}\ \text{Rewr-Type}$$

Figure 8.2: Well-formedness rules for theories of $\lambda\Pi/\mathcal{R}$.

8.1.5 Meta-Properties

We briefly present the main meta-properties of $\lambda\Pi/\mathcal{R}$. The details and proofs are available in [Sai15].

Proposition 8.12: Weakening

Let Γ be a subset of Δ and $\vdash_{\mathbb{T}} \Delta$. If $\Gamma \vdash_{\mathbb{T}} t : T$, then $\Delta \vdash_{\mathbb{T}} t : T$.

Proposition 8.13: Substitution

If $\Gamma, x : A, \Delta \vdash_{\mathbb{T}} t : T$ and $\Gamma \vdash_{\mathbb{T}} u : A$, then $\Gamma, \Delta[x \leftarrow u] \vdash_{\mathbb{T}} t[x \leftarrow u] : T[x \leftarrow u]$.

Typed terms in $\lambda\Pi/\mathcal{R}$ respect the hierarchy between objects, types and kinds.

Proposition 8.14: Stratification

If $\Gamma \vdash_{\mathbb{T}} t : T$, then

— either t is an object, T is a type and $\Gamma \vdash_{\mathbb{T}} T :$ Type,

— or t is a type, T is a kind and $\Gamma \vdash_{\mathbb{T}} T :$ Kind,

— or t is a kind and $T =$ Kind.

Well-formed rewrite rules preserve typing. Additionally, rewriting preserves typing.

Proposition 8.15: Subject Reduction

If $\Gamma \vdash_{\mathbb{T}} t : T$ and $t \hookrightarrow_{\beta\mathcal{R}} u$, then $\Gamma \vdash_{\mathbb{T}} u : T$.

In $\lambda\Pi/\mathcal{R}$, the type of a term is unique modulo rewriting.

Proposition 8.16: Uniqueness of Types

If $\Gamma \vdash_{\mathbb{T}} t : T_1$ and $\Gamma \vdash_{\mathbb{T}} t : T_2$, then $T_1 \equiv_{\beta\mathcal{R}} T_2$.

While type-checking is decidable in LF, it is in general undecidable in $\lambda\Pi/\mathcal{R}$ because the conversion $\equiv_{\beta\mathcal{R}}$ is undecidable. When we restrict our attention to theories for which $\hookrightarrow_{\beta(\eta)\mathcal{R}}$ is confluent and produces terms in (η-long) β-normal forms, the conversion is decidable, as it suffices to check the equality of the (η-long) β-normal forms. In this case, type-checking in $\lambda\Pi/\mathcal{R}$ becomes decidable.

8.2 Encoding Theories in the $\lambda\Pi$-Calculus Modulo Rewriting

We show different encodings in $\lambda\Pi/\mathcal{R}$, namely for propositional logic and natural numbers. For both, we encode two versions: a deductive one that uses axioms, following the approach of LF, and a computational one that uses rewrite rules and fully takes advantage of the rewriting power of $\lambda\Pi/\mathcal{R}$.

8.2.1 Propositional Logic

Example 8.17 (Propositional Logic (Deductive version))**.** *We define the theory* PL*, a fragment of propositional logic with implication and conjunction.* $Prop$ *is the type of propositions and*

Prf maps a proposition to the type of their proofs.

$Prop$: Type

$Prf : Prop \to$ Type

$\Rightarrow : Prop \to Prop \to Prop$

$\wedge : Prop \to Prop \to Prop$

The encoding of the notions of proposition and proof is well-suited for representing judgments and natural deduction inference rules: logical consequences are represented by arrow types, and parameters are represented by dependent types. For example, the elimination of implication

$$\frac{\Gamma \vdash P \Rightarrow Q \qquad \Gamma \vdash P}{\Gamma \vdash Q}\ \text{Imp-E}$$

is represented by the typed constant

$\mathsf{imp_e} : \Pi p, q : Prop.\ Prf\ (p \Rightarrow q) \to (Prf\ p \to Prf\ q)$

When H_{pq} is of type $Prf\ (p \Rightarrow q)$ and H_p is of type $Prf\ p$, the modus ponens step corresponds to the proof term $\mathsf{imp_e}\ p\ q\ H_{pq}\ H_p$ *of type $Prf\ q$.*

The remaining inference rules are encoded using the following axioms.

$\mathsf{imp_i} : \Pi p, q : Prop.\ (Prf\ p \to Prf\ q) \to Prf\ (p \Rightarrow q)$

$\mathsf{and_i} : \Pi p, q : Prop.\ Prf\ p \to Prf\ q \to Prf\ (p \wedge q)$

$\mathsf{and_{e\ell}} : \Pi p, q : Prop.\ Prf\ (p \wedge q) \to Prf\ p$

$\mathsf{and_{er}} : \Pi p, q : Prop.\ Prf\ (p \wedge q) \to Prf\ q$

Example 8.18 (Propositional Logic (Computational version))**.** *Instead of encoding the natural deduction inference rules using axioms, it is possible to resort to rewrite rules [BDG+23]. The rewrite rule for implication is*

$$Prf\ (p \Rightarrow q) \hookrightarrow (Prf\ p \to Prf\ q)$$

with free variables p and q.

Compared to Example 8.17, this means that $Prf\ (p \Rightarrow q) \equiv_{\beta\mathcal{R}} Prf\ p \to Prf\ q$, hence the introduction and elimination of implication, which map back and forth between $Prf\ (p \Rightarrow q)$ and $Prf\ p \to Prf\ q$, can be dropped entirely. For instance, H_{pq} of type $Prf\ (p \Rightarrow q)$ is also of type $Prf\ p \to Prf\ q$. Therefore, the modus ponens step $\mathsf{imp_e}\ p\ q\ H_{pq}\ H_p$ *simply becomes $H_{pq}\ H_p$, erasing the explicit application of* $\mathsf{imp_e}$ *and its arguments p and q. This simplifies the work of the user and leads to smaller proof terms.*

The rewrite rule for the conjunction is

$$Prf\ (p \wedge q) \hookrightarrow \Pi r : Prop.\ (Prf\ p \to Prf\ q \to Prf\ r) \to Prf\ r$$

from which we can derive $\mathsf{and_i}$, $\mathsf{and_{e\ell}}$ *and* $\mathsf{and_{er}}$ *as theorems. For example,* $\mathsf{and_i}$ *is represented by the proof term*

$$\begin{array}{l}\lambda p, q : Prop.\ \lambda H_p : Prf\ p.\ \lambda H_q : Prf\ q.\\ \lambda r : Prop.\ \lambda H_{pqr} : Prf\ p \to Prf\ q \to Prf\ r.\\ H_{pqr}\ H_p\ H_q\end{array}$$

which is indeed of type $\Pi p, q : Prop.\ Prf\ p \to Prf\ q \to Prf\ (p \wedge q)$.

8.2.2 Natural Numbers

Example 8.19 (Natural numbers (Deductive version))**.** *Natural numbers are encoded by the type* nat *of natural numbers, and the constructors* 0 *and* succ*. We declare an infix constant* $+$ *representing the addition function.*

$\mathsf{nat} : \mathsf{Type}$

$0 : \mathsf{nat}$

$\mathsf{succ} : \mathsf{nat} \to \mathsf{nat}$

$+ : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$

We define an equality on natural numbers.

$= : \mathsf{nat} \to \mathsf{nat} \to Prop$

$\mathsf{refl} : \Pi x : \mathsf{nat}.\ Prf\ (x = x)$

$\mathsf{leib} : \Pi x, y : \mathsf{nat}.\ Prf\ (x = y) \to \Pi P : \mathsf{nat} \to Prop.\ Prf\ (P\ x) \to Prf\ (P\ y)$

The behavior of addition can be encoded using axioms.

$\mathsf{addZ} : \Pi x : \mathsf{nat}.\ Prf\ (x + 0 = x)$

$\mathsf{addS} : \Pi x, y : \mathsf{nat}.\ Prf\ (x + \mathsf{succ}\ y = \mathsf{succ}\ (x + y))$

For simplicity, we write 2 *for* $\mathsf{succ}\ (\mathsf{succ}\ 0)$ *and* 4 *for* $\mathsf{succ}\ (\mathsf{succ}\ 2)$*. To prove that* $2 + 2 = 4$*, we have to combine* refl*,* leib*,* addZ *and* addS*. The proof term*

$$\begin{array}{l}\mathsf{leib}\ (2+0)\ 2\ (\mathsf{addZ}\ 2)\ (\lambda x.\ 2+2 = \mathsf{succ}\ (\mathsf{succ}\ x))\\ \quad(\mathsf{leib}\ (2+\mathsf{succ}\ 0)\ (\mathsf{succ}\ (2+0))\ (\mathsf{addS}\ 2\ 0)\ (\lambda x.\ 2+2 = \mathsf{succ}\ x)\\ \quad\quad(\mathsf{leib}\ (2+2)(\mathsf{succ}\ (2+\mathsf{succ}\ 0))\ (\mathsf{addS}\ 2\ (\mathsf{succ}\ 0))\ (\lambda x.\ 2+2 = x)\ (\mathsf{refl}\ (2+2))))\end{array}$$

is of type $Prf\ (2 + 2 = 4)$*.*

Example 8.20 (Natural numbers (Computational version))**.** *Alternatively, the behavior of addition can be encoded using rewrite rules.*

$x + 0 \hookrightarrow x$

$x + \mathsf{succ}\ y \hookrightarrow \mathsf{succ}\ (x + y)$

Using these rewrite rules, we directly have

$$\begin{array}{rcl}\mathsf{succ}\ (\mathsf{succ}\ 0) + \mathsf{succ}\ (\mathsf{succ}\ 0) & \hookrightarrow_{\beta\mathcal{R}} & \mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ 0) + \mathsf{succ}\ 0)\\ & \hookrightarrow_{\beta\mathcal{R}} & \mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ 0) + 0))\\ & \hookrightarrow_{\beta\mathcal{R}} & \mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ 0)))\end{array}$$

which means $2 + 2 \hookrightarrow_{\beta\mathcal{R}} 4$*, and therefore* $2 + 2 \equiv_{\beta\mathcal{R}} 4$*. The advantage of resorting to computation instead of deduction is clear: the complicated proof term we built with the axioms in Example 8.19 is simply replaced by the proof term* refl 4*, which is of type* $Prf\ (4 = 4)$*, and therefore* $Prf\ (2 + 2 = 4)$*.*

Example 8.19 uses a *propositional* equality encoded in $\lambda\Pi/\mathcal{R}$, while Example 8.20 uses the *definitional* equality of $\lambda\Pi/\mathcal{R}$ (i.e., the conversion relation).

8.3 The Dedukti Proof Language

Dedukti [Sai15, ABC+16] is a proof language[1] implementing the $\lambda\Pi$-calculus modulo rewriting. It is written in OCaml and provides a type-checker. The extension for Dedukti files is `.dk`.

Syntax. Dedukti closely follows the syntax of $\lambda\Pi/\mathcal{R}$.

$\lambda\Pi/\mathcal{R}$	Dedukti
Application $t\ u$	`t u`
Abstraction $\lambda x : t.\ u$	`x : t => u`
Dependent product $\Pi x : t.\ u$	`x : t -> u`
Type	`Type`
Constant declaration $c : u$	`c : u.`
Rewrite rule declaration $\ell \hookrightarrow r$ with $\mathrm{fv}(\ell) = \{x_1, \ldots, x_n\}$	`[x_1, ..., x_n] l --> r.`
Theorem labeled n, of statement T and proof term p	`thm n : T := p.`

When a constant is defined using rewrite rules, its declaration `c : u` is prefixed by the keyword `def`. We can give the name n to a term t of type T, using `def n : T := t`.

Type-checking. We can type-check the Dedukti file `file.dk` using the command line `dk check file.dk`. The command `#ASSERT t : u` checks that `t` has type `u`. The command `#ASSERT t == u` checks that `t` and `u` are convertible. The type-checking algorithm relies on the confluence and termination of $\hookrightarrow_{\beta\mathcal{R}}$. If these conditions are not satisfied, type-checking may become undecidable and the algorithm may not be able to type-check a well-typed term.

Example. Let us encode Example 8.20 in Dedukti.

```
nat : Type.
0 : nat.
succ : nat -> nat.
```

The addition on natural numbers is defined through rewrite rules.

```
def add : nat -> nat -> nat.
[x] add x 0 --> x.
[x, y] add x (succ y) --> succ (add x y).
```

We can directly check that $2 + 2 \equiv_{\beta\mathcal{R}} 4$.

```
def 2 : nat := succ (succ 0).
def 4 : nat := succ (succ (succ (succ 0))).
#ASSERT (add 2 2) == 4.
```

We therefore prove that $2 + 2 = 4$.

[1] Available at https://github.com/Deducteam/Dedukti/.

```
Prop : Type.
Prf : Prop -> Type.

eq : nat -> nat -> Prop.
refl : x : nat -> Prf (eq x x).

thm add_2_2_4 : Prf (eq (add 2 2) 4)
:= refl 4.
```

The file type-checks correctly.

Alternative implementation. Dedukti is a very minimal proof language, that is used for translating between proof assistants. So as to develop proofs and formalize libraries in $\lambda\Pi/\mathcal{R}$, the Lambdapi [HB20] interactive proof assistant[2] has been implemented. It features proof tactics and implicit arguments, resulting in a much more user-friendly tool.

[2]Available at https://github.com/Deducteam/lambdapi.

Chapter 9

Case Study: Double-Negation Translation

Summary

In this chapter, we define Kuroda's translation for theories encoded in higher-order logic in the $\lambda\Pi$-calculus modulo rewriting. It is an *extension* of the translation of Chapter 4 to a logical framework that features dependent types, user-defined rewrite rules, and the Curry-Howard correspondence. It is also a *case study* of translation between theories of the $\lambda\Pi$-calculus modulo rewriting. We develop a tool, called Construkti, that implements Kuroda's translation for Dedukti theories encoded in higher-order logic.

This chapter is adapted from [Tra24a].

In Part I, we investigated the interrelations between higher-order logics. In particular, we defined translations from classical logic to intuitionistic logic. This problem is also relevant in the context of interoperability between proof systems. Indeed, some proof systems are based on classical logic, such as HOL Light or Mizar, while others are based on intuitionistic logic, such as Rocq, Lean or Agda. If we want to translate proofs from classical proof systems to intuitionistic proof systems *through* $\lambda\Pi/\mathcal{R}$ and Dedukti, we must be able to translate from classical logic to intuitionistic logic *inside* $\lambda\Pi/\mathcal{R}$ and Dedukti.

Related work. We are interesting in transforming *any* classical proof into an intuitionistic proof, at the price of modifying the proven statements during the process. Alternatively, proof constructivization aims at transforming classical proofs into intuitionistic ones *without* translating the formulas, but such a process does not necessarily succeed. Cauderlier [Cau16] developed heuristics to constructivize proofs in Dedukti, via rewrite systems that try to remove instances of the principle of excluded middle or of the double-negation elimination. Gilbert [Gil17] designed a constructivization algorithm for first-order logic, that was tested in Dedukti and works in practice for large fragments of first-order logic.

Contributions. In this chapter, we propose a case study of translation between theories of $\lambda\Pi/\mathcal{R}$, focusing on translation from an encoding of higher-order classical logic to an encoding of higher-order intuitionistic logic.

First, we extend Kuroda's translation to $\lambda\Pi/\mathcal{R}$. Compared to the translation of Chapter 4, we have to take into account three additional features: dependent types, rewrite rules, and the proofs-as-terms correspondence. We prove the soundness property, and the characterization property assuming functional extensionality and propositional extensionality.

Second, we have implemented the Construkti tool, that applies Kuroda's translation to Dedukti theories encoded in higher-order logic. We test Construkti on a benchmark of a hundred formal proofs.

Outline. In Section 9.1, we detail an encoding of higher-order logic in $\lambda\Pi/\mathcal{R}$. In Section 9.2, we define Kuroda's translation for theories of $\lambda\Pi/\mathcal{R}$ that are encoded in higher-order logic. We prove the soundness property in Section 9.3 and the characterization property in Section 9.4. In Section 9.5, we describe the implementation of Construkti and test it on Dedukti proofs.

9.1 Higher-Order Logic in the $\lambda\Pi$-Calculus Modulo Rewriting

In this section, we present an encoding of higher-order logic in $\lambda\Pi/\mathcal{R}$, and we detail what we mean by "theories encoded in higher-order logic in $\lambda\Pi/\mathcal{R}$".

9.1.1 An Encoding of Higher-Order Logic

In Examples 8.17 and 8.18, we presented encodings of (fragments of) propositional logic inside $\lambda\Pi/\mathcal{R}$. We now present an encoding of higher-order logic, following [BDG+23].

As for propositional logic, we encode the notion of proposition using the constant $Prop$, and the notion of proof using the constant Prf, that maps propositions to the type of their proofs.

$Prop$: Type

Prf : $Prop \to$ Type

We encode the logical connectives.

$\Rightarrow$: $Prop \to Prop \to Prop$

$\wedge$: $Prop \to Prop \to Prop$

$\vee$: $Prop \to Prop \to Prop$

$\neg$: $Prop \to Prop$

$\top$: $Prop$

$\bot$: $Prop$

As we will explain in Remark 9.9, Kuroda's translation cannot be extended if we encode the natural deduction rules using rewrite rules. The constants representing the natural deduction rules for the logical connectives are:

$\mathsf{imp_i} : \Pi p, q : \mathit{Prop}.\ (\mathit{Prf}\ p \to \mathit{Prf}\ q) \to \mathit{Prf}\ (p \Rightarrow q)$
$\mathsf{imp_e} : \Pi p, q : \mathit{Prop}.\ \mathit{Prf}\ (p \Rightarrow q) \to \mathit{Prf}\ p \to \mathit{Prf}\ q$
$\mathsf{and_i} : \Pi p, q : \mathit{Prop}.\ \mathit{Prf}\ p \to \mathit{Prf}\ q \to \mathit{Prf}\ (p \wedge q)$
$\mathsf{and_{e\ell}} : \Pi p, q : \mathit{Prop}.\ \mathit{Prf}\ (p \wedge q) \to \mathit{Prf}\ p$
$\mathsf{and_{er}} : \Pi p, q : \mathit{Prop}.\ \mathit{Prf}\ (p \wedge q) \to \mathit{Prf}\ q$
$\mathsf{or_{i\ell}} : \Pi p, q : \mathit{Prop}.\ \mathit{Prf}\ p \to \mathit{Prf}\ (p \vee q)$
$\mathsf{or_{ir}} : \Pi p, q : \mathit{Prop}.\ \mathit{Prf}\ q \to \mathit{Prf}\ (p \vee q)$
$\mathsf{or_e} : \Pi p, q : \mathit{Prop}.\ \mathit{Prf}\ (p \vee q) \to \Pi r : \mathit{Prop}.\ (\mathit{Prf}\ p \to \mathit{Prf}\ r) \to (\mathit{Prf}\ q \to \mathit{Prf}\ r) \to \mathit{Prf}\ r$
$\mathsf{neg_i} : \Pi p : \mathit{Prop}.\ (\mathit{Prf}\ p \to \mathit{Prf}\ \bot) \to \mathit{Prf}\ (\neg p)$
$\mathsf{neg_e} : \Pi p : \mathit{Prop}.\ \mathit{Prf}\ (\neg p) \to \mathit{Prf}\ p \to \mathit{Prf}\ \bot$
$\mathsf{top_i} : \mathit{Prf}\ \top$
$\mathsf{bot_e} : \mathit{Prf}\ \bot \to \Pi p : \mathit{Prop}.\ \mathit{Prf}\ p$

For convenience, the logical biconditional $\Leftrightarrow$ of type $\mathit{Prop} \to \mathit{Prop} \to \mathit{Prop}$ is encoded through the rewrite rule $p \Leftrightarrow q \hookrightarrow (p \Rightarrow q) \wedge (q \Rightarrow p)$.

First-order logic is obtained by encoding the quantifiers. Instead of defining, for any type A, quantifiers that abstract over terms of type A, we want to encode polymorphic quantifiers that take A as argument. However, following the syntax of terms and the typing rules, we can only abstract over objects in $\lambda\Pi/\mathcal{R}$, and we cannot abstract over types. To get around this issue, we declare the constant Set, which represents the universe of sorts, along with El that maps sorts to the type of their elements.

$\mathit{Set} : \mathsf{Type}$
$\mathit{El} : \mathit{Set} \to \mathsf{Type}$

This allow us to encode polymorphic quantifiers that take a sort as argument.

$\forall : \Pi x : \mathit{Set}.\ (\mathit{El}\ x \to \mathit{Prop}) \to \mathit{Prop}$
$\exists : \Pi x : \mathit{Set}.\ (\mathit{El}\ x \to \mathit{Prop}) \to \mathit{Prop}$

We encode the natural deduction rules for the quantifiers using typed constants.

$\mathsf{all_i} : \Pi a : \mathit{Set}.\ \Pi p : \mathit{El}\ a \to \mathit{Prop}.\ (\Pi x : \mathit{El}\ a.\ \mathit{Prf}\ (p\ x)) \to \mathit{Prf}\ (\forall\ a\ p)$
$\mathsf{all_e} : \Pi a : \mathit{Set}.\ \Pi p : \mathit{El}\ a \to \mathit{Prop}.\ \mathit{Prf}\ (\forall\ a\ p) \to \Pi x : \mathit{El}\ a.\ \mathit{Prf}\ (p\ x)$
$\mathsf{ex_i} : \Pi a : \mathit{Set}.\ \Pi p : \mathit{El}\ a \to \mathit{Prop}.\ \Pi x : \mathit{El}\ a.\ \mathit{Prf}\ (p\ x) \to \mathit{Prf}\ (\exists\ a\ p)$
$\mathsf{ex_e} : \Pi a : \mathit{Set}.\ \Pi p : \mathit{El}\ a \to \mathit{Prop}.\ \mathit{Prf}\ (\exists\ a\ p) \to$
$\qquad \Pi r : \mathit{Prop}.\ (\Pi x : \mathit{El}\ a.\ \mathit{Prf}\ (p\ x) \to \mathit{Prf}\ r) \to \mathit{Prf}\ r$

Higher-order logic is then obtained by allowing quantification on any sort, including sorts representing functions and propositions. The arrow $\rightsquigarrow$ encodes function sorts. The rewrite rule on $\rightsquigarrow$ states that the type embedding a function sort indeed corresponds to a function type. Propositions are considered as objects using the sort o.

$\rightsquigarrow : Set \to Set \to Set$

$El\ (x \rightsquigarrow y) \hookrightarrow El\ x \to El\ y$

$o : Set$

$El\ o \hookrightarrow Prop$

All those constants and rewrite rules define the encoding of higher-order intuitionistic logic in $\lambda\Pi/\mathcal{R}$, written $\mathbb{T}^{\mathsf{IL}}_{\mathsf{hol}}$. The encoding of higher-order classical logic, written $\mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$, is obtained from $\mathbb{T}^{\mathsf{IL}}_{\mathsf{hol}}$ by additionally considering the constant

$\mathsf{pem} : \Pi p : Prop.\ Prf\ (p \vee \neg p)$

representing the principle of excluded middle.

9.1.2 Theories Encoded in Higher-Order Logic

When working with the encoding of higher-order logic in $\lambda\Pi/\mathcal{R}$, it is possible to mix sorts, propositions and proofs. For example, propositions can be inserted in sorts when we have a term of type $Prop \to Set$, and proofs can be inserted in propositions when we have a term of type $Prf\ p \to Prop$ for some proposition p. Such a behavior is not allowed in higher-order logic. To avoid it, we introduce a stratification on types, kinds and Kind.

Definition 9.1: κ-Stratification

The κ-stratification is introduced by the following grammars:

$$\begin{aligned}
\kappa_1 &::= Set \mid \kappa_1 \to \kappa_1 \\
\kappa_2 &::= Prop \mid El\ a \mid \Pi x : \kappa_i.\ \kappa_2 \text{ with } i \in \{1,2\} \\
\kappa_3 &::= Prf\ p \mid \kappa_3 \to \kappa_3 \mid \Pi x : \kappa_i.\ \kappa_3 \text{ with } i \in \{1,2\} \\
\kappa_4 &::= \mathsf{Type} \mid \Pi x : \kappa_i.\ \kappa_4 \text{ with } i \in \{1,2\} \\
\kappa_5 &::= \mathsf{Kind}
\end{aligned}$$

In the grammar κ_2, the case $Prop$ is a particular case of $El\ a$, as $Prop \hookrightarrow El\ o$. The grammar κ_3 generates formulas and inference rules. The grammar κ_4 generates a subclass of kinds, which contains in particular the types of $Prop$, Set, Prf and El.

We characterize the judgments of $\lambda\Pi/\mathcal{R}$ to ensure that types and kinds are generated by one of those grammars.

Definition 9.2: κ-Property

The judgment $\Gamma \vdash_{\mathbb{T}} t : T$ satisfies the κ-property when $T \in \kappa_i$ for some $i \in [\![1,5]\!]$.
The judgment $\vdash_{\mathbb{T}} \Gamma$ satisfies the κ-property when, for each $x : A \in \Gamma$, we have $A \in \kappa_i$ for some $i \in [\![1,3]\!]$.
A derivation satisfies the κ-property when all its judgments satisfy the κ-property.

Let us make two observations when we consider derivations that satisfy the κ-property.

First, it is impossible to have propositions in sorts, proofs in sorts, and proofs in propositions, as types $Prop \to Set$, $Prf\ p \to Set$ and $Prf\ p \to Prop$ are prohibited.

Second, the rewrite rules $\ell \hookrightarrow r$ with ℓ and r of type $A \in \kappa_3$ cannot be used for converting using Conv-Type or Conv-Kind, as ℓ and r cannot occur in types and kinds that satisfy the κ-property. For instance, the term top_i is of type $Prf\ \top \in \kappa_3$. The term top_i cannot occur in types generated by κ_1, κ_2 and κ_3, or in kinds generated by κ_4, as this would break the κ-property. In the rest of this chapter and without loss of generality, we will not consider rewrite rules $\ell \hookrightarrow r$ with ℓ and r of type $A \in \kappa_i$ for $i \neq 3$.

Theories encoded in higher-order logic are theories that feature the base higher-order encoding, and in which the user-defined constants satisfy the κ-property.

Definition 9.3: Theory Encoded in Higher-Order Logic

Let $\mathbb{T}$ be a theory in $\lambda\Pi/\mathcal{R}$. $\mathbb{T}$ is encoded in higher-order logic when:

1. $\mathbb{T} = \mathbb{T}^{\mathsf{L}}_{\mathsf{hol}}, \mathbb{T}_{\mathsf{enc}}$ with $\mathsf{L} \in \{\mathsf{CL}, \mathsf{IL}\}$ and $\mathsf{dom}(\mathbb{T}^{\mathsf{L}}_{\mathsf{hol}}) \cap \mathsf{dom}(\mathbb{T}_{\mathsf{enc}}) = \emptyset$,
2. for every $c : A \in \mathbb{T}_{\mathsf{enc}}$, the judgment $\vdash c : A$ satisfies the κ-property,
3. for every $a : K \in \mathbb{T}_{\mathsf{enc}}$, the judgment $\vdash a : K$ satisfies the κ-property,
4. for every $\ell \hookrightarrow r \in \mathcal{R}_{\mathbb{T}}$, all the occurrences of Prf and $\forall$ in ℓ are applied to their arguments.

The fourth condition will ensure that the translation of a rewrite rule is a well-defined rewrite rule, as we will see in Section 9.2.

Remark 9.4. *Theories encoded in higher-order logic extend higher-order logic with additional user-defined constants, rewrite rules and inference rules. The introduction of rewrite rules to a logic was originally defined in deduction modulo theory [DHK03], while the introduction of inference rules has been developed in superdeduction modulo theory [BHK07, Hou10].*

In the rest of this chapter, we only consider theories that are encoded in higher-order logic, following Definition 9.3. The user-defined constants of $\mathbb{T}_{\mathsf{enc}}$ satisfy the κ-property. Moreover, the constants of $\mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$ and $\mathbb{T}^{\mathsf{IL}}_{\mathsf{hol}}$ directly satisfy the κ-property. But that is not enough to ensure that all the derivations satisfy the κ-property. It is still possible to mix sorts, propositions and proofs through variables or λ-abstractions. For example, the term $(\lambda P : Prop.\ o)$ takes as input a proposition and returns a sort. Hence the type $El\ ((\lambda P : Prop.\ o)\ \bot)$ mixes propositions and sorts. But that is artificial, as this type is β-convertible to $El\ o$, in which no proposition occurs. In the rest of this chapter, we restrict ourselves to variables and λ-abstractions that satisfy the κ-property. As the κ-property is stable under term application, this is sufficient to guarantee that the derivations inside theories encoded in higher-order logic satisfy the κ-property.

The two examples of Section 3.3.2 can be represented as theories encoded in higher-order logic in $\lambda\Pi/\mathcal{R}$.

Example 9.5 (Polymorphic equality)**.** *In higher-order logic, sorts are represented as terms of type Set. It is therefore easy to encode a polymorphic equality.*

$= : \Pi a : Set.\ El\ a \to El\ a \to Prop$

$\mathsf{refl} : \Pi a : Set.\ \Pi x : El\ a.\ Prf\ (=\ a\ x\ x)$

$\mathsf{leib} : \Pi a : Set.\ \Pi x, y : El\ a.\ Prf\ (=\ a\ x\ y) \to \Pi P : El\ a \to Prop.\ Prf\ (P\ x) \to Prf\ (P\ y)$

Here reflexivity and Leibniz's law are encoded by inference rules. It is also possible to encode them by higher-order formulas. For instance, we could have taken refl *of type $\Pi a : Set.\ Prf\ (\forall\ a\ (\lambda x.\ =\ a\ x\ x))$. These two representations are equivalent, using the natural deduction rules of $\forall$. The two types would be convertible if we had considered the computational encoding of $\forall$ instead of the deductive encoding.*

This is a theory encoded in higher-order logic, following Definition 9.3.

From Leibniz's law, we can prove that the equality is indeed symmetric and transitive. For instance, the proof term

$$\lambda a : Set.\ \lambda x, y : El\ a.\ \lambda H_{xy} : Prf\ (=\ a\ x\ y).\ \mathsf{leib}\ a\ x\ y\ H_{xy}\ (\lambda z.\ =\ a\ z\ x)\ (\mathsf{refl}\ a\ x)$$

is of type $\Pi a : Set.\ \Pi x, y : El\ a.\ Prf\ (=\ a\ x\ y) \to Prf\ (=\ a\ y\ x)$, and thus proves symmetry.

Example 9.6 (Natural numbers)**.** *We adapt Example 8.20 so that it is a theory encoded in higher-order logic in $\lambda\Pi/\mathcal{R}$. Natural numbers are now represented by a sort.*

$\mathsf{nat} : Set$

$0 : El\ \mathsf{nat}$

$\mathsf{succ} : El\ \mathsf{nat} \to El\ \mathsf{nat}$

$+ : El\ \mathsf{nat} \to El\ \mathsf{nat} \to El\ \mathsf{nat}$

The rewrite rules $x + 0 \hookrightarrow x$ and $x + \mathsf{succ}\ y \hookrightarrow \mathsf{succ}\ (x + y)$ are unchanged.

Moreover, the induction axiom can be represented as an inference rule

$$\begin{array}{rl} \mathsf{rec} : & \Pi P : El\ \mathsf{nat} \to Prop.\ Prf\ (P\ 0) \to \\ & \big(\Pi x : El\ \mathsf{nat}.\ Prf\ (P\ x) \to Prf\ (P\ (\mathsf{succ}\ x))\big) \to \\ & \Pi x : El\ \mathsf{nat}.\ Prf\ (P\ x) \end{array}$$

or as a higher-order formula

$$\mathsf{rec} : Prf\ \Big(\forall\ (\mathsf{nat} \rightsquigarrow o)\ (\lambda P.\ \big(P\ 0 \wedge (\forall\ \mathsf{nat}\ (\lambda n.\ P\ n \Rightarrow P\ (\mathsf{succ}\ n)))\big) \Rightarrow \big(\forall\ \mathsf{nat}\ (\lambda n.\ P\ n)\big)\Big).$$

These two representations are equivalent, using the natural deduction rules of $\Rightarrow$ and $\forall$.

9.2 Kuroda's Translation for the $\lambda\Pi$-Calculus Modulo Rewriting

In Chapter 4, we defined Kuroda's translation in two steps: an inductive translation $t \mapsto t_{\mathsf{ku}}$ inserts double negations after universal quantifiers, and then $A \mapsto A^{\mathsf{ku}}$ inserts double negations in front of formulas. When working inside a theory encoded in

higher-order logic in $\lambda\Pi/\mathcal{R}$, every formula has head symbol Prf. In that respect, we only need to define a single translation $t \mapsto t^{\mathsf{ku}}$ by induction on the terms of $\lambda\Pi/\mathcal{R}$, that inserts double negations after the Prf and $\forall$ symbols.

As we are in $\lambda\Pi/\mathcal{R}$ with the Curry-Howard correspondence, proofs are terms, and we have to translate them as well. Kuroda's translation relies on the fact that the translation of each natural deduction rule is admissible in intuitionistic logic. In $\lambda\Pi/\mathcal{R}$, this means that each constant c of type A representing a natural deduction rule must be translated by a term c^{IL} of type A^{ku}, where c^{IL} is an intuitionistic proof term of A^{ku}.

For instance, in the standard Kuroda's translation, we have to prove that the translation of the inference rule Imp-I

$$\frac{\Gamma^{\mathsf{ku}}, P^{\mathsf{ku}} \vdash Q^{\mathsf{ku}}}{\Gamma^{\mathsf{ku}} \vdash (P \Rightarrow Q)^{\mathsf{ku}}}$$

is admissible in intuitionistic logic. In $\lambda\Pi/\mathcal{R}$, the *constant* $\mathsf{imp_i}$ is of type $\Pi p, q : Prop.\ (Prf\ p \to Prf\ q) \to Prf\ (p \Rightarrow q)$. We must build a *term* $\mathsf{imp_i}^{\mathsf{IL}}$ of type $\Pi p, q : Prop.\ (Prf\ (\neg\neg p) \to Prf\ (\neg\neg q)) \to Prf\ (\neg\neg(p \Rightarrow q))$, that only depends on the constants representing the *intuitionistic* natural deduction rules.

Definition 9.7: Translation of Terms

Kuroda's translation is inductively defined on the terms of $\lambda\Pi/\mathcal{R}$ by:

$$
\begin{array}{lcll}
x^{\mathsf{ku}} & := & x & \\
c^{\mathsf{ku}} & := & \begin{cases} \lambda x.\ \lambda p.\ \forall\ x\ (\lambda z.\ \neg\neg(p\ z)) & \text{if } c = \forall \\ c^{\mathsf{IL}} & \text{if } c \text{ is a natural deduction rule} \\ c & \text{otherwise} \end{cases} & \\
a^{\mathsf{ku}} & := & \begin{cases} \lambda p.\ Prf\ (\neg\neg p) & \text{if } a = Prf \\ a & \text{otherwise} \end{cases} & \\
(M\ N)^{\mathsf{ku}} & := & M^{\mathsf{ku}}\ N^{\mathsf{ku}} & \\
(A\ M)^{\mathsf{ku}} & := & A^{\mathsf{ku}}\ M^{\mathsf{ku}} & \\
(\lambda x : A.\ M)^{\mathsf{ku}} & := & \lambda x : A^{\mathsf{ku}}.\ M^{\mathsf{ku}} & \\
(\lambda x : A.\ B)^{\mathsf{ku}} & := & \lambda x : A^{\mathsf{ku}}.\ B^{\mathsf{ku}} & \\
(\Pi x : A.\ B)^{\mathsf{ku}} & := & \Pi x : A^{\mathsf{ku}}.\ B^{\mathsf{ku}} & \\
(\Pi x : A.\ K)^{\mathsf{ku}} & := & \Pi x : A^{\mathsf{ku}}.\ K^{\mathsf{ku}} & \\
\mathsf{Type}^{\mathsf{ku}} & := & \mathsf{Type} & \\
\mathsf{Kind}^{\mathsf{ku}} & := & \mathsf{Kind} &
\end{array}
$$

when for each constant $c : A \in \mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$ representing a natural deduction rule, there exists a term c^{IL} such that $\vdash_{\mathbb{T}^{\mathsf{IL}}_{\mathsf{hol}}} c^{\mathsf{IL}} : A^{\mathsf{ku}}$.

We give the parameters c^{IL} in the following proposition.

Proposition 9.8

For every constant $c : A \in \mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$ representing a natural deduction rule, there exists a term c^{IL} such that $\vdash_{\mathbb{T}^{\mathsf{IL}}_{\mathsf{hol}}} c^{\mathsf{IL}} : A^{\mathsf{ku}}$.

Proof. It suffices to encode in $\lambda\Pi/\mathcal{R}$ the proof of Theorem 4.7. In particular, the term $\mathsf{pem}^{\mathsf{IL}}$ of type $\Pi p : Prop.\ Prf\ (\neg\neg(p \vee \neg p))$ is given by

$$\begin{array}{l}\lambda p : Prop.\ \mathsf{neg_i}\ \neg(p \vee \neg p)\\ \quad (\lambda H : Prf\ \neg(p \vee \neg p).\ \mathsf{neg_e}\ (p \vee \neg p)\ H\\ \quad\quad (\mathsf{or_{ir}}\ p\ \neg p\\ \quad\quad\quad (\mathsf{neg_i}\ p\ (\lambda H_p : Prf\ p.\ \mathsf{neg_e}\ (p \vee \neg p)\ H\ (\mathsf{or_{i\ell}}\ p\ \neg p\ H_p)))))\end{array}$$

which directly encodes in $\lambda\Pi/\mathcal{R}$ the proof tree given in Proposition 3.9.

We have formalized all the proof terms c^{IL} in Dedukti[1]. □

Remark 9.9. *Assume that the natural deduction rules are encoded using rewrite rules. For instance, the rewrite rule for implication is $Prf\ (p \Rightarrow q) \hookrightarrow Prf\ p \to Prf\ q$, and the rewrite rule for negation is $Prf\ (\neg p) \hookrightarrow Prf\ p \to Prf\ \bot$. Kuroda's translation would require the conversion $(Prf\ (p \Rightarrow q))^{\mathsf{ku}} \equiv_{\beta\mathcal{R}} (Prf\ p \to Prf\ q)^{\mathsf{ku}}$. However, this does not hold:*

$$\begin{array}{lll}(Prf\ (p \Rightarrow q))^{\mathsf{ku}} & \equiv_{\beta\mathcal{R}} & Prf\ (\neg\neg(p \Rightarrow q))\\ & \equiv_{\beta\mathcal{R}} & (Prf\ (p \Rightarrow q) \to Prf\ \bot) \to Prf\ \bot\\ & \equiv_{\beta\mathcal{R}} & ((Prf\ p \to Prf\ q) \to Prf\ \bot) \to Prf\ \bot\\[1em] (Prf\ p \to Prf\ q)^{\mathsf{ku}} & \equiv_{\beta\mathcal{R}} & Prf\ (\neg\neg p) \to Prf\ (\neg\neg q)\\ & \equiv_{\beta\mathcal{R}} & ((Prf\ p \to Prf\ \bot) \to Prf\ \bot) \to ((Prf\ q \to Prf\ \bot) \to Prf\ \bot)\end{array}$$

The translation works when the natural deduction steps are explicit deduction steps, but not when they are implicit computation steps. That is why we encoded the natural deduction rules using typed constants rather than rewrite rules.

Definition 9.10

Kuroda's translation is inductively defined on contexts and substitutions by:

$$\begin{array}{lll}\varnothing^{\mathsf{ku}} & := & \varnothing\\ (\Gamma, x : A)^{\mathsf{ku}} & := & \Gamma^{\mathsf{ku}}, x : A^{\mathsf{ku}}\\ (\theta, x \leftarrow M)^{\mathsf{ku}} & := & \theta^{\mathsf{ku}}, x \leftarrow M^{\mathsf{ku}}\end{array}$$

We have defined the translation for terms, and we now want to define it for theories. Intuitively, we would like to translate rewrite rules $\ell \hookrightarrow r$ by $\ell^{\mathsf{ku}} \hookrightarrow r^{\mathsf{ku}}$. As defined in Section 8.1, the left-hand side ℓ of rewrite rules must be algebraic but not a variable. We recall that a term is called algebraic when it is a variable or the application of a constant to algebraic arguments. In particular, if ℓ contains Prf or $\forall$, then ℓ^{ku} contains Prf^{ku} or $\forall^{\mathsf{ku}}$, that are λ-abstractions and not constants. Hence $\ell^{\mathsf{ku}} \hookrightarrow r^{\mathsf{ku}}$ may not be a valid rewrite rule in $\lambda\Pi/\mathcal{R}$. By Definition 9.3, all the occurrences of Prf and $\forall$ in ℓ are applied to their arguments. Hence it suffices to β-reduce ℓ^{ku} to obtain an algebraic term. We write $\lfloor \ell^{\mathsf{ku}} \rfloor$ for the term obtained by β-reducing the symbols Prf^{ku} and $\forall^{\mathsf{ku}}$ in ℓ^{ku}.

[1] See https://github.com/Deducteam/Construkti/blob/master/kuroda.dk.

Definition 9.11: Translation of Theories

Kuroda's translation is inductively defined on theories encoded in higher-order logic in $\lambda\Pi/\mathcal{R}$ by:

$$
\begin{array}{lcl}
\varnothing^{\mathsf{ku}} & := & \varnothing \\
(\mathbb{T}, c : A)^{\mathsf{ku}} & := & \begin{cases} \mathbb{T}^{\mathsf{ku}}, c : A & \text{if } c : A \in \mathbb{T}_{\mathsf{hol}} \text{ and } c \neq \mathsf{pem} \\ \mathbb{T}^{\mathsf{ku}} & \text{if } c = \mathsf{pem} \\ \mathbb{T}^{\mathsf{ku}}, c : A^{\mathsf{ku}} & \text{if } c : A \in \mathbb{T}_{\mathsf{enc}} \end{cases} \\
(\mathbb{T}, a : K)^{\mathsf{ku}} & := & \begin{cases} \mathbb{T}^{\mathsf{ku}}, a : K & \text{if } a : K \in \mathbb{T}_{\mathsf{hol}} \\ \mathbb{T}^{\mathsf{ku}}, a : K^{\mathsf{ku}} & \text{if } a : K \in \mathbb{T}_{\mathsf{enc}} \end{cases} \\
(\mathbb{T}, M \hookrightarrow N)^{\mathsf{ku}} & := & \mathbb{T}^{\mathsf{ku}}, \lfloor M^{\mathsf{ku}} \rfloor \hookrightarrow N^{\mathsf{ku}} \\
(\mathbb{T}, A \hookrightarrow B)^{\mathsf{ku}} & := & \mathbb{T}^{\mathsf{ku}}, \lfloor A^{\mathsf{ku}} \rfloor \hookrightarrow B^{\mathsf{ku}}
\end{array}
$$

The classical encoding $\mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$ is translated into the intuitionistic encoding $\mathbb{T}^{\mathsf{IL}}_{\mathsf{hol}}$, and therefore $\mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}, \mathbb{T}_{\mathsf{enc}}$ is translated into $\mathbb{T}^{\mathsf{IL}}_{\mathsf{hol}}, \mathbb{T}^{\mathsf{ku}}_{\mathsf{enc}}$.

9.3 Soundness Property

We aim at proving that the extension of Kuroda's translation to $\lambda\Pi/\mathcal{R}$ indeed embeds classical logic into intuitionistic logic. In other words, we want to show that $\Gamma \vdash_{\mathbb{T}} t : T$ entails $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} t^{\mathsf{ku}} : T^{\mathsf{ku}}$.

The theorem follows the hierarchy of terms in $\lambda\Pi/\mathcal{R}$, as there are items for objects, types and kinds. Additionally, there is an item for well-formed contexts. As the theory $\mathbb{T}$ is translated as well, there is an item for well-formed theories.

We restrict ourselves to theories $\mathbb{T}$ in which rewrite rules $\ell \hookrightarrow r$ such that ℓ and r are well-typed with the same type.

Theorem 9.12: Judgment Preservation

Let $\mathbb{T}$ be a theory encoded in higher-order logic.

1. If $\mathbb{T} \vdash$ then $\mathbb{T}^{\mathsf{ku}} \vdash$.
2. If $\vdash_{\mathbb{T}} \Gamma$ then $\vdash_{\mathbb{T}^{\mathsf{ku}}} \Gamma^{\mathsf{ku}}$.
3. If $\Gamma \vdash_{\mathbb{T}} M : A$ then $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} M^{\mathsf{ku}} : A^{\mathsf{ku}}$.
4. If $\Gamma \vdash_{\mathbb{T}} A : K$ then $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} A^{\mathsf{ku}} : K^{\mathsf{ku}}$.
5. If $\Gamma \vdash_{\mathbb{T}} K : \mathsf{Kind}$ then $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} K^{\mathsf{ku}} : \mathsf{Kind}$.

This theorem relies on the fact that the translation commutes with substitution and preserves conversion.

Proposition 9.13

Let θ be a substitution. We have:

1. $(M\theta)^{\mathsf{ku}} = M^{\mathsf{ku}}\theta^{\mathsf{ku}}$
2. $(A\theta)^{\mathsf{ku}} = A^{\mathsf{ku}}\theta^{\mathsf{ku}}$
3. $(K\theta)^{\mathsf{ku}} = K^{\mathsf{ku}}\theta^{\mathsf{ku}}$

Proof. By induction on terms.

— Constant and Type: We have $(c\theta)^{\mathsf{ku}} = c^{\mathsf{ku}} = c^{\mathsf{ku}}\theta^{\mathsf{ku}}$ since c and c^{ku} are closed terms. Similarly, $(a\theta)^{\mathsf{ku}} = a^{\mathsf{ku}}\theta^{\mathsf{ku}}$ and $(\mathsf{Type}\theta)^{\mathsf{ku}} = \mathsf{Type}^{\mathsf{ku}}\theta^{\mathsf{ku}}$.

— Variable: If θ does not substitute x, then $(x\theta)^{\mathsf{ku}} = x^{\mathsf{ku}} = x^{\mathsf{ku}}\theta^{\mathsf{ku}}$.

If θ substitutes x by w, then $(x\theta)^{\mathsf{ku}} = w^{\mathsf{ku}} = x\theta^{\mathsf{ku}} = x^{\mathsf{ku}}\theta^{\mathsf{ku}}$.

— Dependent type: We have

$$\begin{array}{rcll} ((\Pi x : A.\ B)\theta)^{\mathsf{ku}} & = & (\Pi x : A\theta.\ B\theta)^{\mathsf{ku}} & \\ & = & \Pi x : (A\theta)^{\mathsf{ku}}.\ (B\theta)^{\mathsf{ku}} & \\ & = & \Pi x : A^{\mathsf{ku}}\theta^{\mathsf{ku}}.\ B^{\mathsf{ku}}\theta^{\mathsf{ku}} & \text{by induction hypotheses} \\ & = & (\Pi x : A^{\mathsf{ku}}.\ B^{\mathsf{ku}})\theta^{\mathsf{ku}} & \\ & = & (\Pi x : A.\ B)^{\mathsf{ku}}\theta^{\mathsf{ku}} & \end{array}$$

Similarly, we have $((\Pi x : A.\ K)\theta)^{\mathsf{ku}} = (\Pi x : A.\ K)^{\mathsf{ku}}\theta^{\mathsf{ku}}$.

— The cases for dependent type $\Pi x : A.\ K$, for λ-abstraction $\lambda x : A.\ M$ and $\lambda x : A.\ B$, and for application $M\ N$ and $A\ M$ are treated similarly and follow from the induction hypotheses. □

Proposition 9.14

1. For any types A and B, if $A \equiv_{\beta(\eta)\mathcal{R}} B$ in $\mathbb{T}$ then $A^{\mathsf{ku}} \equiv_{\beta(\eta)\mathcal{R}} B^{\mathsf{ku}}$ in $\mathbb{T}^{\mathsf{ku}}$.
2. For any kinds K_1 and K_2, if $K_1 \equiv_{\beta(\eta)\mathcal{R}} K_2$ in $\mathbb{T}$ then $K_1^{\mathsf{ku}} \equiv_{\beta(\eta)\mathcal{R}} K_2^{\mathsf{ku}}$ in $\mathbb{T}^{\mathsf{ku}}$.

Proof. By induction on the construction of $\equiv_{\beta(\eta)\mathcal{R}}$.

— Suppose $\ell \hookrightarrow r \in \mathbb{T}$. Let us show $(\ell\theta)^{\mathsf{ku}} \equiv_{\beta\mathcal{R}} (r\theta)^{\mathsf{ku}}$ in $\mathbb{T}^{\mathsf{ku}}$ for any substitution θ.

If $\ell \hookrightarrow r \in \mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$, then $\ell^{\mathsf{ku}} = \ell$ and $r^{\mathsf{ku}} = r$. We also have $\ell \hookrightarrow r \in \mathbb{T}^{\mathsf{IL}}_{\mathsf{hol}}$. We use Proposition 9.13.

If $\ell \hookrightarrow r \in \mathbb{T}_{\mathsf{enc}}$, then we have $\lfloor \ell^{\mathsf{ku}} \rfloor \hookrightarrow r^{\mathsf{ku}} \in \mathbb{T}^{\mathsf{ku}}_{\mathsf{enc}}$. Using Proposition 9.13, we derive $(\ell\theta)^{\mathsf{ku}} = \ell^{\mathsf{ku}}\theta^{\mathsf{ku}} \equiv_{\beta\mathcal{R}} \lfloor \ell^{\mathsf{ku}} \rfloor \theta^{\mathsf{ku}} \equiv_{\beta\mathcal{R}} r^{\mathsf{ku}}\theta^{\mathsf{ku}} = (r\theta)^{\mathsf{ku}}$.

— We have $((\lambda x : A.\ M)\ N)^{\mathsf{ku}} = (\lambda x : A^{\mathsf{ku}}.\ M^{\mathsf{ku}})\ N^{\mathsf{ku}}$, which β-reduces to $M^{\mathsf{ku}}[x \leftarrow N^{\mathsf{ku}}]$, that is $(M[x \leftarrow N])^{\mathsf{ku}}$ using Proposition 9.13. Similarly, we have $((\lambda x : A.\ B)\ M)^{\mathsf{ku}} \equiv_{\beta\mathcal{R}} (B[x \leftarrow M])^{\mathsf{ku}}$.

— Suppose that x is free in M. We have $(\lambda x : A.\ M\ x)^{\mathsf{ku}} = \lambda x : A^{\mathsf{ku}}.\ M^{\mathsf{ku}}\ x$, which is η-convertible to M^{ku}. Similarly, we have $(\lambda x : A.\ B\ x)^{\mathsf{ku}} \equiv_{\beta\eta\mathcal{R}} B^{\mathsf{ku}}$.

— The cases for application, λ-abstraction and dependent type directly follow from the induction hypotheses.

— Reflexivity, symmetry, and transitivity are immediate. □

Proof of Theorem 9.12. Assume that the first item holds for a fixed theory $\mathbb{T}$, and let us prove the other items for $\mathbb{T}$ by induction on the typing derivation.

— Const-Obj (case 3): By induction hypothesis, we have $\vdash_{\mathbb{T}^{\mathsf{ku}}} \Gamma^{\mathsf{ku}}$.

If $c : A \in \mathbb{T}_{\mathsf{enc}}$, then $c : A^{\mathsf{ku}} \in \mathbb{T}^{\mathsf{ku}}_{\mathsf{enc}}$, and we derive $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} c : A^{\mathsf{ku}}$ using Const-Obj.

If $c : A \in \mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$ and is neither $\forall$ nor a constant representing a natural deduction rule, then $c : A \in \mathbb{T}^{\mathsf{ku}}$. We know that A does not contain Prf and $\forall$, so $A^{\mathsf{ku}} = A$. We derive $\Gamma^{\mathsf{ku}} \vdash c : A^{\mathsf{ku}}$ using Const-Obj.

If $c = \forall$, then c^{ku} corresponds to the term $\lambda x.\ \lambda p.\ \forall\ x\ (\lambda z.\ \neg\neg(p\ z))$. We easily derive $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} \lambda x.\ \lambda p.\ \forall\ x\ (\lambda z.\ \neg\neg(p\ z)) : \Pi x : Set.\ (El\ x \to Prop) \to Prop$, that is $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} \forall^{\mathsf{ku}} : (\Pi x : Set.\ (El\ x \to Prop) \to Prop)^{\mathsf{ku}}$.

If c represents a natural deduction rule, then we have $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} c^{\mathsf{IL}} : A^{\mathsf{ku}}$ using Proposition 9.8.

— Const-Type (case 4): By induction hypothesis, we have $\vdash_{\mathbb{T}^{\mathsf{ku}}} \Gamma^{\mathsf{ku}}$.

If $a : K \in \mathbb{T}_{\mathsf{enc}}$, then $a : K^{\mathsf{ku}} \in \mathbb{T}^{\mathsf{ku}}_{\mathsf{enc}}$, and we derive $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} a : K^{\mathsf{ku}}$ using Const-Type.

If $a : K \in \mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$ and is not Prf, then $a : K \in \mathbb{T}^{\mathsf{ku}}$. We know that K does not contain Prf and $\forall$, so $K^{\mathsf{ku}} = K$. We derive $\Gamma^{\mathsf{ku}} \vdash a : K^{\mathsf{ku}}$ using Const-Type.

If $c = Prf$, then Prf^{ku} is the term $\lambda p.\ Prf\ (\neg\neg p)$. We easily derive $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} \lambda p.\ Prf\ (\neg\neg p) : Prop \to \mathsf{Type}$, that is $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} Prf^{\mathsf{ku}} : (Prop \to \mathsf{Type})^{\mathsf{ku}}$.

— Conv-Type (case 3): We have $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} M^{\mathsf{ku}} : A^{\mathsf{ku}}$ and $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} B^{\mathsf{ku}} : \mathsf{Type}$ by induction hypotheses. From Proposition 9.14, we have $A^{\mathsf{ku}} \equiv_{\beta(\eta)\mathcal{R}} B^{\mathsf{ku}}$. We conclude that $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} M^{\mathsf{ku}} : B^{\mathsf{ku}}$ using Conv-Type.

— Conv-Kind (case 4): Similar to Conv-Type.

— For the other cases, we directly apply the typing rule in question on the induction hypotheses.

We have proved that, assuming that the first item holds for a fixed $\mathbb{T}$, then the other items hold for that $\mathbb{T}$. Let us now prove the first item by induction on the well-formedness derivation.

— Empty-Thy: as $\varnothing^{\mathsf{ku}} := \varnothing$, we directly have $\varnothing \vdash$ using Empty-Thy.

— Decl-Obj: Suppose $\mathbb{T} \vdash$ and $\vdash_{\mathbb{T}} A : \mathsf{Type}$. By induction hypothesis, we have $\mathbb{T}^{\mathsf{ku}} \vdash$.

If $c : A \in \mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$ and $c \neq \mathsf{pem}$, then we directly have $\vdash_{\mathbb{T}^{\mathsf{ku}}} A : \mathsf{Type}$. We derive $T^{\mathsf{ku}}, c : A \vdash$ using Decl-Obj.

If $c = \mathsf{pem}$, then $(\mathbb{T}, c : A)^{\mathsf{ku}} := \mathbb{T}^{\mathsf{ku}}$, so we have nothing to do.

If $c : A \in \mathbb{T}_{\mathsf{enc}}$, then we have $\vdash_{\mathbb{T}^{\mathsf{ku}}} A^{\mathsf{ku}} : \mathsf{Type}$ using the fourth item of the theorem. We derive $\mathbb{T}^{\mathsf{ku}}, c : A^{\mathsf{ku}} \vdash$ using Decl-Obj.

— Decl-Type: Suppose $\mathbb{T} \vdash$ and $\vdash_{\mathbb{T}} K : \mathsf{Kind}$. By induction hypothesis, we have $\mathbb{T}^{\mathsf{ku}} \vdash$.

If $a : K \in \mathbb{T}^{\mathsf{CL}}_{\mathsf{hol}}$, then we directly have $\vdash_{\mathbb{T}^{\mathsf{ku}}} K : \mathsf{Kind}$. We derive $T^{\mathsf{ku}}, a : K \vdash$ using Decl-Type.

If $a : K \in \mathbb{T}_{\mathsf{enc}}$, then we have $\vdash_{\mathbb{T}^{ku}} K^{\mathsf{ku}} : \mathsf{Kind}$ using the fifth item of the theorem. We derive $\mathbb{T}^{\mathsf{ku}}, a : K^{\mathsf{ku}} \vdash$ using Decl-Type.

— Rewr-Obj: Suppose $\mathbb{T} \vdash$ and $\mathbb{T} \vdash M \hookrightarrow N$. By induction hypothesis, we have $\mathbb{T}^{\mathsf{ku}} \vdash$. In this chapter, $\mathbb{T} \vdash M \hookrightarrow N$ means that there exist some Γ and some A such that $\Gamma \vdash_{\mathbb{T}} M : A$ and $\Gamma \vdash_{\mathbb{T}} N : A$. We have $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} M^{\mathsf{ku}} : A^{\mathsf{ku}}$ and $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} N^{\mathsf{ku}} : A^{\mathsf{ku}}$ using the fourth item of the theorem. We derive $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} \lfloor M^{\mathsf{ku}} \rfloor : A^{\mathsf{ku}}$ by subject reduction. Thus $\mathbb{T}^{\mathsf{ku}} \vdash \lfloor M^{\mathsf{ku}} \rfloor \hookrightarrow N^{\mathsf{ku}}$.

— Rewr-Type: We proceed as for Rewr-Obj, referring to the fifth item of the theorem instead of the fourth. □

Remark 9.15. *The extension of Kuroda's translation to $\lambda\Pi/\mathcal{R}$ is a generalization of Brown and Rizkallah's translation for simple type theory [BR14]. Indeed, if $\mathcal{R}_{\mathbb{T}} = \varnothing$, then we obtain the result in higher-order logic, at the only difference that proofs are represented by terms.*

Example 9.16. *The translation of the theory encoding polymorphic equality (defined in Example 9.5) is given by:*

$$
\begin{aligned}
&= \ : \Pi a : \mathit{Set}.\ \mathit{El}\ a \to \mathit{El}\ a \to \mathit{Prop}\\
&\mathsf{refl} : \Pi a : \mathit{Set}.\ \Pi x : \mathit{El}\ a.\ \mathit{Prf}\ (\neg\neg(=\ a\ x\ x))\\
&\mathsf{leib} : \Pi a : \mathit{Set}.\ \Pi x, y : \mathit{El}\ a.\ \mathit{Prf}\ (\neg\neg(=\ a\ x\ y)) \to\\
&\qquad \Pi P : \mathit{El}\ a \to \mathit{Prop}.\ \mathit{Prf}\ (\neg\neg(P\ x)) \to \mathit{Prf}\ (\neg\neg(P\ y))
\end{aligned}
$$

The equality symbol remains unchanged, while double negations are inserted in refl *and* leib*.*

The theorem stating that the equality is symmetric and its proof are translated as follows: the proof term

$$\lambda a : \mathit{Set}.\ \lambda x, y : \mathit{El}\ a.\ \lambda H_{xy} : \mathit{Prf}\ (\neg\neg(=\ a\ x\ y)).\ \mathsf{leib}\ a\ x\ y\ H_{xy}\ (\lambda z.\ =\ a\ z\ x)\ (\mathsf{refl}\ a\ x)$$

is of type $\Pi a : \mathit{Set}.\ \Pi x, y : \mathit{El}\ a.\ \mathit{Prf}\ (\neg\neg(=\ a\ x\ y)) \to \mathit{Prf}\ (\neg\neg(=\ a\ y\ x))$*.*

Note that, as seen in Section 4.2, we can go further: instead of taking the translated axioms refl *and* leib*, we can prove them in intuitionistic logic using the original* refl *and* leib*.*

9.4 Characterization Property

In the previous section, we have shown the soundness property of Kuroda's translation: if $\Gamma \vdash_{\mathbb{T}} t : A$ then $\Gamma^{\mathsf{ku}} \vdash_{\mathbb{T}^{\mathsf{ku}}} t^{\mathsf{ku}} : A^{\mathsf{ku}}$. We must prove the characterization property, ensuring that the translated results and the original results are equivalent in classical logic.

Let us show that any proposition P and its translation P^{ku} are classically equivalent. As seen in Section 4.3, such a result is not necessarily true in higher-order logic, and the Kuroda-equivalence is a necessary and sufficient condition.

Definition 9.17: Kuroda-Equivalence

Let $\Gamma \vdash M : A_1 \to \ldots \to A_n \to \mathit{Prop}$ and $\Gamma \vdash N_i : A_i$ for $i \in [\![1, n]\!]$. There exists some H such that $\Gamma \vdash H : \mathit{Prf}\ ((M\ N_1\ \ldots\ N_n)^{\mathsf{ku}} \Leftrightarrow M\ N_1\ \ldots\ N_n)$.

Lemma 9.18. *The β-normal form of a proposition p is either a variable, a constant, or an application $M\ N_1\ \ldots\ N_n$ where M is a constant or a variable of type $A_1 \to \ldots \to A_n \to \mathit{Prop}$ and $N_1, \ldots, N_n$ are terms of type $A_1, \ldots, A_n$.*

Proof. As p is a proposition, it has type Prop and thus cannot be an abstraction. □

The constant may be user-defined, $\top$ or $\bot$. The head symbol of the application may be any connective, quantifier or predicate.

Proposition 9.19

Let $\Gamma \vdash p : \mathit{Prop}$. Under Kuroda-equivalence, there exists some proof term M_p such that $\Gamma \vdash M_p : \mathit{Prf}\ (p^{\mathsf{ku}} \Leftrightarrow p)$.

Note that, as a consequence of the κ-property, the constants occurring in p have type $A \in \kappa_1$ or $A \in \kappa_2$, and therefore $A^{\mathsf{ku}} = A$. Thus it makes sense to express p^{ku} inside the original theory $\mathbb{T}$.

Proof. We distinguish cases thanks to Lemma 9.18.

— Suppose that $p \equiv_{\beta\mathcal{R}} N$ where N is a variable or a constant. It follows that $N^{\mathsf{ku}} = N$. We build the term M_N as

$$\mathsf{and_i}\ (N \Rightarrow N)\ (N \Rightarrow N)\ (\mathsf{imp_i}\ N\ N\ (\lambda H : \mathit{Prf}\ N.\ H))\ (\mathsf{imp_i}\ N\ N\ (\lambda H : \mathit{Prf}\ N.\ H))$$

and we have $\Gamma \vdash M_N : \mathit{Prf}\ (N^{\mathsf{ku}} \Leftrightarrow N)$. We get $p^{\mathsf{ku}} \equiv_{\beta\mathcal{R}} N^{\mathsf{ku}}$ using Proposition 9.14. Thus we derive $\Gamma \vdash M_N : \mathit{Prf}\ (p^{\mathsf{ku}} \Leftrightarrow p)$ using Conv-Type.

— Suppose that $p \equiv_{\beta\mathcal{R}} M\ N_1 \ldots\ N_n$ where M is a constant or a variable. We have $p^{\mathsf{ku}} \equiv_{\beta\mathcal{R}} (M\ N_1 \ldots\ N_n)^{\mathsf{ku}}$ using Proposition 9.14. We directly conclude using the Kuroda-equivalence. □

In the encoding of higher-order logic in $\lambda\Pi/\mathcal{R}$, we not only prove formulas but also inference rules, which are terms of type $A \in \kappa_3$. Therefore, we must also prove the characterization property for terms of type $A \in \kappa_3$.

As we are not mixing sorts, propositions and proofs, we know that the symbol $\forall$ and the symbol Prf only occur in types generated by the grammar κ_3. Moreover, the constants representing the natural deduction rules do not occur in types and kinds generated by the grammars. Therefore, any type $A \in \kappa_i$ is modified by Kuroda's translation for $i = 3$, whereas $A^{\mathsf{ku}} = A$ for $i \neq 3$.

Theorem 9.20

Let $\mathbb{T}$ be a theory encoded in higher-order logic and $A \in \kappa_3$. Let Γ be a context. Under Kuroda-equivalence, there exists some M such that $\Gamma \vdash_{\mathbb{T}} M : A^{\mathsf{ku}}$ if and only if there exists some M' such that $\Gamma \vdash_{\mathbb{T}} M' : A$.

Proof. We proceed by induction on the term A, using the fact that $A \in \kappa_3$.

— Suppose that $A = Prf\ p$. Using Proposition 9.19, we derive that $Prf\ p^{\mathsf{ku}}$ is inhabited if and only if $Prf\ p$ is inhabited. As we are in classical logic, it follows that $Prf\ (\neg\neg p^{\mathsf{ku}})$ is inhabited if and only if $Prf\ p$ is inhabited.

— Suppose that $A = \Pi x : B.\ C$ with $B \in \kappa_1$ or $B \in \kappa_2$. Therefore $B^{\mathsf{ku}} = B$. Suppose $\Gamma \vdash_{\mathbb{T}} M : \Pi x : B^{\mathsf{ku}}.\ C^{\mathsf{ku}}$. We have $\Gamma, x : B \vdash_{\mathbb{T}} M\ x : C^{\mathsf{ku}}$ by weakening and App-Obj. We get some M'_C such that $\Gamma, x : B \vdash_{\mathbb{T}} M'_C : C$ by induction hypothesis on C. Using Abs-Obj, we derive $\Gamma \vdash \lambda x : B.\ M'_C : \Pi x : B.\ C$. We proceed similarly for the reverse implication.

— Suppose that $A = B \to C$ with $B, C \in \kappa_3$. Suppose $\Gamma \vdash_{\mathbb{T}} M : B^{\mathsf{ku}} \to C^{\mathsf{ku}}$. Using Var, we derive $\Gamma, x : B \vdash_{\mathbb{T}} x : B$. By induction hypothesis on B, we get some M_B such that $\Gamma, x : B \vdash_{\mathbb{T}} M_B : B^{\mathsf{ku}}$. Using weakening and App-Obj, we derive $\Gamma, x : B \vdash_{\mathbb{T}} M\ M_B : C^{\mathsf{ku}}$. By induction hypothesis on C, we get some M'_C such that $\Gamma, x : B \vdash_{\mathbb{T}} M'_C : C$. Using Abs-Obj, we derive $\Gamma \vdash_{\mathbb{T}} \lambda x : B.\ M'_C : B \to C$. We proceed similarly for the reverse implication. □

9.5 Implementation for Dedukti

Implementation. We developed a tool, called Construkti[2], that implements Kuroda's translation for Dedukti. Construkti takes as input a Dedukti file containing the specification of a user-defined theory encoded in higher-order logic, as well as proofs in this theory. It outputs a Dedukti file containing the specification of the translated theory, as well as the translated proofs. The resulting file can be rechecked by Dedukti. Taking advantage of the Dedukti kernel and parser, Construkti is written in OCaml in less than 150 lines of code.

So as to obtain readable theorems, Construkti directly β-reduces every application of the λ-abstractions Prf^{ku} and $\forall^{\mathsf{ku}}$.

This implementation relies on the formalization of the parameters c^{IL}, for all the constants c representing a natural deduction rule. For example, Construkti replaces every instance of the principle of excluded middle

```
pem : p : Prop -> Prf (or p (not p)).
```

by the theorem proving the double negation of the principle of excluded middle

```
thm pem_i : p : Prop -> Prf (not (not (or p (not p))))
:= p => neg_i (not (or p (not p)))
    (H => neg_e (or p (not p)) H
        (or_ir p (not p)
            (neg_i p (Hp => neg_e (or p (not p)) H
                (or_il p (not p) Hp))))).
```

which encodes in Dedukti the proof tree given in Proposition 3.9. The complete formalization is available in the file `kuroda.dk`.

[2]Available at https://github.com/Deducteam/Construkti.

Demonstration. Let us illustrate Construkti on a proof of the Clavius's law, which states $(\neg A \Rightarrow A) \Rightarrow A$ for any formula A. We encode Clavius's law as a higher-order formula in Dedukti.

```
(Prf (all o (A => (imp (imp (not A) A) A))))
```

Alternatively, we could represent it as an admissible inference rule, without changing much the rest of the demonstration.

```
A : Prop -> (Prf (imp (imp (not A) A) A))
```

In intuitionistic logic, Clavius's law is derivable from (and actually equivalent to) the principle of excluded middle:

```
thm clavius : (Prf (all o (A => (imp (imp (not A) A) A))))
:= (all_i o
    (A : (El o) => (imp (imp (not A) A) A))
    (A : Prop => (imp_i
        (imp (not A) A)
        A
        (pNAA : (Prf (imp (not A) A)) =>
            (or_e A (not A) (pem A) A
                (pA : (Prf A) => pA)
                (pNA : (Prf (not A)) =>
                    (imp_e (not A) A pNAA pNA)))))).
```

After running Construkti, we obtain a proof of the translated Clavius's law:

```
thm clavius : Prf (not (not (all o (A =>
                        not (not (imp (imp (not A) A) A))))))
:= all_i_i o
    (A : (El o) => imp (imp (not A) A) A)
    (A : Prop => imp_i_i
       (imp (not A) A)
       A
       (pNAA : (Prf (not (not (imp (not A) A)))) =>
           or_e_i A (not A) (pem_i A) A
              (pA : (Prf (not (not A))) => pA)
              (pNA : (Prf (not (not (not A)))) =>
                  imp_e_i (not A) A pNAA pNA))).
```

The translation inserted the double negations after the `Prf` and `all` symbols, and replaced the natural deduction inference rules by the proof terms of their translation. In particular, Construkti replaced the classical inference rule `pem` by the intuitionistic proof term `pem_i`.

Benchmark. We have tested Construkti on a benchmark of 101 Dedukti proofs, available in the file `hol-lib.dk`. These proofs encompass results related to connectives and quantifiers, classical formulas, De Morgan's laws, polymorphic equality, and basic arithmetic. The proofs are expressed in propositional, first-order and higher-order logics. The library makes use of the key features of $\lambda\Pi/\mathcal{R}$, namely dependent types and user-defined rewrite rules. We compare in Table 9.1 the different characteristics of the library: the number of proofs, the number of classical proofs, the number of results

expressed in higher-order logic, and the number of results that are expressed via admissible inference rules (thanks to the encoding of the notions of proposition and proofs). Note that, using the natural deduction rules for $\Rightarrow$ and $\forall$, some higher-order results can be turned into admissible inference rules, and vice versa. Our count does not take this into account.

After running Construkti on `hol-lib.dk`, we obtain the translated library `hol-lib_i.dk`, in which all the proofs are expressed in intuitionistic logic. All the translated proofs of the translated theorems type-check.

9.6 Conclusion

In this chapter, we have extended Kuroda's translation to the theories encoded in higher-logic in $\lambda\Pi/\mathcal{R}$. In other words, we have extended the translation of Chapter 4 to dependent types and user-defined rewrite rules. In this logical framework, proofs are represented as terms following the Curry-Howard correspondence, and have to be effectively translated. We have implemented Construkti, a tool that transforms Dedukti proofs following Kuroda's translation, and we have tested it on a benchmark of a hundred formal proofs.

Kuroda's translation is a particular case of translation between theories of $\lambda\Pi/\mathcal{R}$. Throughout this chapter, we have unfolded a general method, that can be applied to various individual translations. We have defined an inductive translation, allowing us to mechanically translate any term from the source theory to the target theory. We have proved the soundness of the translation by induction on the derivation, ensuring that derivations of the source theory are translated into valid derivations in the target theory. In the next chapter, our goal will be to abstract such a method to different translation templates, that can be instantiated to generate various individual translations.

Content of the library	Number of ...			
	proofs	classical proofs	higher-order results	admissible inference rules
Basic logic	40	0	17	26
Classical results	10	10	7	3
De Morgan's law	8	6	4	8
Polymorphic equality	10	0	6	4
Arithmetic	33	0	0	16
All	101	16	34	57

Table 9.1: Comparison of the different libraries.

Chapter 10

Translation Templates

Summary

Translation mechanisms often take the form of an inductive translation that satisfies certain invariants, which are also proved inductively. Theory morphisms and logical relations are common templates of such inductive constructions. Importantly, they allow representing the translation and establishing its soundness by providing parameters that satisfy certain properties. Therefore, in a logical framework supporting theory morphisms and logical relations, formalizing and verifying translations that fit one of these templates becomes much easier.

In this chapter, we extend theory morphisms and logical relations to the $\lambda\Pi$-calculus modulo rewriting. We prove their meta-theorems, guaranteeing the soundness of the generated translations. We apply these templates to define a number of translations between theories. In doing so, we identify some best practices that enable us to formalize challenging translations, in particular sort-erasure translations from sorted logic to unsorted logic.

This chapter is adapted from [TR25] and subsumes [Tra24b].

Representation theorems for logical frameworks often express that, for all terms $t : A$ in the source theory $\mathbb{S}$, there is a term $\mu(t) : \mu(A)$ in the target theory $\mathbb{T}$ (where $\mathbb{S}$ and $\mathbb{T}$ are independently formalized in the framework). There are two approaches to formalizing such representation functions μ.

First, μ can be implemented in a programming language that treats the terms of $\mathbb{S}$ and $\mathbb{T}$ as data. These programs are logic programs in [PS99] or functional programs in [PD10, PS09]. The framework then has to verify the meta-theorem by proving the correctness and termination of the program. Among early large case studies are verifications of cut-elimination [Pfe00] and logic translations [SS06].

Second, the framework can provide explicit support for certain restricted classes of representation functions for which correctness and termination are guaranteed. These are usually significantly easier for the user to define and implement. However, their expressivity is often limited, and intended applications often hit these limitations. *Theory morphisms* (also called signature morphisms) were introduced for LF in [HST94]. Here the user provides the translation of the constants of $\mathbb{S}$, and then the function μ is induced homomorphically on all terms of $\mathbb{S}$. The function μ is guaranteed to be total and

to preserve all typing judgments. Theory morphisms were added to Isabelle in [KWP99], to Twelf in [RS09], and to MMT in [RK13]. They were used to build the LATIN library of modular logics and representation theorems [CHK+11, Rab14].

To extend the expressivity of theory morphisms while retaining their simplicity, [RS13] introduced *logical relations* for LF. Here the morphism μ is coupled with a function ρ that establishes additional invariants about μ. Logical relations can be generalized to several morphisms. Binary logical relations are typically used to show an isomorphism between two theories. Logical relations, designed for LF, correspond to parametricity translations [BJP10, BJP12], designed for Pure Type systems. Parametricity was developed for Isabelle in [Lam13, HKKN13] and for the Calculus of (Inductive) Constructions in [KL12, TTS21, CCM25]. In particular, Trocq [CCM24, CCM25] is a parametricity-based plugin for automated proof transfer in Rocq. One of the main application of parametricity is to show the equivalence between different data structures, allowing the transfer of proofs between human-friendly representations and computer-friendly representations.

All of these developments were done in the absence of rewriting. This raises the question of whether such representation theorems for LF carry over to $\lambda\Pi/\mathcal{R}$, or if rewriting subtly interfere with established intuitions and results. In fact, other than the Maude tool [CELM96], which is based on rewriting logic, we are not aware of any tool supporting theory morphisms or related concepts on top of a rewrite system, and none that does so for a dependently typed λ-calculus. Felicissimo [Fel22] effectively defined theory morphisms in $\lambda\Pi/\mathcal{R}$ to establish the soundness of an encoding of functional and explicitly-typed Pure Type Systems. Similarly, [Tra24b] encoded individual interpretations in which the theory morphism and the logical relation are mutually recursive. However, both lacked general definitions of the concepts and general meta-theorems establishing their properties once and for all. More importantly, such theory morphisms and individual interpretations were not leveraged in practice to exchange proofs between Dedukti theories.

Contributions. In this chapter, we extend this morphism-like approach to $\lambda\Pi/\mathcal{R}$, and use it to exchange proofs between theories.

First, we generalize theory morphisms and logical relations from LF to $\lambda\Pi/\mathcal{R}$, stating all definitions and meta-theorems in full generality. The resulting formalism extends the translation templates of [HST94, RS13] and simplifies the special interpretations of [Tra24b]. We use theory morphisms and logical relations to define several translations between theories of $\lambda\Pi/\mathcal{R}$. The soundness of these translations is obtained for free thanks to the meta-theorems.

Second, we apply this formalism to define translations that have previously been out of reach for morphism-like methods. In particular, we formalize a novel translation from hard-sorted to soft-sorted to unsorted logic. To our knowledge, this is the first time that such a translation is defined in a fully declarative way. Our formalization relies on several design choices, taking advantage of rewriting to encode artifacts that have previously proved to be very cumbersome to use at scale in the absence of rewriting.

Third, we have implemented the `TranslationTemplates` tool, which realizes theory morphisms and unary logical relations for Dedukti. We have used `TranslationTemplates` to formalize all the examples shown in this chapter. The implementation and examples are available at

https://github.com/Deducteam/TranslationTemplates.

Outline. We define theory morphisms for $\lambda\Pi/\mathcal{R}$ in Section 10.1 and highlight several applications in Section 10.2. We define logical relations for $\lambda\Pi/\mathcal{R}$ in Section 10.3. In Section 10.4 and Section 10.5, we apply such templates to formalize challenging sort-erasure translations and data type embeddings. We describe our implementation for Dedukti in Section 10.6.

10.1 Theory Morphisms

In this section, we define theory morphisms for $\lambda\Pi/\mathcal{R}$, and we prove the basic Judgment Preservation theorem. As a running example, we give theory morphisms that translate between two representations of multiplicative groups.

10.1.1 Running Example

Before giving two encodings of multiplicative groups, let us first define propositional logic extended with equality.

Example 10.1 (PLeq)**.** *The theory* `PLeq` *extends propositional logic* `PL` *(as defined in Example 8.17) with equality.* ι *is the type of individuals. We define an equality symbol for elements of type* ι*, along with the reflexivity principle and Leibniz's law.*

ι : Type

$= : \iota \to \iota \to Prop$

refl : $\Pi x : \iota.\ Prf\ (x = x)$

leib : $\Pi x, y : \iota.\ Prf\ (x = y) \to \Pi P : \iota \to Prop.\ Prf\ (P\ x) \to Prf\ (P\ y)$

We now define two encodings of multiplicative groups: one based on multiplication and one based on division.

Example 10.2 (Groups based on multiplication)**.** *The theory* `MulGr` *encodes multiplicative groups by extending* `PLeq` *with a multiplication symbol* $\times$*, an inverse operation* inv*, and a neutral element* 1*.*

$\times : \iota \to \iota \to \iota$

$1 : \iota$

inv : $\iota \to \iota$

We use rewrite rules to encode that 1 *is a neutral element.*

$x \times 1 \hookrightarrow x$

$1 \times x \hookrightarrow x$

The inverse operation inv *is characterized by the following rewrite rules.*

$x \times (\mathsf{inv}\ x) \hookrightarrow 1$
$(\mathsf{inv}\ x) \times x \hookrightarrow 1$
$\mathsf{inv}\ 1 \hookrightarrow 1$
$\mathsf{inv}\ (\mathsf{inv}\ x) \hookrightarrow x$
$(\mathsf{inv}\ x) \times (x \times y) \hookrightarrow y$
$x \times ((\mathsf{inv}\ x) \times y) \hookrightarrow y$
$\mathsf{inv}\ (x \times y) \hookrightarrow (\mathsf{inv}\ y) \times (\mathsf{inv}\ x)$

The operation $\times$ *is associative.*

$(x \times y) \times z \hookrightarrow x \times (y \times z)$

These rewrite rules are generated by the Knuth-Bendix completion algorithm [KB70] from $1 \times x = x$, $\mathsf{inv}\ x \times x = 1$ *and* $(x \times y) \times z = x \times (y \times z)$. *This guarantees that such rewrite rules are confluent and terminating.*

The axioms for neutrality, inverseness and associativity derive from the rewrite rules. For instance, the axiom of associativity

$$\Pi x, y, z : \iota.\ Prf\ ((x \times y) \times z = x \times (y \times z))$$

is simply proved using refl.

In this encoding, we benefit from the computational power of $\lambda\Pi/\mathcal{R}$. *For example, we obtain the conversion* $(\mathsf{inv}\ (\mathsf{inv}\ x)) \times (\mathsf{inv}\ x \times y) \equiv_{\beta\mathcal{R}} y$ *for free.*

Example 10.3 (Groups based on division)**.** *The theory* `DivGr` *encodes multiplicative groups by extending* `PLeq` *with a division operation* $\div$ *and an element* 1.

$\div : \iota \to \iota \to \iota$
$1 : \iota$

The following rewrite rules encode the semantics of $\div$.

$x \div 1 \hookrightarrow x$
$x \div x \hookrightarrow 1$
$1 \div (x \div y) \hookrightarrow y \div x$
$(x \div (1 \div y)) \div (1 \div z) \hookrightarrow x \div ((1 \div z) \div y)$

Using these rewrite rules, we have for example $((y \div x) \div y) \div (1 \div x) \equiv_{\beta\mathcal{R}} 1$ *for free.*

Note that these rewrite rules are not confluent, as we have

$$((1 \div y) \div (1 \div y)) \div (1 \div z) \hookrightarrow 1 \div (1 \div z) \hookrightarrow z \div 1 \hookrightarrow z$$

and $((1 \div y) \div (1 \div y)) \div (1 \div z) \hookrightarrow (1 \div y) \div ((1 \div z) \div y)$, *but there is no common reduct. For the sake of simplicity, we still consider these rewrite rules.*

In this encoding with division, the usual group operation $x \times y$ *can be defined as* $x \div (1 \div y)$.

`MulGr` and `DivGr` are two different encodings of multiplicative groups. Our goal is to translate between these two representations.

10.1.2 Formal Definition

Theory morphisms from theory $\mathbb{S}$ to theory $\mathbb{T}$ are translations that replace the *constants* of $\mathbb{S}$ by *terms* of $\mathbb{T}$. Such terms are the parameters of the translation and must be provided to perform the translation.

Definition 10.4: Theory Morphism

The mapping μ defined inductively from a set of parameters μ_c and μ_a by

$$\begin{array}{lcl}
\mu(x) & := & x \\
\mu(c) & := & \mu_c \\
\mu(a) & := & \mu_a \\
\mu(M\ N) & := & \mu(M)\ \mu(N) \\
\mu(A\ M) & := & \mu(A)\ \mu(M) \\
\mu(\lambda x : A.\ M) & := & \lambda x : \mu(A).\ \mu(M) \\
\mu(\lambda x : A.\ B) & := & \lambda x : \mu(A).\ \mu(B) \\
\mu(\Pi x : A.\ B) & := & \Pi x : \mu(A).\ \mu(B) \\
\mu(\Pi x : A.\ K) & := & \Pi x : \mu(A).\ \mu(K) \\
\mu(\mathsf{Type}) & := & \mathsf{Type} \\
\mu(\mathsf{Kind}) & := & \mathsf{Kind}
\end{array}$$

is a theory morphism from theory $\mathbb{S}$ to theory $\mathbb{T}$ when:

1. for every constant $c : A \in \mathbb{S}$, there exists a term μ_c such that $\vdash_{\mathbb{T}} \mu_c : \mu(A)$,
2. for every constant $a : K \in \mathbb{S}$, there exists a term μ_a such that $\vdash_{\mathbb{T}} \mu_a : \mu(K)$,
3. for every rewrite rule $\ell \hookrightarrow r \in \mathbb{S}$, we have $\mu(\ell) \equiv_{\beta(\eta)\mathcal{R}} \mu(r)$,

where μ is defined on contexts and substitutions by

$$\begin{array}{lcl}
\mu(\varnothing) & := & \varnothing \\
\mu(\Gamma, x : A) & := & \mu(\Gamma), x : \mu(A) \\
\mu(\theta, x \leftarrow M) & := & \mu(\theta), x \leftarrow \mu(M).
\end{array}$$

The first two conditions are the same as in LF: the constants of $\mathbb{S}$ must be mapped to terms of $\mathbb{T}$ that have the correct type. When extending theory morphisms from LF to $\lambda\Pi/\mathcal{R}$, we require as a third condition that, for every rewrite rule $\ell \hookrightarrow r$ of $\mathbb{S}$, we have the conversion $\mu(\ell) \equiv_{\beta(\eta)\mathcal{R}} \mu(r)$ in $\mathbb{T}$. Under this condition, we will see that *convertibility* is preserved, i.e., if $t \equiv_{\beta(\eta)\mathcal{R}} u$ in $\mathbb{S}$ then $\mu(t) \equiv_{\beta(\eta)\mathcal{R}} \mu(u)$ in $\mathbb{T}$. Intuitively, the three conditions of theory morphisms ensure that the target theory has at least the same deductive and computational strength as the source theory.

Remark 10.5. *Felicissimo [Fel22, see Long version] required as a third condition that, for every rewrite rule $\ell \hookrightarrow r$ of $\mathbb{S}$, we have the rewriting $\mu(\ell) \hookrightarrow^*_{\beta\mathcal{R}} \mu(r)$ in $\mathbb{T}$ (where $\hookrightarrow^*_{\beta\mathcal{R}}$ is the reflexive and transitive closure of $\hookrightarrow_{\beta\mathcal{R}}$). Under this condition,* rewritability *is preserved, i.e., if $t \hookrightarrow^*_{\beta\mathcal{R}} u$ in $\mathbb{S}$ then $\mu(t) \hookrightarrow^*_{\beta\mathcal{R}} \mu(u)$ in $\mathbb{T}$. Felicissimo's condition on rewriting is sufficient to prove our condition on conversion, but it is not necessary. For instance, consider A_1, A_2, A_3 of type* Type *in $\mathbb{S}$, with $A_1 \hookrightarrow A_3$, and B_1, B_2, B_3 of type* Type *in $\mathbb{T}$, with $B_1 \hookrightarrow B_2$ and $B_3 \hookrightarrow B_2$. We set $\mu(A_i) := B_i$ for $i \in [\![1, 3]\!]$. We indeed have $\mu(A_1) = B_1 \equiv_{\beta\mathcal{R}} B_3 = \mu(A_3)$ in $\mathbb{T}$, but we do not*

have $\mu(A_1) \hookrightarrow^*_{\beta\mathcal{R}} \mu(A_3)$. For the Judgment Preservation theorem, we only need to preserve convertibility, that is why we take a definition of theory morphisms that is more general than Felicissimo's definition.

For simplicity, when assigning values to the parameters, we will write $\mu(c)$ and $\mu(a)$ instead of μ_c and μ_a.

Example 10.6 (Identity morphism)**.** *Let $\mathbb{T}$ be a theory. The identity morphism from $\mathbb{T}$ to $\mathbb{T}$ maps each constant to itself, and therefore each term to itself. The conditions of theory morphism are trivially satisfied.*

Example 10.7 (Morphism $\texttt{MulDivGr} : \texttt{MulGr} \to \texttt{DivGr}$)**.** *We define a morphism* `MulDivGr` *from* `MulGr` *to* `DivGr`*. All the constants of* `PLeq` *are mapped to themselves. The parameters for $\times$, 1 and* inv *follow the intuition.*

$\mu(\times) := \lambda x, y : \iota.\ x \div (1 \div y)$
$\mu(1) := 1$
$\mu(\mathsf{inv}) := \lambda x : \iota.\ 1 \div x$

For every rewrite rule $\ell \hookrightarrow r$ of the multiplication group, we can easily show that $\mu(\ell)$ and $\mu(r)$ are convertible using the rewrite rules of `DivGr`*. For the rewrite rule $x \times (\mathsf{inv}\ x) \hookrightarrow 1$, we have $\mu(x \times (\mathsf{inv}\ x)) \equiv_{\beta\mathcal{R}} x \div (1 \div (1 \div x)) \equiv_{\beta\mathcal{R}} x \div (x \div 1) \equiv_{\beta\mathcal{R}} x \div x \equiv_{\beta\mathcal{R}} 1 \equiv_{\beta\mathcal{R}} \mu(1)$.*

Example 10.8 (Morphism $\texttt{DivMulGr} : \texttt{DivGr} \to \texttt{MulGr}$)**.** *We define a morphism* `DivMulGr` *from* `DivGr` *to* `MulGr`*. All the constants of* `PLeq` *are mapped to themselves.*

$\mu(\div) := \lambda x, y : \iota.\ x \times (\mathsf{inv}\ y)$
$\mu(1) := 1$

For every rewrite rule $\ell \hookrightarrow r$ of `DivGr`*, we can easily show that $\mu(\ell)$ and $\mu(r)$ are convertible using the rewrite rules of* `MulGr`*. For the the rewrite rule $1 \div (1 \div x) \hookrightarrow x$, we have $\mu(1 \div (1 \div x)) \equiv_{\beta\mathcal{R}} 1 \times (\mathsf{inv}\ (1 \times \mathsf{inv}\ x)) \equiv_{\beta\mathcal{R}} \mathsf{inv}\ (\mathsf{inv}\ x) \equiv_{\beta\mathcal{R}} x \equiv_{\beta\mathcal{R}} \mu(x)$.*

To show that `MulDivGr` and `DivMulGr` are isomorphisms, we have to show that the composition $\texttt{MulDivGr};\texttt{DivMulGr}$ and the identity morphism of `MulGr` are equal, and accordingly for the opposite composition.

Example 10.9 (Morphism $\texttt{MulGr} \to \texttt{MulGr}$)**.** *The composition* $\texttt{MulDivGr};\texttt{DivMulGr}$ *is a theory morphism from the multiplication group* `MulGr` *to itself. In particular, we get the following parameters.*

$\mu(\times) := \lambda x, y : \iota.\ x \times \mathsf{inv}\ (1 \times \mathsf{inv}\ y)$
$\mu(1) := 1$
$\mu(\mathsf{inv}) := \lambda x : \iota.\ 1 \times \mathsf{inv}\ x$

Note that, thanks to the rewrite rules of `MulGr`*, we have $\mu(\times) \equiv_{\beta\mathcal{R}} \lambda x, y : \iota.\ x \times y$ and $\mu(\mathsf{inv}) \equiv_{\beta\mathcal{R}} \lambda x : \iota.\ \mathsf{inv}\ x$. Therefore, the variant of $\lambda\Pi/\mathcal{R}$ with η-conversion suffices to have $\mu(\times) \equiv_{\beta\eta\mathcal{R}} \times$ and $\mu(\mathsf{inv}) \equiv_{\beta\eta\mathcal{R}} \mathsf{inv}$. In that case, the morphism* $\texttt{MulDivGr};\texttt{DivMulGr}$ *and the identity morphism of* `MulGr` *are definitionally equal.*

Later in Example 10.18, we will show how a *propositional* equality of morphisms can be proved in a situation where definitional equality is not strong enough.

10.1.3 Judgment Preservation Theorem

The main property of theory morphisms is that this translation preserves convertibility and judgments. Once we have specified the parameters of a theory morphism, we can therefore translate any typing judgment from the source theory to the target theory. In particular, we can transfer proofs between different theories of $\lambda\Pi/\mathcal{R}$.

Theorem 10.10: Judgment Preservation

Let μ be a theory morphism from $\mathbb{S}$ to $\mathbb{T}$.

1. If $\vdash_{\mathbb{S}} \Gamma$, then $\vdash_{\mathbb{T}} \mu(\Gamma)$.
2. If $\Gamma \vdash_{\mathbb{S}} M : A$, then $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(M) : \mu(A)$.
3. If $\Gamma \vdash_{\mathbb{S}} A : K$, then $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(A) : \mu(K)$.
4. If $\Gamma \vdash_{\mathbb{S}} K : \mathsf{Kind}$, then $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(K) : \mathsf{Kind}$.

This theorem extends the Judgment Preservation theorem of [RS13] from LF to $\lambda\Pi/\mathcal{R}$, and generalizes the one of [Fel22] as we consider a broader definition of theory morphisms. The theorem relies on the substitution lemma, which states that morphism and substitution commute with each other, and on the conversion lemma, which states that convertibility is preserved by the morphism.

Lemma 10.11 (Substitution)**.** *Let μ be a theory morphism from $\mathbb{S}$ to $\mathbb{T}$, and θ be a substitution.*

1. $\mu(M\theta) = \mu(M)\mu(\theta)$,
2. $\mu(A\theta) = \mu(A)\mu(\theta)$,
3. $\mu(K\theta) = \mu(K)\mu(\theta)$.

Proof. By induction on terms.

— Variable: If θ does not substitute x, then by definition $\mu(\theta)$ does not substitute x either. Therefore $\mu(x\theta) = \mu(x) = \mu(x)\mu(\theta)$.

If θ substitutes x by N, then by definition $\mu(\theta)$ substitutes x by $\mu(N)$. Therefore $\mu(x\theta) = \mu(N) = \mu(x)\mu(\theta)$.

— Constant: We have $\mu(c\theta) = \mu(c) = \mu_c = \mu_c\mu(\theta) = \mu(c)\mu(\theta)$, as c and μ_c are closed terms. Similarly, $\mu(a\theta) = \mu(a)\mu(\theta)$.

— The other cases are shown using the induction hypotheses. □

Lemma 10.12 (Conversion)**.** *Let μ be a theory morphism from $\mathbb{S}$ to $\mathbb{T}$.*

1. *If $A \equiv_{\beta(\eta)\mathcal{R}} B$ in $\mathbb{S}$, then $\mu(A) \equiv_{\beta(\eta)\mathcal{R}} \mu(B)$ in $\mathbb{T}$.*
2. *If $K \equiv_{\beta(\eta)\mathcal{R}} K'$ in $\mathbb{S}$, then $\mu(K) \equiv_{\beta(\eta)\mathcal{R}} \mu(K')$ in $\mathbb{T}$.*

Proof. By induction on the construction of $\equiv_{\beta(\eta)\mathcal{R}}$.

— We have $\mu((\lambda x : A.\ M)\ N) = (\lambda x : \mu(A).\ \mu(M))\ \mu(N) \equiv_{\beta\mathcal{R}} \mu(M)\mu([x \leftarrow N])$. By Lemma 10.11, we get $\mu((\lambda x : A.\ M)\ N) \equiv_{\beta\mathcal{R}} \mu(M[x \leftarrow N])$. Similarly, we prove $\mu((\lambda x : A.\ B)\ M) \equiv_{\beta\mathcal{R}} \mu(B[x \leftarrow M])$.

— Suppose that x is not free in M. We have $\mu(\lambda x : A.\ M\ x) = \lambda x : \mu(A).\ \mu(M)\ x$, which is η-convertible to $\mu(M)$.

— Let $\ell \hookrightarrow r \in \mathbb{S}$ and θ be a substitution. We have $\mu(\ell\theta) = \mu(\ell)\mu(\theta)$ and $\mu(r\theta) = \mu(r)\mu(\theta)$ using Lemma 10.11. By definition of theory morphism, we derive $\mu(\ell\theta) = \mu(\ell)\mu(\theta) \equiv_{\beta(\eta)\mathcal{R}} \mu(r)\mu(\theta) = \mu(r\theta)$.

— The cases for applications, abstractions and dependent types follow from the induction hypotheses.

— Reflexivity, symmetry, and transitivity are immediate. □

Proof of Theorem 10.10. By induction on the typing derivations.

— Const-Obj: By induction hypothesis, we have $\vdash_{\mathbb{T}} \mu(\Gamma)$. By definition of theory morphism, we have $\vdash_{\mathbb{T}} \mu_c : \mu(A)$. Hence, we derive $\mu(\Gamma) \vdash_{\mathbb{T}} \mu_c : \mu(A)$ by weakening. We proceed similarly for Const-Type.

— App-Obj: We have $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(M) : \Pi x : \mu(A).\ \mu(B)$ and $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(N) : \mu(A)$ by induction hypotheses. We derive $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(M)\ \mu(N) : \mu(B)[x \leftarrow \mu(N)]$ using App-Obj. We get $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(M)\ \mu(N) : \mu(B[x \leftarrow N])$ using Lemma 10.11. We proceed similarly for App-Type.

— Conv-Type: We have $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(M) : \mu(A)$ and $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(A) :$ Type by induction hypotheses. As $A \equiv_{\beta(\eta)\mathcal{R}} B$ in $\mathbb{S}$, we get $\mu(A) \equiv_{\beta(\eta)\mathcal{R}} \mu(B)$ in $\mathbb{T}$ using Lemma 10.12. We derive $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(M) : \mu(B)$ using Conv-Type. We proceed similarly for Conv-Kind.

— The remaining cases follow from the induction hypotheses. □

Remark 10.13. *Theory morphisms capture that derivability in the source theory implies derivability in the target theory. The reverse is not necessarily true: it fails when the target theory has greater deductive or computational strength than the source theory.*

Let us show that $\mu(\Gamma) \vdash_{\mathbb{T}} N : \mu(A)$ for some N does not necessarily imply $\Gamma \vdash_{\mathbb{S}} M : A$ for some M. Let $\mathbb{S}$ be the theory with types A_1, A_2 and with constant $a : A_1$. Let $\mathbb{T}$ be the theory with types B_1, B_2 such that $B_1 \hookrightarrow B_2$, and with constant $b : B_1$. Let μ be the morphism such that $\mu(A_1) := B_1$, $\mu(A_2) := B_2$ and $\mu(a) := b$. We have $\vdash_{\mathbb{T}} b : \mu(A_2)$ in the target theory thanks to the rewrite rule, but A_2 is not inhabited in the source theory.

This property was already not satisfied for theory morphisms in LF: it suffices to replace the rewrite rule $B_1 \hookrightarrow B_2$ by a constant $f : B_1 \to B_2$. The type $\mu(A_2)$ is inhabited in the target theory by $f\ b$, but A_2 is not inhabited in the source theory.

10.2 Applications

Theory morphisms encompass many different translations between logics and between data structures. We give here four examples. The first one is a translation from a theory with axiomatized natural deduction to a theory with computational natural deduction. The second one is a translation from propositional logic to Q_0 logic. The third one is a translation from higher-order classical logic to higher-order intuitionistic logic. The fourth one is a translation from lists to binary trees.

10.2.1 From Deduction to Computation

In Example 8.17 and Example 8.18, we have encoded a fragment of propositional logic with implication and conjunction. The natural deduction rules are represented via axioms in Example 8.17 and via rewrite rules in Example 8.18. We now define a theory morphism μ from the deductive encoding to the computational encoding of the natural deduction rules, thus showing that the latter is a refinement of the former.

The constants shared by both encodings are mapped to themselves. The constants representing natural deduction rules are mapped to theorems proving them by using the rewrite rules. We map the introduction of implication to

$$\mu(\mathsf{imp_i}) := \lambda p, q : Prop.\ \lambda H : Prf\ p \to Prf\ q.\ H$$

of type $\Pi p, q : Prop.\ (Prf\ p \to Prf\ q) \to Prf\ (p \Rightarrow q)$, since $Prf\ p \to Prf\ q$ and $Prf\ (p \Rightarrow q)$ are convertible. Similarly, the left-elimination of the conjunction is mapped to

$$\mu(\mathsf{and_{e\ell}}) := \lambda p, q : Prop.\ \lambda H_{pq} : Prf\ (p \wedge q).\ H_{pq}\ p\ (\lambda H_p : Prf\ p.\ \lambda H_q : Prf\ q.\ H_p)$$

which has type $\Pi p, q : Prop.\ Prf\ (p \wedge q) \to Prf\ p$. The same idea applies for the remaining natural deduction rules.

10.2.2 From Propositional Logic to Q_0 Logic

Andrews formulated Q_0 [And02], a version of higher-order logic in which the only primitive symbol is equality. All the other connectives and quantifiers are then built upon equality. Q_0 is for instance used in the HOL Light proof assistant.

Example 10.14 (Q_0 logic)**.** *To encode Q_0 in $\lambda\Pi/\mathcal{R}$, we use the encoding of the notions of sort, function sort, proposition and proof [BDG+23].*

Set : Type

$El : Set \to$ Type

$\rightsquigarrow : Set \to Set \to Set$

$El\ (x \rightsquigarrow y) \hookrightarrow El\ x \to El\ y$

$o : Set$

$Prf : El\ o \to$ Type

We define a polymorphic equality that satisfies Leibniz's law, functional extensionality and propositional extensionality. For simplicity, $= \ a \ x \ y$ is written $x =_a y$. Unlike the usual presentation of Q_0, we take advantage of $\lambda\Pi/\mathcal{R}$ and define Leibniz's law through rewriting.

$= : \Pi a : Set.\ El\ a \to El\ a \to El\ o$

$Prf\ (x =_a y) \hookrightarrow \Pi P : El\ a \to Prop.\ Prf\ (P\ x) \to Prf\ (P\ y)$

propext : $\Pi p, q : Prop.\ (Prf\ p \to Prf\ q) \to (Prf\ q \to Prf\ p) \to Prf\ (p =_o q)$

funext : $\Pi a, b : Set.\ \Pi f, g : El\ (a \rightsquigarrow b).\ (\Pi x : El\ a.\ Prf\ (f\ x =_b g\ x)) \to Prf\ (f =_{a \rightsquigarrow b} g)$

In funext*, the applications $f\ x$ and $g\ x$ are well-typed thanks to the rewrite rule on $\rightsquigarrow$.*

We define a theory morphism from PL (as described in Example 8.17) to Q_0. The constant Prf is mapped to itself, and $Prop$ is mapped to $El\ o$. Each connective is mapped to its encoding in Q_0.

$\mu(\wedge) := \lambda p, q : El\ o.\ (\lambda f.\ f\ p\ q) =_{(o \rightsquigarrow o \rightsquigarrow o) \rightsquigarrow o} (\lambda f.\ f\ \top\ \top)$

$\mu(\Rightarrow) := \lambda p, q : El\ o.\ (\mu(\wedge)\ p\ q) =_o p$

where $\top$ is an abbreviation for the term $(\lambda x.\ x) =_{o \rightsquigarrow o} (\lambda x.\ x)$. Note that the translation of $\wedge$ is well-typed only because of the rewrite rule on $\rightsquigarrow$. The translations of the natural deduction rules rely on funext, on propext and on intermediate lemmas on equality.

In LF, i.e., without rewriting, we would have to define an application constructor and an abstraction constructor

$app : \Pi a, b : Set.\ El\ (a \rightsquigarrow b) \to El\ a \to El\ b$

$lam : \Pi a, b : Set.\ (El\ a \to El\ b) \to El\ (a \rightsquigarrow b)$

instead of the rewrite rule $El\ (x \rightsquigarrow y) \hookrightarrow El\ x \to El\ y$, along with an axiom for the β-reduction rule. Similarly, Leibniz's law would have been encoded by an axiom. Therefore, in LF, the user has to explicitly apply these constructors and axioms. In $\lambda\Pi/\mathcal{R}$, this work is discharged to the framework through rewriting, resulting in much simpler proof obligations for the user.

10.2.3 From Classical Logic to Intuitionistic Logic

We have seen in Section 9.1 an encoding of higher-order logic, for both the classical variant and the intuitionistic variant. The version of Kuroda's translation defined for $\lambda\Pi/\mathcal{R}$ in Section 9.2 can be expressed by a theory morphism. The insertion of double negations is encoded using the following parameters.

$\mu(\forall) := \lambda a : Set.\ \lambda p : El\ a \to Prop.\ \forall\ a\ (\lambda z : El\ a.\ \neg\neg(p\ z))$

$\mu(Prf) := \lambda p : Prop.\ Prf\ (\neg\neg p)$

All the other constants, except the ones encoding the natural deduction rules, are mapped to themselves. Every constant c representing a natural deduction rule is mapped to an intuitionistic proof term $\mu(c)$ representing its translation. In particular, the axiom of excluded middle

pem : $\Pi p : Prop.\ Prf\ (p \vee \neg p)$

is mapped to

$$\begin{aligned}\mu(\mathsf{pem}) \quad := \quad & \lambda p : Prop.\ \mathsf{neg_i}\ \neg(p \vee \neg p) \\ & \quad (\lambda H : Prf\ (\neg(p \vee \neg p)).\ \mathsf{neg_e}\ (p \vee \neg p)\ H \\ & \quad\quad (\mathsf{or_{ir}}\ p\ \neg p \\ & \quad\quad\quad (\mathsf{neg_i}\ p\ (\lambda H_p : Prf\ p.\ \mathsf{neg_e}\ (p \vee \neg p)\ H\ (\mathsf{or_{i\ell}}\ p\ \neg p\ H_p)))))\end{aligned}$$

which is an *intuitionistic* proof of $\Pi p : Prop.\ Prf\ (\neg\neg(p \vee \neg p))$. The theory morphism is well-defined, as the conversions $\mu(El\ o) \equiv_{\beta\mathcal{R}} \mu(Prop)$ and $\mu(El\ (x \rightsquigarrow y)) \equiv_{\beta\mathcal{R}} \mu(El\ x \to El\ y)$ trivially hold in the encoding of intuitionistic logic.

10.2.4 From Lists to Binary Trees

Example 10.15 (Lists)**.** *The theory* `List` *defines the data structure of lists indexed by the sort of their elements. We use* Set *and* El *defined in Example 10.14 to define the polymorphic constructor* list*, which maps any sort* a *to the type of the lists containing elements of sort* a*.* nil *creates an empty list.* cons *appends an element to a list.* hd *returns the head of a list and* tl *returns the tail of a list.* concat *concatenates two lists.*

$\mathsf{list} : Set \to Set$

$\mathsf{nil} : \Pi a : Set.\ El\ (\mathsf{list}\ a)$

$\mathsf{cons} : \Pi a : Set.\ El\ a \to El\ (\mathsf{list}\ a) \to El\ (\mathsf{list}\ a)$

$\mathsf{hd} : \Pi a : Set.\ El\ (\mathsf{list}\ a) \to El\ a$

$\mathsf{tl} : \Pi a : Set.\ El\ (\mathsf{list}\ a) \to El\ (\mathsf{list}\ a)$

$\mathsf{concat} : \Pi a : Set.\ El\ (\mathsf{list}\ a) \to El\ (\mathsf{list}\ a) \to El\ (\mathsf{list}\ a)$

The semantic of these symbols is encoded using linearized rewrite rules, as explained in Example 8.10.

$\mathsf{hd}\ a\ (\mathsf{cons}\ a'\ x\ l) \hookrightarrow x$

$\mathsf{tl}\ a\ (\mathsf{cons}\ a'\ x\ l) \hookrightarrow l$

$\mathsf{concat}\ a\ (\mathsf{nil}\ a')\ l \hookrightarrow l$

$\mathsf{concat}\ a\ (\mathsf{cons}\ a'\ x\ l_1)\ l_2 \hookrightarrow \mathsf{cons}\ a\ x\ (\mathsf{concat}\ a\ l_1\ l_2)$

The left-hand side of these rewrite rules are obviously ill-typed, but this is not a problem: the rules preserve typing. Due to the typing constraints on the head symbols, these rewrite rules can only be used when a *and* a' *are instantiated by the same sort.*

Example 10.16 (Trees)**.** *The theory* `Tree` *defines the data structure of binary trees indexed by the sort of their elements.* leaf *creates the empty tree.* node *creates a new binary tree, by taking as arguments its root and the two children.* left *returns the left child,* right *returns the right child, and* root *returns the root.*

$\mathsf{tree} : Set \to Set$

$\mathsf{leaf} : \Pi a : Set.\ El\ (\mathsf{tree}\ a)$

$\mathsf{node} : \Pi a : Set.\ El\ a \to El\ (\mathsf{tree}\ a) \to El\ (\mathsf{tree}\ a) \to El\ (\mathsf{tree}\ a)$

$\mathsf{left} : \Pi a : Set.\ El\ (\mathsf{tree}\ a) \to El\ (\mathsf{tree}\ a)$

$\mathsf{right} : \Pi a : Set.\ El\ (\mathsf{tree}\ a) \to El\ (\mathsf{tree}\ a)$

$\mathsf{root} : \Pi a : Set.\ El\ (\mathsf{tree}\ a) \to El\ a$

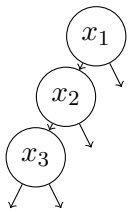
We encode the semantics of left, right *and* root *using rewriting. Again, we use linearized rules.*

left a (node a' x l r) $\hookrightarrow l$
right a (node a' x l r) $\hookrightarrow r$
root a (node a' x l r) $\hookrightarrow x$

We define a composition of binary trees: the second tree is inductively composed with the left child of the first tree.

compo : $\Pi a : Set.\ El$ (tree a) $\to El$ (tree a) $\to El$ (tree a)
compo a (leaf a') $t \hookrightarrow t$
compo a (node a' x l r) $t \hookrightarrow$ node a x (compo a l t) r

We now define a theory morphism from `List` to `Tree`. The idea is to represent lists as binary trees with only left children. For instance, the list $[x_1, x_2, x_3]$ is mapped to the following binary tree.



Formally, the morphism maps Set and El to themselves. list, nil and concat are mapped to their respective counterparts in `Tree`. The parameters for cons, hd and tl are chosen so that we encode lists inside the leftmost branch of binary trees.

μ(list) := tree
μ(nil) := leaf
μ(cons) := $\lambda a : Set.\ \lambda x : El\ a.\ \lambda l : El$ (tree a). node a x l (leaf a)
μ(hd) := root
μ(tl) := left
μ(concat) := compo

The conditions of Definition 10.4 on the rewrite rules of `List` are satisfied in `Tree`. We only show the most interesting case.

$$\begin{array}{lll} \mu(\text{concat } a\ (\text{cons } a\ x\ l_1)\ l_2) & \equiv_{\beta\mathcal{R}} & \text{compo } a\ (\text{node } a\ x\ l_1\ (\text{leaf } a))\ l_2 \\ & \equiv_{\beta\mathcal{R}} & \text{node } a\ x\ (\text{compo } a\ l_1\ l_2)\ (\text{leaf } a) \\ & \equiv_{\beta\mathcal{R}} & \mu(\text{cons } a\ x\ (\text{concat } a\ l_1\ l_2)) \end{array}$$

Note that the conditions on the rewrite rules only need to be satisfied by the versions of the rewrite rules *before* linearization. Indeed, we only use instances of the linearized rules when the left-hand side is well-typed, i.e., we only use instances of the rewrite rules before linearization.

We are therefore able to translate lists into binary trees, where both data structures are encoded using the computational power of $\lambda\Pi/\mathcal{R}$.

10.3 Logical Relations

In this section, we extend logical relations in the sense of [RS13] to $\lambda\Pi/\mathcal{R}$, and we prove the main theorem about them, often called the Basic lemma or Abstraction theorem. The technical details of logical relations are much trickier than those of theory morphisms. We illustrate logical relations on the running example of multiplicative groups.

10.3.1 Formal Definition

A theory morphism μ maps the judgment $\Gamma \vdash_{\mathbb{S}} M : A$ to $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(M) : \mu(A)$. A logical relation ρ on μ states and proves an additional invariant satisfied by μ: it maps the judgment $\Gamma \vdash_{\mathbb{S}} M : A$ to the judgment $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(M) : \rho(A)\ \mu(M)$. Here every type A is mapped to a predicate $\rho(A) : \mu(A) \to \mathsf{Type}$, and every term $M : A$ is mapped to a proof $\rho(M)$ that $\mu(M)$ satisfies $\rho(A)$.

The function ρ duplicates every (free or bound) variable so that every fresh variable $x : A$ yields both its translation $x : \mu(A)$ and an assumption $x^* : \rho(A)\ x$ that x satisfies the invariant. Thus, the translation of an abstraction $\lambda x : A.\ M$ is given by $\lambda x : \mu(A).\ \lambda x^* : \rho(A)\ x.\ \rho(M)$. Accordingly, the translation of an application $M\ N$ is given by $\rho(M)\ \mu(N)\ \rho(N)$, i.e., it supplies both the translated argument $\mu(N)$ and a proof that $\mu(N)$ satisfies its invariant.

The definition of the logical relation of a function type $\Pi x : A.\ B$ is the well-known condition that functions must preserve the relation: $\rho(\Pi x : A.\ B)$ holds for a function $f : \mu(\Pi x : A.\ B)$ if for every $x : \mu(A)$ satisfying $\rho(A)$, the term $(f\ x)$ satisfies $\rho(B)$. Thus, $\rho(\Pi x : A.\ B)$ is given by $\lambda f : \mu(\Pi x : A.\ B).\ \Pi x : \mu(A).\ \Pi x^* : \rho(A)\ x.\ \rho(B)\ (f\ x)$.

Following the same idea, we would like to define

$$\rho(\Pi x : A.\ K) = \lambda f : \mu(\Pi x : A.\ K).\ \Pi x : \mu(A).\ \Pi x^* : \rho(A)\ x.\ \rho(K)\ (f\ x)$$

This works, with a little extra effort, in type theories with higher universes. However, in $\lambda\Pi/\mathcal{R}$, such a term is ill-typed because $\mu(\Pi x : A.\ K)$ is a kind. To get around this issue, we insert an extra parameter R to the translation: if R has type $\Pi x : A.\ K$, then we define

$$\rho^R(\Pi x : A.\ K) = \Pi x : \mu(A).\ \Pi x^* : \rho(A)\ x.\ \rho^{R\ x}(K)$$

and if R has type Type, then we define $\rho^R(\mathsf{Type}) = \mu(R) \to \mathsf{Type}$.

More generally, we can define n-ary logical relations on the n theory morphisms $\mu_1, \ldots, \mu_n$ from $\mathbb{S}$ to $\mathbb{T}$. Such a n-ary logical relation ρ maps every type to an n-ary predicate, and every term $M : A$ to a proof of $\rho(A)\ \mu_1(M)\ \ldots\ \mu_n(M)$.

Conventions. Let $\mu_1, \ldots, \mu_n$ be n theory morphisms from $\mathbb{S}$ to $\mathbb{T}$. Without loss of generality, we consider that each μ_i maps variables x to x_i. We use the following notations:

— We write $\vec{x} : \vec{\mu}(A)$ for the context $x_1 : \mu_1(A), \ldots, x_n : \mu_n(A)$.

— We write $[\vec{x} \leftarrow \vec{\mu}(M)]$ for the substitution $[x_1 \leftarrow \mu_1(M), \ldots, x_n \leftarrow \mu_n(M)]$.

— We write $\lambda\vec{x} : \vec{\mu}(A).\ t$ for $\lambda x_1 : \mu_1(A).\ \ldots\ \lambda x_n : \mu_n(A).\ t$.

— We write $\Pi\vec{x} : \vec{\mu}(A).\ t$ for $\Pi x_1 : \mu_1(A).\ \ldots\ \Pi x_n : \mu_n(A).\ t$. Similarly, we write $\vec{\mu}(A) \to t$ for $\mu_1(A) \to \ldots \to \mu_n(A) \to t$.

— Given a list of terms $\vec{M} = M_1, \ldots, M_n$, we write $t\ \vec{M}$ for the application $t\ M_1\ \ldots\ M_n$.

Definition 10.17: Logical Relation

Let $\mu_1, \ldots, \mu_n$ be theory morphisms from $\mathbb{S}$ to $\mathbb{T}$. The mapping ρ defined inductively from a set of parameters ρ_c and ρ_a by

$$\begin{array}{lcl}
\rho(x) & := & x^* \\
\rho(c) & := & \rho_c \\
\rho(a) & := & \rho_a \\
\rho(M\ N) & := & \rho(M)\ \vec{\mu}(N)\ \rho(N) \\
\rho(A\ M) & := & \rho(A)\ \vec{\mu}(M)\ \rho(M) \\
\rho(\lambda x : A.\ M) & := & \lambda\vec{x} : \vec{\mu}(A).\ \lambda x^* : \rho(A)\ \vec{x}.\ \rho(M) \\
\rho(\lambda x : A.\ B) & := & \lambda\vec{x} : \vec{\mu}(A).\ \lambda x^* : \rho(A)\ \vec{x}.\ \rho(B) \\
\rho(\Pi x : A.\ B) & := & \lambda\vec{f} : \vec{\mu}(\Pi x : A.\ B).\ \Pi\vec{x} : \vec{\mu}(A).\ \Pi x^* : \rho(A)\ \vec{x}. \\
 & & \rho(B)\ (f_1\ x_1)\ \ldots\ (f_n\ x_n) \\
\rho^R(\Pi x : A.\ K) & := & \Pi\vec{x} : \vec{\mu}(A).\ \Pi x^* : \rho(A)\ \vec{x}.\ \rho^{R\ x}(K) \\
\rho^R(\mathsf{Type}) & := & \vec{\mu}(R) \to \mathsf{Type} \\
\rho(\mathsf{Kind}) & := & \mathsf{Kind}
\end{array}$$

is a logical relation on μ when:

1. for every constant $c : A \in \mathbb{S}$, there exists a term ρ_c such that $\vdash_{\mathbb{T}} \rho_c : \rho(A)\ \vec{\mu}(c)$,
2. for every constant $a : K \in \mathbb{S}$, there exists a term ρ_a such that $\vdash_{\mathbb{T}} \rho_a : \rho^a(K)$,
3. for every rewrite rule $\ell \hookrightarrow r \in \mathbb{S}$, we have $\rho(\ell) \equiv_{\beta(\eta)\mathcal{R}} \rho(r)$,

where ρ is defined on contexts and substitutions by

$$\begin{array}{lcl}
\rho(\varnothing) & := & \varnothing \\
\rho(\Gamma, x : A) & := & \rho(\Gamma),\ \vec{x} : \vec{\mu}(A),\ x^* : \rho(A)\ \vec{x} \\
\rho(\theta, x \leftarrow N) & := & \rho(\theta),\ \vec{x} \leftarrow \vec{\mu}(N),\ x^* \leftarrow \rho(N).
\end{array}$$

The first two conditions are the same as in LF. The third condition, specific to $\lambda\Pi/\mathcal{R}$, is necessary so that convertibility is preserved by logical relations.

Note that for every variable x that occurs in a term t, the $n+1$ variables $x_1, \ldots, x_n, x^*$ occur in the translated term $\rho(t)$. Therefore, if Γ declares m variables, $\rho(\Gamma)$ declares $m(n+1)$ variables.

For simplicity, when assigning values to the parameters, we will write $\rho(c)$ and $\rho(a)$ instead of ρ_c and ρ_a.

10.3.2 Running Example

In Example 10.9, we discussed how the equality between `MulDivGr;DivMulGr` and the identity of `MulGr` can be offloaded to the definitional equality of $\lambda\Pi/\mathcal{R}$ extended with η-conversion. In general, however, definitional equality—even with η-conversion—is not

sufficient to show the equality between two morphisms, and we have to resort to a propositional equality. Those situations are one of the applications of logical relations.

For the sake of example, we now show how to prove propositional equality between `MulDivGr`; `DivMulGr` and the identity of `MulGr`.

Example 10.18. *We show the equality between* `MulDivGr`; `DivMulGr` *and the identity of* `MulGr` *using a propositional equality on terms and an equivalence on propositions. To formalize that argument, we capture the invariant as a binary logical relation on* `MulDivGr`; `DivMulGr`*, abbreviated* μ_1*, and on the identity of* `MulGr`*, abbreviated* μ_2*.*

We first define a binary logical relation on `PLeq`*. The parameter* $\rho(\iota)$ *must be of type* $\mu_1(\iota) \to \mu_2(\iota) \to$ Type*, that is* $\iota \to \iota \to$ Type*. It captures the following invariant: the two translations of any element of type* ι *must be equal.*

$$\rho(\iota) := \lambda x_1, x_2 : \iota.\ Prf\ (x_1 = x_2)$$

$\rho(Prop)$ *must be of type* $\mu_1(Prop) \to \mu_2(Prop) \to$ Type*, that is* $Prop \to Prop \to$ Type*. It captures the following invariant: the two translations of any proposition must be equivalent.*

$$\rho(Prop) := \lambda p_1, p_2 : Prop.\ Prf\ ((p_1 \Rightarrow p_2) \wedge (p_2 \Rightarrow p_1))$$

The parameter $\rho(Prf)$ *must be of type*

$$\Pi p_1 : \mu_1(Prop).\ \Pi p_2 : \mu_2(Prop).\ \Pi H : \rho(Prf)\ p_1\ p_2.\ \mu_1(Prf\ p) \to \mu_2(Prf\ p) \to \text{Type}$$

that is $\Pi p_1, p_2 : Prop.\ \Pi H : Prf\ ((p_1 \Rightarrow p_2) \wedge (p_2 \Rightarrow p_1)).\ Prf\ p_1 \to Prf\ p_2 \to$ Type*. There is no invariant for proofs, so we only require a proof of a trivially true proposition* $\top$*.*

$$\begin{aligned}\rho(Prf) := \lambda p_1, p_2 : Prop.\ \lambda H : Prf((p_1 \Rightarrow p_2) \wedge (p_2 \Rightarrow p_1)).\\ \lambda H_1 : Prf\ p_1.\ \lambda H_2 : Prf\ p_2.\ Prf\ \top\end{aligned}$$

The parameters for the remaining constants of `PLeq` *capture that the invariant on* $Prop$ *is satisfied by* $\Rightarrow$*,* $\wedge$ *and* $=$*, and that the trivial invariant on* Prf *is satisfied by the constants encoding the natural deduction inference rules. Such parameters are easy to define.*

We now extend ρ *to the constants of* `MulGr`*. The parameter* $\rho(\times)$ *must be of type*

$$\begin{aligned}&\Pi x_1 : \mu_1(\iota).\ \Pi x_2 : \mu_2(\iota).\ \Pi H_x : \rho(\iota)\ x_1\ x_2.\\ &\Pi y_1 : \mu_1(\iota).\ \Pi y_2 : \mu_2(\iota).\ \Pi H_y : \rho(\iota)\ y_1\ y_2.\\ &\rho(\iota)\ (\mu_1(\times)\ x_1\ y_1)\ (\mu_2(\times)\ x_2\ y_2)\end{aligned}$$

that is $\Pi x_1, x_2 : \iota.\ Prf\ (x_1 = x_2) \to \Pi y_1, y_2 : \iota.\ Prf\ (y_1 = y_2) \to Prf\ (x_1 \times y_1 = x_2 \times y_2)$*. It follows that* $\rho(\times)$ *captures the following proof obligation: if* $x_1 = x_2$ *and* $y_1 = y_2$ *then* $x_1 \times y_1 = x_2 \times y_2$*.*

$$\begin{aligned}\rho(\times) := \lambda x_1, x_2 : \iota.\ \lambda H_x : Prf\ (x_1 = x_2).\ \lambda y_1, y_2 : \iota.\ \lambda H_y : Prf\ (y_1 = y_2).\\ \text{leib}\ x_1\ x_2\ H_x\ (\lambda z.\ x_1 \times y_1 = z \times y_2)\\ (\text{leib}\ y_1\ y_2\ H_y\ (\lambda z.\ x_1 \times y_1 = x_1 \times z)\ (\text{refl}\ (x_1 \times y_1)))\end{aligned}$$

Similarly, the parameter $\rho(1)$ *captures the proof obligation that* $1 = 1$*, and the parameter* $\rho(\text{inv})$ *captures the proof obligation that if* $x_1 = x_2$ *then* inv $x_1 =$ inv x_2*.*

$$\rho(1) := \text{refl}\ 1$$

$$\rho(\text{inv}) := \lambda x_1, x_2 : \iota.\ \lambda H : Prf\ (x_1 = x_2).\ \text{leib}\ x_1\ x_2\ H\ (\lambda z : \iota.\ \text{inv}\ x_1 = \text{inv}\ z)\ (\text{refl}\ (\text{inv}\ x_1))$$

To ensure this is a well-formed logical relation, we have to check the condition on the rewrite rules. For instance, for the rewrite rule $x \times 1 \hookrightarrow x$, we have $\rho(x \times 1)$ of type $Prf\ (x_1 \times 1 = x_2 \times 1)$ and $\rho(x)$ of type $Prf\ (x_1 = x_2)$. These two types are definitionally equal, and therefore $\rho(x \times 1)$ and $\rho(x)$ are two proofs of the same proposition. If we assume that our framework implements definitional proof irrelevance (two proofs of the same proposition are definitionally equal), then the condition $\rho(x \times 1) \equiv_{\beta(\eta)\mathcal{R}} \rho(x)$ holds. The other rewrite rules are treated similarly.

10.3.3 Abstraction Theorem

We have seen that, if M has type A, we want $\rho(M)$ to be of type $\rho(A)\ \vec{\mu}(M)$, where $\rho(A)$ is intuitively an invariant that must be satisfied by $\vec{\mu}(M)$. The Abstraction theorem extends this idea to the three-level hierarchy of LF and $\lambda\Pi/\mathcal{R}$.

Theorem 10.19: Abstraction

Let ρ be a logical relation on $\mu_1, \ldots, \mu_n$.

1. If $\vdash_{\mathbb{S}} \Gamma$, then $\vdash_{\mathbb{T}} \rho(\Gamma)$.
2. If $\Gamma \vdash_{\mathbb{S}} M : A$, then $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(M) : \rho(A)\ \vec{\mu}(M)$.
3. If $\Gamma \vdash_{\mathbb{S}} A : K$, then $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(A) : \rho^A(K)$.
4. If $\Gamma \vdash_{\mathbb{S}} K :$ Kind, then $\Gamma \vdash_{\mathbb{S}} R : K$ entails $\rho(\Gamma) \vdash_{\mathbb{T}} \rho^R(K) :$ Kind.

Note that, in [RS13], the preservation of conversion is part of the Abstraction theorem. This is because in [RS13], the conversion is typed, so that both conversion and typing depend on each other, and therefore both proofs must be intertwined. With our untyped conversion (see Remark 8.8), the preservation of conversion can be proved separately and before proving the preservation of typing, making the proofs much simpler. Our proof needs a simple substitution lemma and a conversion lemma.

Lemma 10.20 (Substitution)**.** *Let ρ be a logical relation on $\mu_1, \ldots, \mu_n$, and θ be a substitution.*

1. $\rho(M\theta) = \rho(M)\rho(\theta)$,
2. $\rho(A\theta) = \rho(A)\rho(\theta)$,
3. $\rho^{R\theta}(K\theta) = \rho^R(K)\rho(\theta)$.

Proof. By induction on the terms.

— Variable: If θ does not substitute x, then by definition $\rho(\theta)$ does not substitute x^*. Therefore $\rho(x\theta) = \rho(x) = x^* = x^*\rho(\theta) = \rho(x)\rho(\theta)$.

If θ substitutes x by N, then by definition $\rho(\theta)$ substitutes x^* by $\rho(N)$. Therefore $\rho(x\theta) = \rho(N) = x^*\rho(\theta) = \rho(x)\rho(\theta)$.

— Application: We have $\rho((M\ N)\theta) = \rho(M\theta\ N\theta) = \rho(M\theta)\ \vec{\mu}(N\theta)\ \rho(N\theta)$. We have $\rho(M\theta) = \rho(M)\rho(\theta)$ and $\rho(N\theta) = \rho(N)\rho(\theta)$ using the induction hypotheses. Using Lemma 10.12, we have $\mu_i(N\theta) = \mu_i(N)\mu_i(\theta) = \mu_i(N)\rho(\theta)$ for every $i \in [\![1, n]\!]$. Therefore we derive $\rho((M\ N)\theta) = (\rho(M)\ \vec{\mu}(N)\ \rho(N))\rho(\theta) = \rho(M\ N)\rho(\theta)$.

— Type: We have $\rho^{R\theta}(\mathsf{Type}\theta) = \rho^{R\theta}(\mathsf{Type}) = \vec{\mu}(R\theta) \to \mathsf{Type}$. Using Lemma 10.12, we have $\mu_i(R\theta) = \mu_i(R)\mu_i(\theta) = \mu_i(R)\rho(\theta)$ for every $i \in [\![1, n]\!]$. Therefore we derive $\rho^{R\theta}(\mathsf{Type}\theta) = \vec{\mu}(R)\rho(\theta) \to \mathsf{Type} = (\vec{\mu}(R) \to \mathsf{Type})\rho(\theta) = \rho^R(\mathsf{Type})\rho(\theta)$.

— The other cases are shown similarly. □

Lemma 10.21 (Conversion)**.** *Let ρ be a logical relation on $\mu_1, \ldots, \mu_n$.*

1. *If $A \equiv_{\beta(\eta)\mathcal{R}} B$ in $\mathbb{S}$, then $\rho(A) \equiv_{\beta(\eta)\mathcal{R}} \rho(B)$ in $\mathbb{T}$.*
2. *If $K \equiv_{\beta(\eta)\mathcal{R}} K'$ in $\mathbb{S}$ then $\rho^R(K) \equiv_{\beta(\eta)\mathcal{R}} \rho^R(K')$ in $\mathbb{T}$.*

Proof. By induction on the construction of $\equiv_{\beta(\eta)\mathcal{R}}$.

— We have $\rho((\lambda x : A.\ M)\ N) = (\lambda \vec{x} : \vec{\mu}(A).\ \lambda x^* : \rho(A)\ \vec{x}.\ \rho(M))\ \vec{\mu}(N)\ \rho(N)$, which β-reduces to $\rho(M)\rho([x \leftarrow N])$. We derive $\rho((\lambda x : A.\ M)\ N) \equiv_{\beta\mathcal{R}} \rho(M[x \leftarrow N])$ using Lemma 10.20. Similarly, we have $\rho((\lambda x : A.\ B)\ M) \equiv_{\beta\mathcal{R}} \rho(B[x \leftarrow M])$.

— Suppose that x is not free in M. Then the variables $\vec{x}, x^*$ are not free in $\rho(M)$. We have $\rho(\lambda x : A.\ M\ x) = \lambda \vec{x} : \vec{\mu}(A).\ \lambda x^* : \rho(A)\ \vec{x}.\ \rho(M)\ \vec{x}\ x^*$. Using η-conversion, we get to $\rho(\lambda x : A.\ M\ x) \equiv_{\beta\eta\mathcal{R}} \rho(M)$.

— Let $\ell \hookrightarrow r \in \mathbb{S}$ and θ be a substitution. Using Lemma 10.20, we have $\rho(\ell\theta) = \rho(\ell)\rho(\theta)$ and $\rho(r\theta) = \rho(r)\rho(\theta)$. We derive $\rho(\ell\theta) = \rho(\ell)\rho(\theta) \equiv_{\beta(\eta)\mathcal{R}} \rho(r)\rho(\theta) = \rho(r\theta)$ by definition of theory morphism.

— The cases for applications, abstractions and dependent types follow from the induction hypotheses.

— Reflexivity, symmetry, and transitivity are immediate. □

The following lemma will be useful to prove Theorem 10.19.

Lemma 10.22. *If $R \equiv_{\beta(\eta)\mathcal{R}} R'$ then $\rho^R(K) \equiv_{\beta(\eta)\mathcal{R}} \rho^{R'}(K)$.*

Proof. By induction on K. The case $\rho^R(\Pi x : A.\ K)$ follows from the induction hypothesis on K using $R\ x \equiv_{\beta(\eta)\mathcal{R}} R'\ x$. The case $\rho^R(\mathsf{Type})$ follows from Lemma 10.12. □

Proof of Theorem 10.19. By induction on the typing derivations.

— Empty-Ctx (case 1): Since $\rho(\varnothing) = \varnothing$, we derive $\vdash_{\mathbb{T}} \rho(\varnothing)$ using Empty-Ctx.

— Decl-Var (case 1): By induction hypotheses, we have

$$\begin{array}{rl} & \vdash_{\mathbb{T}} \rho(\Gamma) \\ \text{and} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho(A) : \vec{\mu}(A) \to \mathsf{Type}. \end{array}$$

Using Theorem 10.10, we have $\mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(A) : \mathsf{Type}$. Since $x \notin \Gamma$, we have $x_i \notin \rho(\Gamma)$ and $x^* \notin \rho(\Gamma)$. We derive $\vdash_{\mathbb{T}} \rho(\Gamma), \vec{x} : \vec{\mu}(A)$ using weakening and Decl-Var several times. We have $\rho(\Gamma), \vec{x} : \vec{\mu}(A), x^* : \rho(A) \vdash_{\mathbb{T}} \rho(A)\ \vec{x}$ using App-Type and weakening. We derive $\vdash_{\mathbb{T}} \rho(\Gamma), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x}$ using Decl-Var.

— Sort (case 4): Suppose that $\Gamma \vdash_{\mathbb{S}} R : \mathsf{Type}$ for some R. We have $\vdash_{\mathbb{T}} \rho(\Gamma)$ by induction hypothesis. Using Theorem 10.10, we get $\mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(R) : \mathsf{Type}$. We derive $\rho(\Gamma) \vdash_{\mathbb{T}} \vec{\mu}(R) \to \mathsf{Type} : \mathsf{Kind}$ using Sort, weakening and Prod-Kind several times.

— Const-Obj (case 2): We get $\vdash_{\mathbb{T}} \rho(\Gamma)$ by induction hypothesis. By definition, we have $\vdash_{\mathbb{T}} \rho_c : \rho(A)\ \vec{\mu}(c)$. Using weakening, we derive $\rho(\Gamma) \vdash_{\mathbb{T}} \rho_c : \rho(A)\ \vec{\mu}(c)$.

— Const-Type (case 3): We get $\vdash_{\mathbb{T}} \rho(\Gamma)$ by induction hypothesis. By definition, we have $\vdash_{\mathbb{T}} \rho_a : \rho^a(K)$. Using weakening, we derive $\rho(\Gamma) \vdash_{\mathbb{T}} \rho_a : \rho^a(K)$.

— Var (case 2): By induction hypothesis, we have $\vdash_{\mathbb{T}} \rho(\Gamma)$. Since $x : A \in \Gamma$, we have $x^* : \rho(A)\ \vec{x} \in \rho(\Gamma)$. We derive $\rho(\Gamma) \vdash_{\mathbb{T}} x^* : \rho(A)\ \vec{x}$ using Var.

— Prod-Type (case 3): By induction hypotheses, we have

$$\begin{aligned} &\rho(\Gamma) \vdash_{\mathbb{T}} \rho(A) : \vec{\mu}(A) \to \mathsf{Type} \\ \text{and}\quad &\rho(\Gamma), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x} \vdash_{\mathbb{T}} \rho(B) : \vec{\mu}(B) \to \mathsf{Type}. \end{aligned}$$

Using Theorem 10.10, we have

$$\begin{aligned} &\mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(A) : \mathsf{Type} \\ \text{and}\quad &\mu_i(\Gamma), x_i : \mu_i(A) \vdash_{\mathbb{T}} \mu_i(B) : \mathsf{Type}. \end{aligned}$$

We have $\rho(\Gamma), \vec{f} : \vec{\mu}(\Pi x : A.\ B), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x} \vdash_{\mathbb{T}} f_i\ x_i : \mu_i(B)$ using App-Obj. Using weakening and then App-Type several times, we get

$$\rho(\Gamma), \vec{f} : \vec{\mu}(\Pi x : A.\ B), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x} \vdash_{\mathbb{T}} \rho(B)\ (f_1\ x_1)\ \ldots\ (f_n\ x_n) : \mathsf{Type}.$$

Using Prod-Type, Abs-Type and weakening several times, we derive

$$\begin{aligned} \rho(\Gamma) \vdash_{\mathbb{T}}\ & \lambda \vec{f} : \vec{\mu}(\Pi x : A.\ B).\ \Pi \vec{x} : \vec{\mu}(A).\ \Pi x^* : \rho(A)\ \vec{x}. \\ & \rho(B)\ (f_1\ x_1)\ \ldots\ (f_n\ x_n) : \vec{\mu}(\Pi x : A.\ B) \to \mathsf{Type}. \end{aligned}$$

— Prod-Kind (case 4): Suppose that $\Gamma \vdash_{\mathbb{S}} R : \Pi x : A.\ K$ for some R. Therefore we have $\Gamma, x : A \vdash_{\mathbb{S}} R\ x : K$. By induction hypotheses, we have

$$\begin{aligned} &\rho(\Gamma) \vdash_{\mathbb{T}} \rho(A) : \vec{\mu}(A) \to \mathsf{Type} \\ \text{and}\quad &\rho(\Gamma), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x} \vdash_{\mathbb{T}} \rho^{R\ x}(K) : \mathsf{Kind}. \end{aligned}$$

Using Theorem 10.10, we have $\mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(A) : \mathsf{Type}$. Using Prod-Kind and weakening several times, we derive

$$\rho(\Gamma) \vdash_{\mathbb{T}} \Pi \vec{x} : \vec{\mu}(A).\ \Pi x^* : \rho(A)\ \vec{x}.\ \rho^{R\ x}(K) : \mathsf{Kind}.$$

— Abs-Obj (case 2): By induction hypotheses, we have

$$\begin{aligned} &\rho(\Gamma) \vdash_{\mathbb{T}} \rho(A) : \vec{\mu}(A) \to \mathsf{Type} \\ \text{and}\quad &\rho(\Gamma), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x} \vdash_{\mathbb{T}} \rho(B) : \vec{\mu}(B) \to \mathsf{Type} \\ \text{and}\quad &\rho(\Gamma), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x} \vdash_{\mathbb{T}} \rho(M) : \rho(B)\ \vec{\mu}(M). \end{aligned}$$

Using Theorem 10.10, we have

$$\begin{array}{ll} & \mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(A) : \mathsf{Type} \\ \text{and} & \mu_i(\Gamma), x_i : \mu_i(A) \vdash_{\mathbb{T}} \mu_i(B) : \mathsf{Type} \\ \text{and} & \mu_i(\Gamma), x_i : \mu_i(A) \vdash_{\mathbb{T}} \mu_i(M) : \mu_i(B). \end{array}$$

Using Abs-Obj and weakening several times, we derive

$$\rho(\Gamma) \vdash_{\mathbb{T}} \lambda \vec{x} : \vec{\mu}(A).\ \lambda x^* : \rho(A)\ \vec{x}.\ \rho(M) : \Pi \vec{x} : \vec{\mu}(A).\ \Pi x^* : \rho(A)\ \vec{x}.\ \rho(B)\ \vec{\mu}(M).$$

Using Conv-Type, we get $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(\lambda x : A.\ M) : \rho(\Pi x : A.\ B)\ \vec{\mu}(\lambda x : A.\ M)$, as we easily have $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(\Pi x : A.\ B)\ \vec{\mu}(\lambda x : A.\ M) : \mathsf{Type}$.

— Abs-Type (case 3): By induction hypotheses, we have

$$\begin{array}{ll} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho(A) : \vec{\mu}(A) \to \mathsf{Type} \\ \text{and} & \rho(\Gamma), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x} \vdash_{\mathbb{T}} \rho^{(\lambda x : A.\ B)\ x}(K) : \mathsf{Kind} \\ \text{and} & \rho(\Gamma), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x} \vdash_{\mathbb{T}} \rho(B) : \rho^B(K). \end{array}$$

Note that we have applied the fourth item with $R := (\lambda x : A.\ B)\ x$. We have $\rho^B(K) \equiv_{\beta\mathcal{R}} \rho^{(\lambda x : A.\ B)\ x}(K)$ using Lemma 10.22. Using Conv-Kind, it follows that $\rho(\Gamma), \vec{x} : \vec{\mu}(A), x^* : \rho(A)\ \vec{x} \vdash_{\mathbb{T}} \rho(B) : \rho^{(\lambda x : A.\ B)\ x}(K)$.

Using Theorem 10.10, we have $\mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(A) : \mathsf{Type}$. Using Abs-Type and weakening several times, we derive

$$\rho(\Gamma) \vdash_{\mathbb{T}} \lambda \vec{x} : \vec{\mu}(A).\ \lambda x^* : \rho(A)\ \vec{x}.\ \rho(B) : \Pi \vec{x} : \vec{\mu}(A).\ \Pi x^* : \rho(A)\ \vec{x}.\ \rho^{(\lambda x : A.\ B)\ x}(K).$$

— App-Obj (case 2): By induction hypotheses and Conv-Type, we have

$$\begin{array}{ll} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho(M) : \Pi \vec{x} : \vec{\mu}(A).\ \Pi x^* : \rho(A)\ \vec{x}.\ \rho(B)\ (\mu_1(M)\ x_1)\ \ldots\ (\mu_n(M)\ x_n) \\ \text{and} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho(N) : \rho(A)\ \vec{\mu}(N). \end{array}$$

Using Theorem 10.10, we get $\mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(N) : \mu_i(A)$. Using App-Obj and weakening several times, we derive

$$\begin{array}{l} \rho(\Gamma) \vdash_{\mathbb{T}} \rho(M)\ \vec{\mu}(N)\ \rho(N) : \\ \qquad (\rho(B)\ (\mu_1(M)\ x_1)\ \ldots\ (\mu_n(M)\ x_n))[\vec{x} \leftarrow \vec{\mu}(N), x^* \leftarrow \rho(N)] \end{array}$$

that is

$$\begin{array}{l} \rho(\Gamma) \vdash_{\mathbb{T}} \rho(M)\ \vec{\mu}(N)\ \rho(N) : \\ \qquad \rho(B)[\vec{x} \leftarrow \vec{\mu}(N), x^* \leftarrow \rho(N)]\ (\mu_1(M)\ \mu_1(N))\ \ldots\ (\mu_n(M)\ \mu_n(N)). \end{array}$$

Using Lemma 10.20, we derive $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(M\ N) : \rho(B[x \leftarrow N])\ \vec{\mu}(M\ N)$.

— App-Type (case 3): By induction hypotheses, we have

$$\begin{array}{ll} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho(A) : \Pi \vec{x} : \vec{\mu}(B).\ \Pi x^* : \rho(B)\ \vec{x}.\ \rho^{A\ x}(K) \\ \text{and} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho(M) : \rho(B)\ \vec{\mu}(M). \end{array}$$

Using Theorem 10.10, we get $\mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(M) : \mu_i(B)$. Using App-Type and weakening several times, we derive

$$\rho(\Gamma) \vdash_{\mathbb{T}} \rho(A)\ \vec{\mu}(M)\ \rho(M) : \rho^{A\ x}(K)[\vec{x} \leftarrow \vec{\mu}(M), x^* \leftarrow \rho(M)].$$

Using Lemma 10.20, we obtain $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(A\ M) : \rho^{A\ M}(K[x \leftarrow M])$.

— Conv-Type (case 2): By induction hypotheses, we have

$$\begin{array}{rl} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho(M) : \rho(A)\ \vec{\mu}(M) \\ \text{and} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho(B) : \vec{\mu}(B) \to \mathsf{Type}. \end{array}$$

Using Theorem 10.10, we have

$$\begin{array}{rl} & \mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(M) : \mu_i(A) \\ \text{and} & \mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(B) : \mathsf{Type}. \end{array}$$

Since $A \equiv_{\beta(\eta)\mathcal{R}} B$ in $\mathbb{S}$, we have $\mu_i(A) \equiv_{\beta(\eta)\mathcal{R}} \mu_i(B)$ in $\mathbb{T}$ using Lemma 10.12 and $\rho(A) \equiv_{\beta(\eta)\mathcal{R}} \rho(B)$ in $\mathbb{T}$ using Lemma 10.21. Therefore, using weakening, Conv-Type and App-Type, we get $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(B)\ \vec{\mu}(M) : \mathsf{Type}$. We derive $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(M) : \rho(B)\ \vec{\mu}(M)$ using Conv-Type.

— Conv-Kind (case 3): By induction hypotheses, we have

$$\begin{array}{rl} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho(A) : \rho^A(K) \\ \text{and} & \rho(\Gamma) \vdash_{\mathbb{T}} \rho^A(K') : \mathsf{Kind}. \end{array}$$

Using Theorem 10.10, we have

$$\begin{array}{rl} & \mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(A) : \mu_i(K) \\ \text{and} & \mu_i(\Gamma) \vdash_{\mathbb{T}} \mu_i(K') : \mathsf{Kind}. \end{array}$$

Since $K \equiv_{\beta(\eta)\mathcal{R}} K'$ in $\mathbb{S}$, we have $\mu_i(K) \equiv_{\beta(\eta)\mathcal{R}} \mu_i(K')$ in $\mathbb{T}$ using Lemma 10.12 and $\rho^A(K) \equiv_{\beta(\eta)\mathcal{R}} \rho^A(K')$ in $\mathbb{T}$ using Lemma 10.21. We derive $\rho(\Gamma) \vdash_{\mathbb{T}} \rho(A) : \rho^A(K')$ using Conv-Kind. □

10.4 Sort-Erasure Translations

In this section, we represent sort-erasure translations from hard-sorted to soft-sorted to unsorted logic. Here "sort" is the word that we will use for object-logic types to avoid any confusion with the types of $\lambda\Pi/\mathcal{R}$. The first translation erases hard sorting and instead captures the sorting relation via a special judgment. The second translation erases sorts entirely and captures them as unary predicates. Both of these translations have proved challenging, and to our knowledge, this is the first time that such translations are fully defined and verified. Our treatment will reveal several subtle critical design choices.

A key insight to formalize these is to adjust the target theories with additional features that allow carrying the sorting properties throughout the translation. Specifically, we encode dependent pairs to group a term with a proof about it, and we add a dependent variant of implication in order to access such proofs while building propositions. We define dependent pairs and dependent implication by taking advantage of rewriting.

10.4.1 Hard-Sorted, Soft-Sorted and Unsorted Logic

We first formalize unsorted logic UFOL, soft-sorted logic SFOL, and hard-sorted logic HFOL.

Example 10.23 (Unsorted Logic)**.** *In unsorted logic* UFOL*, all terms have the generic type* tm*.*

tm : Type
$Prop$: Type
Prf : $Prop \to$ Type

We define an implication $\Rightarrow$*, along with a rewrite rule that subsumes its two natural deduction rules, as discussed in Example 8.18.*

$\Rightarrow : Prop \to Prop \to Prop$
$Prf\ (p \Rightarrow q) \hookrightarrow Prf\ p \to Prf\ q$

We also add the usual universal quantifier to exemplify the treatment of binders.

$\forall : (tm \to Prop) \to Prop$
$Prf\ (\forall\ p) \hookrightarrow \Pi x : tm.\ Prf\ (p\ x)$

Sorted logic arises by adding a constant Set for object-logic sorts and allows quantification over sorted object-logic terms. There are two variants to define, going back to the definitions by Curry and Church, which we will refer to as soft-sorted logic SFOL and hard-sorted logic HFOL.

Example 10.24 (Soft-Sorted Logic)**.** *The theory* SFOL *arises from that of* UFOL *by adding a type* Set *of sorts and an external predicate* $\#$ *that captures the sorting of terms.*

tm : Type
Set : Type
$Prop$: Type
Prf : $Prop \to$ Type
$\# : tm \to Set \to$ Type

The definition of the implication is the same as in UFOL*. But we change the universal quantifier, which is unsorted in* UFOL*, to a polymorphic version that takes as an additional argument the sort* a *over which it quantifies.*

$\forall : \Pi a : Set.\ (\Pi x : tm.\ x \# a \to Prop) \to Prop$
$Prf\ (\forall\ a\ p) \hookrightarrow \Pi x : tm.\ \Pi h : x \# a.\ Prf\ (p\ x\ h)$

The body of the quantifier binds two assumptions: the bound variable and a proof that it has the sort a*. The latter ensures that bound variables are always well-sorted and thus that variables range over well-sorted objects only.*

Example 10.25 (Hard-Sorted Logic)**.** *Hard-sorted logic* HFOL *uses the type of sorts* Set*, and* El *that maps a sort to the type of their elements.*

Set : Type

$El : Set \to$ Type

$Prop$: Type

$Prf : Prop \to$ Type

Thus, object-logic terms of sort a *have type* $El\ a$*.*

The encoding of the implication is the same as for UFOL *and* SFOL*. The polymorphic universal quantifier of* HFOL *differs from that of* SFOL*: instead of binding a variable* $x : tm$ *and a proof* $h : x \# a$*, we directly bind a variable* $x : El\ a$*.*

$\forall : \Pi a : Set.\ (El\ a \to Prop) \to Prop$

$\mathsf{all_i} : \Pi a : Set.\ \Pi p : El\ a \to Prop.\ (\Pi x : El\ a.\ Prf\ (p\ x)) \to Prf\ (\forall\ a\ p)$

$\mathsf{all_e} : \Pi a : Set.\ \Pi p : El\ a \to Prop.\ Prf\ (\forall\ a\ p) \to \Pi x : El\ a.\ Prf\ (p\ x)$

Note that we encode the semantics of the universal quantifier here using two axioms as opposed to the rewrite rule used in UFOL *and* SFOL*. This is due to a technical issue that we will explain in Section 10.4.4.*

10.4.2 From Hard-Sorted Logic to Soft-Sorted Logic

Overview. To translate from HFOL to SFOL, the first intuition is to proceed in two steps: we erase the sort information using a theory morphism, and then we recover it using a logical relation. This approach is one of the example of [RS13], where bindings are exemplified by λ-abstractions instead of quantifiers.

The first step is to define a morphism, mapping sorts to themselves and *erasing* the hard sort information by mapping every type $El\ a$ to the type tm. The constants Set, $Prop$ and Prf are mapped to themselves.

$\mu(El) := \lambda a : Set.\ tm$

$\mu(\forall) := \lambda a : Set.\ \lambda p : tm \to Prop.\ \forall\ a\ (\lambda x : tm.\ \lambda h : x \# a.\ p\ x)$

Then, in a second step, we recover the sort information by proving that whenever we have $M : El\ a$ in HFOL, we can show (i.e., give a term of type) $\mu(M) \# \mu(a)$ in SFOL. To do so, we define a unary logical relation on the morphism. Terms of type Set are mapped by the morphism to themselves, so $\rho(Set)$ can be any trivially satisfied predicate, for instance $\top$.

$\rho(Set) := \lambda a : Set.\ Prf\ \top$

$\rho(El) := \lambda a : Set.\ \lambda a^* : Prf\ \top.\ \lambda x : tm.\ x \# a$

So far, we have only translated the *syntax* of HFOL. We also need to translate the *proof rules* of HFOL. However, we encounter a problem. The proof rule $\mathsf{all_e}$ takes as argument $x : El\ a$. Therefore $\mu(\mathsf{all_e})$ takes as argument $x : tm$. But to define $\mu(\mathsf{all_e})$ in SFOL, we must have access to the sort-preservation invariant $\mu(x) \# \mu(a)$. Thus, we cannot define the

morphism for the proof rules without already using the logical relation on the syntax—whereas the logical relation must be defined after the theory morphism. The translation in [RS13] worked out because it did not cover the proof rules.

One solution is to define a mutually recursive morphism and logical relation. This is essentially the approach followed in [Tra24b] for particular theories of $\lambda\Pi/\mathcal{R}$. We generalized that idea to a systematic meta-theorem for $\lambda\Pi/\mathcal{R}$, allowing the morphism to flexibly make use of the invariant established by the logical relation whenever it was needed. However, the use of this formalism was very complex.

Instead, we identify a third approach as the most scalable: we pair up the morphism and the logical relation, and define both at once. For example, we map $El\ a$ to the dependent pair type $\Sigma x : tm.\ x \# a$. Thus, every translated term always carries its well-sortedness proof. While seemingly heavyweight in the use of pairs, this approach has the key advantage that it can technically be represented as a theory morphism, thus keeping the framework simple.

Of course, the syntax of $\lambda\Pi/\mathcal{R}$ does not feature dependent pairs. But that is not a problem, as we can encode inside $\lambda\Pi/\mathcal{R}$ the specific instances of dependent pairs we need for the translation.

Extending the Target Language. For our specific translation, we only need the types $\Sigma x : tm.\ x \# a$ for every $a : Set$. The declarations below extend `SFOL` to `SFOL'` by adding this type, called pair a.

pair : $Set \to$ Type

mk_pair : $\Pi a : Set.\ \Pi x : tm.\ x \# a \to$ pair a

fst : $\Pi a : Set.\ \Pi m :$ pair $a.\ tm$

snd : $\Pi a : Set.\ \Pi m :$ pair $a.$ (fst $a\ m$) $\# a$

We also specify rewrite rules, effectively making pair a behave like $\Sigma x : tm.\ x \# a$.

fst a (mk_pair $a\ x\ h$) $\hookrightarrow x$

snd a (mk_pair $a\ x\ h$) $\hookrightarrow h$

mk_pair a (fst $a\ m$) (snd $a\ m$) $\hookrightarrow m$

In particular, the second rewrite rule only preserves typing because of the first rewrite rule: the left-hand side is of type (fst a (mk_pair $a\ x\ h$)) $\# a$ while the right-hand side is of type $x \# a$.

Remark 10.26. *The combination of these rewrite rules with β-reduction is not confluent on untyped terms [Klo80], but is confluent in several typed cases [Pot81, CC96]. The case of typed terms of $\lambda\Pi/\mathcal{R}$ remains to be investigated.*

Defining the Translation. We can now give a morphism `HFOL` $\to$ `SFOL'`.

$\mu(Set) := Set$

$\mu(El) := \lambda a : Set.$ pair a

$\mu(Prop) := Prop$

$\mu(Prf) := Prf$

We set $\mu(\Rightarrow) := \Rightarrow$, and the condition of Definition 10.4 on the rewrite rule of $\Rightarrow$ is trivially satisfied.

To define $\mu(\forall)$, we have a predicate p that takes a pair as an argument, but we need to use the universal quantifier of soft-sorted logic, in which the predicate takes two arguments sequentially. We pair of the arguments together and apply p. The translations for $\mathsf{all_i}$ and $\mathsf{all_e}$ can be easily derived, as it suffices to pack elements into a pair or unpack them from a pair.

$$
\begin{array}{lcl}
\mu(\forall) & := & \lambda a : Set.\ \lambda p : \mathsf{pair}\ a \to Prop.\ \forall\ a\ (\lambda x.\ \lambda h.\ p\ (\mathsf{mk\_pair}\ a\ x\ h)) \\
\mu(\mathsf{all_i}) & := & \lambda a : Set.\ \lambda p : \mathsf{pair}\ a \to Prop. \\
 & & \lambda H : (\Pi m : \mathsf{pair}\ a.\ Prf\ (p\ m)). \\
 & & \lambda x : tm.\ \lambda h : x \# a. \\
 & & H\ (\mathsf{mk\_pair}\ a\ x\ h) \\
\mu(\mathsf{all_e}) & := & \lambda a : Set.\ \lambda p : \mathsf{pair}\ a \to Prop. \\
 & & \lambda H : Prf\ (\forall\ a\ (\lambda x.\ \lambda h.\ p\ (\mathsf{mk\_pair}\ a\ x\ h))) \\
 & & \lambda m : \mathsf{pair}\ a. \\
 & & H\ (\mathsf{fst}\ a\ m)\ (\mathsf{snd}\ a\ m)
\end{array}
$$

Note that we build a term of type $Prf\ (p\ (\mathsf{mk\_pair}\ a\ (\mathsf{fst}\ a\ m)\ (\mathsf{snd}\ a\ m)))$, although $\mu(\mathsf{all_e})$ must return an object of type $Prf\ (p\ m)$. These two types are convertible thanks to the third rewrite rule.

Without rewriting, we would have to represent the rewrite rules of dependent pairs as equality axioms in the logic. But these rewrite rules are used extensively in practice: because our translation pairs every term with the proof of its invariant, every operation on terms must unpair its arguments, operate on the components, and pair the results, thus introducing reducible expressions all over the places. For example, [IR11] used a similar pairing approach to translate type theory into set theory. However, it used LF, and practical applications of that translation were made difficult by the deduction steps required to reduce these expressions. In $\lambda\Pi/\mathcal{R}$, this burdensome task is discharged to the framework through rewriting.

10.4.3 From Soft-Sorted Logic to Unsorted Logic

Overview. To translate from `SFOL` to `UFOL`, the key intuition is to map every *sort* to a unary *predicate* on unsorted terms, and to use that predicate to relativize the quantifiers. Such a relativization process has already been used to translate from many-sorted logic to one-sorted logic [Her30, Wan52].

We map every sort a to a predicate $\mu(a)$ on unsorted terms, and then the external sorting relation $M \# a$ can be mapped to the proposition $\mu(a)\ \mu(M)$. The critical part of the translation is the treatment of bound variables in the universal quantifier. The intuitive idea is to translate a quantification $\forall\ a\ p$ over some sort a with predicate p to a relativized unsorted quantification of the form $\forall\ (\lambda x.\ (\mu(a)\ x) \Rightarrow (\mu(p)\ x))$. But in `SFOL`, the predicate p takes two arguments—a term x and a proof of $x \# a$. The translation of p therefore takes two arguments—a term x and a proof of $\mu(a)\ x$. The straightforward relativization fails here: the application of $\mu(p)$ is ill-typed.

We could try to erase the assumption $x \# a$, e.g., by translating it to a trivial type. But then we would run into the same problem as discussed above for `HFOL` $\to$ `SFOL`: we need

these assumptions when translating the proof rules. One way to realize the translation is to again extend the target language.

Extending the Target Language. We extend UFOL to UFOL′ by adding *dependent* implication [IR11], where the construction of the second argument may assume a proof of the first. UFOL′ replaces the declarations regarding implication by an encoding of dependent implication [BDG+23].

$$\Rightarrow_d : \Pi p : Prop.\ (Prf\ p \to Prop) \to Prop$$
$$Prf\ (p \Rightarrow_d q) \hookrightarrow \Pi h : Prf\ p.\ Prf\ (q\ h)$$

Dependent implication is the Curry-Howard analogue of dependent function type, just like plain implication is the analogue of simple function type. It is easy to give a theory morphism from UFOL to UFOL′ that shows that the usual implication is a special case of the dependent one.

Defining the Translation. We now define a theory morphism SFOL → UFOL′. The translation of most constants is straightforward:

$$\mu(tm) := tm$$
$$\mu(Set) := tm \to Prop$$
$$\mu(Prop) := Prop$$
$$\mu(Prf) := Prf$$
$$\mu(\#) := \lambda x : tm.\ \lambda a : tm \to Prop.\ Prf\ (a\ x)$$
$$\mu(\Rightarrow) := \Rightarrow$$

The condition on the rewrite rule of $\Rightarrow$ is trivially satisfied.

The translation of the universal quantifier is the key case. It succeeds now because we access a proof of $a\ x$.

$$\begin{aligned}\mu(\forall) \quad := \quad & \lambda a : tm \to Prop.\ \lambda p : (\Pi x : tm.\ Prf\ (a\ x) \to Prop).\\ & \forall\ (\lambda x : tm.\ (a\ x) \Rightarrow_d (\lambda h.\ p\ x\ h))\end{aligned}$$

The condition on the rewrite rule of $\forall$ is satisfied in UFOL′.

$$\begin{aligned}\mu(Prf\ (\forall\ a\ p)) \quad & \equiv_{\beta\mathcal{R}} \quad Prf\ (\forall\ (\lambda x : tm.\ (a\ x) \Rightarrow_d (\lambda h.\ p\ x\ h)))\\ & \equiv_{\beta\mathcal{R}} \quad \Pi x : tm.\ Prf\ ((a\ x) \Rightarrow_d (\lambda h.\ p\ x\ h))\\ & \equiv_{\beta\mathcal{R}} \quad \Pi x : tm.\ \Pi h : Prf\ (a\ x).\ Prf\ (p\ x\ h)\\ & \equiv_{\beta\mathcal{R}} \quad \mu(\Pi x : tm.\ \Pi h : x \#\ a.\ Prf\ (p\ x\ h))\end{aligned}$$

10.4.4 Axioms vs. Rewrite Rules

We can give both a computational and a deductive variant for HFOL, SFOL, and UFOL. In each case, we can obtain a morphism $\mu_{\mathbb{T}}$ from the deductive variant of $\mathbb{T}$ to its computational variant, following the blueprint of Section 10.2.1. Let us discuss how the translations from HFOL to SFOL and from SFOL to UFOL are impacted by the choice of the variants.

Typically, if the source theory is deductive, it does not matter if the target theory is deductive or computational—the translations can be established in essentially the same way. In particular, if the target theory $\mathbb{T}$ is deductive, we can simply compose the translation with $\mu_{\mathbb{T}}$ to obtain a translation into the computational variant of $\mathbb{T}$. But giving the translation into the computational variant directly can be much simpler because the rewrite rules of the target theory makes the necessary proofs a lot easier.

It would be unreasonable to expect a translation from a computational to a deductive variant, for example from computational `SFOL` to deductive `UFOL`. Indeed, if two types are convertible in the source theory, then their translation need to be convertible in the target theory.

But translations between computational variants can also pose subtle issues. Indeed, in Section 10.4.2, we have only given a translation from deductive `HFOL` to computational `SFOL`, as the translation from computational `HFOL` to computational `SFOL` fails. To understand what goes wrong, consider the computational variant of the universal quantifier using the rewrite rule

$$Prf\ (\forall\ a\ p) \hookrightarrow \Pi x : El\ a.\ Prf\ (p\ x)$$

After applying $\texttt{HFOL} \to \texttt{SFOL}'$, the translations of the left-hand side and the right-hand side are not convertible in $\texttt{SFOL}'$:

$$\begin{array}{lll} \mu(Prf\ (\forall\ a\ p)) & \equiv_{\beta\mathcal{R}} & \Pi x : tm.\ \Pi h : x \#\ \mu(a).\ Prf\ (\mu(p)\ (\mathsf{mk\_pair}\ \mu(a)\ x\ h)) \\ \mu(\Pi x : El\ a.\ Prf\ (p\ x)) & \equiv_{\beta\mathcal{R}} & \Pi x : \mathsf{pair}\ a.\ Prf\ (\mu(p)\ x) \end{array}$$

and therefore the theory morphism is not well-formed. The problem here is, essentially, that the former is uncurried while the latter is curried. Díaz-Caro and Dowek [DCD19] designed an extension of simply typed λ-calculus with Cartesian product, in which we have the currying/uncurrying conversion $(A \times B) \to C \equiv A \to B \to C$. It would be interesting to investigate whether such a conversion can be extended to dependent types, and whether $\lambda\Pi/\mathcal{R}$ can be extended with such a feature.

10.5 Embedding Data Types

Our use of dependent implication and dependent pairs in the previous translations is not a one-off trick. They are general techniques for realizing challenging translations. One common application is to define embeddings that identify a smaller data type as a fragment of a larger one. Here the difficulty is that a binding over the smaller source data type must be relativized when translating it to a binding of the larger target data type.

As an example, we give an embedding of natural numbers into integers, in which we need to relativize bindings of integers with the property of being non-negative. This property serves as an invariant that the translated terms must satisfy. As for the sort-erasure translations, it is critical to prove this invariant by induction over terms, and the translation must be intertwined with the proof of the invariant.

We start by encoding the theory of natural numbers and the theory of integers as sorted languages, i.e., as extensions of `HFOL`.

Example 10.27 (Natural Numbers)**.** *The theory* $\texttt{HFOL} + \texttt{Nat}$ *of natural numbers is built on hard-sorted logic. For simplicity, we encode the universal quantifier using a rewrite rule.*

$\forall : \Pi a : Set.\ (El\ a \to Prop) \to Prop$

$Prf\ (\forall\ a\ p) \hookrightarrow \Pi x : El\ a.\ Prf\ (p\ x)$

We define the sort nat *for natural numbers, with the two constructors* 0 *and* succ*. The relation* $\geq$ *is reflexive and transitive.*

$\mathsf{nat} : Set$

$0 : El\ \mathsf{nat}$

$\mathsf{succ} : El\ \mathsf{nat} \to El\ \mathsf{nat}$

$\geq\ : El\ \mathsf{nat} \to El\ \mathsf{nat} \to Prop$

$\mathsf{ax}_1 : \Pi x : El\ \mathsf{nat}.\ Prf\ (x \geq x)$

$\mathsf{ax}_2 : \Pi x, y, z : El\ \mathsf{nat}.\ Prf\ (x \geq y) \to Prf\ (y \geq z) \to Prf\ (x \geq z)$

For any natural number x*, its successor* $\mathsf{succ}\ x$ *is greater than* x*. In other words, we have a proof of* $\mathsf{succ}\ x \geq x$*, and any proof of* $x \geq \mathsf{succ}\ x$ *leads to an inconsistency.*

$\mathsf{ax}_3 : \Pi x : El\ \mathsf{nat}.\ Prf\ (\mathsf{succ}\ x \geq x)$

$\mathsf{ax}_4 : \Pi x : El\ \mathsf{nat}.\ Prf\ (x \geq \mathsf{succ}\ x) \to \Pi P : Prop.\ Prf\ P$

Finally, we have the induction principle on natural numbers:

$$\begin{aligned}\mathsf{rec} :\ & \Pi P : El\ \mathsf{nat} \to Prop.\ Prf\ (P\ 0) \to \\ & \big(\Pi x : El\ \mathsf{nat}.\ Prf\ (P\ x) \to Prf\ (P\ (\mathsf{succ}\ x))\big) \to \\ & \Pi x : El\ \mathsf{nat}.\ Prf\ (P\ x)\end{aligned}$$

In this theory, we can prove that every natural number is non-negative using the proof term

$$\begin{aligned}&\mathsf{rec}\ (\lambda z : El\ \mathsf{nat}.\ z \geq 0)\ (\mathsf{ax}_1\ 0) \\ &\qquad (\lambda x : El\ \mathsf{nat}.\ \lambda H_x : Prf\ (x \geq 0).\ \mathsf{ax}_2\ (\mathsf{succ}\ x)\ x\ 0\ (\mathsf{ax}_3\ x)\ H_x)\end{aligned}$$

of type $\Pi x : El\ \mathsf{nat}.\ Prf\ (x \geq 0)$*.*

Example 10.28 (Integers)**.** *The theory of integers* $\texttt{HFOL} + \texttt{Int}$ *is like* $\texttt{HFOL} + \texttt{Nat}$*, with the sort* nat *renamed to* int*. We introduce a predecessor symbol* pred*. The fact that* pred *and* succ *are inverses is encoded using rewrite rules.*

$\mathsf{int} : Set$

$0 : El\ \mathsf{int}$

$\mathsf{succ} : El\ \mathsf{int} \to El\ \mathsf{int}$

$\mathsf{pred} : El\ \mathsf{int} \to El\ \mathsf{int}$

$\mathsf{succ}\ (\mathsf{pred}\ x) \hookrightarrow x$

$\mathsf{pred}\ (\mathsf{succ}\ x) \hookrightarrow x$

$\geq\ : El\ \mathsf{int} \to El\ \mathsf{int} \to Prop$

The normal terms of type $El\ \mathsf{int}$ *are in bijection with the mathematical integers.*

The relation $\geq$ is defined as in $\texttt{HFOL} + \texttt{Nat}$*. Additionally, for any integer x, its predecessor* pred x *is lower than x.*

$\mathsf{ax}_1 : \Pi x : El\ \mathsf{int}.\ Prf\ (x \geq x)$

$\mathsf{ax}_2 : \Pi x, y, z : El\ \mathsf{int}.\ Prf\ (x \geq y) \to Prf\ (y \geq z) \to Prf\ (x \geq z)$

$\mathsf{ax}_3 : \Pi x : El\ \mathsf{int}.\ Prf\ (\mathsf{succ}\ x \geq x)$

$\mathsf{ax}_4 : \Pi x : El\ \mathsf{int}.\ Prf\ (x \geq \mathsf{succ}\ x) \to \Pi P : Prop.\ Prf\ P$

$\mathsf{ax}_5 : \Pi x : El\ \mathsf{int}.\ Prf\ (x \geq \mathsf{pred}\ x)$

$\mathsf{ax}_6 : \Pi x : El\ \mathsf{int}.\ Prf\ (\mathsf{pred}\ x \geq x) \to \Pi P : Prop.\ Prf\ P$

The induction principle on integers

$$\begin{aligned}\mathsf{rec} : \quad & \Pi P : El\ \mathsf{int} \to Prop.\ Prf\ (P\ 0) \to \\ & \big(\Pi x : El\ \mathsf{int}.\ Prf\ (x \geq 0) \to Prf\ (P\ x) \to Prf\ (P\ (\mathsf{succ}\ x))\big) \to \\ & \big(\Pi x : El\ \mathsf{int}.\ Prf\ (0 \geq x) \to Prf\ (P\ x) \to Prf\ (P\ (\mathsf{pred}\ x))\big) \to \\ & \Pi x : El\ \mathsf{int}.\ Prf\ (P\ x)\end{aligned}$$

involves an additional induction step for pred*. The conditions $x \geq 0$ and $0 \geq x$ of the inductive steps may appear inessential, but they will be necessary to formalize the embedding of natural numbers into integers.*

Overview. The translation $\texttt{HFOL} + \texttt{Nat} \to \texttt{HFOL} + \texttt{Int}$ is structurally very similar to the translation $\texttt{HFOL} \to \texttt{SFOL}$. We intuitively map the sort nat to the sort int, but we need to recover the information that any natural number is mapped to a non-negative integer. Here the invariant of the translation is the predicate $x \geq 0$. Like the translation $\texttt{HFOL} \to \texttt{SFOL}$, we will employ dependent pairs. We need dependent pairs of the form $\Sigma x : El\ a.\ Prf\ (p\ x)$. The definitions below extend $\texttt{HFOL} + \texttt{Int}$ to $\texttt{HFOL} + \texttt{Int}'$ with an encoding of those dependent pairs.

$\mathsf{pair} : \Pi a : Set.\ (El\ a \to Prop) \to Set$

$\mathsf{mk\_pair} : \Pi a : Set.\ \Pi p : El\ a \to Prop.\ \Pi x : El\ a.\ Prf\ (p\ x) \to El\ (\mathsf{pair}\ a\ p)$

$\mathsf{fst} : \Pi a : Set.\ \Pi p : El\ a \to Prop.\ El\ (\mathsf{pair}\ a\ p) \to El\ a$

$\mathsf{snd} : \Pi a : Set.\ \Pi p : El\ a \to Prop.\ \Pi m : El\ (\mathsf{pair}\ a\ p).\ Prf\ (p\ (\mathsf{fst}\ a\ p\ m))$

$\mathsf{fst}\ a\ p\ (\mathsf{mk\_pair}\ a\ p\ x\ h) \hookrightarrow x$

$\mathsf{snd}\ a\ p\ (\mathsf{mk\_pair}\ a\ p\ x\ h) \hookrightarrow h$

$\mathsf{mk\_pair}\ a\ p\ (\mathsf{fst}\ a\ p\ m)\ (\mathsf{snd}\ a\ p\ m) \hookrightarrow m$

Defining the Translation. We can now define $\texttt{HFOL} + \texttt{Nat} \to \texttt{HFOL} + \texttt{Int}'$. The constants of hard-sorted logic are mapped to themselves. The sort of natural numbers is mapped to the sort that pairs an integer and a proof that it is non-negative.

$\mu(\mathsf{nat}) := \mathsf{pair}\ \mathsf{int}\ (\lambda x.\ x \geq 0)$

The parameters $\mu(0)$ requires a proof that 0 satisfies the invariant, and $\mu(\mathsf{succ})$ requires a proof that succ preserves the invariant.

$$\begin{aligned}
&\mu(0) := \mathsf{mk_pair}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\ 0\ (\mathsf{ax}_1\ 0)\\
&\mu(\mathsf{succ}) := \lambda m : El\ (\mathsf{pair}\ \mathsf{int}\ (\lambda x.\ x \geq 0)).\ \mathsf{mk_pair}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\\
&\qquad\qquad (\mathsf{succ}\ (\mathsf{fst}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\ m))\\
&\qquad\qquad \big(\mathsf{ax}_2\ (\mathsf{succ}\ (\mathsf{fst}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\ m))\ (\mathsf{fst}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\ m)\ 0\\
&\qquad\qquad\qquad (\mathsf{ax}_3\ (\mathsf{fst}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\ m))\\
&\qquad\qquad\qquad (\mathsf{snd}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\ m)\big)
\end{aligned}$$

We define $\mu(\geq)$ so that comparing two pairs means comparing their first elements. The parameter $\mu(\mathsf{ax}_1)$ is straightforward.

$$\begin{aligned}
&\mu(\geq) := \lambda m_1, m_2 : El\ (\mathsf{pair}\ \mathsf{int}\ (\lambda x.\ x \geq 0)).\\
&\qquad\qquad (\mathsf{fst}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\ m_1) \geq (\mathsf{fst}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\ m_2)\\
&\mu(\mathsf{ax}_1) := \lambda m : El\ (\mathsf{pair}\ \mathsf{int}\ (\lambda x.\ x \geq 0)).\ \mathsf{ax}_1\ (\mathsf{fst}\ \mathsf{int}\ (\lambda x.\ x \geq 0)\ m)
\end{aligned}$$

The parameters for ax_2, ax_3 and ax_4 are defined similarly.

Setting the Parameter $\mu(\mathsf{rec})$. The parameter $\mu(\mathsf{rec})$ is trickier. It takes as input a predicate P of type $El\ (\mathsf{pair}\ \mathsf{int}\ (\lambda x.\ x \geq 0)) \to Prop$, while the induction principle on integers requires a predicate of type $El\ \mathsf{int} \to Prop$. As for the translation `HFOL` $\to$ `SFOL`, we relativize the predicate using our invariant, thanks to dependent implication.

$$\begin{aligned}
&\Rightarrow_d : \Pi p : Prop.\ (Prf\ p \to Prop) \to Prop\\
&Prf\ (p \Rightarrow_d q) \hookrightarrow \Pi h : Prf\ p.\ Prf\ (q\ h)
\end{aligned}$$

We apply the induction principle of integers to the predicate

$$\lambda x : El\ \mathsf{int}.\ (x \geq 0) \Rightarrow_d (\lambda h.\ P\ (\mathsf{mk\_pair}\ \mathsf{int}\ (\lambda z.\ z \geq 0)\ x\ h))$$

of type $El\ \mathsf{int} \to Prop$. We then have to prove the base case (for 0) and the two inductive steps (for succ and pred).

For the base case, we are given a proof H_0 of type $Prf\ (P\ \mu(0))$, and we need to build a term of type $Prf\ ((0 \geq 0) \Rightarrow_d (\lambda h.\ P\ (\mathsf{mk\_pair}\ \mathsf{int}\ (\lambda z.\ z \geq 0)\ 0\ h)))$. Assuming h_0 of type $Prf\ (0 \geq 0)$, it remains to build a term of type $Prf\ (P\ (\mathsf{mk\_pair}\ \mathsf{int}\ (\lambda z.\ z \geq 0)\ 0\ h_0))$. This is not exactly H_0, as h_0 is not necessarily the second element of the pair $\mu(0)$. To get around this issue, we add an axiom for proof irrelevance—the principle stating that two proofs of the same proposition are equal.

$$\mathsf{proof\_irr} : \Pi p : Prop.\ \Pi h_1, h_2 : Prf\ p.\ \Pi q : Prf\ p \to Prop.\ Prf\ (q\ h_1) \to Prf\ (q\ h_2)$$

The induction step for succ is proved similarly. For the induction step for pred, we are in particular given a proof $H_\geq$ of type $Prf\ (0 \geq x)$. We need to build a term of type $Prf\ ((\mathsf{pred}\ x \geq 0) \Rightarrow_d (\lambda h.\ P\ (\mathsf{mk\_pair}\ \mathsf{int}\ (\lambda z.\ z \geq 0)\ (\mathsf{pred}\ x)\ h)))$. Assuming h_{pred} of type $Prf\ (\mathsf{pred}\ x \geq 0)$, it remains to build a term of type

$$Prf\ (P\ (\mathsf{mk\_pair}\ \mathsf{int}\ (\lambda z.\ z \geq 0)\ (\mathsf{pred}\ x)\ h_{\mathsf{pred}})).$$

Using ax_2 on $H_{\geq}$ and h_{pred}, we get a proof of pred $x \geq x$. Using the explosion principle ax_6, we obtain a term of type $Prf\ (P\ (\mathsf{mk\_pair}\ \mathsf{int}\ (\lambda z.\ z \geq 0)\ (\mathsf{pred}\ x)\ h_{\mathsf{pred}}))$, which achieves the case. The complete parameter $\mu(\mathsf{rec})$ has been formalized in Dedukti.

Remark 10.29. *An encoding of dependent pair with predicate subtyping has been proposed in [BDG+23]. These dependent pairs are proof-irrelevant on their second element. In other words, two pairs that have the same first element are convertible. We could use such proof-irrelevant dependent pairs instead of the explicit axiom* proof_irr*, leveraging even more the computational power of* $\lambda\Pi/\mathcal{R}$.

Remark 10.30. *In [Tra24b], we defined an embedding of natural numbers into integers using a mutually recursive morphism and logical relation. This construction is particularly intricate, and trivially true invariants must be carried through the induction. In contrast, the present translation is much simpler and only resorts to a morphism.*

Our embedding shows that natural numbers are mapped to non-negative integers. We can extend the theories `Nat` and `Int` with operations such as addition. Then we can extend the morphism accordingly, so that it additionally shows that the corresponding operations on integers preserve the invariant.

10.6 Implementation for Dedukti

Implementation. We developed a tool, called `TranslationTemplates`[1], that implements theory morphisms and unary logical relations for Dedukti.

`TranslationTemplates` takes two files representing a source theory $\mathbb{S}$ and a target theory $\mathbb{T}$, and outputs a new file that contains the result of the translation template. An additional argument controls if the generated file should capture a theory morphism or a logical relation. For every constant declared in the source file $\mathbb{S}$, the user must provide an appropriate parameter. The resulting file can be rechecked by Dedukti so that our code does not have to be a part of the trusted code base. The tool is written in OCaml in less than 400 lines of code and benefits from the Dedukti kernel and parser. All the examples of theory morphisms given in this chapter have been implemented in Dedukti and mechanically checked.

Demonstration. We illustrate `TranslationTemplates` on the theory morphism described in Section 10.2.1. For simplicity, we only consider the implication symbol. The source file `deduction.dk` contains the theory where natural deduction rules are encoded via axioms.

```
Prop : Type.
Prf : Prop -> Type.
imp : Prop -> Prop -> Prop.
imp_i : p : Prop -> q : Prop -> (Prf p -> Prf q) ->
        Prf (imp p q).
imp_e : p : Prop -> q : Prop -> Prf (imp p q) -> Prf p ->
        Prf q.
```

[1] Available at https://github.com/Deducteam/TranslationTemplates.

The theorem $\Pi p : Prop.\ Prf\ (p \Rightarrow p)$ is proved by $\lambda p : Prop.\ \mathsf{imp_i}\ p\ p\ (\lambda H : Prf\ p.\ H)$.

```
thm lemma_imp : p : Prop -> Prf (imp p p)
  := p => imp_i p p (H => H).
```

The target file `computation.dk` contains the theory where natural deduction rules are encoded via rewrite rules. Note that the symbol `Prf` is now declared with the keyword `def`, because it is definable with rewrite rules.

```
Prop : Type.
def Prf : Prop -> Type.
imp : Prop -> Prop -> Prop.
[p, q] Prf (imp p q) --> Prf p -> Prf q.
```

When asking a theory morphism from `deduction.dk` to `computation.dk`, our tool generates the following file.

```
#REQUIRE computation.

def Prop_mu : Type := TODO.
def Prf_mu : Prop_mu -> Type := TODO.
def imp_mu : Prop_mu -> Prop_mu -> Prop_mu := TODO.
def imp_i_mu : p : Prop_mu -> q : Prop_mu ->
               (Prf_mu p -> Prf_mu q) -> Prf_mu (imp_mu p q)
  := TODO.
def imp_e_mu : p : Prop_mu -> q : Prop_mu ->
               Prf_mu (imp_mu p q) -> Prf_mu p -> Prf_mu q
  := TODO.

thm lemma_imp_mu : p : Prop_mu -> Prf_mu (imp_mu p p)
  := p => imp_i_mu p p (H => H).
```

This is a skeleton of the translation, where the parameters must be filled in by the user instead of the `TODO`s. To help the user, the type of each parameter is displayed. Such parameters must be expressed in the target theory `computation.dk`. We can do so following the blueprint of Section 10.2.1. The statement and the proof of the theorem `lemma_imp` of the source theory have been *automatically* translated to the target theory. The file type-checks provided that the conditions of the theory morphism are fulfilled.

10.7 Conclusion

We have defined theory morphisms and logical relations for the $\lambda\Pi$-calculus modulo rewriting. These translation templates allow mechanically generating translations from parameters supplied by the user. The correctness of the translations is guaranteed, provided certain conditions on the parameters. Such translation templates therefore allow shifting most of the work to the framework and leaving only a small amount of work to the formalizer of an individual translation. Apart from an example of theory morphism in [Fel22], this is the first time that such translation templates are used systematically for a logical framework with rewriting.

Throughout our examples, we have dealt with the subtle trade-off between axioms and rewrite rules. On the one hand, when using rewrite rules instead of axioms, parts of

the proofs are directly discharged by the framework through computation. In particular, the presence of rewrite rules in the target theory simplifies the proof obligations generated by the translation templates. On the other hand, the presence of rewrite rules in the source theory creates additional constraints that need to be satisfied by the translation templates. These observations allow us to fully leverage the computational power of the $\lambda\Pi$-calculus modulo rewriting to formalize translations between theories.

Moreover, we have identified two subtle practices that allow representing translations that have previously proved challenging: the use of dependent pairs and of dependent implication. This observation is independent of rewriting and applies to other logical frameworks as well—but their use is considerably facilitated by rewriting. More generally, it indicates that efficient translations across languages may be critically enabled by deep technical tweaks to the framework or the target language.

We have implemented both templates in the Dedukti proof language, and we have applied them to mechanically check translations between a number of logics. These kinds of templates are critical for the interoperability of proof systems, as they allow for the batch translation of libraries along a theory morphism or a logical relation.

Chapter 11

Conclusion and Perspectives

11.1 Summary of the Contributions

In this thesis, we have investigated the interrelations between higher-order logics (Part I) and have strengthened the interoperability between different theories of the $\lambda\Pi$-calculus modulo rewriting (Part II). The first research axis is based on simply typed λ-calculus (used as a language for the logic), while the second research axis is based on a dependently typed λ-calculus with rewriting (used as a language for logics and proofs). Both research axes are carried out by establishing translations and proving that they are sound. We have defined individual translations (Chapters 4 and 9), as well as parameterized translations (Chapters 5 and 10). Such parameterized translations can be instantiated to generate various individual translations.

In Part I, we have defined embeddings of higher-order classical logic into intuitionistic logic, and of higher-order intuitionistic logic into minimal logic. We have done so by extending double-negation translations (Chapter 4) and monad translations (Chapter 5). Some results, straightforward in first-order logic, do not necessarily hold in higher-order logic, due to its intensional nature, but do under functional extensionality and propositional extensionality. These embeddings translate any proof, but in doing so they also modify the statement of the theorem. For particular theorems, we can constructivize the proof without modifying the statement. We have shown how to constructivize proofs of higher-order coherent formulas (Chapter 5) and higher-order geometric formulas (Chapter 6). Such results extend Barr's theorem to higher-order logic and to higher-order infinitary logic.

On this subject, another contribution of Part I is the definition and study of a higher-order infinitary logic (Chapter 6), that features higher-order quantifiers, infinite conjunctions and infinite disjunctions. We have investigated its syntax and semantics. In particular, we have shown a completeness theorem that entails the cut-elimination theorem for natural deduction.

In Part II, we have focused on the exchange of formal proofs between theories of the $\lambda\Pi$-calculus modulo rewriting. As a case study, we have extended a double-negation translation to the $\lambda\Pi$-calculus modulo rewriting, and have implemented it in Dedukti (Chapter 9). Furthermore, we have extended two translation templates to the $\lambda\Pi$-calculus modulo rewriting, namely theory morphisms and logical relations (Chapter 9). Theory morphisms allow us to formalize translations from a source theory

to a target theory, while logical relations allow us to show that certain invariants are preserved by the morphisms. Provided that the parameters supplied by the user satisfy certain conditions, the generated translations are guaranteed to be correct. We have instantiated these templates to formalize various translations between theories of the $\lambda\Pi$-calculus modulo rewriting, in particular sort-erasure translations. We have experienced that the presence of rewrite rules in the source theory creates additional constraints to satisfy, whereas the presence of rewrite rules in the target theory allows to discharge some proof obligations to the framework and therefore simplifies the work of the user. We have implemented these translation templates in Dedukti, boosting interoperability between Dedukti theories and thus between proof systems.

In the following, we present potential applications of the results of this thesis, identify goals for future work, and raise open questions.

11.2 Perspectives on Part I

Characterization Property for Double-Negation Translations. As seen in Chapter 4, Kuroda's translation and the Kolmogorov-Dowek-Werner translation do not satisfy the characterization property in higher-order logic, because of the lack of extensionality. More generally, we will have the same problem for any translation that inserts double negations inside the arguments of predicates. For this reason, we conjecture that it is not possible to define a double-negation translation that satisfies the characterization property in higher-order logic and without additional assumptions.

Infinitary Reduction. In Chapter 6, we have extended simple type theory with families of propositions indexed by an arbitrary countable set, and we have shown that the resulting reduction is neither confluent nor terminating. The problem is that, even if each A_i reduces to B_i, the family $\{i \mapsto A_i\}_{\mathcal{I}}$ does not necessarily reduce to $\{i \mapsto B_i\}_{\mathcal{I}}$ in a finite number of steps. Intuitively, we would like to allow the reduction of a family of propositions when all its components reduce in a finite number of steps. As future work, we could investigate whether this is feasible, and we could analyze the consequences of such reduction on the meta-theory.

Sequent Calculus for Higher-Order Infinitary Logic. We have defined a natural deduction proof calculus for higher-order infinitary logic, along with a cut-free variant. We have chosen natural deduction, as it allowed us to build upon the previous chapters.

In particular, we have extended Barr's theorem following the approach based on Friedman's trick [Pal02, vdB19]. An alternative approach is based on the cut-elimination theorem for sequent calculus [Neg03, Rat16, Neg21]. Negri's proof relies on the fact that cut-free proofs of geometric implications in the classical sequent calculus only use the rules of her intuitionistic multi-conclusion sequent calculus. Such an argument does not apply for natural deduction, so the cut-elimination theorem we have established cannot be used to prove Barr's theorem.

It would be interesting to investigate a sequent calculus for our higher-order infinitary logic, along with an intuitionistic multi-conclusion variant. This would pave the way for another proof of Barr's theorem for higher-order geometric formulas.

Topos-based Semantics of Higher-Order Infinitary Logic. In Chapter 6, we have studied a semantics of higher-order logic based on Heyting algebras. Alternatively, topos theory can be used to give a semantics to many logics. In particular, first-order geometric logic can be interpreted in a geometric category [MM92], and higher-order logic can be interpreted in an elementary topos [Joh02]. Grothendieck toposes are both complete and co-complete geometric categories and elementary toposes. We could thus investigate a semantics of higher-order infinitary logic in terms of Grothendieck toposes.

11.3 Perspectives on Part II

Exchange of Proofs between B and TLA+. B (or its variant EventB) [Abr96] is a formal method tool, widely used for the specification and verification of safety-critical systems such as the Paris metro line 14 [BKK+20, Lec24]. TLA+ [Lam02] is a specification formalism used in the industry, for example by Amazon to design complex software [NRZ+15]. Both B/Event-B and TLA+ are based on set theory, but they differ on their logical foundations: B/Event-B uses a typed logic, while TLA+ uses an untyped logic that does not distinguish terms and formulas.

The ICSPA project (Interoperable and Confident Set-based Proof Assistants) [AABB+26] aims at having interoperability between B/EventB and TLA+, using Dedukti as a middleware. To do so, both B/Event-B and TLA+ have been encoded in the $\lambda\Pi$-calculus modulo rewriting [GB25]. Preliminary results[1] show that the sort-erasure translation of Chapter 10 can be easily applied to translate from the encoding of the typed logic of B to the encoding of the untyped logic of TLA+. As a future work, it would be interesting to investigate whether theory morphisms can also be used to translate from the encoding of the set theory of B to the encoding of the set theory of TLA+.

Reflection Property. Theory morphisms and logical relations capture preservation-style meta-theorems ("if it holds in the source, it is preserved in the target"), that are typically shown by induction on the derivations. Reflection-style meta-theorems ("if it holds in the target, it already held in the source") are generally much harder to prove.

One solution is to establish an inverse morphism from the target theory to the source theory, and use a logical relation to show that the morphism and the inverse morphism are isomorphic. In that case, both the preservation property and the reflection property are captured by the isomorphisms.

Alternatively, the reflection property may hold in some restricted form, even if it is not possible to define an inverse morphism. For example, consider the straightforward morphism from an encoding of sequent calculus without cuts to an encoding of sequent calculus with cuts. The cut-elimination theorem formalized in [Pfe00] corresponds to the reflection theorem, although there is no inverse morphism.

Both solutions only work for certain kinds of translations. Future work would be to provide formalization support for proving reflection-style theorems.

[1] Obtained by Zéphyr Ricci, during an internship that I co-supervised with Catherine Dubois.

Computationally-blocked translation. We have seen in Chapter 10 that rewriting in the target theory can be heavily leveraged to discharge proof obligations to the logical framework, but that rewriting in the source theory creates additional constraints. In some cases, the computations of the source theory cannot be represented computationally in the target theory, hence blocking the translation. We thus have to pre-process the source theory and replace computations with deductions.

Object-level rewrite rules can be replaced by equational axioms. For example, $x + 0 \hookrightarrow x$ can be replaced by a typed constant of type $Prf\ (x + 0 = x)$. The replacement of a large class of object-level rewrite rules by equational axioms was studied in [BDTW24]. In particular, the interplay with dependent types is intricate, and requires the use of transports.

Type-level rewrite rules can be replaced by typed constants. For example, the rule $Prf\ (p \Rightarrow q) \hookrightarrow Prf\ p \to Prf\ q$ can be replaced by the two constants imp_i and imp_e. Although such replacement is in appearance easy, we also have to take care of the interplay with dependent types.

It would be interesting to further study translations from computational encodings to deductive encodings (both for object-level and type-level rewrite rules), and to implement practical tools to do so in Dedukti.

11.4 Long-Term Perspectives

In the following, we underscore long-term perspectives that emerge from this thesis, and we discuss how theorem proving and formal methods in general are impacted by the developments of artificial intelligence.

Concept alignment. When translating a library between proof assistants, we obtain theorems and proofs that are expressed using the translated definitions of the original library. However, the translated definitions generally do not match the definitions of the target proof assistant. Additional work has to be done in order to obtain a library that uses the definitions of the target proof assistant. This is called concept alignments. A classification of different concept alignments has been proposed in [MGK+17]. Heuristics for automatically identifying isomorphic concepts and similar concepts have been developed [GK14, GK19].

Remark that concept alignment concerns both direct translations between proof assistants and translations that use a logical framework as a middleware.

Blanqui [Bla24] developed a tool that automatically translates proofs from HOL Light to Rocq, via Dedukti. He proved, by hand, the equality between the concepts coming from HOL Light and their counterparts in Rocq. It would be interesting to investigate to what extent this task could be automated, for example using the Trocq tool [CCM24, CCM25], based on parametricity.

More generally, theory morphisms and logical relations/parametricity translations could be used to align concepts. Theory morphisms could be applied to translate from one representation of the concept to another, and logical relations/parametricity translations could be applied to show that both representations are equal.

Artificial Intelligence and Formal Proofs. The recent developments of AI, in particular with Large Language Models (LLMs), disrupt how we do mathematics and solve problems [Avi26]. The output of LLMs is not necessarily reliable, and the generated textual proofs require human checking. The use of AI tools to solve mathematical problems thus strengthens the need for formal proofs and proof assistants. Conversely, AI appears promising for enhancing theorem proving tools [YPH+26], at different levels.

First, AI might prove useful for turning pen-and-paper proofs into formal proofs, a task that is often cumbersome for humans. Recently, a proof of the sphere packing problem in dimension 8 has been formalized in Lean [HBL+26] from a pen-and-paper proof. The project initially relied on human intervention, but it was finalized using an autoformalization model.

Second, AI could be useful for guiding humans that use interactive theorem prover, for example by suggesting intermediate lemmas or proof techniques. The process of writing a formal proof would therefore be a back-and-forth process between the user and the AI. Moreover, AI might also be used for automated theorem proving. Among such tools is DeepHOL, an automated neural theorem prover for higher-order logic [BLR+19].

Third, AI performs well for translating between natural languages, suggesting that it may also be helpful for having a better interoperability between proof assistants. For example, LLMs have been used to translate proof terms between Lean and Rocq [SCT25], and to translate Lean and Isabelle proofs into Rocq [VBAL25].

In short, this AI boom opens new perspectives for theorem proving. Regarding the contributions of this thesis, it would be interesting to investigate the potential use of AI. For instance, the parameters of the translation templates could be inferred by an AI instead of being supplied by the user. Conversely, translation templates could be used for aligning concepts of a library translated by an AI. On a broader perspective, AI could offer helpful tools to formalize new results in proof assistants, and could be leveraged to exchange formal proofs between different systems.

Artificial Intelligence and Software Engineering. The recent AI boom not only impacts mathematics and problem solving, but also computer science and programming. LLMs are increasingly used to generate code, but they are opaque tools that are difficult to explain. The generated programs may contain errors, or may not match the specification required by the user. This reinforces the need for software engineering and formal methods. Following the principle of certifiable machine learning, when asking an AI to generate some code, we should also ask for a certificate that the generated code satisfies the specification [BHS25]. This program verification requires the use of formal methods, such as static analysis.

Conversely, AI also seems promising for improving software quality. AI tools not only generate code, but can also be used for code completion, code commenting, bug detection, or bug repair [WGH+23]. For example, the Claude LLM was used to find bugs in Rocq and Lean, including bugs that can be exploited to prove false in Rocq [Tri26]. In cybersecurity, AI can be leveraged to detect vulnerabilities of critical software [WMR26].

In summary, the recent developments of AI caused a paradigm shift for programming, that intensified the need for program verification with proof assistants. At the same time, AI appears promising for improving formal method techniques and tools.

Bibliography

[AABB+26] Yamine Aït-Ameur, Frédéric Blanqui, Jean-Paul Bodeveix, Guillaume Burel, Alessio Coltellacci, David Deharbe, David Delahaye, Gilles Dowek, Catherine Dubois, Ciaran Dunne, Mamoun Filali, Anne Grieu, Olivier Hermant, Dominique Méry, Stephan Merz, and Claude Stolze. Assistants de preuve basés sur la théorie des ensembles interopérables et sûrs – le projet ANR ICSPA. In *Approches Formelles dans l'Assistance au Développement du Logiciel (AFADL 2026)*, 2026.

[ABC+04] Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, and Lor Lorigo. FDL: A Prototype Formal Digital Library, June 2004.

[ABC+16] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti: a Logical Framework based on the $\lambda\Pi$-Calculus Modulo Theory. Manuscript, 2016.

[Abr96] Jean-Raymond Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[ACTZ07] Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. Crafting a Proof Assistant. In Thorsten Altenkirch and Conor McBride, editors, *Types for Proofs and Programs*, pages 18–32, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[Acz01] Peter Aczel. The Russell–Prawitz modality. *Mathematical. Structures in Comp. Sci.*, 11(4):541–554, August 2001.

[Ada06] Robin Adams. Pure type systems with judgemental equality. *Journal of Functional Programming*, 16(2):219–246, 2006.

[AH14] Jeremy Avigad and John Harrison. Formally verified mathematics. *Commun. ACM*, 57(4):66–75, April 2014.

[AHMP92] Arnon Avron, Furio Honsell, Ian A. Mason, and Robert Pollack. Using typed lambda calculus to implement formal systems on a machine. *J. Autom. Reason.*, 9(3):309–354, December 1992.

[AHMP98] Arnon Avron, Furio Honsell, Marino Miculan, and Cristian Paravano. Encoding Modal Logics in Logical Frameworks. *Studia Logica: An International Journal for Symbolic Logic*, 60(1):161–208, 1998.

[And71] Peter B Andrews. Resolution in Type Theory. *The Journal of Symbolic Logic*, 36(3):414–432, September 1971.

[And02] Peter B. Andrews. *Type Theory*, pages 201–256. Springer Netherlands, Dordrecht, 2002.

[Avi06] Jeremy Avigad. A Variant of the Double-Negation Translation. Carnegie Mellon Technical Report CMU-PHIL 179, 2006.

[Avi26] Jeremy Avigad. Mathematicians in the age of AI, 2026.

[Bar69] Jon Barwise. Infinitary Logic and Admissible Sets. *Journal of Symbolic Logic*, 34(2):226–252, 1969.

[Bar74] Michael Barr. Toposes without points. *Journal of Pure and Applied Algebra*, 5(3):265–280, 1974.

[Bar84] Hendrik Pieter Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland Publishing, 1984.

[BC04] Yves Bertot and Pierre Casteran. *Interactive Theorem Proving and Program Development*. SpringerVerlag, 2004.

[BDG+23] Frédéric Blanqui, Gilles Dowek, Émilie Grienenberger, Gabriel Hondet, and François Thiré. A modular construction of type theories. *Logical Methods in Computer Science*, Volume 19, Issue 1, February 2023.

[BDTW24] Valentin Blot, Gilles Dowek, Thomas Traversié, and Théo Winterhalter. From Rewrite Rules to Axioms in the $\lambda\Pi$-Calculus Modulo Theory. In Naoki Kobayashi and James Worrell, editors, *Foundations of Software Science and Computation Structures*, pages 3–23, Cham, 2024. Springer Nature Switzerland.

[BH13] Mélanie Boudard and Olivier Hermant. Polarizing Double-Negation Translations. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 182–197, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BHK07] Paul Brauner, Clement Houtmann, and Claude Kirchner. Principles of Superdeduction. In *LICS 2007 - 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 41–50, Wroclaw, Poland, July 2007. ISSN: 1043-6871.

[BHS25] Clark Barrett, Thomas A. Henzinger, and Sanjit A. Seshia. Certificates in AI: Learn but Verify. *Commun. ACM*, 69(1):66–75, December 2025.

[BJP10] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and dependent types. In *ICFP 2010 - 15th ACM SIGPLAN International Conference on Functional Programming*, page 345–356, Baltimore, USA, 2010. Association for Computing Machinery.

[BJP12] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for free: Parametricity for dependent types. *Journal of Functional Programming*, 22(2):107–152, 2012.

[BKK+20] Michael Butler, Philipp Körner, Sebastian Krings, Thierry Lecomte, Michael Leuschel, Luis-Fernando Mejia, and Laurent Voisin. The First Twenty-Five Years of Industrial Use of the B-Method. In *Formal Methods for Industrial Critical Systems: 25th International Conference, FMICS 2020, Vienna, Austria, September 2–3, 2020, Proceedings*, page 189–209, Berlin, Heidelberg, 2020. Springer-Verlag.

[Bla01] Frédéric Blanqui. Definitions by rewriting in the calculus of constructions. In *Proceedings 16th Annual IEEE Symposium on Logic in Computer Science*, pages 9–18, 2001.

[Bla16] Frédéric Blanqui. Termination of rewrite relations on λ-terms based on Girard's notion of reducibility. *Theoretical Computer Science*, 611:50–86, 2016.

[Bla24] Frédéric Blanqui. Translating HOL-Light proofs to Coq. In *LPAR 2024 - 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pages 1–18, Balaclava, Mauritius, 2024.

[BLR+19] Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An Environment for Machine Learning of Higher Order Logic Theorem Proving. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 454–463. PMLR, 09–15 Jun 2019.

[Bou70] Nicolas Bourbaki. *Théorie des ensembles*. Hermann Paris, 1970.

[BR13] Chad E. Brown and Christine Rizkallah. From Classical Extensional Higher-Order Tableau to Intuitionistic Intentional Natural Deduction. In Jasmin Christian Blanchette and Josef Urban, editors, *PxTP 2013. Third International Workshop on Proof Exchange for Theorem Proving*, volume 14 of *EPiC Series in Computing*, pages 27–42. EasyChair, 2013.

[BR14] Chad E. Brown and Christine Rizkallah. Glivenko and Kuroda for simple type theory. *The Journal of Symbolic Logic*, 79(2):485–495, 2014.

[CAB+86] Robert Constable, Stuart Allen, Hank Bromley, Walter Rance Cleaveland, J.F. Cremer, Robert Harper, Douglas Howe, Todd Knoblock, Nax Mendler, Prakash Panangaden, James Sasaki, and Scott Smith. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, 1986.

[Cau16] Raphaël Cauderlier. A Rewrite System for Proof Constructivization. In *LFMTP 2016 - International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, pages 1 – 7, Porto, Portugal, June 2016.

[CC96] Pierre-Louis Curien and Roberto Di Cosmo. A confluent reduction for the λ-calculus with surjective pairing and terminal object. *Journal of Functional Programming*, 6(2):299–327, 1996.

[CCM24] Cyril Cohen, Enzo Crance, and Assia Mahboubi. Trocq: Proof Transfer for Free, With or Without Univalence. In *ESOP 2024 - 33rd European Symposium*

on Programming, pages 239–268, Luxembourg, Luxembourg, 2024. Springer Nature Switzerland.

[CCM25] Cyril Cohen, Enzo Crance, and Assia Mahboubi. Trocq: Proof Transfer for Free, Beyond Equivalence and Univalence. *ACM Trans. Program. Lang. Syst.*, 47(3), September 2025.

[CD07] Denis Cousineau and Gilles Dowek. Embedding Pure Type Systems in the Lambda-Pi-Calculus Modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications*, pages 102–117, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CdJV16] Almudena Colacito, Dick de Jongh, and Ana Lucia Vargas. Subminimal negation. *Soft Computing*, 21:165–174, 2016.

[CELM96] Manuel Clavel, Steven Eker, Patrick D. Lincoln, and José Meseguer. Principles of Maude. volume 4, pages 65–89, 1996. RWLW96, First International Workshop on Rewriting Logic and its Applications.

[CH88] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2):95–120, 1988.

[CHK+11] Mihai Codescu, Fulya Horozal, Michael Kohlhase, Till Mossakowski, and Florian Rabe. Project Abstract: Logic Atlas and Integrator (LATIN). In James H. Davenport, William M. Farmer, Josef Urban, and Florian Rabe, editors, *Intelligent Computer Mathematics*, pages 289–291, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Chu40] Alonzo Church. A Formulation of the Simple Theory of Types. *The Journal of Symbolic Logic*, 5(2):56–68, 1940.

[dB80] Nicolaas G. de Bruijn. *A survey of the project Automath*, pages 579–606. Academic Press Inc., United States, 1980.

[DCD19] Alejandro Díaz-Caro and Gilles Dowek. Proof Normalisation in a Logic Identifying Isomorphic Propositions. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[DCK93] Roberto Di Cosmo and Delia Kesner. A confluent reduction for the extensional typed λ-calculus with pairs, sums, recursion and terminal object. In Andrzej Lingas, Rolf Karlsson, and Svante Carlsson, editors, *Automata, Languages and Programming*, pages 645–656, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[DCK94] Roberto Di Cosmo and Delia Kesner. Combining first order algebraic rewriting systems, recursion and extensional lambda calculi. In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming*, pages 462–472, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[DCK96] Roberto Di Cosmo and Delia Kesner. Rewriting with extensional polymorphic λ-calculus. In Hans Kleine Büning, editor, *Computer Science Logic*, pages 215–232, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[DH07] Gilles Dowek and Olivier Hermant. A simple proof that super-consistency implies cut elimination. In Franz Baader, editor, *Proceedings of the 18th International Conference on Term Rewriting and Applications (RTA)*, volume 4533 of *Lecture Notes in Computer Science*, pages 93–106, Berlin, Heidelberg, 2007. Springer.

[DH12] Gilles Dowek and Olivier Hermant. A Simple Proof that Super-Consistency Implies Cut Elimination. *Notre Dame Journal of Formal Logic*, 53(4):439 – 456, 2012.

[DHK03] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning*, 31:33–72, 2003.

[Dia75] Radu Diaconescu. Axiom of Choice and Complementation. *Proceedings of the American Mathematical Society*, 51(1):176–178, 1975.

[DJ91] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite Systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, 1991.

[DL05] Mary DeMarco and James Lipton. Completeness and Cut-elimination in the Intuitionistic Theory of Types. *Journal of Logic and Computation*, 15(6):821–854, December 2005.

[dMKA+15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean Theorem Prover (System Description). In Amy Felty and Aart Middeldorp, editors, *Automated Deduction*, pages 378–388. Springer, 2015.

[DNR04] René David, Karim Nour, and Christophe Raffalli. *Introduction à la Logique, Théorie de la démonstration (2nd édition)*. Sciences Sup. Dunod, 2004.

[Dou93] Daniel J. Dougherty. Some lambda calculi with categorical sums and products. In Claude Kirchner, editor, *Rewriting Techniques and Applications*, pages 137–151, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[Dow15] Gilles Dowek. On the definition of the classical connectives and quantifiers. In Edward Hermann Haeusler, Wagner de Campos Sanz, and Bruno Lopes, editors, *Why is this a Proof?, Festschrift for Luiz Carlos Pereira*. College Publications, 2015.

[DT20] Gilles Dowek and François Thiré. Logipedia: a multi-system encyclopedia of formal proofs, 2020.

[DW03] Gilles Dowek and Benjamin Werner. Proof Normalization Modulo. *The Journal of Symbolic Logic*, 68(4):1289–1316, 2003.

[EO12] Martín Escardó and Paulo Oliva. The Peirce translation. *Annals of Pure and Applied Logic*, 163(6):681–692, 2012. Computability in Europe 2010.

[ER08] Eric Eide and John Regehr. Volatiles are miscompiled, and what to do about it. In *Proceedings of the 8th ACM International Conference on Embedded Software*, EMSOFT '08, page 255–264, New York, NY, USA, 2008. Association for Computing Machinery.

[Far08] William M. Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6(3):267–286, 2008.

[FB24] Thiago Felicissimo and Frédéric Blanqui. Sharing proofs with predicative theories through universe-polymorphic elaboration. *Logical Methods in Computer Science*, Volume 20, Issue 3, Sep 2024.

[Fef68] Solomon Feferman. Lectures on proof theory. In M. H. Löb, editor, *Proceedings of the Summer School in Logic Leeds, 1967*, pages 1–107, Berlin, Heidelberg, 1968. Springer Berlin Heidelberg.

[Fel22] Thiago Felicissimo. Adequate and Computational Encodings in the Logical Framework Dedukti. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[FM97] Matt Fairtlough and Michael Mendler. Propositional Lax Logic. *Information and Computation*, 137:1–33, 1997.

[FO12a] Hadi Farahani and Hiroakira Ono. Glivenko Theorems and Negative Translations in Substructural Predicate Logics. *Archive for Mathematical Logic*, 51(7-8):695–707, 2012.

[FO12b] Gilda Ferreira and Paulo Oliva. On the Relation Between Various Negative Translations. In Ulrich Berger, Hannes Diener, Peter Schuster, and Monika Seisenberger, editors, *Logic, Construction, Computation*, pages 227–258. De Gruyter, Berlin, Boston, 2012.

[Fri78] Harvey Friedman. Classically and intuitionistically provably recursive functions. In Gert H. Müller and Dana S. Scott, editors, *Higher Set Theory*, pages 21–27, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[Gab72] Dov M. Gabbay. Applications of Trees to Intermediate Logics. *The Journal of Symbolic Logic*, 37(1):135–138, 1972.

[Gan56] R. O. Gandy. On the Axiom of Extensionality–Part I. *The Journal of Symbolic Logic*, 21(1):36–48, 1956.

[Gas13] Jaime Gaspar. Negative Translations Not Intuitionistically Equivalent to the Usual Ones. *Studia Logica: An International Journal for Symbolic Logic*, 101(1):45–63, 2013.

[GB25] Anne Grieu and Jean-Paul Bodeveix. Encodage du langage mathématique d'Event-B dans Lambdapi. In *36es Journées Francophones des Langages Applicatifs (JFLA 2025)*, Roiffé, France, January 2025.

[Gen36] Gerhard Gentzen. Die Widerspruchsfreiheit der Reinen Zahlentheorie. *Mathematische Annalen*, 112:493–565, 1936.

[Geu09] Herman Geuvers. Proof assistants: History, ideas and future. *Sadhana*, 34(1):3–25, 2009.

[GH15] Gaëtan Gilbert and Olivier Hermant. Normalisation by Completeness with Heyting Algebras. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 469–482. Springer, 2015.

[Gil15] Frederic Gilbert. A Lightweight Double-negation Translation. In Ansgar Fehnker, Annabelle McIver, Geoff Sutcliffe, and Andrei Voronkov, editors, *LPAR-20. 20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations*, volume 35 of *EPiC Series in Computing*, pages 81–93. EasyChair, 2015.

[Gil17] Frédéric Gilbert. Automated Constructivization of Proofs. In *FOSSACS 2017 - 20th International Conference on Foundations of Software Science and Computation Structures*, pages 480–495, Uppsala, Sweden, 2017.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris 7, jun 1972.

[GK14] Thibault Gauthier and Cezary Kaliszyk. Matching Concepts across HOL Libraries. In Stephen M. Watt, James H. Davenport, Alan P. Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics*, pages 267–281, Cham, 2014. Springer International Publishing.

[GK19] Thibault Gauthier and Cezary Kaliszyk. Aligning concepts across proof assistant libraries. *Journal of Symbolic Computation*, 90:89–123, 2019. Symbolic Computation in Software Science.

[Gli28] Valery Glivenko. Sur quelques points de la logique de M. Brouwer. *Bulletins de la classe des sciences*, 15:183–188, 1928.

[GM78] Nicholas D. Goodman and John Myhill. Choice Implies Excluded Middle. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 24(25-30):461–461, 1978.

[Gon08] Georges Gonthier. Formal Proof – The Four-Color Theorem. *Notices of the American Mathematical Society*, 55(11):1382–1393, 2008.

[Gor88] Michael Gordon. HOL: A Proof Generating System for Higher-Order Logic. In Graham Birtwistle and P.A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128. Kluwer-Academic Publishers, 1988.

[Gri25] Emilie Grienenberger. *Combining computational theories*. Phd thesis, Université Paris-Saclay, February 2025.

[GW94] Herman Geuvers and Benjamin Werner. On the Church-Rosser property for expressive type systems and its consequences for their metatheoretic study. In *Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 320–329, 1994.

[Gö33] Kurt Gödel. Zur intuitionistischen Arithmetik und Zahlentheorie. *Ergebnisse eines Mathematischen Kolloquiums*, 4:34–38, 1933.

[HAB+17] Thomas C. Hales, Mark Adams, Gertrud Josefine Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, C. Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason M. Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A Formal Proof of the Kepler Conjecture. *Forum of Mathematics, Pi*, 5, 2017.

[Har96] John Harrison. HOL Light: A Tutorial Introduction. In *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pages 265–269. Springer, 1996.

[HB20] Gabriel Hondet and Frédéric Blanqui. The New Rewriting Engine of Dedukti. In *FSCD 2020 - 5th International Conference on Formal Structures for Computation and Deduction*, number 167, page 16, Paris, France, June 2020.

[HBL+26] Sidharth Hariharan, Christopher Birkbeck, Seewoo Lee, Ho Kiu Gareth Ma, Bhavik Mehta, Auguste Poiroux, and Maryna Viazovska. Progress in Formalizing Sphere Packing in Dimension 8, 2026.

[Her30] Jacques Herbrand. *Recherches sur la théorie de la démonstration*. PhD thesis, Université de Paris, 1930.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A Framework for Defining Logics. *Journal of the ACM*, 40(1):143–184, January 1993.

[Hil27] David Hilbert. The Foundations of Mathematics. 1927. Reprinted in [vH67].

[HKKN13] Florian Haftmann, Alexander Krauss, Ondřej Kunčar, and Tobias Nipkow. Data refinement in Isabelle/HOL. In *Proceedings of the 4th International Conference on Interactive Theorem Proving*, ITP'13, page 100–115, Berlin, Heidelberg, 2013. Springer-Verlag.

[HL08a] Olivier Hermant and James Lipton. A Constructive Semantic Approach to Cut Elimination in Type Theories with Axioms. In Michael Kaminski and Simone Martini, editors, *CSL*, volume 5213 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2008.

[HL08b] Olivier Hermant and James Lipton. Cut Elimination in the Intuitionistic Theory of Types with Axioms and Rewriting Cuts, Constructively. In Christoph Benzmüller, Chad E. Brown, Jörg Siekmann, and Richard Statman, editors,

Reasoning in Simple Type Theory – Festschrift in Honor of Peter B. Andrews on His 70th Birthday, volume 17 of *Studies in Logic and the Foundations of Mathematics*, pages 101–134. College Publications London, 2008.

[HLMS17] Furio Honsell, Luigi Liquori, Petar Maksimovic, and Ivan Scagnetto. LLFP: a logical framework for modeling external evidence, side conditions, and proof irrelevance using monads. *Logical Methods in Computer Science*, 2017.

[Hof97] Martin Hofmann. *Syntax and Semantics of Dependent Types*, page 79–130. Publications of the Newton Institute. Cambridge University Press, 1997.

[Hou10] Clément Houtmann. *Représentation et interaction des preuves en superdéduction modulo*. PhD thesis, Université Henri Poincaré - Nancy I, March 2010.

[HST94] Robert Harper, Donald Sannella, and Andrzej Tarlecki. Structured theory presentations and logic representations. *Annals of Pure and Applied Logic*, 67(1):113–160, 1994.

[IR11] Mihnea Iancu and Florian Rabe. Formalising foundations of mathematics. *Mathematical Structures in Computer Science*, 21(4):883–911, 2011.

[JG95] C. Barry Jay and Neil Ghani. The virtues of eta-expansion. *Journal of Functional Programming*, 5(2):135–154, 1995.

[Joh02] Peter Johnstone. *Sketches of an Elephant: A Topos Theory Compendium. Volume 1 and Volume 2*. Oxford University Press, 2002.

[Kar64] Carol R. Karp. *Languages with Expressions of Infinite Length*. Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1964.

[KB70] Donald E. Knuth and Peter B. Bendix. Simple Word Problems in Universal Algebras. In John Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon, 1970.

[Kei71] H. Jerome Keisler. *Model Theory for Infinitary Logic*. North-Holland, Amsterdam, 1971.

[Kie71] B. Melvin Kiernan. The Development of Galois Theory from Lagrange to Artin. *Archive for History of Exact Sciences*, 8(1/2):40–154, 1971.

[KL12] Chantal Keller and Marc Lasson. Parametricity in an Impredicative Sort. In *CSL 2012 - 26th EACSL Annual Conference on Computer Science Logic*, volume 16, pages 381–395, Fontainebleau, France, 2012. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Klo80] Jan Willem Klop. *Combinatory Reduction Systems*. Mathematisch centrum, Amsterdam, 1980.

[Kol25] Andrey Nikolaevich Kolmogorov. O principe tertium non datur. *Matematicheskiĭ Sbornik*, 32:646–667, 1925.

[Kri90] Jean-Louis Krivine. Opérateurs de mise en mémoire et traduction de gödel. *Arch. Math. Log.*, 30(4):241–267, 1990.

[Kur51] Sigekatu Kuroda. Intuitionistische Untersuchungen der formalistischen Logik. *Nagoya Mathematical Journal*, 2:35–47, 1951.

[KW10] Chantal Keller and Benjamin Werner. Importing HOL Light into Coq. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving*, pages 307–322, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[KWP99] Florian Kammüller, Markus Wenzel, and Lawrence C. Paulson. Locales - A Sectioning Concept for Isabelle. In Yves Bertot, Gilles Dowek, Laurent Théry, André Hirschowitz, and Christine Paulin, editors, *Theorem Proving in Higher Order Logics*, pages 149–165, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Lam02] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[Lam13] Peter Lammich. Automatic Data Refinement. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 84–99, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[LBK+16] Xavier Leroy, Sandrine Blazy, Daniel Kästner, Bernhard Schommer, Markus Pister, and Christian Ferdinand. CompCert - A Formally Verified Optimizing Compiler. In *ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress*, Toulouse, France, January 2016. SEE.

[LE65] E. G. K. López-Escobar. An Interpolation Theorem for Denumerably Long Formulas. *Fundamenta Mathematicae*, 57(3):253–272, 1965.

[Lec24] Thierry Lecomte. *Proving B with Atelier B*, pages 329–345. Springer Nature Switzerland, Cham, 2024.

[Ler09] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, July 2009.

[LM24] Henri Lombardi and Assia Mahboubi. Théories géométriques pour l'algèbre des nombres réels sans test de signe ni axiome de choix dépendant, 2024.

[LPR17] Tadeusz Litak, Miriam Polzer, and Ulrich Rabenstein. Negative Translations and Normal Modality. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:18, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Mac71] Saunders Maclane. *Categories for the Working Mathematician*. Springer, 1971.

[Mae91] Shôji Maehara. Lattice-valued Representation of the Cut-elimination Theorem. *Tsukuba Journal of Mathematics*, 15(2):509–521, 1991.

[Mag03] Nicolas Magaud. Changing Data Representation within the Coq System. In David Basin and Burkhart Wolff, editors, *Theorem Proving in Higher Order Logics*, pages 87–102, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[MB02] Nicolas Magaud and Yves Bertot. Changing Data Structures in Type Theory: A Study of Natural Numbers. In Paul Callaghan, Zhaohui Luo, James McKinna, Robert Pollack, and Robert Pollack, editors, *Types for Proofs and Programs*, pages 181–196, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[McL06] Sean McLaughlin. An Interpretation of Isabelle/HOL in HOL Light. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 192–204, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[MGK+17] Dennis Müller, Thibault Gauthier, Cezary Kaliszyk, Michael Kohlhase, and Florian Rabe. Classification of Alignments between Concepts of Formal Mathematical Systems. In Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, editors, *Intelligent Computer Mathematics*, pages 83–98. Springer, 2017.

[ML84] Per Martin-Löf. Intuitionistic Type Theory. Notes by Giovanni Sambin. *Studies in Proof Theory*, 1984.

[MM92] Saunders McLane and Ieke Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer Verlag, 1992.

[MN12] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, 2012.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991. Selections from 1989 IEEE Symposium on Logic in Computer Science.

[Neg03] Sara Negri. Contraction-free sequent calculi for geometric theories with an application to Barr's theorem. *Archive for Mathematical Logic*, 42:389–401, 05 2003.

[Neg21] Sara Negri. Geometric Rules in Infinitary Logic. In Ofer Arieli and Anna Zamansky, editors, *Arnon Avron on Semantics and Proof Theory of Non-Classical Logics*, pages 265–293. Springer International Publishing, Cham, 2021.

[Nip02] Tobias Nipkow. *Hoare Logics in Isabelle/HOL*, pages 341–367. Springer Netherlands, Dordrecht, 2002.

[Nor05] Ulf Norell. The Agda WiKi, 2005. http://wiki.portal.chalmers.se/agda.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[NRZ+15] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon web services uses formal methods. *Commun. ACM*, 58(4):66–73, March 2015.

[Oka02] Mitsuhiro Okada. A uniform semantic proof for cut-elimination and completeness of various first and higher order logics. *Theoretical Computer Science*, 281:471–498, 2002.

[ORS92] Sam Owre, John M. Rushby, and Natarajan Shankar. PVS: A Prototype Verification System. In *Proceedings of the 11th International Conference on Automated Deduction: Automated Deduction*, CADE-11, page 748–752, Berlin, Heidelberg, 1992. Springer-Verlag.

[OS06] Steven Obua and Sebastian Skalberg. Importing HOL into Isabelle/HOL. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 298–302, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Pal02] Erik Palmgren. An Intuitionistic Axiomatisation of Real Closed Fields. *Mathematical Logic Quarterly*, 48(2):297–299, 2002.

[Pau93] Lawrence C. Paulson. Set theory for verification. I: From foundations to functions. *J. Autom. Reason.*, 11(3):353–389, December 1993.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 1994.

[PBNP26] Luiz Carlos Pereira, Victor Barroso-Nascimento, and Elaine Pimentel. Glivenko's theorems from an ecumenical perspective, 2026.

[PD10] Brigitte Pientka and Jana Dunfield. Beluga: a framework for programming and reasoning with deductive systems (system description). In *Proceedings of the 5th International Conference on Automated Reasoning*, IJCAR'10, page 15–21, Berlin, Heidelberg, 2010. Springer-Verlag.

[Pes04] Peter Pesic. *Abel's proof. An essay on the sources and meaning of mathematical unsolvability*. Cambridge, MA: MIT Press, 2004.

[Pfe00] Frank Pfenning. Structural Cut Elimination: I. Intuitionistic and Classical Logic. *Information and Computation*, 157(1):84–141, 2000.

[Pfe01] Frank Pfenning. Chapter 17 - Logical Frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, Handbook of Automated Reasoning, pages 1063–1147. North-Holland, Amsterdam, 2001.

[PH15] Luiz Carlos Pereira and Edward Hermann Haeusler. On Constructive Fragments of Classical Logic. In Heinrich Wansing, editor, *Dag Prawitz on Proofs and Meaning*, pages 281–292. Springer International Publishing, Cham, 2015.

[PM68] Dag Prawitz and Per-Erik Malmnäs. A Survey of Some Connections Between Classical, Intuitionistic and Minimal Logic. In Hermann Arnold Schmidt, Kurt Schütte, and H.-J. Thiele, editors, *Contributions to Mathematical Logic*, volume 50 of *Studies in Logic and the Foundations of Mathematics*, pages 215–229. Elsevier, 1968.

[PM93] Christine Paulin-Mohring. Inductive definitions in the system Coq rules and properties. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, pages 328–345, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[Pot81] Garrel Pottinger. The Church-Rosser theorem for the typed λ-calculus with surjective pairing. *Notre Dame Journal of Formal Logic*, 22(3):264 – 268, 1981.

[PP23] Elaine Pimentel and Luiz Carlos Pereira. A Tour on Ecumenical Systems. In Paolo Baldan and Valeria de Paiva, editors, *10th Conference on Algebra and Coalgebra in Computer Science (CALCO 2023)*, volume 270 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[PPdP18] Elaine Pimentel, Luiz Carlos Pereira, and Valeria de Paiva. A proof theoretical view of ecumenical systems. In *13th Workshop on Logical and Semantic Frameworks with Applications*, pages 109–114, 2018.

[PPdP19] Elaine Pimentel, Luiz Carlos Pereira, and Valeria de Paiva. An Ecumenical Notion of Entailment. *Synthese*, 198(S22):5391–5413, 2019.

[PPdP25] Luiz Carlos Pereira, Elaine Pimentel, and Valeria de Paiva. Translations and Prawitz's Ecumenical System. *Studia Logica*, 113(2):523–538, 2025.

[PR17] Luiz Carlos Pereira and Ricardo Osar Rodriguez. Normalization, Soundness and Completeness for the Propositional Fragment of Prawitz' Ecumenical System. *Revista Portuguesa de Filosofia*, 73(3/4):1153–1168, 2017.

[Pra68] Dag Prawitz. Hauptsatz for Higher Order Logic. *The Journal of Symbolic Logic*, 33(3):452–457, September 1968.

[Pra15] Dag Prawitz. Classical versus intuitionistic logic. In Edward Hermann Haeusler, Wagner de Campos Sanz, and Bruno Lopes, editors, *Why is this a Proof?, Festschrift for Luiz Carlos Pereira*. College Publications, 2015.

[PS99] Frank Pfenning and Carsten Schürmann. System Description: Twelf — A Meta-Logical Framework for Deductive Systems. In *Automated Deduction — CADE-16*, pages 202–206, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[PS07] Lawrence C. Paulson and Kong Woei Susanto. Source-Level Proof Reconstruction for Interactive Theorem Proving. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics*, pages 232–245, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[PS09] Adam Poswolsky and Carsten Schürmann. System Description: Delphin – A Functional Programming Language for Deductive Systems. *Electronic Notes in Theoretical Computer Science*, 228:113–120, 2009. Proceedings of the International Workshop on Logical Frameworks and Metalanguages: Theory and Practice (LFMTP 2008).

[Rab14] Florian Rabe. How to identify, translate and combine logics? *Journal of Logic and Computation*, 27(6):1753–1798, 12 2014.

[Rab18] Florian Rabe. A Modular Type Reconstruction Algorithm. *ACM Trans. Comput. Logic*, 19(4), December 2018.

[Rat16] Michael Rathjen. Remarks on Barr's Theorem: Proofs in Geometric Theories. In Dieter Probst and Peter Schuster, editors, *Concepts of Proof in Mathematics, Philosophy, and Computer Science*, pages 347–374. De Gruyter, Berlin, Boston, 2016.

[RK13] Florian Rabe and Michael Kohlhase. A scalable module system. *Information and Computation*, 230:1–54, 2013.

[RR25] Rubén Rubio and Adrián Riesco. Maude2Lean: Theorem proving for Maude specifications using Lean. *Journal of Logical and Algebraic Methods in Programming*, 142:101005, 2025.

[RS09] Florian Rabe and Carsten Schürmann. A practical module system for LF. In *Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, LFMTP '09, page 40–48, New York, NY, USA, 2009. Association for Computing Machinery.

[RS13] Florian Rabe and Kristina Sojakova. Logical relations for a logical framework. *ACM Transactions on Computational Logic*, 14(4), November 2013.

[SAB+20] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq Project. *J. Autom. Reason.*, 64(5):947–999, June 2020.

[Sai15] Ronan Saillard. *Typechecking in the lambda-Pi-Calculus Modulo : Theory and Practice*. PhD thesis, Ecole Nationale Supérieure des Mines de Paris, September 2015.

[Sch60] Kurt Schütte. Syntactical and Semantical Properties of Simple Type Theory. *The Journal of Symbolic Logic*, 25(4):305–326, December 1960.

[SCT25] Théo Stoskopf, Cyril Cohen, and Nicolas Tabareau. Babel-formal: Translation of Proofs between Lean and Rocq. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025.

[SH12] Vincent Siles and Hugo Herbelin. Pure Type System conversion is always typable. *Journal of Functional Programming*, 22(2):153 – 180, May 2012.

[Sil10] Vincent Siles. *Investigation on the typing of equality in type systems*. PhD thesis, Ecole Polytechnique, November 2010.

[Smu68] Raymond M. Smullyan. *First-Order Logic*. Springer-Verlag, Berlin Heidelberg New York, 1968.

[SOR+13] Aaron Stump, Duckki Oe, Andrew Reynolds, Liana Hadarean, and Cesare Tinelli. SMT proof checking using a logical framework. *Formal Methods System Design*, 42(1):91–118, February 2013.

[SPK+] Carsten Schurmann, Frank Pfenning, Michael Kohlhase, Natarajan Shankar, and Sam Owre. Logosphere. A Formal Digital Library.

[SS06] Carsten Schürmann and Mark-Oliver Stehr. An Executable Formalization of the HOL/Nuprl Connection in the Metalogical Framework Twelf. In Miki Hermann and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 150–166, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Tai67] William W. Tait. Intensional Interpretations of Functionals of Finite Type I. *The Journal of Symbolic Logic*, 32(2):198–212, 1967.

[Tai68] William W. Tait. Normal derivability in classical logic. In Jon Barwise, editor, *The Syntax and Semantics of Infinitary Languages*, pages 204–236, Berlin, Heidelberg, 1968. Springer Berlin Heidelberg.

[Tak53] Gaisi Takeuti. On a generalized logic calculus. *Japanese journal of mathematics: transactions and abstracts*, 23:39–96, 1953.

[Tak67] Moto-o Takahashi. A proof of cut-elimination theorem in simple type-theory. *Journal of the Mathematical Society of Japan*, 19(4):399–410, 1967.

[Tak75] Gaisi Takeuti. Chapter 4 Infinitary Logic. In *Proof Theory*, volume 81 of *Studies in Logic and the Foundations of Mathematics*, pages 188–284. Elsevier, 1975.

[Tay10] Ralph Gregory Taylor. Zermelo 1932a. In Heinz-Dieter Ebbinghaus, Craig G. Fraser, and Akihiro Kanamori, editors, *Ernst Zermelo - Collected Works/Gesammelte Werke: Volume I - Set Theory, Miscellanea / Band I - Mengenlehre, Varia*, pages 532–549. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[TB85] Andrzej Trybulec and Howard Blair. Computer Assisted Reasoning with MIZAR. In Aravind Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 26–28. Morgan Kaufmann, 1985.

[THA26] Thomas Traversié, Olivier Hermant, and Marc Aiguier. Investigations on Higher-Order Infinitary Logic. Accepted for publication at the 11th International Conference on Formal Structures for Computation and Deduction , 2026.

[Thi20] François Thiré. *Interoperability between proof systems using the logical framework Dedukti*. PhD thesis, Université Paris-Saclay, 2020.

[Tho98] Robin Thomas. An Update on the Four-Color Theorem. *Notices of the American Mathematical Society*, 45(7):848–859, 1998.

[TR25] Thomas Traversié and Florian Rabe. Formalizing Representation Theorems for a Logical Framework with Rewriting, 2025.

[Tra24a] Thomas Traversié. Kuroda's Translation for the $\lambda\Pi$-Calculus Modulo Theory and Dedukti. In Florian Rabe and Claudio Sacerdoti Coen, editors, Proceedings Workshop on *Logical Frameworks and Meta-Languages: Theory and Practice,* Tallinn, Estonia, 8th July 2024, volume 404 of *Electronic Proceedings in Theoretical Computer Science*, pages 35–48. Open Publishing Association, 2024.

[Tra24b] Thomas Traversié. Proofs for Free in the $\lambda\Pi$-Calculus Modulo Theory. In Florian Rabe and Claudio Sacerdoti Coen, editors, Proceedings Workshop on *Logical Frameworks and Meta-Languages: Theory and Practice,* Tallinn, Estonia, 8th July 2024, volume 404 of *Electronic Proceedings in Theoretical Computer Science*, pages 49–63. Open Publishing Association, 2024.

[Tra25] Thomas Traversié. Monad Translations for Higher-Order Logic. In Maribel Fernández, editor, *10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025)*, volume 337 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:14, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Tra26] Thomas Traversié. Kuroda's translation for higher-order logic. *The Journal of Symbolic Logic*, page 1–12, 2026. Published online.

[Tri26] Tristan Stérin. In search of falsehood (blog post), 2026.

[TTS21] Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. The Marriage of Univalence and Parametricity. *J. ACM*, 68(1), January 2021.

[TvD88] Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in Mathematics: An Introduction, Volume 1*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing, Amsterdam, 1988.

[Ume59] Toshio Umezawa. On Logics Intermediate between Intuitionistic and Classical Predicate Logic. *The Journal of Symbolic Logic*, 24(2):141–153, 1959.

[VBAL25] Jules Viennot, Guillaume Baudart, Emilio Jesús Gallego Arias, and Marc Lelarge. Minif2f in Rocq: Automatic Translation Between Proof Assistants — A Case Study. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025.

[vdB19] Benno van den Berg. A Kuroda-style j-translation. *Archive for Mathematical Logic*, 58(5–6):627–634, August 2019.

[vH67] Jean van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Harvard University Press, Cambridge, MA, USA, 1967.

[Wan52] Hao Wang. Logic of Many-Sorted Theories. *The Journal of Symbolic Logic*, 17(2):105–116, 1952.

[Wen99] Markus Wenzel. Isar — A Generic Interpretative Approach to Readable Formal Proof Documents. In Yves Bertot, Gilles Dowek, Laurent Théry, André Hirschowitz, and Christine Paulin, editors, *Theorem Proving in Higher Order Logics*, pages 167–183, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[WGH+23] Man-Fai Wong, Shangxin Guo, Ching-Nam Hang, Siu-Wai Ho, and Chee-Wei Tan. Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review. *Entropy*, 25(6), 2023.

[WMR26] Dylan Wolff, Martin Mirchev, and Abhik Roychoudhury. Large Language Models in Software Security Analysis. *Commun. ACM*, 69(6):60–67, May 2026.

[YCER11] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in C compilers. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, page 283–294, New York, NY, USA, 2011. Association for Computing Machinery.

[YPH+26] Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal Reasoning Meets LLMs: Toward AI for Mathematics and Verification. *Commun. ACM*, 69(3):66–73, February 2026.