Integrating Automated Theorem Provers in Proof Assistants

Yacine El Haddad

université
PARIS-SACLAY

Integrating Automated Theorem Provers in Proof Assistants

Utiliser des démonstrateurs automatiques dans un assistant à la preuve

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580,Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique
Unité de recherche: Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France.
Référent: : École Normale Supérieure Paris-Saclay

Thèse présentée et soutenue à École Normale Supérieure Paris-Saclay, le 9 septembre 2021, par

Mohamed Yacine EL HADDAD

Composition du jury:

David DELAHAYE Professeur, Université de Montpellier	Président de jury
Pascal FONTAINE Professeur, Université de Liège	Rapporteur & Examinateur
Stéphane GRAHAM-LENGRAND Professeur, SRI International	Rapporteur & Examinateur
Véronique BENZAKEN Professeur, Université Paris-Saclay	Examinatrice
Cezary KALISZYK Associate Professor, University of Innsbruck	Examinateur
Sophie TOURRET Chargée de recherche, INRIA Nancy & LORIA	Examinatrice
Frédéric BLANQUI Chargé de recherche, Inria Saclay	Directeur de thèse
Guillaume BUREL Maître de conférence, ENSIIE	Coencadrant

Thèse de doctorat

NNT: 2021UPASG052

PhD thesis: Integrating Automated Provers in Proof Assistants

Mohamed Yacine EL HADDAD

September 9, 2021

عَلَّمَ الْإِنسَانَ مَا لَمْ يَعْلَمْ

Taught man what he never knew

A enseigné à l'homme ce qu'il ne savait pas

In the memory of my dear grandfather "*Mohamed Haddouche"*

Remerciements

Je tiens ici, et en premier lieu, à exprimer mes plus profonds sentiments de gratitude à Frédéric Blanqui et Guillaume Burel qui ont dirigé mes travaux. Leurs patiences et leurs sens de la pédagogie m'ont marqué pendant cette aventure scientifique. Merci d'avoir su comment m'encourager dans les moments les plus difficiles, que ce soit d'ordre professionnel ou personnel.

Merci aux membres du jury de soutenance de cette thèse qui m'ont fait l'honneur d'accepter d'en faire part : Stéphane Graham-Lengrand et Pascal Fontaine pour leurs rapports et la pertinence de leurs retours, et David Delahaye, Véronique Benzaken, Cezary Kaliszyk et Sophie Tourret pour avoir bien voulu évaluer ce travail.

Je remercie tout aussi chaleureusement Gilles Dowek qui a mis à ma disposition, durant ma thèse, toutes les ressources nécessaires à son bon déroulement. Pour moi, Gilles faisait partie de mes encadrants. Je n'oublierai jamais mes discussions avec lui sur divers sujets : le métier de chercheur, politique, religion, anecdotes.

Un très grand merci à Guillaume Genestier, Gaspard Férey et François Thiré pour l'accueil qu'ils m'ont réservé à mon arrivée au sein de l'équipe. Ils étaient toujours disponibles à chaque fois que j'avais besoin d'une aide que soit d'ordre personnel ou professionnel.

Mes sentiments d'amitié et de gratitude vont aussi à l'ensemble des membres de l'équipe Deducteam, Jean-Pierre Jouannaud, Valentin Blot, Bruno Barras, Rodolphe Lepigre, Franck Slama, Michael Färber, Rehan Malak, Pierre Vial, Guillaume Bury, Gabriel Hondet, Emilie Grienenberger, Amélie Ledein, Louise Dubois de Prisque et Farzad Jafarrahmani.

A M. Mezghiche Mohamed, je ne vous remercierai jamais assez et vous dédie ce travail en témoignage de mon grand respect et mon estime envers vous. Sachez que c'est grâce à vous que j'ai pu accéder à la recherche depuis mes premiers jours à l'université.

Je profite de cette occasion pour remercier ma famille qui n'a jamais cessé de croire en moi et m'a soutenue durant ces dernières années malgré la distance qui nous séparent. Un grand merci à mon très cher père Kamal pour son amour, sa générosité et son soutien. Je remercie aussi ma très chère mère Safia. Quoique je puisse dire et écrire, je ne pourrais exprimer ma grande affection et ma profonde reconnaissance. Je tiens aussi à remercier mon frère Adel et son épouse Sofia et leur fils Riyane pour leur présence et encouragements durant mes deux dernières années de thèse. Mon frère Aymen et ma soeur Ikram, aucune dédicace ne peut exprimer la profondeur des sentiments fraternels et d'amour, d'attachement que j'éprouve à votre égard.

Merci à ma grand-mère Yamina, qui m'a élevée comme son fils, c'est la personne la plus sympa et la plus gentille que je connais. Je dédie ce travail à mon arrière-grand-mère Fatima que dieu lui procure la bonne santé et la longue vie. Mes remerciements vont également à mes oncles Rabie, Sofiane, Tchico et Abdelghani que je considère à la fois des frères et des amis, sans

vous je ne serai jamais ce que je suis. Mes remerciements vont aussi à mes tantes Rafika, Karima et Djahida qui ont toujours cru en moi et n'ont jamais cesser de m'aimer comme leurs fils. Je remercie cette famille au sein de laquelle je me suis toujours senti chez moi et qui m'ont toujours considéré comme un des leurs. Je tiens aussi à remercier les épouses de mes oncles : Nawel, Mimi, Selma et Amina qui font partie de ma vie depuis des années ainsi que les époux de mes tantes : Ramadan et Hamza que j'admire beaucoup. Je clôture cette liste par mes cousins et cousines qui me considèrent comme leur grand frère étant donné que je suis l'aîné de cette génération : Amine, Imane, Aya, Amina, Rayane, Rinade, Ishak, Zakaria, Assouma, Ouwais et Mohamed Anes.

Mes remerciements vont également à mes amis de longue date pour leurs encouragements et leur soutien qui m'ont permis de faire ce travail : Hamza Ouali, Raouf Belakrouf, Mohamed Anis Djebrane, Hamza Belabed, Yacine Zelmat, Youcef Zitoun, Zaki Zemmour, Mehdi Djeddai, Sofiane Amazouz, Raid Nour, Mehdi Nadour, Salim Nadour, Karim Kecir, Anes Hocine, Mahmoud Benzamouche, Chihab Dahmani et Abdelhamid Bouzidi.

Je remercie aussi mes amis que j'ai connus pendant ma thèse : Abdelhak Bougouffa, Guia Brahim Fouad, Farid Faidi, Hicham Djanaouissine, Reda Bakalem et Jad Itani.

Enfin, merci à toutes celles et tous ceux qui ont contribué, de près ou de loin, d'une manière ou d'une autre, à l'élaboration de ce travail.

Contents

Introduction (en Français)

Contexte

Aujourd'hui, l'informatique est présente dans tous les domaines et a beaucoup plus d'impact sur nos vies qu'auparavant. Par conséquent, il est nécessaire de vérifier si les programmes informatiques sont construits correctement et sont exempts de bugs, en particulier dans les logiciels sensibles traitant des comptes bancaires, des dispositifs nucléaires et des programmes de pilotage automatique, qui peuvent conduire à une catastrophe ou une tragédie. Pour vérifier ces programmes, on peut utiliser des méthodes formelles puisqu'elles ont atteint un niveau de maturité significatif au cours des dernières décennies. Une de ces méthodes est la méthode déductive, où l'on spécifie le comportement attendu d'un programme en utilisant des propriétés mathématiques, qui sont ensuite formellement prouvées.

Il existe différentes approches et méthodes permettant de vérifier la véracité d'une formule mathématique. L'une de ces approches utilise des outils de preuve, des outils conçus pour vérifier l'exactitude des preuves mathématiques générées sur ordinateur [RPS+19, Ler09, KEH+09]. Il existe deux types d'outil de preuve: les outils automatiques et les outils interactifs. Les premiers (ATPs pour *Automated Theorem Provers*) sont entièrement automatisés et ne nécessitent aucune interaction humaine. Les seconds sont appelés assistants de preuve (ITPs pour *Interactive Theorem Provers*). Il y a par exemple Coq [CH86], Isabelle [Nip21], PVS [OS70], Lean [dMKA+15], etc. Ces prouveurs de théorèmes aident un utilisateur à spécifier et à prouver des déclarations mathématiques, y compris les propriétés des systèmes informatiques.

Cependant, bien qu'il y ait eu beaucoup d'améliorations dans les interfaces utilisateur, les assistants de preuve ne sont pas encore très faciles à utiliser car ils nécessitent une solide connaissance en logique. Pour faciliter l'utilisation de ces outils ou tout simplement pour augmenter la productivité, on peut utiliser des prouveurs automatiques pour gérer les buts pour lesquels des algorithmes efficaces sont connus (par ex. logique propositionnelle, arithmétique linéaire).

Prouveurs Automatiques

Nous pouvons distinguer certaines familles de Provers automatisés :

- Les solveurs SAT [GKSS08] tels que Glucose [SA09] et PicoSat [Bie08];
- Les solveurs SMT tels que veriT [BdODF09], Z3 [dMB08], CVC4 [DRK+14], Yices [Dut14] et *ArchSAT* [BCD18];
- Les prouveurs du premier ordre tels que **E** [Sch13], *ZenonModulo* [DDG+13] et VAMPIRE [KV13];
- Les prouveurs de logique d'ordre supérieur comme ZipperPosition [Cru15] et Satallax [Bro12].

Les solveurs SAT (SATisfiablité) et SMT (Satisfiabilité Modulo Théories) prennent comme entrée des formules sans quantificateur et essaient de trouver une interprétation booléenne qui satisfasse cette formule. Sinon, ils renvoient une preuve de l'inexistence d'une telle interprétation. Les solveurs SMT combinent un solveur SAT avec des solveurs pour des théories particulières comme l'arithmétique linéaire, la théorie des tableaux ou la théorie des vecteurs de bits (e.g. $(x > 0 \vee x < 10) \wedge (y > 3) \wedge (y = x + 2)$). Les prouveurs pour la logique du premier ordre ou la logique d'ordre supérieur prennent comme entrée des formules avec des quantificateurs et s'appuient sur un calcul de preuve pour démontrer la formule donnée.

Interopérabilité

L'utilisation de prouveurs automatiques dans les assistants de preuve est un exemple d'interopérabilité entre les systèmes de preuve. L'interopérabilité entre les systèmes de preuve interactifs est également utile pour éviter la duplication inutile du travail [AC15, Ass15, Cau16, CD, Thi18, Thi20, Gen20, Fér21].

L'interopérabilité entre les systèmes de preuve exige une traduction à deux niveaux: une traduction au niveau des propositions, et une traduction au niveau des preuves. Cela soulève naturellement un certain nombre de questions. Premièrement, la traduction est-elle correcte? Autrement dit, si une proposition P est traduite en une proposition P', est-ce qu'une preuve de P est traduite en une preuve de P'? Deuxièmement, la traduction est-elle complète? La prouvabilité de P' implique-t-elle la prouvabilité de P?

Contrairement à d'autres domaines de l'informatique, l'interopérabilité

entre les outils de preuve n'est pas bien développée. Il y a une raison profonde à cela: les prouveurs de théorèmes reposent sur des bases logiques différentes et parfois incompatibles. Il est donc souvent difficile, et parfois impossible, de traduire une preuve d'un système à un autre. Il est toutefois important de faire de notre mieux chaque fois que cela est possible.

Ce travail contribue à améliorer l'interopérabilité entre les assistants de preuves et les prouveurs automatiques, dans les deux sens.

Cadres logiques

Développer (et maintenir!) un traducteur pour chaque paire de systèmes utilisés dans le monde serait très coûteux ($O(n^2)$ traducteurs pour n systèmes). Une stratégie plus durable consiste à utiliser un langage intermédiaire, à traduire chaque système dans ce langage intermédiaire et à traduire ce langage intermédiaire dans tous les systèmes (traducteurs $O(n)$ pour n systèmes).

Un tel langage intermédiaire commun est appelé un cadre logique (*logical framework*). Un cadre logique permet la définition précise et donc la comparaison des fondements logiques des prouveurs de théorème, et éventuellement de leurs preuves. Par exemple, le calcul des prédicats est un cadre logique que les mathématiciens utilisent pour exprimer leurs théories comme la théorie des ensembles, l'arithmétique, la théorie des groupes, la géométrie, etc. Cependant, le calcul des prédicats ne permet pas de gérer facilement des termes avec des variables liées, des termes de preuve, etc. Ainsi, depuis les années 80, divers nouveaux cadres logiques ont été développés pour mieux gérer ces fonctionnalités: Isabelle [Pau94], LF [HHP93a], λProlog [Mil91], etc. Et, plus récemment, Le $\lambda\Pi$-calcul modulo theorie [CD07, ABC$^+$] qui subsume les précédents.

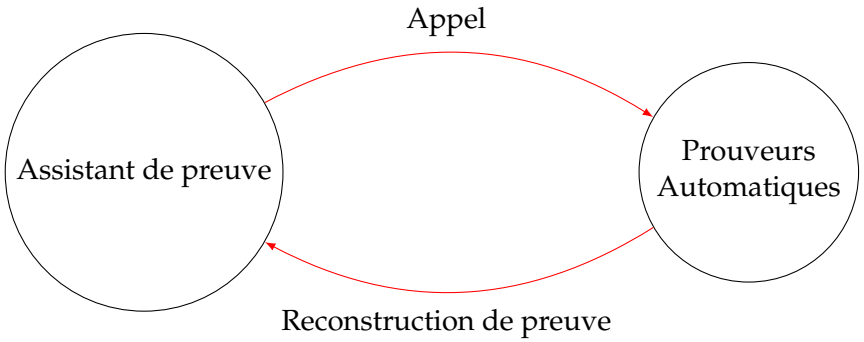
L'outil DEDUKTI (`https://github.com/deducteam/dedukti`) [Boe11] [Sai15] fournit un vérificateur de type pour ce cadre logique. LAMBDAPI (`https://github.com/deducteam/lambdapi`) est une extension avec métavariables pour gérer des termes et des preuves incomplets, et permettre leur définition interactive.

Plusieurs outils ont été développés autour de DEDUKTI pour permettre la traduction de bibliothèques entre différents assistants de preuve: LOGIPEDIA permet la traduction des preuves DEDUKTI vers Coq, HOL-Light, Lean, Matita et PVS [Thi19]; Holide traduit les preuves OpenTheory de HOL-Light ou HOL4 vers DEDUKTI [Ass15]; Krajono traduit les preuves de Matita vers DEDUKTI [Thi20]; Focalide traduit les développements de FoCaLiZe en DEDUKTI [Cau16]; Coqine traduit des preuves de Coq vers

DEDUKTI [BB12, ADJL16, Fér21]; Agda2dk traduit des preuves de Agda vers DEDUKTI [Gen20].

Reconstruction des preuves

En général, les assistants de preuve ne comprennent pas les preuves générées par les prouveurs automatiques puisqu'ils ne partagent pas la même logique ni le même format pour les preuves. Ainsi, une reconstruction de preuve est nécessaire. Cette reconstruction traduit principalement des preuves de la logique des prouveurs automatiques vers la logique de l'assistant de preuve. En outre, les démonstrateurs automatiques omettent parfois certaines informations qui sont importantes pour les assistants de preuve. Ainsi, nous devons retrouver ces parties manquantes.



Plusieurs travaux ont été entrepris pour tenter de résoudre ces problèmes tels que SledgeHammer [BN10], un outil qui utilise les prouveurs automatiques et les solveurs SMT pour générer des preuves dans l'assistant de preuve Isabelle. Un autre exemple est SMTCoq [AFG+11], un plugin utilisé dans l'assistant de preuve Coq pour vérifier les résultats des solveurs SMT.

Cependant, ces outils fonctionnent pour un assistant de preuve spécifique et ne peuvent pas être utilisés avec d'autres assistants de preuves et, en particulier, ni avec DEDUKTI ni avec LAMBDAPI. Pourtant, avoir un outil de reconstruction de preuve pour LAMBDAPI, qui est utilisé comme langage intermédiaire pour de nombreux assistants de preuve, permettra l'utilisation de prouveurs automatiques dans beaucoup d'autres assistants de preuve.

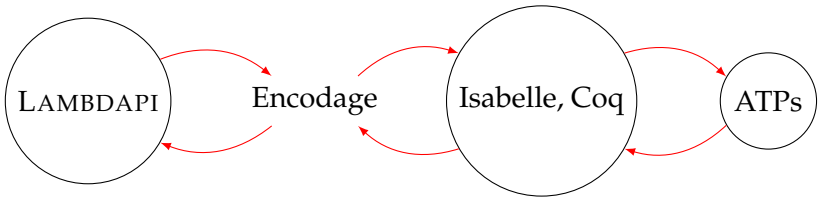
La construction de preuves peut également être utile pour traduire certains assistants de preuves vers LAMBDAPI. Par exemple, PVS [Sha96] utilise en interne des prouveurs automatiques pour lesquels aucune trace

n'est générée ou pour lesquels il serait difficile d'obtenir des traces car il faudrait instrumenter le code de ces prouveurs automatiques.

En fait, la reconstruction de preuves pourrait même être utilisée comme stratégie de traduction. Au lieu de définir une traduction au niveau de la preuve, on pourrait extraire les lemmes utilisés dans une preuve et demander à un prouveur automatique de prouver un théorème en utilisant ces lemmes, puis utiliser un outil de reconstruction de preuve pour obtenir une preuve complète.

Une solution pourrait être de modifier Sledgehammer ou CoqHammer [CK18] afin qu'ils génèrent du code Lambdapi au lieu de générer du code Metis [PS] ou Coq. Cependant, nous pourrions faire face à certains problèmes avec les encodages utilisés dans ces systèmes de preuves :

- Nous devons construire une traduction entre l'encodage de la logique de premier ordre dans Isabelle ou Coq et l'encodage de la logique de premier ordre dans $\lambda\Pi/\equiv$, ce qui n'est pas une tâche simple et peut varier d'un système de preuve à l'autre.
- L'encodage de la logique du premier ordre dans ces systèmes de preuve (Isabelle, Coq) pourrait utiliser certaines fonctionnalités qui ne sont pas nécessaires. Ainsi, une implémentation de ces fonctionnalités devrait être ajoutée dans LAMBDAPI. Par exemple, Isabelle pourrait utiliser l'axiome du choix pour gérer les étapes de Skolemisation effectuées par le prouveur automatique.
- Beaucoup de transformations seront faites au sous-but que nous voulons prouver afin d'avoir une preuve en $\lambda\Pi/\equiv$, ce qui conduit à avoir une superposition d'encodages. Il serait donc difficile de retracer les erreurs, comme on le voit ci-dessous :



Contributions

Pour résoudre les problèmes mentionnés ci-dessus, nous avons conçu plusieurs solutions théoriques et techniques, qui sont présentées dans cette thèse.

Appeler des prouveurs externes

Puisque les prouveurs n'utilisent pas la même logique ni le même système que l'assistant de preuve, nous avons élaboré une traduction entre la logique de LAMBDAPI qui est le $\lambda\Pi$-calcul Modulo Théorie et la logique du premier ordre, utilisée par les prouveurs automatiques. Cette traduction est implémentée à l'intérieur de LAMBDAPI comme une partie de son langage tactique. De plus, nous avons montré que cette traduction est correcte, c'est-à-dire que s'il y a une preuve de la traduction d'une formule dans la logique de premier ordre, alors il y a une preuve de cette formule dans le $\lambda\Pi$-calcul Modulo Théorie.

Reconstitution des preuves

Comme nous l'avons souligné dans la section précédente, les assistants à la preuve ne supportent pas toujours la sortie des prouveurs externes. Cependant, pour certains assistants de preuve, il existe des prouveurs produisant un format supporté par cet assistant de preuve. C'est le cas de LAMBDAPI avec *ZenonModulo* [DDG+13] et *ArchSAT* [BCD18], ainsi que de Coq avec Zenon [BDD07]. Nous avons conçu une architecture qui reconstruit, vers LAMBDAPI, des preuves générées par des prouveurs, et ce en détectant quelles informations sont manquantes et en demandant à un prouveur supporté par Dedukti de générer cette information. La solution proposée ne dépend pas d'un prouveur spécifique et permet de reconstruire un nombre considérable de preuves. Le format de la sortie des prouveurs choisi est TSTP [Sut17] étant donné que c'est le format standard utilisé pour tester les prouveur automatiques. Cette solution est implémentée dans un outil séparé appelé EKSTRAKTO[1].

Transformer les preuves Skolemisées

Les démonstrateurs automatiques peuvent utiliser de nombreuses techniques pour prouver une formule. L'une de ces techniques est la Skolemisation. Cette étape transforme la formule originale en une formule équisatifiable mais pas syntaxiquement équivalente. Nous avons mis au point un algorithme qui transforme la preuve de la nouvelle formule en une preuve de la formule originale. L'algorithme présenté s'est avéré correct et pourrait être utilisé séparément si la preuve correspond à une spécification particulière. Cet algorithme est implémenté comme un outil nommé **SKon-**

[1] L'outil est disponible sur `https://github.com/elhaddadyacine/ekstrakto`

verto[2], qui peut être utilisé avec EKSTRAKTO pour construire des preuves dans LAMBDAPI.

Aperçu

Ce document est organisé comme suit :
Dans le Chapitre 1, nous rappelons certaines notions et définitions de la logique du premier ordre, qui sont utilisées dans les Chapitres 2 et 5. Nous rappelons également les règles de la déduction naturelle comme calcul de preuve pour la logique du premier ordre. Ensuite, nous présentons les formats TPTP et TSTP avec un exemple. Ces formats sont les formats standards d'entrée et de sortie utilisés par la plupart des prouveurs automatiques. La compréhension de ces formats est essentielle pour comprendre le Chapitre 3, qui traite de la reconstruction des preuves. Nous présentons ensuite le $\lambda\Pi$-calcul Modulo Théorie, les notions de règles de réécriture, de règles de typage et certaines définitions et résultats qui sont utilisées principalement dans le Chapitre 5 mais aussi dans les Chapitres 2 et 3. Nous rappelons également, dans ce chapitre, l'encodage de la logique du premier ordre dans le $\lambda\Pi$-calcul Modulo Théorie. Nous terminons le chapitre en présentant le calcul mis en œuvre dans le prouveur automatique *ZenonModulo*.

Dans le Chapitre 2, nous introduisons une façon d'appeler les prouveurs depuis LAMBDAPI en expliquant la traduction faite du $\lambda\Pi$-calcul Modulo Théorie vers la logique propositionnelle avec la preuve de sa correction. En outre, nous présentons la traduction entre $\lambda\Pi$-calcul Modulo Théorie et la logique du premier ordre, ainsi que les changements nécessaires pour adapter la preuve de correction.

Au Chapitre 3, nous présentons la technique utilisée pour reconstruire les preuves à partir des formats TSTP en utilisant *ZenonModulo* et *ArchSAT*. Le chapitre est divisé en trois parties: l'extraction des problèmes dans TPTP, la reconstruction de la preuve elle-même, et quelques expériences pour montrer le succès de l'outil.

Dans le Chapitre 4, nous introduisons les coupures et leurs représentations dans le $\lambda\Pi$-calcul Modulo Théorie. Par la suite, nous montrons ce que sont les coupures commutatives ainsi que leur impact. La compréhension de ce chapitre est nécessaire au chapitre suivant.

Dans le Chapitre 5, nous présentons le théorème de Skolem ainsi qu'un algorithme qui transforme une preuve contenant le symbole de Skolem en

[2]L'outil est disponible sur `https://github.com/elhaddadyacine/SKonverto`

une autre ne contenant pas ce symbol. Nous fournissons également une preuve de correction de l'algorithme, et nous terminons en présentant son implémentation.

Nous poursuivons avec le Chapitre 6 dans lequel nous présentons divers travaux connexes et leurs différences avec notre travail.

Nous concluons par le Chapitre 7, qui contient les différentes perspectives qui peuvent utiliser ce travail de recherche.

Chapter 0

Introduction (in English)

Context

Nowadays, computer science is present in every field and has much more impact on our lives than before. Hence, it is necessary to check if the programs are built correctly and are free of bugs, especially in software dealing with bank accounts, nuclear devices and autopilot programs, which can lead to a catastrophe or a tragedy. To check these programs, one can use formal methods since they reached a significant level of maturity in the last decade. One of these methods is the deductive method, where we specify the expected behaviour of a program using mathematical properties, which are then formally proved.

Checking the veracity of a mathematical formula is done in various ways and approaches. One of these approaches uses theorem provers, tools designed to check the correctness of mathematical proofs generated on computers [RPS+19, Ler09, KEH+09]. There are two types of theorem provers: automated and interactive. Automated Theorem Provers (ATPs) are fully automated and do not require any human interaction. Interactive Theorem Provers (ITPs), also called proof assistants such as Coq [CH86], Isabelle [Nip21], PVS [OS70], Lean [dMKA+15], etc. These theorem provers help a user specify and prove mathematical statements, including properties of computer systems.

However, although there has been a lot of improvements in user interfaces, proof assistants are still not very easy to use as they require a strong background in logic. To facilitate the use of these tools or increase productivity, one can allow the use of Automated Theorem Provers to take care of goals for which good algorithms are known (e.g. propositional logic, linear

arithmetic).

Automated Theorem Provers

We can distinguish some families of Automated Theorem Provers:

- SAT solvers [GKSS08] such as Glucose [SA09] and PicoSat [Bie08];
- SMT solvers such as veriT [BdODF09], Z3 [dMB08], CVC4 [DRK+14], Yices [Dut14] and *ArchSAT* [BCD18];
- First-order theorem provers such as **E** [Sch13], *ZenonModulo* [DDG+13] and VAMPIRE [KV13];
- Provers for higher-order logic such as ZipperPosition [Cru15] and Satallax [Bro12].

SAT (SATisfiabiliy) and SMT (Satisfiability Modulo Theories) solvers take as input quantifier-free formulas and try to find a boolean interpretation that satisfies that formula. Otherwise, they return a proof of the non-existence of such an interpretation. However, SMT solvers are combined with a set of theories such as arithmetic, array and bit-vector (e.g. $(x > 0 \vee x < 10) \wedge (y > 3) \wedge (y = x + 2)$). Provers for first-order or higher-order logic take formulas with quantifiers as input and rely on a proof calculus to prove the given formula.

Interoperability

Using automated theorem provers in proof assistants is an example of interoperability between proof systems. Interoperability between interactive proof systems is equally useful to avoid useless work duplication [AC15, Ass15, Cau16, CD, Thi18, Thi20, Gen20, Fér21].

Interoperability between proof systems requires a two-level translation: a translation at the level of statements, and a translation at the level of proofs. This naturally raises a number of questions. First, is the translation correct? That is, if a statement P is translated into a statement P', is a proof of P translated into a proof of P'? Second, is the translation complete? Does the provability of P' implies the provability of P?

In contrast with other areas of computer technology, the interoperability between theorem provers is not well developed. There is a deep reason for that: theorem provers are based on different, and sometimes incompatible, logical foundations. It is therefore often difficult, and sometimes impossi-

ble, to translate a proof from one system into another one. It is, however, important to try our best whenever it is possible.

This work contributes to improving the interoperability between proof assistants and automated theorem provers, in both ways.

Logical Frameworks

Developing (and maintaining!) a translator for every pair of systems used in the world would be very expensive ($O(n^2)$ translators for n systems). A more sustainable strategy is to use an intermediate language, translate every system to this intermediate language, and translate this intermediate language to every system ($O(n)$ translators for n systems).

Such a common intermediate language is called a logical framework. Logical frameworks allow the precise definition and thus the comparison of the logical foundations of theorem provers, and possibly their proofs. For instance, the Predicate calculus is a logical framework in which mathematicians are used to express their theories like set theory, arithmetic, group theory, geometry, etc. However, the Predicate calculus does not allow to easily handle terms with bound variables, proof terms, etc. Hence, since the 80's, various new logical frameworks have been developed to better handle these features: Isabelle [Pau94], LF [HHP93a], λProlog [Mil91], etc. and, more recently, the $\lambda\Pi$-calculus modulo theory [CD07, ABC$^+$] which subsumes the previous ones.

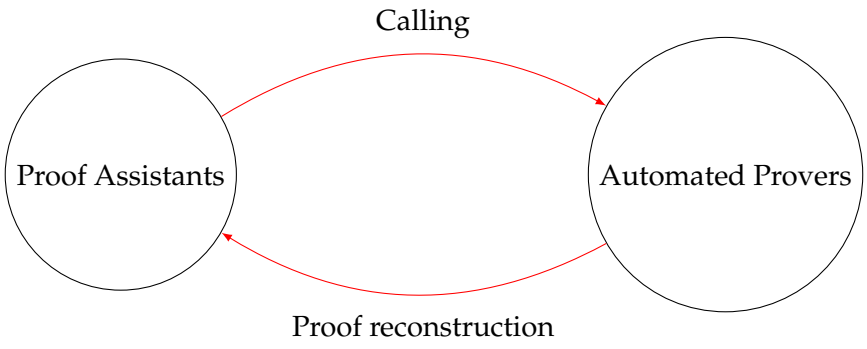
The DEDUKTI[1] tool [Boe11, Sai15] provides a type-checker for this logical framework. LAMBDAPI[2] is an extension of DEDUKTI with metavariables for handling incomplete terms and proofs, and allow their interactive definition.

Several tools have been developed around DEDUKTI to allow the translation of libraries between various proof assistants: LOGIPEDIA allows the translation of DEDUKTI proofs to Coq, Lean, HOL-Light, PVS and Lean [Thi19]; Holide translates OpenTheory proofs from HOL-Light or HOL4 to DEDUKTI [Ass15]; Krajono translates Matita proofs to DEDUKTI [Thi20]; Focalide translates FoCaLiZe developments to DEDUKTI [Cau16]; Coqine translates Coq proofs to DEDUKTI [BB12, ADJL16, Fér21]; Agda2dk translates Agda proofs to DEDUKTI [Gen20].

[1] `https://github.com/deducteam/dedukti`
[2] `https://github.com/deducteam/lambdapi`

Proof reconstruction

In general, proof assistants do not understand the proofs generated by automated theorem provers since they do not share the same logic and the same format for proofs. Thus, a proof reconstruction is necessary. This reconstruction mainly translates proofs from the logic of the automated theorem provers to the logic of the proof assistant. Also, automated theorem provers sometimes omit some information that are important for proof assistants. Thus, we need to find out the missing parts.



Several works have been done to try to solve these problems such as SledgeHammer [BN10], a tool that uses automated theorem provers and SMT solvers to generate proofs in the proof assistant Isabelle. Another example is SMTCoq [AFG+11], a plugin used in the proof assistant Coq to check SMT solvers' output.

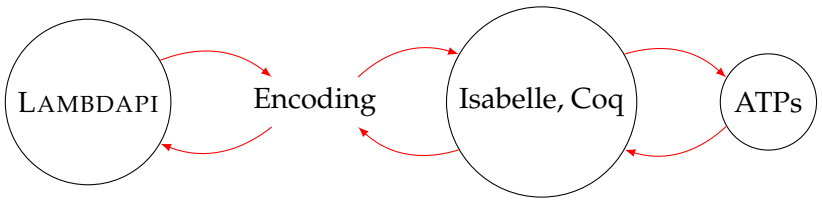
However, these tools work for a specific proof assistant and cannot be used with other proof assistants and, in particular, neither with DEDUKTI nor LAMBDAPI. Yet, having a proof reconstruction tool for LAMBDAPI, which is used as an intermediate language for many proof assistants, will allow the use of automated theorem provers in many other proof assistants.

Proof reconstruction may also be useful to translate some proof assistants to LAMBDAPI. For instance, PVS [Sha96] internally uses automated theorem provers for which no trace are generated or for which traces would be difficult to obtain as it would require to instrument the code of those automated theorem provers.

In fact, proof reconstruction could even be used as a translation strategy. Instead of defining a translation at the proof level, one could extract the lemmas used in a proof and ask an automated theorem prover to prove a theorem by using these lemmas, and then use a proof reconstruction tool to get a full proof.

A solution could be to modify Sledgehammer or CoqHammer [CK18] so that they output some LAMBDAPI code instead of Metis [PS] or Coq code. However, we could face some problems with the encodings used in these proof systems:

- We need to build a translation between the encoding of first-order logic in Isabelle or Coq and the encoding of first-order logic in $\lambda\Pi/\equiv$, which is not a simple task and can vary from one proof system to the other.
- The encoding of first-order logic in these proof systems (Isabelle, Coq) could use some features that are not necessary. Thus, an implementation of these features should be added in LAMBDAPI. For instance, Isabelle could use the axiom of choice to handle Skolemization steps performed by the automated prover.
- Plenty of transformations will be done to the subgoal that we want to prove in order to have a proof in $\lambda\Pi/\equiv$, which leads to having an overlay of encodings. Thus, it would be difficult to trace back errors as shown below:



Contributions

To solve the problems pointed above, we conceived several theoretical and technical solutions that are presented in this thesis.

Calling external provers

Since the provers do not use the same logic and system as the proof assistant, we elaborated a translation between LAMBDAPI's logic which is the $\lambda\Pi$-calculus Modulo Theory and the provers' logic which is first-order logic. This translation is implemented inside LAMBDAPI as part of its tactic language. Moreover, we showed that this translation is correct, i.e., if there

is a proof of the translation of a formula in first-order logic, then there is a proof of that formula in the $\lambda\Pi$-calculus Modulo Theory.

Reconstructing proofs

As pointed out in the previous section, proof assistants do not always support the output of external provers. However, for some proof assistants, there is a prover that outputs a format supported by that proof assistant. This is the case of LAMBDAPI with *ZenonModulo* [DDG+13] and *ArchSAT* [BCD18], and Coq with Zenon [BDD07]. We designed an architecture to reconstruct proofs generated by provers into LAMBDAPI by detecting which information is missing and asking a prover supported by LAMBDAPI to generate that missing information. The proposed solution does not depend on a specific prover and can reconstruct a considerable amount of proofs. The output format of the provers chosen is TSTP [Sut17] since it is the standard format used to test automated theorem provers. This solution is implemented as an independent tool named EKSTRAKTO[3].

Transforming Skolemized proofs

Provers can use many techniques to prove a formula, and one of these techniques is Skolemization. This transformation step transforms the original formula into an equi-satisfiable but not syntactically equivalent formula. We implemented an algorithm that transforms the proof of the new formula into a proof of the original one. The presented algorithm is proved correct and could be used separately if the proof fits a particular specification. This algorithm is implemented as a tool named bfSKonverto [4], which could be used with EKSTRAKTO to build proofs in LAMBDAPI.

Outline

This document is organized as follows:

- In Chapter 1, we recall some notions and definitions of First-Order Logic that are used in Chapter 2 and Chapter 5. We also recall Natural Deduction as a proof calculus for First-Order Logic. Then we

[3]The tool is available in `https://github.com/elhaddadyacine/ekstrakto`
[4]The tool is available in `https://github.com/elhaddadyacine/SKonverto`

present the formats TPTP and TSTP with an example. These formats are the standard formats of output used in most ATPs, and the understanding of TPTP and TSTP is essential to comprehend Chapter 3, which talks about proof reconstruction. We follow by presenting $\lambda\Pi$-calculus Modulo Theory , the notions of rewriting rules, typing rules and some definitions that are used mainly in Chapter 5 but also in Chapter 2 and Chapter 3. We also recall, in this chapter, the encoding of First-Order Logic in $\lambda\Pi$-calculus Modulo Theory. We finish the chapter by presenting the calculus implemented in the automated theorem prover *ZenonModulo*.

- In Chapter 2, we introduce a way to call provers from LAMBDAPI by explaining the translation done from $\lambda\Pi$-calculus Modulo Theory to Propositional Logic with a proof of its correctness. Also, we present the translation between $\lambda\Pi$-calculus Modulo Theory and First-Order Logic and the necessary changes to adapt the proof of correctness.

- In Chapter 3, we present the technique used to reconstruct proofs from TSTP formats by using *ZenonModulo* and *ArchSAT*. The chapter is split into three parts: the extraction of problems (in TPTP), the proof reconstruction itself and some experiments to show the success of the tool.

- In Chapter 4, we introduce cuts and their representations in $\lambda\Pi$ Modulo Theory. After, we show what commutative cuts are, as well as what their impact might be. This chapter is mandatory for the next chapter.

- In Chapter 5, we present the Skolem theorem and an algorithm hat transforms a proof containing a Skolem symbol into a proof not containing that Skolem symbol. We also provide a proof of correctness of the algorithm, and finish by presenting its implementation.

- We follow by Chapter 6, where we present various related works with their differences with our work.

- We conclude by Chapter 7, which contains different perspectives that could build on this research.

Chapter 1

Preliminaries

This chapter introduces necessary preliminaries and basic definitions of First order logic, TPTP and TSTP formats, $\lambda\Pi$ Modulo Theory, the encoding of first order logic in $\lambda\Pi/\equiv$ and *ZenonModulo* that will be used later.

1.1 First order logic ($\mathcal{FOL}$)

First-order logic is a formalism that represents mathematical objects. It is used, for instance, in the theorem proving field to represent formulas that we want to prove. It has two levels of presentation: terms and predicates. A term is either a variable or a function symbol applied to other terms. A predicate is either a predicate symbol, an equality, a quantified formula, or two formulas linked with a connector.

Definition 1.1.1 (Terms and propositions)
Let $\mathcal{V}$ be a set of variables, $\mathcal{F}$ be a set of function symbols and $\mathcal{P}$ a set of predicate symbols. Every function or predicate symbol has an arity $n \in \mathbb{N}$, indicating the number of arguments it takes. The terms and propositions of first-order logic are then defined as follows:

- *(variables) x is a term if $x \in \mathcal{V}$.*
- *(functions) $f\ t_1 \dots t_n$ is a term if $f \in \mathcal{F}$ is of arity n and $t_1, \dots, t_n$ are n terms.*

The set of propositions of first-order logic is defined as follows:

- *(predicates) $P\ t_1 \dots t_n$ is a proposition if $P \in \mathcal{P}$ of arity n and $t_1, \dots, t_n$ are n terms.*

- *(top)* $\top$ *is a proposition.*
- *(bot)* $\bot$ *is a proposition.*
- *(equality)* $t_1 = t_2$ *is a proposition if* t_1 *and* t_2 *are terms.*
- *(connectors)* $A \bullet B$ *is a proposition if* A *and* B *are propositions and* $\bullet \in \{\Rightarrow, \wedge, \vee, \Leftrightarrow\}$
- *(universal quantifier)* $\forall x A$ *is a proposition if* $x \in \mathcal{V}$ *and* A *is a proposition.*
- *(existential quantifier)* $\exists x A$ *is a proposition if* $x \in \mathcal{V}$ *and* A *is a proposition.*

Remark 1.1.2
In the following, we assume that $\neg A$ *is a short hand for* $A \Rightarrow \bot$.

Notation 1.1.3
- *The expression* $\neg(\neg A)$ *is denoted by* $\neg\neg A$.
- *The expression* $\neg(A = B)$ *is denoted by* $A \neq B$.
- *We note the sequence* $x_1, \ldots, x_n$ *by* $\bar{x}$, *and* $\forall x_1, \ldots, \forall x_n A$ *by* $\forall \bar{x} A$.

Notation 1.1.4
The notation $[t/x]A$, *denotes the substitution of the variable* x *by the term* t *in the formula* A.

Definition 1.1.5 (Natural Deduction)
Natural Deduction [Gen34] is a proof calculus built by using inference rules on top of a logic, which is here first-order logic. The set of inference rules defining Natural Deduction is as follows:

- *(axiom):*

$$\frac{F \in \Gamma}{\Gamma \vdash F}$$

- *(*$\top$*-intro):*

$$\frac{}{\Gamma \vdash \top}$$

- *(*$\bot$*-elim):*

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash F}$$

- *(*$\wedge$*-intro):*

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

- *($\wedge$-elim$_1$):*

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$$

- *($\wedge$-elim$_2$):*

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$

- *($\vee$-elim):*

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}$$

- *($\vee$-intro$_1$):*

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}$$

- *($\vee$-intro$_2$):*

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

- *($\Rightarrow$-intro):*

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$$

- *($\Rightarrow$-elim):*

$$\frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

- *($\neg$-intro):*

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

- *($\neg$-elim):*

$$\frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot}$$

- *(=-elim):*

$$\frac{\Gamma \vdash t = v \qquad \Gamma \vdash [t/x]A}{\Gamma \vdash [v/x]A}$$

- *(=-intro):*

$$\frac{}{\Gamma \vdash t = t}$$

- *(∀-intro):*

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x A}\ x \notin FV(\Gamma)$$

- *(∀-elim):*

$$\frac{\Gamma \vdash \forall x A}{\Gamma \vdash [t/x]A}$$

- *(∃-intro):*

$$\frac{\Gamma \vdash [t/x]A}{\Gamma \vdash \exists x A}$$

- *(∃-elim):*

$$\frac{\Gamma \vdash \exists x A \qquad \Gamma, A \vdash C}{\Gamma \vdash C}\ x \notin FV(\Gamma; C)$$

All these rules represent the natural deduction for intuitionistic logic. In order to have classical logic, we need to add a new rule (nnpp):

- (nnpp):

$$\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$$

Remark 1.1.6
With (nnpp), the rule (⊥-elim) above is admissible.

Classical logic is the logic where we can prove the excluded middle, i.e., the formulas $P \vee \neg P$ for all propositions P, which is not possible in intuitionistic logic. In intuitionistic logic, the notion of veracity is replaced by the notion of a proof, i.e, if we want to prove a formula F in intuitionistic logic, we need to provide a proof π of F and not just enumerate all cases for every variable appearing in F.

1.2 TPTP and TSTP

TPTP [Sut17] is a standard library of problems to test automated theorem provers [Sut18]. Each TPTP file represents a problem in propositional, first-order or higher-order logic. We distinguish the type of formulas by using one of the keywords: CNF (mono-sorted first-order formulas in clausal normal form), FOF (general mono-sorted first-order formulas), TFF (multi-sorted first-order formulas) and THF (typed higher-order formulas).

Apart from an include instruction, each line in a TPTP file is a declaration of a formula given with its role, e.g. axiom, hypothesis, definition or conjecture:

```
cnf(name, role, formula).
```

TSTP [Sut17] is a library of solutions to TPTP problems. In the following, we call a TSTP file a trace. It is obtained after running an automated theorem prover on a TPTP problem. The syntax used in a TSTP file is the same as TPTP except for a new field called *information* that is added after the formula in a TSTP line:

```
cnf(name, role, formula, information).
```

This field contains general information about how the current formula is obtained. Here is the grammar used to describe a source in the *information* field:

```
<source>     :== <dag_source> | [ <sources> ] | ..
<dag_source> :== <name> | inference(name, infos, <inference_parents>)
<inference_parents> :== [] | [ <sources> ]
<sources>    :== <source> (, <source>)*
```

For our purpose, only 3 cases are of interest, as shown in the grammar above:

1) When it is the name of a formula previously declared.

2) When it is a list of several sources:

   ```
   [s_0, s_1, ..., s_n]
   ```

3) When it is an inference:

```
inference(name, infos, [s_0, s_1, ..., s_n])
```

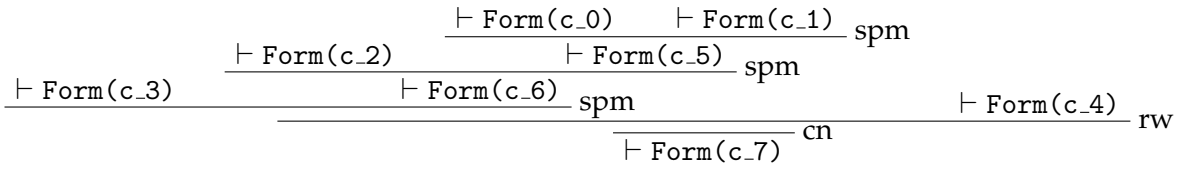
The name of the inference refers to the name of the rule used by the prover to prove the current step. The *infos* field contains more information about the inference like status. Note that each s_i is a source and therefore can contain sub-inferences.

Here is an example of a TSTP file obtained after running **E** on the TPTP problem SET001-1:

SET001-1.p

```
cnf(c_0,axiom,
    ( subset(X1,X2)
    | ~ equal_sets(X1,X2) )).
cnf(c_1,hypothesis,
    ( equal_sets(b,bb) )).
cnf(c_2,axiom,
    ( member(X1,X3)
    | ~ member(X1,X2)
    | ~ subset(X2,X3) )
).
cnf(c_3,negated_conjecture,
    ( ~ member(element_of_b,bb) )).
cnf(c_4,hypothesis,
    ( member(element_of_b,b) )
).
cnf(c_5,hypothesis,
    ( subset(b,bb) ),
    inference(spm,[status(thm)],[c_0,c_1])
).
cnf(c_6,hypothesis,
    ( member(X1,bb)
    | ~ member(X1,b) ),
    inference(spm,[status(thm)],[c_2,c_5])
).
cnf(c_7,negated_conjecture,
    ( $false ),
    inference(cn,[status(thm)],[inference(rw,[status(thm)],
    [inference(spm,[status(thm)],[c_3,c_6]),c_4])]),
    [proof]
).
```

We can represent this trace as the following tree:



where:

```
Form(c_0) = subset(X1,X2) | ~equal_sets(X1,X2)
Form(c_1) = equal_sets(b,bb)
Form(c_2) = member(X1,X3) | ~member(X1,X2) | ~subset(X2,X3)
Form(c_3) = ~member(element_of_b,bb)
Form(c_4) = member(element_of_b,b)
Form(c_5) = subset(b,bb)
Form(c_6) = member(X1,bb) | ~member(X1,b)
Form(c_7) = $false
```

As shown in this proof tree, there are some derivations where the formula is not specified, and this is due to some provers that perform multiple inference rules in one step without providing the information about the intermediate formula.

1.3 λΠ-calculus Modulo Theory (λΠ/≡)

The λΠ-calculus [dB70, HHP93a] is an extension of the λ-calculus [HS08] with dependent types. The λΠ-calculus Modulo Theory [CD07, ABC+] is an extension of the λΠ-calculus where function and type symbols can be defined by rewriting rules, that is, oriented equations. In this section, we recall the definition and basic properties of the λΠ-calculus Modulo Theory.

Definition 1.3.1 (λΠ-terms)
Let $\mathcal{F}$ be a set of function symbols, $\mathcal{V}$ a set of variables. The set of λΠ-terms Λ is defined as follows:

$$t = TYPE \mid KIND \mid x \mid f \mid tt \mid \lambda x : t, t \mid \Pi x : t, t$$

where $x \in \mathcal{V}$ and $f \in \mathcal{F}$.

Definition 1.3.2 (Free variables)
Let t be a term. $FV(t)$ is the set of free variables that t contains. It is defined as follows:

$$\begin{array}{lcl} FV(x) & = & \{x\} \text{ if } x \text{ is a variable.} \\ FV(AB) & = & FV(A) \cup FV(B) \\ FV(\lambda x : A, B) & = & FV(A) \cup (FV(B) \backslash \{x\}) \\ FV(\Pi x : A, B) & = & FV(A) \cup (FV(B) \backslash \{x\}) \end{array}$$

Definition 1.3.3 (Substitution)
$[N/x]M$ is the term obtained by substituting by N every free occurrence of x in M.

Definition 1.3.4 (β-reduction)
The β-reduction relation $\rightarrow_\beta$ is the smallest relation stable by context (and substitution) containing the pairs $(\lambda x : t, M)N \hookrightarrow_\beta [N/x]M$.

Definition 1.3.5 (Rewriting)
A rewrite rule is a pair of terms (l, r), written $l \hookrightarrow r$, with l of the form $f\ l_1\ \ldots\ l_n$ and $FV(r) \subseteq FV(l)$.

Given a set R of rewrite rules, let $\rightarrow_R$ be the smallest relation stable by context and substitution containing R.

In some rare cases, we will use a more general notion of rewriting rule using higher-order pattern-matching [Mil91] like in Combinatory Reduction Systems [Klo80, KvOvR93] that is supported by Dedukti [Sai15] and Lambdapi [HB20]. In this case, a pattern variable M is applied to arguments between square brakets. In a left-hand side, these arguments must be pairwise distinct bound variables indicating which variables the instantiation of M may depend on. For instance, in an encoding of pure λ-calculus, this allows one to encode the η-reduction as a non-conditional rule as follows:

$$\begin{array}{rcl} app\ (lam\ (\lambda x, M[x]))\ N & \hookrightarrow & M[N] \\ lam\ (\lambda x, app\ M[]\ x) & \hookrightarrow & M[] \end{array}$$

Definition 1.3.6 (Signature)
A $\lambda\Pi/\equiv$ signature is given by:

- *for every function symbol f a term τ_f, called its type, and a sort $\sigma_f \in \{TYPE, KIND\}$, called its sort;*
- *a set R of rewrite rules.*

Let $\rightarrow_{\beta R} = \rightarrow_\beta \cup \rightarrow_R$, $\rightarrow^*_{\beta R}$ be the reflexive and transitive closure of $\rightarrow_{\beta R}$, and $\equiv_{\beta R}$ be the smallest equivalence relation containing $\rightarrow_{\beta R}$.

Definition 1.3.7 (Normal form)
A term t is in normal form (NF) if there is no term u such that $t \hookrightarrow_{\beta R} u$.

Definition 1.3.8 (Well-typed terms)
A typing environment is a possibly empty ordered sequence of type declarations for variables.

Given a signature Σ, a term t is said to be of type A in the signature Σ and the environment Γ if the judgment $\Gamma \vdash t : A$ can be recursively obtained by using the following inference rules, where $sort \in \{TYPE, KIND\}$:

- *TYPE:*

$$\frac{}{\vdash TYPE : KIND}$$

- *Variable:*

$$\frac{\Gamma \vdash A : sort}{\Gamma, x : A \vdash x : A}\ var$$

- *Weakening:*

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : sort}{\Gamma, x : B \vdash t : A}\ weak$$

- *Function symbol:*

$$\frac{\vdash \tau_f : \sigma_f}{\vdash f : \tau_f}\ fun$$

- *Abstraction:*

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash B : sort}{\Gamma \vdash \lambda x : A, t : \Pi x : A, B}\ abs$$

- *Application:*

$$\frac{\Gamma \vdash t : \Pi x : A, B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : [u/x]B}\ app$$

- *Product:*

$$\frac{\Gamma \vdash A : TYPE \qquad \Gamma, x : A \vdash B : sort}{\Gamma \vdash \Pi x : A, B : sort}\ prod$$

- *Conversion:*

$$\frac{\Gamma \vdash t : A \qquad A \equiv_{\beta R} B \qquad \Gamma \vdash B : sort}{\Gamma \vdash t : B}\ conv$$

Type-checking is decidable if the relation $\rightarrow_{\beta R}$ satisfies the following properties [Sai15]:

- $\rightarrow_{\beta R}$ is confluent, that is, if $t \rightarrow^*_{\beta R} u$ and $t \rightarrow^*_{\beta R} v$ then there is w such that $u \rightarrow^*_{\beta R} w$ and $v \rightarrow^*_{\beta R} w$.
- $\rightarrow_{\beta R}$ preserves typing, that is, if $\Gamma \vdash t : A$ and $t \rightarrow_{\beta R} u$, then $\Gamma \vdash u : A$.
- $\rightarrow_{\beta R}$ terminates, that is, there is no infinite sequences of terms $t_0 \rightarrow_{\beta R} t_1 \rightarrow_{\beta R} \ldots$

Confluence implies that every term has at most one normal form, and termination implies that every term has at least one normal form. Therefore, the combination of both implies that every term has a unique normal form.

An important criterion for confluence is orthogonality [KvOvR93]. A system is orthogonal if it is left-linear (no variable occurs more than once in a LHS) and has no critical pairs (no two LHS non-variable subterms overlap). For instance, $\rightarrow_\beta$ is orthogonal.

Note that $\rightarrow_\beta$ always preserves typing when $\rightarrow_{\beta R}$ is confluent [Bla05]. A criterion for type preservation is given in [Bla20]. Both Dedukti and LAMBDAPI implement algorithms for automatically checking type preservation when $\rightarrow_{\beta R}$ is confluent.

We recall hereafter some basic properties of $\lambda\Pi/\equiv$ that we will use later:

Lemma 1.3.9 (Permutation)
[Bla01, Lemma 52, page 51] If $\Gamma, x : A, y : B, \Gamma' \vdash t : T$ *and* $x \notin FV(B)$*, then* $\Gamma, y : B, x : A, \Gamma' \vdash t : T$.

1.4 Encoding of first-order logic in $\lambda\Pi/\equiv$

The encoding of first-order logic formulas and their proofs in $\lambda\Pi/\equiv$ is based on the *Curry-Howard* correspondence, i.e., formulas are interpreted as types and their proofs as terms.

We first define a $\lambda\Pi/\equiv$ signature for representing terms and formulas:

$$
\begin{aligned}
&\Sigma_{\mathcal{FOL}} = \\
&\qquad \iota : TYPE, \\
&\qquad f : \iota \to \ldots \to \iota, \\
&\quad Prop : TYPE, \\
&\qquad P : \iota \to \ldots \to Prop, \\
&\qquad \top : Prop, \\
&\qquad \bot : Prop, \\
&\qquad \neg : Prop \to Prop, \\
&\qquad \wedge : Prop \to Prop \to Prop, \\
&\qquad \vee : Prop \to Prop \to Prop, \\
&\qquad \Rightarrow : Prop \to Prop \to Prop, \\
&\qquad \Leftrightarrow : Prop \to Prop \to Prop, \\
&\qquad \forall : (\iota \to Prop) \to Prop, \\
&\qquad \exists : (\iota \to Prop) \to Prop, \\
&\qquad = : \iota \to \iota \to Prop, \\
&\qquad \epsilon : Prop \to TYPE
\end{aligned}
$$

with a declaration $f : \iota \to \ldots \to \iota$, with n arrow, for each $\mathcal{FOL}$ symbol of arity n, and a declaration $P : \iota \to \ldots \to Prop$, with n arrows, for each $\mathcal{FOL}$ predicate symbol P of arity n.

The symbol ϵ allows us to interpret propositions as types by lifting propositions, which are objects, to $\lambda\Pi/\equiv$ types.

The representation of this signature in LAMBDAPI is given in Appendix 8.1.

Translating $\mathcal{FOL}$ formulas to $\lambda\Pi/\equiv$ terms and types

The function φ defined hereafter translates every $\mathcal{FOL}$ term into a $\lambda\Pi/\equiv$ term of type ι, and every $\mathcal{FOL}$ formula into a $\lambda\Pi/\equiv$ term of type $Prop$:

$$
\begin{array}{rcll}
\varphi(x) & := & x & \text{if } x \text{ is a variable} \\
\varphi(f\ t_1\ t_2\ \ldots\ t_n) & := & f\ \varphi(t_1)\ \varphi(t_2)\ \ldots\ \varphi(t_n) & \text{if } f \text{ is a function symbol} \\
\varphi(P\ t_1\ t_2\ \ldots\ t_n) & := & P\ \varphi(t_1)\ \varphi(t_2)\ \ldots\ \varphi(t_n) & \text{if } P \text{ is a predicate symbol} \\
\varphi(\bot) & := & \bot & \\
\varphi(\top) & := & \top & \\
\varphi(A \wedge B) & := & \varphi(A)\ \wedge\ \varphi(B) & \\
\varphi(A \vee B) & := & \varphi(A)\ \vee\ \varphi(B) & \\
\varphi(A \Rightarrow B) & := & \varphi(A)\ \Rightarrow\ \varphi(B) & \\
\varphi(\forall x A) & := & \forall\ (\lambda x : \iota, \varphi(A)) & \\
\varphi(\exists x A) & := & \exists\ (\lambda x : \iota, \varphi(A)) & \\
\varphi(x = y) & := & \varphi(x)\ =\ \varphi(y) &
\end{array}
$$

Property 1.4.1 (Typable terms)
If t is a $\mathcal{FOL}$ term and $FV(t) = \{x_1, \ldots, x_n\}$ then $x_1 : \iota, \ldots, x_n : \iota \vdash \varphi(t) : \iota$

Property 1.4.2 (Typable propositions)
If F is a proposition in $\mathcal{FOL}$ and $FV(F) = \{x_1, \ldots, x_n\}$ then $x_1 : \iota, \ldots, x_n : \iota \vdash \varphi(F) : Prop$

Deep embedding of Natural Deduction in $\lambda\Pi/\equiv$

Natural Deduction proofs for first-order logic can be represented by $\lambda\Pi/\equiv$ terms by mapping every inference rule of 1.1.5 to a $\lambda\Pi/\equiv$ symbol from the

following signature:

$$
\begin{aligned}
&\Sigma_{ND} = \\
&\quad \bot_E : \epsilon\bot \to \Pi p, \epsilon p \\
&\quad \top_I : \epsilon\top \\
&\quad \wedge_{El} : \Pi p : Prop, \Pi q : Prop, \epsilon(p \wedge q) \to \epsilon p \\
&\quad \wedge_{Er} : \Pi p : Prop, \Pi q : Prop, \epsilon(p \wedge q) \to \epsilon q \\
&\quad \wedge_I : \Pi p : Prop, \Pi q : Prop, \epsilon p \to \epsilon q \to \epsilon(p \wedge q) \\
&\quad \vee_E : \Pi p : Prop, \Pi q : Prop, \epsilon(p \vee q) \to \Pi x, (\epsilon p \to \epsilon x) \to (\epsilon q \to \epsilon x) \to \epsilon x \\
&\quad \vee_{Il} : \Pi p : Prop, \Pi q : Prop, \epsilon p \to \epsilon(p \vee q) \\
&\quad \vee_{Ir} : \Pi p : Prop, \Pi q : Prop, \epsilon q \to \epsilon(p \vee q) \\
&\quad \Rightarrow_E : \Pi p : Prop, \Pi q : Prop, \epsilon(p \Rightarrow q) \to \epsilon p \to \epsilon q \\
&\quad \Rightarrow_I : \Pi p : Prop, \Pi q : Prop, (\epsilon p \to \epsilon q) \to \epsilon(p \Rightarrow q) \\
&\quad =_E : \Pi t : \iota, \Pi v : \iota, \epsilon(t = v) \to \Pi p : \iota \to Prop, \epsilon(p\ t) \to \epsilon(p\ v) \\
&\quad =_I : \Pi t : \iota, \epsilon(t = t) \\
&\quad \forall_I : \Pi p : \iota \to Prop, (\Pi x : \iota, \epsilon(p\ x)) \to \epsilon(\forall p) \\
&\quad \forall_E : \Pi p : \iota \to Prop, \Pi t : \iota, \epsilon(\forall p) \to \epsilon(p\ t) \\
&\quad \exists_E : \Pi p : \iota \to Prop, \epsilon(\exists p) \to \Pi P, (\Pi x : \iota, \epsilon(p\ x) \to \epsilon P) \to \epsilon P \\
&\quad \exists_I : \Pi p : \iota \to Prop, \Pi t, \epsilon(p\ t) \to \epsilon(\exists p)
\end{aligned}
$$

Its representation in LAMBDAPI is given in Appendix 8.2.

Proposition 1.4.3 (Correctness)
Let $F_1, \ldots, F_p \vdash F$ be a $\mathcal{FOL}$ judgement in Natural Deduction (either intuitionist or classical) where $x_1, \ldots, x_n$ are the free variables of $F, F_1, \ldots, F_p$.

If $F_1, \ldots, F_p \vdash F$ is provable, then $\exists \pi \in \Lambda$ where $x_1 : \iota, \ldots, x_n : \iota, h_1 : \epsilon(\varphi(F_1)), \ldots, h_p : \epsilon(\varphi(F_p)) \vdash \pi : \epsilon(\varphi(F))$ is provable in $\lambda\Pi/\equiv$.

Proof. By induction on the set of inference rules of Natural Deduction.

- axiom: If $F \in \{F_1, \ldots, F_p\}$ then $\pi = h_k$ where h_k has the type $\epsilon(\varphi(F))$.
- $\top$-intro: $\pi = \top_I$.
- $\bot$-elim: By induction we have $\pi_\bot : \epsilon(\bot)$ then $\pi = \bot_E\ \pi_\bot\ \varphi(F)$.
- $\wedge$-intro: If $F = A \wedge B$ then by induction hypothesis, we have $\pi_A : \epsilon(\varphi(A))$ and $\pi_B : \epsilon(\varphi(B))$ then $\pi = \wedge_I\ \varphi(A)\ \varphi(B)\ \pi_A\ \pi_B$.

- $\wedge$-elim$_1$: If $F = A$ then by induction hypothesis we have $\pi_{A\wedge B} : \epsilon(\varphi(A \wedge B))$ then $\pi = \wedge_{El}\ \varphi(A)\ \varphi(B)\ \pi_{A\wedge B}$.

- $\wedge$-elim$_2$: If $F = B$ then by induction hypothesis, we have $\pi_{A\wedge B} : \epsilon(\varphi(A \wedge B))$ then $\pi = \wedge_{Er}\ \varphi(A)\ \varphi(B)\ \pi_{A\wedge B}$.

- $\vee$-elim: If $F = C$ then by induction hypothesis, we have $\Gamma \vdash \pi_{A\vee B} : \epsilon(\varphi(A \vee B))$,$\Gamma, a : \epsilon(\varphi(A)) \vdash \pi_{ac} : \epsilon(\varphi(C))$ and $\Gamma, b : \epsilon(\varphi(B)) \vdash \pi_{bc} : \epsilon(\varphi(C))$ then by applying (abs) rule we can have $\Gamma \vdash \lambda a : \epsilon(\varphi(A)), \pi_{ac} : \epsilon(\varphi(A)) \to \epsilon(\varphi(C))$ and $\Gamma \vdash \lambda b : \epsilon(\varphi(B)), \pi_{bc} : \epsilon(\varphi(B)) \to \epsilon(\varphi(C))$ thus $\Gamma \vdash \vee_E\ \varphi(A)\ \varphi(B)\ \pi_{A\vee B}\ \varphi(C)\ (\lambda a : \epsilon(\varphi(A)), \pi_{ac})\ (\lambda b : \epsilon(\varphi(B)), \pi_{bc})$.

- $\vee$-intro$_1$: If $F = A \vee B$ then by induction hypothesis, we have $\pi_A : \epsilon(\varphi(A))$ then $\pi = \vee_{Il}\ \varphi(A)\ \varphi(B)\ \pi_A$.

- $\vee$-intro$_2$: If $F = A \vee B$ then by induction hypothesis, we have $\pi_B : \epsilon(\varphi(B))$ then $\pi = \vee_{Ir}\ \varphi(A)\ \varphi(B)\ \pi_B$.

- $\Rightarrow$-elim: If $F = B$ then by induction hypothesis, we have $\pi_{A\Rightarrow B} : \epsilon(\varphi(A \Rightarrow B))$ and $\pi_A : \epsilon(\varphi(A))$ then $\pi = \Rightarrow_E\ \varphi(A)\ \varphi(B)\ \pi_{A\Rightarrow B}\ \pi_A$.

- $\Rightarrow$-intro: If $F = A \Rightarrow B$ then by induction hypothesis, we have $\Gamma, a : \epsilon(\varphi(A)) \vdash \pi_B : \epsilon(\varphi(B))$ then by applying (abs) rule we get $\Gamma \vdash \lambda a : \epsilon(\varphi(A)), \pi_B : \epsilon(\varphi(A)) \to \epsilon(\varphi(B))$.
  Thus $\Gamma \vdash \Rightarrow_I \varphi(A)\varphi(B)(\lambda a : \epsilon(\varphi(A)), \pi_B) : \epsilon(\varphi(A) \Rightarrow \varphi(B))$.

- $\neg$-intro: If $F = \neg F'$ then by induction hypothesis, we have $\pi_{F'\bot} : \epsilon(\varphi(F \Rightarrow \bot))$ then $\pi = \Rightarrow_I\ \varphi(A)\ \bot\ (\lambda a, \pi_{A\bot})$.

- $\neg$-elim: If $F = \bot$ then by induction hypothesis, we have $\pi_A : \epsilon(\varphi(A))$ and $\pi_{\neg A} : \epsilon(\varphi(\neg A))$ then $\pi = \Rightarrow_E\ \varphi(A)\ \bot\ \pi_{\neg A}\ \pi_A$.

- $=$-elim: If $F = A[v]$ then by induction hypothesis, we have $\pi_= : \epsilon(\varphi(t) = \varphi(v))$ and $\pi_{At} : \epsilon(\varphi(A[t]))$ then $\pi = =_E\ \varphi(t)\varphi(v)\pi_=(\lambda x : \iota, \varphi(A[x]))\pi_{At}$.

- $=$-intro: If $F = (t = t)$ then $\pi = =_I\ \varphi(t)$.

- $\forall$-intro: If $F = \forall x A$ then by induction hypothesis, we have $x_1 : \iota, \ldots, x_n : \iota, x : \iota, h_1 : \varphi(F_1), \ldots, h_p : \varphi(F_p) \vdash \pi_A : \epsilon(\varphi(A))$ with $x \notin FV(h_1 : \varphi(F_1), \ldots h_p : \varphi(F_p))$ then by Lemma 1.3.9 we get $x_1 : \iota, \ldots, x_n : \iota, h_1 : \varphi(F_1), \ldots h_p : \varphi(F_p), x : \iota \vdash \pi_A : \epsilon(\varphi(A))$ then by applying (abs) rule we get $\Gamma \vdash \lambda x : \iota, \pi_A : \Pi x : \iota, \epsilon(\varphi(A))$ where

$\Gamma = x_1 : \iota, \ldots, x_n : \iota, h_1 : \varphi(F_1), \ldots h_p : \varphi(F_p)$. Thus $\Gamma \vdash \forall_I\ (\lambda x : \iota, \varphi(A))\ (\lambda x : \iota, \pi_A) : \epsilon(\forall(\lambda x : \iota, \varphi(A)))$.

- $\forall$-elim: If $F = [t/x]A$ then by induction hypotheis, we have $\pi_{\forall xA} : \epsilon(\forall(\lambda x : \iota, \varphi(A)))$ then $\pi = \forall_E\ (\lambda x : \iota, \varphi(A))\ \varphi(t)\ \pi_{\forall xA}$.

- $\exists$-elim: if $F = C$ then by induction hypothesis, we have $\Gamma \vdash \pi_{\exists xA} : \epsilon(\exists A)$ and $x : \iota, \Gamma, a : \epsilon(\varphi(A)) \vdash \pi_C : \epsilon(\varphi(C))$ with $x \notin FV(\Gamma; \varphi(C))$ then by 1.3.9 on all elements of Γ we get $\Gamma, x : \iota, a : \epsilon(\varphi(A)) \vdash \pi_C : \epsilon(\varphi(C))$ then we apply the rule (abs) twice to get $\Gamma \vdash \lambda x : \iota, \lambda a : \epsilon(\varphi(A)), \pi_C : \Pi x : \iota, \epsilon(\varphi(A)) \Rightarrow \epsilon(\varphi(C))$ thus $\Gamma \vdash \exists_E\ (\lambda x : \iota, \varphi(A))\ \pi_{\exists xA}\ \varphi(C)\ (\lambda x : \iota, \lambda a : \epsilon(\varphi(A)), \pi_C) : \epsilon(\varphi(C))$.

- $\exists$-intro: If $F = \exists xA$ then, by induction hypothesis,
  we have $\pi_{At} : \epsilon(\varphi([t/x]A))$ then $\pi = \exists_I\ (\lambda x : \iota, \varphi(A))\ \varphi(t)\ \pi_{At}$.

- *nnpp*: In order to have the classical version of this proof, we just need to add this case that uses the rule *nnpp* declared in 1.1. If $F = A$ then by induction hypothesis, we have $\Gamma \vdash \pi_{\neg\neg A} : \epsilon(\varphi(\neg\neg A))$ then $\pi = nnpp\ (\varphi(A))\ \pi_{\neg\neg A}$ where $nnpp : \Pi(P : Prop), \epsilon(\neg\neg P) \to \epsilon(P)$ is added to Σ_{ND}.

□

Having the correctness of the translation is necessary step but it is insufficient. For instance, we could define the translation of any formula in first-order logic to a single well-typed and inhabited term in $\lambda\Pi/\equiv$, which remains correct only in one direction. Thus, the completeness of this translation is required.

Proposition 1.4.4 (Completeness)
If there exists $\pi \in \Lambda$ such that $x_1 : \iota, \ldots, x_n : \iota, h_1 : \varphi(F_1), \ldots, h_p : \varphi(F_p) \vdash \pi : \varphi(F)$ is provable then $F_1, \ldots, F_p \vdash F$ is provable in Natural Deduction. [Dor11].

Remark 1.4.5
This result is proved in [Dor11] for intuitionistic first-order logic. To get the completeness for the classical first-order logic, the author showed a way to prove it by adding a constant for the excluded middle but without any formal proof.

A more shallow embedding of Natural Deduction

A more shallow embedding of Natural Deduction can be obtained by adding the following rules resulting in the signature Σ^{ϵ}_{ND}:

$$\begin{aligned} \epsilon(p \Rightarrow q) &\hookrightarrow \epsilon\ p \to \epsilon\ q \\ \epsilon(\forall\ p) &\hookrightarrow \Pi x, \epsilon(p\ x) \end{aligned}$$

Then, $\Rightarrow_E, \Rightarrow_I, \forall_I, \forall_E$ can be defined as follows:

$$\begin{aligned} \Rightarrow_E &\hookrightarrow \lambda p, \lambda q, \lambda \pi, \pi \\ \Rightarrow_I &\hookrightarrow \lambda p, \lambda q, \lambda \pi, \pi \\ \forall_E &\hookrightarrow \lambda t, \lambda A, \lambda \pi, \pi \\ \forall_I &\hookrightarrow \lambda x, \lambda A, \lambda \pi, \pi \end{aligned}$$

The resulting system Σ^{ϵ}_{ND} is orthogonal, hence confluent [KvOvR93]. It is easy to check that it also preserves typing (this can be automatically checked by LAMBDAPI, see Appendix 8.4). As for the termination, it follows from the criterion described in [Bla05]. The type *Prop* is a strictly-positive inductive type: every constructor has all its arguments accessible, hence smaller. p and q are smaller than $p \Rightarrow q$, and $(p\ x)$ is smaller than $\forall p$ since $\forall : (\iota \to Prop) \to Prop$ and *Prop* occurs only positively in $\iota \to Prop$.

We need the following definitions to ensure that the environment that we consider correspond to contexts in $\mathcal{FOL}$.

Definition 1.4.6 (ϵ-term)
An ϵ-term is a term that is convertible to a term of the form ϵF, where F is the translation of some $\mathcal{FOL}$-formula.

Definition 1.4.7 ($\mathcal{FOL}$-type)
A term is a $\mathcal{FOL}$-type if it is either $\iota \to \cdots \to \iota \to \iota$, or $\iota \to \cdots \to \iota \to Prop$, or an ϵ-term.

Definition 1.4.8 ($\mathcal{FOL}$-environment)
A $\mathcal{FOL}$-environment maps variables to $\mathcal{FOL}$-types.

An even more shallow embedding of Natural Deduction

We can have a complete shallow embedding by defining ϵ on the other connectives, following their impredicative encoding, resulting in the signature $\Sigma^{\epsilon,full}_{ND}$:

$$
\begin{aligned}
\epsilon(\bot) &\hookrightarrow \Pi r, \epsilon r \\
\epsilon(\top) &\hookrightarrow \Pi r, \epsilon(r \Rightarrow r) \\
\epsilon(p \Rightarrow q) &\hookrightarrow \epsilon\ p \to \epsilon\ q \\
\epsilon(p \vee q) &\hookrightarrow \Pi r, \epsilon((p \Rightarrow r) \Rightarrow (q \Rightarrow r) \Rightarrow r) \\
\epsilon(x = y) &\hookrightarrow \Pi r, \epsilon((r\ x) \Rightarrow (r\ y)) \\
\epsilon(p \wedge q) &\hookrightarrow \Pi r, \epsilon((p \Rightarrow q \Rightarrow r) \Rightarrow r) \\
\epsilon(\forall\ p) &\hookrightarrow \Pi x, \epsilon(p\ x) \\
\epsilon(\exists\ p) &\hookrightarrow \Pi r, \epsilon((\forall(\lambda x, px \Rightarrow r)) \Rightarrow r)
\end{aligned}
$$

With these rules, we can now define all the symbols of Σ_{ND}:

$$
\begin{aligned}
\bot_E &\hookrightarrow \lambda p, p \\
\top_I &\hookrightarrow \lambda p, \lambda \pi_p, \pi_p \\
\wedge_{El} &\hookrightarrow \lambda p, \lambda q, \lambda \pi_{p\wedge q}, \pi_{p\wedge q}\ p\ (\lambda x\_, x) \\
\wedge_{Er} &\hookrightarrow \lambda p, \lambda q, \lambda \pi_{p\wedge q}, \pi_{p\wedge q}\ q\ (\lambda x\_, x) \\
\wedge_I &\hookrightarrow \lambda \pi_p, \lambda \pi_q, \lambda x, \lambda \pi_{p\Rightarrow q\Rightarrow x}, \pi_{p\Rightarrow q\Rightarrow x}\ \pi_p\ \pi_q \\
\vee_E &\hookrightarrow \lambda \pi, \pi \\
\vee_{Il} &\hookrightarrow \lambda \pi_p, \lambda x, \lambda \pi_{p\Rightarrow x}, \lambda \pi_{q\Rightarrow x}, \pi_{p\Rightarrow x}\ \pi_p \\
\vee_{Ir} &\hookrightarrow \lambda \pi_p, \lambda x, \lambda \pi_{p\Rightarrow x}, \lambda \pi_{q\Rightarrow x}, \pi_{q\Rightarrow x}\ \pi_q \\
\neg_I &\hookrightarrow \lambda p, \pi, \pi \\
\neg_E &\hookrightarrow \lambda p, \pi, \pi \\
=_E &\hookrightarrow \Pi t : \iota, \Pi v : \iota, \epsilon(t = v) \to \Pi p : \iota \to Prop, \epsilon(p\ t) \to \epsilon(p\ v) \\
=_I &\hookrightarrow \lambda t, \lambda r, \pi, \pi \\
\exists_E &\hookrightarrow \lambda p, \lambda \pi_{\exists p}, \pi_{\exists p} \\
\exists_I &\hookrightarrow \lambda p, \lambda t, \lambda \pi_{p\ t}, \forall_I\ t\ \pi_{p\ t}
\end{aligned}
$$

The system $\Sigma_{ND}^{\epsilon,full}$ is still orthogonal, hence confluent.

The correctness of these definitions reduce to checking that they preserve typing, which can be automatically done by LAMBDAPI (see Appendix 8.5).

On the other hand, we know no termination criterion that can handle this system. We however, know that it terminates since it can be embed-

ded into the encoding of higher-order logic, which is proved terminating in [Dow17].

1.5 ZenonModulo

ZenonModulo [DDG+13] is an extension of the Automated Theorem Prover Zenon [BDD07], based on the tableaux method. *ZenonModulo* produces proofs in LAMBDAPI by using a system called LL Proofs. We will present here the LL Proofs system and its encoding in $\lambda\Pi/\equiv$.

1.5.1 LL Proofs

The LL Proofs system [CH15] is presented as a set of inference rules where every right side of a judgement is $\bot$.

$$\frac{}{\bot \vdash \bot}\, R_{\bot}$$

$$\frac{}{\neg\top \vdash \bot}\, R_{\neg\top}$$

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

$$\frac{}{\Gamma, \neg(t = t) \vdash \bot}\, R_{\neq}$$

$$\frac{}{\Gamma, t = u, \neg(u = t) \vdash \bot}\, R_{=}$$

$$\frac{\Gamma, P \vdash \bot \qquad \Gamma, \neg P \vdash \bot}{\Gamma \vdash \bot}\, R_{cut}$$

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

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

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

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

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

$$\frac{\Gamma, \neg(P \wedge Q), \neg P \vdash \bot \qquad \Gamma, \neg(P \wedge Q), \neq Q \vdash \bot}{\Gamma, \neg(P \wedge Q) \vdash \bot} R_{\neg\wedge}$$

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

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

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

$$\frac{\Gamma, P\ t \vdash \bot}{\Gamma, \exists P \vdash \bot} R_{\exists}$$

$$\frac{\Gamma, P\ t \vdash \bot}{\Gamma, \forall P \vdash \bot} R_{\forall}$$

$$\frac{\Gamma, \neg(P\ t) \vdash \bot}{\Gamma, \neg(\exists P) \vdash \bot} R_{\neg\exists}$$

$$\frac{\Gamma, \neg(P\ t) \vdash \bot}{\Gamma, \neg(\forall P) \vdash \bot} R_{\neg\forall}$$

$$\frac{\Gamma, \neg(t_1 = t_2) \vdash \bot \qquad \Gamma, P\ t_2 \vdash \bot}{\Gamma, P\ t_1 \vdash \bot} R_{subst}$$

1.5.2 Encoding of LL Proofs in $\lambda\Pi/\equiv$

In order to encode LL Proofs in $\lambda\Pi/\equiv$, we need to declare symbols to represent each inference rule of the LL Proof system. These symbols are declared as follows using the encoding of $\mathcal{FOL}$ defined in 1.4:
Let $\Sigma_{\mathcal{LL}}$ be the signature containing the encoding of LL Proofs in $\lambda\Pi$ Modulo Theory by using the signature $\Sigma_{\mathcal{FOL}}$ (the corresponding LAMBDAPI file is in Appendix 8.6).

$$
\begin{aligned}
\Sigma_{\mathcal{LL}} &= \Sigma_{\mathcal{FOL}} + \\
R_{\bot} &: \epsilon\bot \to \epsilon\bot, \\
R_{\neg\top} &: \epsilon(\neg\top) \to \epsilon\bot, \\
R_{ax} &: \Pi p : Prop, \epsilon p \to \epsilon(\neg p) \to \epsilon\bot, \\
R_{\neq} &: \Pi t : \iota, \epsilon(t \neq t) \to \epsilon\bot, \\
R_{=} &: \Pi t : \iota, \Pi u : \iota, \epsilon(t = u) \to \epsilon(u \neq t) \to \epsilon\bot, \\
R_{cut} &: \Pi p : Prop, (\epsilon p \to \epsilon\bot) \to (\epsilon(\neg p) \to \epsilon\bot) \to \epsilon\bot, \\
R_{\neg\neg} &: \Pi p : Prop, (\epsilon p \to \epsilon\bot) \to \epsilon(\neg\neg p) \to \epsilon\bot, \\
R_{\wedge} &: \Pi pq : Prop, (\epsilon p \to \epsilon q \to \epsilon\bot) \to \epsilon(p \wedge q) \to \epsilon\bot, \\
R_{\vee} &: \Pi pq : Prop, (\epsilon p \to \epsilon\bot) \to (\epsilon q \to \epsilon\bot) \to \epsilon(p \vee q) \to \epsilon\bot, \\
R_{\Rightarrow} &: \Pi p : Prop, \Pi q : Prop, (\epsilon(\neg p) \to \epsilon\bot) \to (\epsilon q \to \epsilon\bot) \to \epsilon(p \Rightarrow q) \to \epsilon\bot, \\
R_{\equiv} &\ \Pi p : Prop, \Pi q : Prop, (\epsilon(\neg p) \to \epsilon(\neg q) \to \epsilon\bot) \\
&\quad \to (\epsilon p \to \epsilon q \to \epsilon\bot) \to \epsilon(p \Leftrightarrow q) \to \epsilon\bot, \\
R_{\neg\wedge} &: \Pi p : Prop, \Pi q : Prop, (\epsilon(\neg p) \to \epsilon\bot) \to (\epsilon(\neg q) \to \epsilon\bot) \to \epsilon(\neg(p \wedge q)) \to \epsilon\bot, \\
R_{\neg\vee} &: \Pi p : Prop, \Pi q : Prop, (\epsilon(\neg p) \to \epsilon(\neg q) \to \epsilon\bot) \to \epsilon(\neg(\vee p\ q)) \to \epsilon\bot, \\
R_{\neg\Rightarrow} &: \Pi p : Prop, \Pi q : Prop, (\epsilon p \to \epsilon(\neg q) \to \epsilon\bot) \to \epsilon(\neg(p \Rightarrow q)) \to \epsilon\bot, \\
R_{\neg\equiv} &: \Pi p : Prop, \Pi q : Prop, (\epsilon(\neg p) \\
&\quad \to \epsilon q \to \epsilon\bot) \to (\epsilon p \to \epsilon(\neg q) \to \epsilon\bot) \to \epsilon(\neg(p \Leftrightarrow q)) \to \epsilon\bot, \\
R_{\exists} &: \Pi p : \iota \to Prop, (\Pi t : \iota, \epsilon(p\ t) \to \epsilon\bot) \to \epsilon(\exists p) \to \epsilon\bot, \\
R_{\forall} &: \Pi p : \iota \to Prop, \Pi t : \iota, (\epsilon(p\ t) \to \epsilon\bot) \to \epsilon(\forall p) \to \epsilon\bot, \\
R_{\neg\exists} &: \Pi p : \iota \to Prop, \Pi t : \iota, (\epsilon(\neg(p\ t)) \to \epsilon\bot) \to \epsilon(\neg(\exists p)) \to \epsilon\bot, \\
R_{\neg\forall} &: \Pi p : \iota \to Prop, (\Pi t : \iota, \epsilon(\neg(p\ t)) \to \epsilon\bot) \to \epsilon(\neg(\forall p)) \to \epsilon\bot, \\
R_{subst} &: \Pi p : \iota \to Prop, \Pi t : \iota, \Pi u : \iota, (\epsilon(t \neq u) \to \epsilon\bot) \to (\epsilon(p\ u) \to \epsilon\bot) \to \epsilon(p\ t) \to \epsilon\bot, \\
R_{conglr} &: \Pi p : \iota \to Prop, \Pi t : \iota, \Pi u : \iota, (\epsilon(p\ u) \to \epsilon\bot) \to \epsilon(p\ t) \to \epsilon(t = u) \to \epsilon\bot, \\
R_{congrl} &: \Pi p : \iota \to Prop, \Pi t : \iota, \Pi u : \iota, (\epsilon(p\ u) \to \epsilon\bot) \to \epsilon(p\ t) \to \epsilon(u = t) \to \epsilon\bot
\end{aligned}
$$

Before defining each symbol, we introduce here some auxiliary defini-

tions that are used in the definitions of LL Proof symbols: $\Sigma_{ND}^{\epsilon,aux} =$

$$
\begin{aligned}
\top_{intro} &: \epsilon\top \\
&:= \lambda p : Prop, \lambda x : \epsilon p, x \\
L_{contraposition} &: \Pi p : Prop, \Pi q : Prop, \epsilon(p \Rightarrow q) \to \epsilon((\neg q) \Rightarrow (\neg p)) \\
&:= \lambda h_1 : \epsilon(p \Rightarrow q), \lambda h_2 : \epsilon(\neg q), \lambda h_3 : \epsilon p, h_2\ (h_1\ h_3) \\
L_{\Leftrightarrow 1} &: \Pi p : Prop, \Pi q : Prop, \Pi h_5 : \epsilon q \to \epsilon p, \Pi h_2 : \epsilon p \to \epsilon(\neg q), \Pi h_q : \epsilon q, \epsilon(\neg q) \\
&:= h_2\ (h_5\ h_q) \\
L_{\Leftrightarrow 2} &: \Pi p : Prop, \Pi q : Prop, \Pi h_5 : \epsilon q \to \epsilon p, \Pi h_2 : \epsilon p \to \epsilon(\neg q), \epsilon(\neg q) \\
&:= \lambda h_q : \epsilon q, L_{\Leftrightarrow 1}\ p\ q\ h_5\ h_2\ h_q\ h_q \\
L_{\Leftrightarrow 3} &: \Pi p : Prop, \Pi q : Prop, \Pi h_5 : \epsilon q \to \epsilon p, \Pi h_2 : \epsilon p \to \epsilon(\neg q), \Pi h_4 : \epsilon p \to \epsilon q, \epsilon(\neg p) \\
&:= L_{contraposition}\ p\ q\ h_4\ (L_{\Leftrightarrow 2}\ p\ q\ h_5\ h_2) \\
L_{\Leftrightarrow 4} &: \Pi p : Prop, \Pi q : Prop, \Pi h_5 : \epsilon q \to \epsilon p, \Pi h_2 : \epsilon p \to \epsilon(\neg q), \Pi h_4 : \epsilon p \to \epsilon q, \\
&\quad \Pi h_1 : \epsilon(\neg p) \to \epsilon(\neg\neg q), \epsilon(\neg\neg q) \\
&:= h_1(L_{\Leftrightarrow 3}\ p\ q\ h_5\ h_2\ h_4) \\
L_{\neg\vee 1} &: \Pi p : Prop, \Pi q : Prop, \epsilon p \to \epsilon(p \vee q) \\
&:= \lambda h_1 : \epsilon p, \lambda z : Prop, \lambda h_2 : \epsilon p \to \epsilon z, \lambda h_3 : \epsilon q \to \epsilon z, h_2\ h_1 \\
L_{\neg\vee 2} &: \Pi p : Prop, \Pi q : Prop, \epsilon q \to \epsilon(p \vee q) \\
&:= \lambda h_1 : \epsilon q, \lambda z : Prop, \lambda h_2 : \epsilon p \to \epsilon z, \lambda h_3 : (\epsilon q \to \epsilon z), h_3\ h_1 \\
L_{\neg\vee 3} &: \Pi p : Prop, \Pi q : Prop, \Pi h_2 : \epsilon(\neg(p \vee q)), \epsilon(\neg p) \\
&:= L_{contraposition}\ p(p \vee q)(L_{\neg\vee 1}\ p\ q)\ h_2 \\
L_{\neg\vee 4} &: \Pi p : Prop, \Pi q : Prop, \Pi h_2 : \epsilon(\neg(p \vee q)), \epsilon(\neg q) \\
&:= L_{contraposition}\ q(p \vee q)(L_{\neg\vee 2}\ p\ q)\ h_2 \\
L_{\neg\Rightarrow 1} &: \Pi p : Prop, \Pi q : Prop, \Pi h_2 : \epsilon(\neg(p \Rightarrow q)), \epsilon(\neg q) \\
&:= \lambda h_3 : \epsilon q, h_2(\lambda(h_4 : \epsilon p), h_3) \\
L_{\neg\Rightarrow 2} &: \Pi p : Prop, \Pi q : Prop, \Pi h_1 : \epsilon p \to \epsilon(\neg\neg q), \Pi h_2 : \epsilon(\neg(p \Rightarrow q)), \epsilon(\neg p) \\
&:= \lambda h_3 : \epsilon p, (h_1\ h_3)(L_{\neg\Rightarrow 1}\ p\ q\ h_2) \\
L_{\neg\Rightarrow 3} &: \Pi p : Prop, \Pi q : Prop, \Pi h_3 : \epsilon(\neg p), \epsilon(p \Rightarrow q) \\
&:= \lambda h_4 : \epsilon p, \bot_E\ (h_3\ h_4)\ q \\
L_{\neg\Leftrightarrow 1} &: \Pi p : Prop, \Pi q : Prop, \Pi h_2 : \epsilon p \to \epsilon(\neg\neg q), \Pi h_3 : \epsilon(\neg(p \Leftrightarrow q)), \epsilon(\neg p) \\
&:= \lambda h_p : \epsilon p, h_2 h_p(\lambda h_q : \epsilon q, h_3 \\
&\quad (\lambda z : Prop, \lambda h_4 : (\epsilon(p \Rightarrow q) \to \epsilon(q \Rightarrow p) \to \epsilon z), \\
&\quad h_4(\lambda x : \epsilon p, h_q)(\lambda x : \epsilon q, h_p)))
\end{aligned}
$$

We define now LL proofs symbols by $\lambda\Pi/\equiv$ terms as follows:

$\Sigma_{ND}^{\mathcal{LL}} = \Sigma_{LL} + \Sigma_{ND}^{\epsilon,full} + \Sigma_{ND}^{\epsilon,aux} +$

$R_{\bot} \hookrightarrow \lambda x : \epsilon\bot, x$

$R_{\neg\top} \hookrightarrow \lambda h : \epsilon(\neg\top), h\ \top_{intro}$

$R_{ax} \hookrightarrow \lambda p, \lambda h : \epsilon p, \lambda\pi_{\neg p} : \epsilon(\neg p), \pi_{\neg p}\ h$

$R_{\neq} \hookrightarrow \lambda t, \lambda h_1 : \epsilon(t \neq t), h_1(=_I\ (\lambda z : \iota \to Prop, (\lambda h_2 : \epsilon(z\ t), h_2)))$

$R_{=} \hookrightarrow \lambda t, \lambda u, \lambda h_1 : \epsilon(t = u), \lambda h_2 : \epsilon(u \neq t), h_2(=_I\ (\lambda z : \iota \to Prop, \lambda h_3 : \epsilon(z\ u),$
$=_E\ h_1(\lambda x : \iota, ((z\ x) \Rightarrow (z\ t)))(\lambda h_4 : \epsilon(z\ t), h_4)h_3))$

$R_{cut} \hookrightarrow \lambda p, \lambda h_1 : \epsilon p \to \epsilon\bot, \lambda h_2 : \epsilon(\neg p) \to \epsilon\bot, h_2\ h_1$

$R_{\neg\neg} \hookrightarrow \lambda p, \lambda h_1 : \epsilon p \to \epsilon\bot, \lambda h_2 : \epsilon(\neg\neg p), h_2\ h_1$

$R_{\wedge} \hookrightarrow \lambda p, \lambda q, \lambda h_1 : \epsilon p \to \epsilon q \to \epsilon\bot, \lambda h_2 : \epsilon(p \wedge q), h_1(\wedge_{El} p q h_2)(\wedge_{Er}\ p\ q\ h_2)$

$R_{\vee} \hookrightarrow \lambda p, \lambda q, \lambda h_1 : \epsilon p \to \epsilon\bot, \lambda h_2 : \epsilon q \to \epsilon\bot, \lambda h_3 : \epsilon(p \vee q), \vee_E h_3 \bot h_1\ h_2$

$R_{\Rightarrow} \hookrightarrow \lambda p, \lambda q, \lambda h_1 : \epsilon(\neg p) \to \epsilon\bot, \lambda h_2 : \epsilon q \to \epsilon\bot, \lambda h_3 : \epsilon(p \Rightarrow q), h_1(L_{contraposition}\ p\ q\ h_3\ h_2)$

$R_{\Leftrightarrow} \hookrightarrow \lambda p, \lambda q, \lambda h_1 : \epsilon(\neg p) \to \epsilon(\neg q) \to \epsilon\bot, \lambda h_2 : \epsilon p \to \epsilon q \to \epsilon\bot, \lambda h_3 : \epsilon(p \Leftrightarrow q), L_{\Leftrightarrow 4}$
$\lambda p, \lambda q, (\wedge_{Er}(p \Rightarrow q)(q \Rightarrow p)h_3)h_2(\wedge_{El}(p \Rightarrow q)(q \Rightarrow p)h_3)h_1(L_{\Leftrightarrow 2}\ p\ q(\wedge_{Er}$
$(p \Rightarrow q)(q \Rightarrow p)h_3)h_2)$

$R_{\neg\wedge} \hookrightarrow \lambda p, \lambda q, \lambda h_1 : \epsilon(\neg p) \to \epsilon\bot, \lambda h_2 : \epsilon(\neg q) \to \epsilon\bot, \lambda h_3 : \epsilon(\neg(p \wedge q)), h_1(\lambda h_5 : \epsilon p,$
$h_2(\lambda h_6 : \epsilon q, h_3(\wedge_I\ h_5\ h_6)))$

$R_{\neg\vee} \hookrightarrow \lambda p, \lambda q, \lambda h_1 : \epsilon(\neg p) \to \epsilon(\neg q) \to \epsilon\bot, \lambda h_2 : \epsilon(\neg(p \vee q)), h_1(L_{\neg\vee 3}\ p\ q\ h_2)(L_{\neg\vee 4}\ p\ q\ h_2)$

$R_{\neg\Rightarrow} \hookrightarrow \lambda p, \lambda q, \lambda h_1 : \epsilon p \to \epsilon(\neg q) \to \epsilon\bot, \lambda h_2 : \epsilon(\neg(p \Rightarrow q)), h_2(\lambda h_3 : \epsilon p,$
$\bot_E((h_1\ h_3)(\lambda h_4 : \epsilon q, h_2(\lambda h_5 : \epsilon p, h_4)))q)$

$R_{\neg\Leftrightarrow} \hookrightarrow \lambda p, \lambda q, \lambda h_1 : \epsilon(\neg p) \to \epsilon(\neg q), \lambda h_2 : \epsilon p \to \epsilon(\neg(\neg q)), \lambda h_3 : \epsilon(\neg(p \Leftrightarrow q)),$
$(\lambda h_{\neg p} : \epsilon(\neg p), h_3(\wedge_I(\lambda h_P : \epsilon p, \bot_E(h_{\neg p} h_P)q)$
$(\lambda h_q : \epsilon q, \bot_E(h_1\ h_{\neg p}\ h_q)p)))(L_{\neg\Leftrightarrow 1}\ p\ q\ h_2\ h_3)$

$R_{\exists} \hookrightarrow \lambda p, \lambda h_1 : \Pi t : \iota, \epsilon(p\ t) \to \epsilon\bot, \lambda h_2 : \epsilon(\exists p), \exists_E\ p\ h_2 \perp h_1$

$R_{\forall} \hookrightarrow \lambda p, \lambda t, \lambda h_1 : \epsilon(pt) \to \epsilon\bot, \lambda h_2 : \epsilon(\forall p), h_1(h_2\ t)$

$R_{\neg\exists} \hookrightarrow \lambda p, \lambda t, \lambda h_1 : \epsilon(\neg(p\ t)) \to \epsilon\bot, \lambda h_2 : \epsilon(\neg(\exists p)), h_1(\lambda h_4 : \epsilon(p\ t), h_2(\exists_I\ p\ t\ h_4))$

$R_{\neg\forall} \hookrightarrow \lambda p, \lambda h_1 : \Pi(t : \iota), \epsilon(\neg(p\ t)) \to \epsilon\bot, \lambda h_2 : \epsilon(\neg(\forall p)), h_2(\lambda t : \iota, nnpp(p\ t)(h_1\ t))$

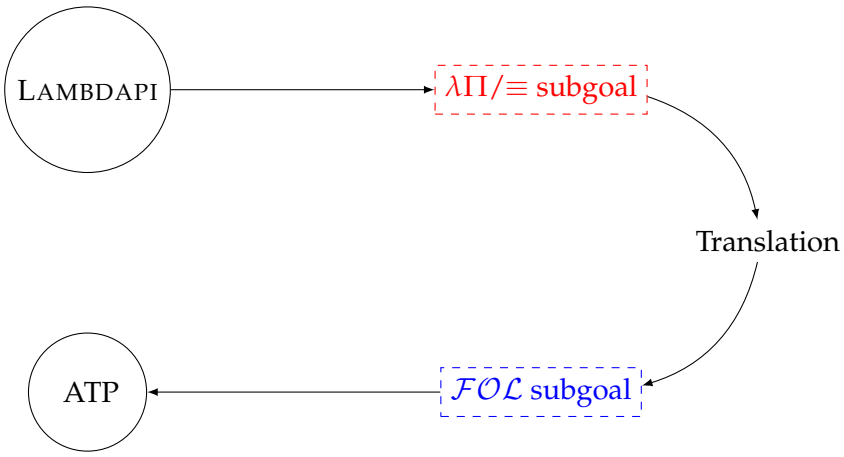
$R_{subst} \hookrightarrow \lambda p, \lambda t_1, \lambda t_2, \lambda h_1 : \epsilon(t_1 \neq t_2) \to \epsilon\bot, \lambda h_2 : (\epsilon(p\ t_2) \to \epsilon\bot), \lambda h_3 : \epsilon(p\ t_1),$
$h_1(\lambda h_4 : \epsilon(t_1 = t_2), h_2(=_E\ t_1\ t_2\ h_4\ p\ h_3))$

Again, the correctness of these definitions reduces to the type preservation property of the above rewrite rules, which can be automatically checked by LAMBDAPI (see Appendix 8.8).

Chapter 2

Calling provers

Proof assistants usually call automated theorem provers to prove goals automatically. In this chapter, we will show a way to call ATPs from LAMBDAPI. In order to call an ATP from LAMBDAPI, we need to translate the current goal from the logic used in LAMBDAPI, which is $\lambda\Pi/\equiv$, to the logic of the ATP, which is here $\mathcal{FOL}$.



2.1 Propositional logic

Let S be a signature extending $\Sigma_{\mathcal{FOL}}$. Let Γ be a typing environment. Let $\Gamma_1 = x_1 : \iota, \ldots, x_m : \iota$ be the environment made of the declarations of Γ of the form $x : \iota$. Let Γ_2 be an environment made of the declarations of Γ of the form $h : \epsilon(t)$.

Let ψ be the function that translates back $\lambda\Pi/\equiv$ terms to $\mathcal{FOL}$ propositions defined as follows:

$$\begin{aligned}
\psi(A \vee B) &= \psi(A) \vee \psi(B) \\
\psi(A \wedge B) &= \psi(A) \wedge \psi(B) \\
\psi(A \Rightarrow B) &= \psi(A) \Rightarrow \psi(B) \\
\psi(A \Leftrightarrow B) &= \psi(A) \Leftrightarrow \psi(B) \\
\psi(\neg A) &= \neg\psi(A) \\
\psi(\top) &= \top \\
\psi(\bot) &= \bot \\
\psi(u) &= P_u\ x_1 \ldots\ x_m \text{ otherwise}
\end{aligned}$$

where P_u is a new predicate symbol abstracting the term u when u is not a $\mathcal{FOL}$ proposition. So, ψ is in some sense the inverse of the function φ encoding $\mathcal{FOL}$ propositions into $\lambda\Pi/\equiv$ (cf. section 1.4).

Suppose $\Gamma \vdash a : Prop$. The formula $\psi(a)$ is a $\mathcal{FOL}$ proposition whose free variables are declared in Γ_1. Let S' be the extension of S with the symbols P_u that appear in $\psi(a)$ and the rewriting rules

$$P_u\ x_1 \ldots\ x_m \hookrightarrow u.$$

For all terms v in the signature S', we have

$$\varphi(\psi(v)) \equiv v,$$

by a simple induction on v.

Let now Θ be the function on typing environments defined as follows:

$$\begin{aligned}
\Theta([]) &= [] \\
\Theta(x : \epsilon(v), \Gamma) &= x : \epsilon(\varphi(\psi(v))), \Theta(\Gamma)
\end{aligned}$$

So, for all typing environment Δ in the signature S', we have $\Theta(\Delta) \equiv \Delta$.

Finally, we extend the function ψ on typing environments as follows:

$$\begin{aligned}
\psi([]) &= [] \\
\psi(x : \epsilon(v), \Gamma) &= \psi(v), \psi(\Gamma)
\end{aligned}$$

Theorem 2.1.1
Let S be a left-linear confluent extension of $\Sigma_{\mathcal{FOL}}$. If $\psi(\Gamma_2) \vdash_{ND} \psi(a)$ then there is t in $\lambda\Pi/\equiv$ such that $\Gamma \vdash_S t : \epsilon(a)$

Proof. Let Γ_1' be the sub-environment of Γ_1 containing the free variables of $\psi(\Gamma_2)$ and $\psi(a)$. By Proposition 1.4.3, there is a term t such as $\Gamma_1', \Theta(\Gamma_2) \vdash_{S'}$

$t : \epsilon(\varphi(\psi(a)))$. Since $\varphi(\psi(v)) \equiv v$ and $\Theta(\Delta) \equiv \Delta$, by conversion, $\Gamma_1', \Gamma_2 \vdash_{S'} t : \epsilon(a)$. By replacing in t all terms $P_u\ x_1 \dots\ x_m$ by their definition u (this is confluent, terminating and type-preserving), we get a term t' such that $\Gamma_1', \Gamma_2 \vdash_{S'} t' : \epsilon(a)$. Then, by [BDG+21, Theorem 7], there is D' such that $\epsilon(a) \hookrightarrow^* D'$ and $\Gamma_1', \Gamma_2 \vdash_S t : D'$ because:

- S' is confluent: it is the union of two left-linear confluent systems with no critical pairs between them [vO94]
- S is a fragment of S' preserving typing:
  - all symbols and rewriting rules of S are in S',
  - if c is a symbol of S then the type of c only contains symbols of S,
  - if $l \hookrightarrow r$ is a rule of S' whose LHS only contains symbols of S then r only contains symbols of S and $l \hookrightarrow r \in S$.
- the symbols of Γ_1', Γ_2 and t are in S.

Since Γ_1' is a sub-environment of Γ_1 and Γ_1 is a sub-environment of Γ, by weakening, we have $\Gamma \vdash_S t : D'$. Since $\epsilon(a)$ is in S, the reduction $\epsilon(a) \hookrightarrow^* D'$ is in S too. Therefore, by conversion, $\Gamma \vdash_S t : \epsilon(a)$. □

2.2 First-order logic

To extend the previous result to first-order logic, we need to take care of additional bound variables introduced by quantifiers. We therefore need ψ to take the typing environment as argument. For abstracting a term u of type $Prop$ that is not a $\mathcal{FOL}$ proposition, we introduced a new constant P_u applied to the variables of the typing environment (that was fixed before and is changing now) that reduces to u. Similarly, for handling a term u of type ι that is not a $\mathcal{FOL}$ term, we also need to introduce a new constant c_u that reduces to u.

We therefore extend ψ as follows:

$$
\begin{aligned}
\psi(\Gamma; A \vee B) &= \psi(\Gamma; A) \vee \psi(\Gamma; B) \\
\psi(\Gamma; A \wedge B) &= \psi(\Gamma; A) \wedge \psi(\Gamma; B) \\
\psi(\Gamma; A \Rightarrow B) &= \psi(\Gamma; A) \Rightarrow \psi(\Gamma; B) \\
\psi(\Gamma; A \Leftrightarrow B) &= \psi(\Gamma; A) \Leftrightarrow \psi(\Gamma; B) \\
\psi(\Gamma; \neg A) &= \neg\psi(\Gamma; A) \\
\psi(\Gamma; \top) &= \top \\
\psi(\Gamma; \bot) &= \bot \\
\psi(\Gamma, \forall(\lambda x : \iota, A)) &= \forall x, \psi(\Gamma, x : \iota; A) \\
\psi(\Gamma; \exists(\lambda x : \iota, A)) &= \exists x, \psi(\Gamma, x : \iota; A) \\
\psi(\Gamma; P\ t_1 \dots\ t_n) &= P\ \psi_\iota(\Gamma; t_1) \dots\ \psi_\iota(\Gamma, t_n) \\
\psi(\Gamma; t_1 = t_2) &= \psi_\iota(\Gamma, t_1) = \psi_\iota(\Gamma, t_2) \\
\psi(x_1 : \iota, \dots, x_m : \iota; u) &= P_u\ x_1 \dots\ x_m \text{ otherwise}
\end{aligned}
$$

$$
\begin{aligned}
\psi_\iota(x) &= x \\
\psi_\iota(f\ t_1 \dots\ t_n) &= f\ \psi_\iota(t_1) \dots\ \psi_\iota(t_n) \\
\psi_\iota(x_1 : \iota, \dots, x_m : \iota; u) &= c_u\ x_1 \dots\ x_m \text{ otherwise}
\end{aligned}
$$

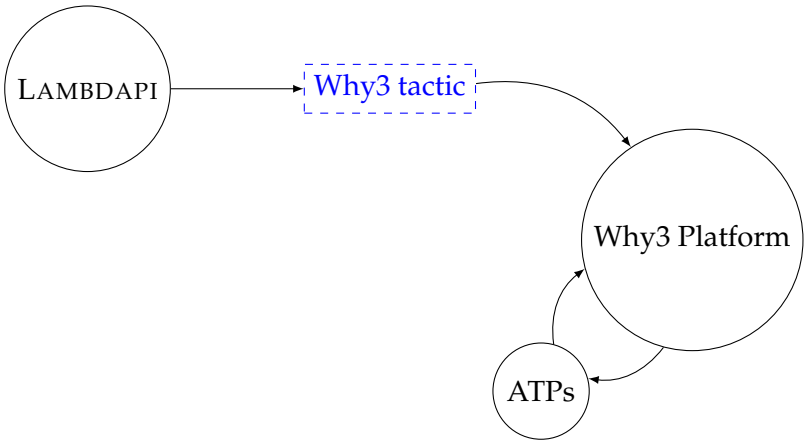
2.3 Implementation

2.3.1 Tactics in proof assistants

In order to simplify and automate some tasks, proof assistants can use tactics. Tactics are a mechanism built into proof assistants to prove goals or subgoals. They change the state of a proof to another one by solving the current goal or by generating new subgoals.

2.3.2 Why3 tactic in LAMBDAPI

The translation presented in 2.1 is implemented as a tactic in the proof assistant LAMBDAPI. It accepts the encoding presented in 1.4, which is the encoding of first-order logic in $\lambda\Pi/\equiv$. The implemented tactic allows us to call an external prover using the Why3 platform [BFMP11]. It translates the current goal from $\lambda\Pi/\equiv$ to first-order logic and sends the translated goal to the prover specified as argument. If the prover cannot find a proof for the selected goal, then the tactic fails. If the prover solves the goal, we declare the solved goal as an axiom and apply that axiom to discharge the goal. Instead of adding the current goal as an axiom, we could try to prove it by using the the prover output.



Example 2.3.1
As shown in 2.3.2, the tactic depends on the encoding of first-order logic which is included in this file:

```
require open logic.fol;
```

We specify then which external prover[1] *to use and set its timeout:*

```
prover "Alt-Ergo"; // set the prover used to Alt-Ergo
prover_timeout 2; // set prover's timeout to 2 seconds
```

Let $F = a \wedge b \Rightarrow a$ *be a first-order formula. The translation of this formula in* $\lambda\Pi/\equiv$ *is* $\epsilon(a \wedge b \Rightarrow a)$

```
opaque symbol example1 a b : ϵ (a ∧ b ⇒ a) :=
begin
  assume a b; // first we move a and b to the context
  why3; // we call the tactic to find a proof
end;
```

If the prover succeeds to find a proof, then an axiom of type $\Pi a : Prop, \Pi b : Prop, \epsilon(a \wedge b \Rightarrow a)$ *is added to the current signature and applied to the current goal to solve it.*

Example 2.3.2
Let $T = \Pi\, d : (\iota \to \iota) \to \iota, \Pi\, h : Prop \to \iota, \Pi\, s : \iota \to Prop, \epsilon(s\ (h\ \bot) \Rightarrow\ d\ (\lambda\ x,\ x) = d\ (\lambda\ y,\ y))$ *be a term in* $\lambda\Pi/\equiv$*. The translation of this term in first-order logic is* $s\ T_1 \Rightarrow\ T_2 = T_2$ *since* $d\ (\lambda\ x,\ x)$ *and* $d\ (\lambda\ y,\ y)$ *are abstracted to the same term* T_2*.*

[1] The prover needs to be installed in the machine and configured by Why3

```
require open logic.fol;
opaque symbol example2 :
    Π d : (ι → ι) → ι, Π h : Prop → ι, Π s : ι → Prop,
        ϵ(s (h ⊥) ⇒ d (λ x, x) = d (λ y, y)) ≔
    begin
        assume d h s;
        why3 "eprover";
    end;
```

The corresponding Why3 code that was generated by the tactic is:

```
theory Task
    (* The type of terms *)
    type iota
    (* The term that abstracts d (λ x, x) *)
    constant a : iota
    (* The predicate that abstracts h ⊥ *)
    predicate P
    (* The final formula sent to the prover *)
    goal main_goal : P -> a = a
end
```

Remark 2.3.3
In order to use the why3 tactic, we need to provide the encoding used by maping each symbol of our encoding with the one used inside LAMBDAPI. LAMBDAPI *offers a way to do this by using the* ***builtin*** *command. By using this feature, we let the user chooses his own encoding of First-order logic.*

```
    builtin "P"   ≔ ϵ;
    builtin "bot" ≔ ⊥;
    builtin "top" ≔ ⊤;
    builtin "imp" ≔ ⇒;
    builtin "not" ≔ ¬;
    builtin "and" ≔ ∧;
    builtin "or"  ≔ ∨;
    builtin "T"   ≔ τ; // used for polymorphism.
```

2.4 Conclusion

In this chapter, we presented a translation from $\lambda\Pi/\equiv$ to First-order logic and its correctness. The translation is implemented inside LAMBDAPI as a tactic. This tactic assigns the translated formula to the Why3 platform to

find a proof. The advantage of this tactic to LAMBDAPI is to have more automation; hence, to gain of time. For now, this tactic does not return a proof, we will present in the next chapter a way to use the prover's output to reconstruct the found proof in LAMBDAPI.

Chapter 3

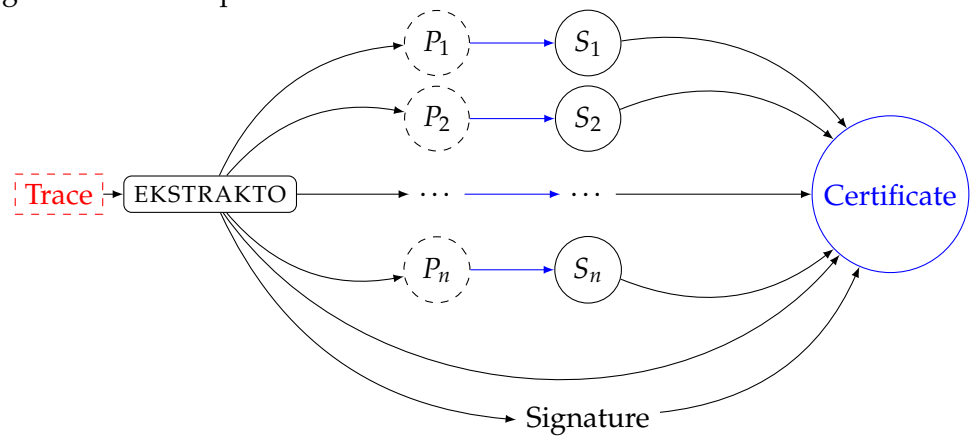
Proof reconstruction

In order to discharge more burden from users of interactive theorem provers, and thus to widen the use of these tools, it is crucial to automate them more. To achieve this goal, in the process of checking the validity of formulas, proof assistants could use an external theorem prover to automate their tasks and obtain a proof of a specific formula. Once a proof is found, the proof assistant applies this proof on the current goal and tells the user that all is done in the background. However, this can work only if the prover builds a complete proof that is easily checkable by the proof assistant. We distinguish two families of automated theorem provers: some provers, like *ZenonModulo* [DDG+13] and *ArchSAT* [BCD18], output complete proofs but are not very efficient at finding proofs; others, like **E** [Sch13] and *ZipperPosition* [Cru15], are more powerful but return only proof traces, i.e. proofs with less details. Our goal is to transform proof traces into complete proofs. In this chapter, we present EKSTRAKTO, a tool that translates a TSTP proof trace for a CNF problem into a complete LAMBDAPI proof. We explain how proof reconstruction works and give some experiments to show the success of this approach.

3.1 Architecture

In this section, we explain in details how EKSTRAKTO works. In order to produce a LAMBDAPI proof from a TSTP file, EKSTRAKTO extracts a TPTP problem for each formula declaration containing at least one inference, and calls *ZenonModulo* (or any other automated prover producing LAMBDAPI proofs, see discussion below) on each generated problem to get a LAMBDAPI proof for this problem. If the external prover succeeds to find a proof

of all the generated problems, then we combine those proofs in another file to get a LAMBDAPI proof of the whole TSTP file.



3.1.1 Extracting TPTP problems

To extract a TPTP problem from a trace step, we need to find the premises used in it. We define the function $\mathcal{P}$, which takes a TSTP source as input and returns the set of premises used by the prover:

$$\mathcal{P}(name) = \{name\}$$

$$\mathcal{P}([s_0, s_1, \ldots, s_n]) = \bigcup_{i=0}^{n} \mathcal{P}(s_i)$$

$$\mathcal{P}(inference(name, infos, [s_0, s_1, \ldots, s_n])) = \bigcup_{i=0}^{n} \mathcal{P}(s_i)$$

Note that if we have an inference t inside another one, say s, we will repeat the process for each sub-inference and omit s from the set of premises, i.e., if we represent an inference step by a proof tree, we take only the leaves of this tree as premises.

We omit all information that is not needed (*status*, *name*, ...). In particular, we do not consider the inference name field. Even if it could be used to fine-tune the problem, we prefer to ignore it in order to remain generic since the names are specific to the prover that produced the trace. Hence, we have:

$$\mathcal{P}(inference([inference([inference([c\_3, c\_6]), c\_4])])) = \{c\_3, c\_6, c\_4\}$$

After getting all the premises used for proving `Form`($name$), say $name_0$, ..., $name_k$, we generate the following TPTP problem:

$$\texttt{Form}(name_0) \Rightarrow \ldots \Rightarrow \texttt{Form}(name_k) \Rightarrow \texttt{Form}(name)$$

Note that the generated TPTP problem is a FOF formula. The reason of this choice is to keep the same formula when we combine the sub-proofs. If we generated a CNF problem, then we would need to negate the goal and it would be more complex to reconstruct the proof.

Since we are using FOF formulas in sub-problems that are obtained from a CNF trace, we need to quantify over each free variable to get a closed formula.

In our example, there are 3 steps (colored in blue in the file `SET001-1.p` above). EKSTRAKTO will generate the following 3 first-order formulas:

`Form(c_0)` $\Rightarrow$ `Form(c_1)` $\Rightarrow$ `Form(c_5)`

`Form(c_2)` $\Rightarrow$ `Form(c_5)` $\Rightarrow$ `Form(c_6)`

`Form(c_3)` $\Rightarrow$ `Form(c_6)` $\Rightarrow$ `Form(c_4)` $\Rightarrow$ `Form(c_7)`

Each formula will be written in a separate TPTP file as follows:

c_5.p

```
fof(c_5, conjecture, (
        (![X1, X2] : (s (X1, X2)|~equal_sets (X1, X2)))
    =>  ((equal_sets (b, bb))
    =>  (subset (b, bb))))).
```

c_6.p

```
fof(c_6, conjecture,(
        (![X1, X2, X3] : (member (X1, X3)|~member (X1, X2)
                          |~subset (X2, X3)))
    =>  ((subset (b, bb))
    =>  (![X1] : (member (X1, bb)|~member (X1, b)))))).
```

c_7.p

```
fof(c_7, conjecture, (
        (~member (element_of_b, bb))
    =>  ((![X1] : (member (X1, bb)|~member (X1, b)))
    =>  ((member (element_of_b, b))
    =>  ($false))))).
```

3.1.2 Proof reconstruction

If the automated theorem prover succeeds to solve all the generated TPTP problems, then we can reconstruct a proof in LAMBDAPI directly by using the proof tree of the trace that we are trying to certify and all the proofs of the sub-problems. The proof term of each sub-problem is irrelevant since it has the right type.

The global proof is reconstructed from each sub-proof. We just need to apply each proof term of a sub-proof to its premises by following the proof tree of the TSTP file. Indeed, the type of the sub-proof of `Form`$(name)$ using premises $name_0, \ldots, name_k$ is $\epsilon(\varphi(Form(name_0)) \to \ldots \varphi(Form(name_k)) \to \varphi(Form(name)))$, which is convertible to $\epsilon(\varphi(Form(name_0))) \to \quad \ldots \quad \to \epsilon(\varphi(Form(name_k))) \to \epsilon(\varphi(Form(name)))$, thanks to the rule $\epsilon(A \Rightarrow B) \hookrightarrow \epsilon A \to \epsilon B$,

Hence, the proof term of a sub-problem is a function whose arguments are proofs of the premises and which returns a proof of its conclusion. Since we are handling only CNF formulas, the proof that we want to reconstruct at the end is always a proof of $\bot$. Before applying those proof terms, we need to declare our hypotheses. With our example file, we get:

proof_SET001-1.lp

```
// Axioms
symbol hyp_c_0 : ϵ(φ(Form(c_0)));
symbol hyp_c_1 : ϵ(φ(Form(c_1)));
symbol hyp_c_2 : ϵ(φ(Form(c_2)));
symbol hyp_c_3 : ϵ(φ(Form(c_3)));
symbol hyp_c_4 : ϵ(φ(Form(c_4)));

// Lemmas
opaque symbol lemma_c_5 ≔ c_5.delta hyp_c_0 hyp_c_1;
opaque symbol lemma_c_6 ≔ c_6.delta hyp_c_2 lemma_c_5;
opaque symbol lemma_c_7 ≔ c_7.delta hyp_c_3 lemma_c_6
                                    hyp_c_4;

// Proof
opaque symbol proof_trace : ϵ⊥
    ≔ lemma_c_7;
```

where `delta` is the name of the proof term in each file.

Remark 3.1.1
For each TSTP file, we generate a LAMBDAPI *file defining its signature by declaring a* LAMBDAPI *symbol f for each function symbol f of the TSTP file:*

SET001-1.lp

```
symbol element_of_b : ι;
symbol subset       : ι → ι → Prop;
symbol b            : ι;
symbol member       : ι → ι → Prop;
symbol bb           : ι;
symbol equal_sets   : ι → ι → Prop;
```

All this has been implemented in a tool called EKSTRAKTO[1], consisting of 2,000 lines of OCaml.

Remark 3.1.2
The version of the tool presented in [HBB19] uses the `let in` *mechanism that* LAMBDAPI *offers (see the example file* `proof_SET001-1.lp` *in [HBB19]). In every step, the definition of the lemma is unfolded by* LAMBDAPI *although it is not necessary. In order to get more efficiency and gain time for type-checking, we replaced those* `let`*'s by* `opaque symbol` *definitions that cannot be unfolded.*

3.2 Experiments

We chosed two well-known first-order provers, **E** [Sch13] and VAMPIRE [KV13], to generate TSTP files. We run **E** (version 2.5) and VAMPIRE (version 4.5.1) on the set of CNF problems of the TPTP library v7.4.0 (8118 files) with 2GB of memory space and a timeout of 5 minutes. The time and memory limits used here are those of the CASC competition until recently. We obtained 4760 TSTP files for **E** and 2449 TSTP files for VAMPIRE.

On these TSTP files, EKSTRAKTO generated 379412 TPTP files for **E** and 127430 TPTP files for VAMPIRE. Note that EKSTRAKTO could not handle 1689 VAMPIRE TSTP files because they are using definitions introduced by VAMPIRE. A solution is proposed in Section 7.3 to fix this problem, which should therefore increase these percentages further.

We then used two other provers to generate LAMBDAPI proofs, *ZenonModulo* [DDG+13] and *ArchSAT* [BCD18]. The timeout used for *ZenonModulo* and *ArchSAT* is only 10 seconds since these provers are called on lemmas that are supposed to be simple to prove. *ZenonModulo* generated a LAMBDAPI proof for 87% of **E** TPTP files and 60% of VAMPIRE TPTP files. *ArchSAT* generated 92% for **E** TPTP files and 83% for VAMPIRE TPTP files. The union of both produced 95% LAMBDAPI proofs for **E** TPTP files and 83% LAMBDAPI proofs for VAMPIRE TPTP files:

[1] `https://github.com/elhaddadyacine/ekstrakto`

Table 3.1: Percentage of LAMBDAPI proofs on the extracted TPTP files

Prover	% **E**	% VAMPIRE
ZenonModulo	87%	60%
ArchSAT	92%	81%
ZenonModulo ∪ *ArchSAT*	95%	85%

However, as it suffices that no LAMBDAPI proof is found for only one TPTP file for getting no global proof, EKSTRAKTO can generate a complete proof for only 45% of **E** TSTP files by using *ZenonModulo* only, 56% by using *ArchSAT* only, but 62% by using both provers. As for VAMPIRE files, EKSTRAKTO can generate a complete proof for 54% of VAMPIRE TSTP files using *ZenonModulo* only, 74% using *ArchSAT* only, but 83% by using both provers:

Table 3.2: Percentage of LAMBDAPI proofs on the TSTP files

Prover	% **E** TSTP	% VAMPIRE TSTP
ZenonModulo	45%	54%
ArchSAT	56%	74%
ZenonModulo ∪ *ArchSAT*	69%	83%

Indeed, it may be the case that a generated TPTP file cannot be proved by *ZenonModulo* but can be proved by *ArchSAT*, or vice versa. However, in the end, we get a complete proof some part of it has been proved by *ZenonModulo* and some other part by *ArchSAT*.

Consequently, we are now able to produce 2145 LAMBDAPI proofs from the TPTP library using **E** and *ZenonModulo* (resp. 2684 using **E** and *ArchSAT* and 3295 using **E**, *ZenonModulo* and *ArchSAT*), whereas, under the same conditions, *ZenonModulo* alone is only able to produce 1026 LAMBDAPI proofs (resp. 500 for *ArchSAT* alone).

Sometimes, *ZenonModulo* and *ArchSAT* fail to find a proof even if the sub-problem is simpler than the main one. This is justified by the fact that the proof calculus used in *ZenonModulo* and *ArchSAT* is based on a different method from the one used in **E**. In fact, some steps that are trivial for a prover based on resolution or superposition may not be trivial for *ZenonModulo* or *ArchSAT*, which use the tableaux method.

We give in Table 3.3 and Table 3.4 the number of proofs reconstructed for each category of TPTP problems. Categories abbreviated in Table 3.3 and Table 3.4 are associated with a specific domain field or topics in sci-

ence, such as `ALG` for General Algebra, `GEO` for Geometry, `GRP` for Group Theory, `LCL` for Logic Calculi, and `SWV` for Software Verification[2]. Having a great number of reconstructed proofs (69% for **E** and 83% for VAMPIRE) is satisfactory. However, there are some categories where we have almost the totality of reconstructed proofs, such as Software Verification (90% for **E** and 96% for VAMPIRE) and Set Theory (90% for **E** and 99% for VAMPIRE) and some categories where we have a less amount of reconstructed proofs such as Lattices Algebra (29% for **E** and 69% for VAMPIRE) and Group Theory (23% for **E** and 39% for VAMPIRE). This matter is expected due to the performance of the provers used to prove steps (*ZenonModulo* and *ArchSAT*) since both provers are based on the tableaux method, which is not as efficient as the equational Superposition Calculus and Saturation algorithm used in **E** and VAMPIRE.

iProverModulo [Bur11] is another candidate to prove TSTP steps, but it performs some transformations before outputting a LAMBDAPI proof. Therefore, the proof reconstruction is hard in the sense that we need to justify each transformation made by *iProverModulo*.

Each step of this experiment has been run on a cluster with these specifications:

- A set of 56 processors with 4096 KB of cache memory and a frequency of 2294 MHz.
- 115GB amount of memory.
- 515GB amount of disk.

We give in Table 3.5 the time spent for each step.

We notice that the amount of time spent on generating the TSTP files of **E** is less than the amount to prove the generated steps even though the generated TPTP problems are less complex to prove than the original problem. This issue is due to two primary reasons:

- First, the proof objects returned by *ZenonModulo* and *ArchSAT* contain more details than the TSTP files. The majority of steps in a TSTP file are a trivial instantiation of a specific rule used by the prover. However, it is not the case for *ZenonModulo* and *ArchSAT* since they need to perform a proof search and produce a proof witness in their proof system, which is not an instant process.

[2] A full list of these categories is available at
`http://www.tptp.org/cgi-bin/SeeTPTP?Category=Documents&File=OverallSynopsis`

- Second, some TSTP files could contain more than thousands of steps, such as the generated file `SWV424-1-050` that contains more than 50 000 steps which theoretically could take more than five days to reconstruct it using *ZenonModulo* or *ArchSAT* if we use only 10 seconds timeout for each step. We observed that *ZenonModulo* for example, spend the entire timeout (10s) each time it does not directly find a proof at the beginning of the proof search.

Remark 3.2.1
The type checking of 10 LAMBDAPI *files*[3] *generated from* ***E*** *proofs takes a significant amount of time (several hours) so that we could not compute the total amount of time necessary for type-checking if we take all the files. The amount presented in Table 3.5 refers to the time of type-checking of all files except the 10 files that need more than one hour to type-check and one file (*`SWV418-1-060`*) that type-checks in 42 minutes.*

We did not type-check the files generated by *ArchSAT* since, for now, they cannot be used with the files generated by *ZenonModulo* since they do not share the same logic files. This problem could be fixed by mapping every symbol of the logic files used in *ArchSAT* with the corresponding symbol of *ZenonModulo* logic files.

In Table 3.6, we give the size of all the generated files.

We give also in Table 3.7 the compressed size of all the generated files.

Remark 3.2.2 (Reproducibility)
A bench of scripts and instructions are provided in the default repository [4] *of* EKSTRAKTO. *The folder* `scripts` *contains the scripts to run the benchmarks by following this order:*

- *We start generating TSTP files by using one of the provers (****E*** *or* VAMPIRE*) by using the script* `eprove.sh`.
- *We then run* EKSTRAKTO *on the set of the generated TSTP files by using the script* `ekstrakto.sh`.
- *After generating the TPTP files with* EKSTRAKTO, *we now launch ZenonModulo or ArchSAT by using* `zenon_modulo.sh` *or* `archsat.sh`.

[3] `PUZ037-3`, `MSC015-1-027`, `MSC015-1-025`, `MSC015-1-020` ,`MSC015-1-030`, `PUZ037-2`, `PUZ037-1`, `SWV424-1-050` , `MSC015-1-022`, `SWV418-1-100`.

[4] `https://github.com/elhaddadyacine/ekstrakto`

3.3 Conclusion

We have presented a tool that reconstructs proofs generated by first-order theorem provers and described how we could implement a simple proof reconstruction.

The advantage of EKSTRAKTO is to be generic since it does not depend on the rules used by the automated prover to find the proof. Another advantage is the fact that the proofs are expressed in LAMBDAPI: we can translate them to many other systems (Coq, HOL, Lean, Matita, PVS).

In our experiments, we used *ZenonModulo* and *ArchSAT* to prove each trace step since they are tools that produce LAMBDAPI proof terms.

Some prover's rules are hard to prove, like Skolemization steps. Hence, we will introduce in Chapter 5 an algorithm that transforms proofs of a Skolemized formula to a proof without Skolem symbol.

Table 3.3: Number of LAMBDAPI proofs for each category (**E**)

Category	Total	*ZenonModulo*		*ArchSAT*		both	
ALG	82	46	56%	55	67%	61	74%
ANA	69	39	57%	54	78%	56	81%
BOO	72	10	14%	11	15%	22	31%
CAT	53	38	72%	48	91%	48	91%
COL	200	72	36%	105	53%	114	57%
COM	12	8	67%	10	83%	12	100%
CSR	8	2	25%	8	100%	8	100%
FLD	143	73	51%	128	90%	137	96%
GEO	145	84	58%	128	88%	141	97%
GRA	1	1	100%	1	100%	1	100%
GRP	740	99	13%	159	21%	167	23%
HEN	62	14	23%	17	27%	29	47%
HWC	3	0	0%	0	0%	0	0%
HWV	88	66	75%	68	77%	78	89%
KLE	4	0	0%	0	0%	0	0%
KRS	9	9	100%	7	78%	9	100%
LAT	191	49	26%	54	28%	55	29%
LCL	526	135	26%	204	39%	408	78%
LDA	7	0	0%	6	86%	6	86%
MGT	67	52	78%	51	76%	67	100%
MSC	19	18	95%	19	100%	19	100%
NLP	22	21	95%	9	41%	21	95%
NUM	47	18	38%	31	66%	41	87%
PLA	36	13	36%	25	69%	32	89%
PUZ	61	53	87%	53	87%	58	95%
REL	81	0	0%	2	2%	4	5%
RNG	62	5	8%	17	27%	21	34%
ROB	17	2	12%	6	35%	6	35%
SCT	33	15	45%	21	64%	22	67%
SET	362	246	68%	286	79%	324	90%
SEU	1	0	0%	0	0%	0	0%
SWC	353	278	79%	213	60%	288	82%
SWV	480	339	71%	370	77%	430	90%
SWW	22	0	0%	1	5%	1	5%
SYN	613	293	48%	505	82%	553	90%
SYO	64	42	66%	7	11%	51	80%
TOP	5	5	100%	5	100%	5	100%

Table 3.4: Number of LAMBDAPI proofs for each category (VAMPIRE)

Category	Total	*ZenonModulo*		*ArchSAT*		both	
ALG	39	26	67%	29	74%	34	87%
ANA	34	19	56%	31	91%	34	100%
BOO	60	10	17%	21	35%	33	55%
CAT	44	31	70%	39	89%	41	93%
COL	134	52	39%	108	81%	112	84%
COM	7	5	71%	7	100%	7	100%
CSR	7	3	43%	7	100%	7	100%
FLD	60	40	67%	60	100%	60	100%
GEO	96	61	64%	93	97%	96	100%
GRP	376	62	16%	130	35%	147	39%
HEN	58	12	21%	43	74%	49	84%
HWC	2	0	0%	0	0%	1	50%
HWV	41	34	83%	40	98%	40	98%
KRS	9	9	100%	9	100%	9	100%
LAT	77	46	60%	52	68%	53	69%
LCL	295	189	64%	196	66%	271	92%
LDA	7	0	0%	7	100%	7	100%
MGT	10	9	90%	9	90%	10	100%
MSC	6	5	83%	6	100%	6	100%
NLP	2	2	100%	2	100%	2	100%
NUM	21	9	43%	19	90%	20	95%
PLA	28	5	18%	19	68%	28	100%
PUZ	12	9	75%	12	100%	12	100%
REL	18	1	6%	2	11%	10	56%
RNG	45	21	47%	29	64%	34	76%
ROB	14	3	21%	9	64%	11	79%
SCT	13	10	77%	10	77%	10	77%
SET	227	198	87%	221	97%	225	99%
SEU	1	0	0%	1	100%	1	100%
SWC	88	88	100%	88	100%	88	100%
SWV	238	199	84%	228	96%	229	96%
SWW	14	5	36%	2	14%	6	43%
SYN	310	126	41%	259	84%	288	93%
SYO	51	39	76%	8	16%	44	86%
TOP	5	5	100%	5	100%	5	100%

Table 3.5: Time spent for each step

Step	**E**	VAMPIRE
Generating TSTP files	6h 08m 01s	6h 42m 06s
Extracting TPTP files	0h 08m 09s	0h 01m 46s
Proving TPTP with *ZenonModulo*	2h 46m 14s	2h 58m 50s
Proving TPTP with *ArchSAT*	9h 00m 39s	3h 46m 20s
Type-checking with LAMBDAPI*	0h 13m 32s	0h 05m 40s

* See Remark 3.2.1.

Table 3.6: Size of each type of files (in MB)

Type of files	**E**	VAMPIRE
TSTP files	200	700
TPTP files	1,800	1,000
LAMBDAPI files with *ZenonModulo*	28,000	8,800
DEDUKTI files with *ArchSAT*	26,000	5,000

Table 3.7: Compressed size of each type of files (in MB)

Type of files	**E**	VAMPIRE
TSTP files	9	13
TPTP files	19	62
LAMBDAPI files with *ZenonModulo*	531	214
DEDUKTI files with *ArchSAT*	903	178

Chapter 4

Handling commutative cuts in $\lambda\Pi/\equiv$

4.1 Intuitionistic logic

In the next chapter on de-skolemization, we will need an important property, namely the subformula property, saying that a derivation tree whose conclusion is a formula F only contains sub-formulas of F. This property doesn't hold in general, but it holds for the so-called cut-free proofs.

4.1.1 Standard cuts

A cut is a derivation tree where an introduction rule is followed by an elimination rule like:

$$\dfrac{\dfrac{\dfrac{\pi_A}{A} \quad \dfrac{\pi_B}{B}}{A \wedge B}\wedge_I}{A}\wedge_{El}$$

A cut represents a useless detour which we may want to eliminate:

$$\dfrac{\pi_A}{A}$$

This can be simulated by adding the following rewriting rule in the deep encoding of Section 1.4:

$$\wedge_{El}\ a\ b\ (\wedge_I\ a\ b\ \pi_a\ \_) \hookrightarrow \pi_a$$

Within the shallow embedding of Section 1.4, there is no need to add rules as a cut is translated into a redex. For instance, the proof above is represented by the $\lambda\Pi/\equiv$ term

$$\wedge_{El}\ A\ B\ (\wedge_I\ A\ B\ \pi_A\ \pi_B)$$

which reduces to π_A by using the definitions of the symbols and β-reduction:

$$\begin{array}{ll} & \wedge_{El}\ A\ B\ (\wedge_I\ A\ B\ \pi_A\ \pi_B) \\ \hookrightarrow & (\lambda\pi, \pi\ A\ (\lambda\pi_A, \lambda\_, \pi_A))\ (\lambda X, \lambda h, h\ \pi_A\ \pi_B) \\ \hookrightarrow & (\lambda X, \lambda h, h\ \pi_A\ \pi_B)\ A\ (\lambda\pi_A, \lambda\_, \pi_A) \\ \hookrightarrow & (\lambda\pi_A, \lambda\_, \pi_A)\ \pi_A\ \pi_B \\ \hookrightarrow & \pi_A \end{array}$$

4.1.2 Commutative cuts

For handling the $\vee$ and $\exists$ connectors, we also need to consider the so-called commutative cuts [Pra71, GLT88], that are, a sequence of two rules, such that the second rule is an elimination and its main formula is the conclusion of the first rule, like:

$$\dfrac{\dfrac{A \vee A \vdash A \vee A \qquad \dfrac{A \vee A, A \vdash A \qquad A \vee A, A \vdash A}{A \vee A, A \vdash A \wedge A}\wedge_I \qquad \dfrac{A \vee A, A \vdash A \qquad A \vee A, A \vdash A}{A \vee A, A \vdash A \wedge A}\wedge_I}{A \vee A \vdash A \wedge A}\vee_E}{A \vee A \vdash A}\wedge_{El}$$

After one step of cut elimination we end up with the following proof tree:

$$\dfrac{A \vee A \vdash A \vee A \qquad \dfrac{\dfrac{A \vee A, A \vdash A \quad A \vee A, A \vdash A}{A \vee A, A \vdash A \wedge A}\wedge_I}{A \vee A, A \vdash A}\wedge_{El} \qquad \dfrac{\dfrac{A \vee A, A \vdash A \quad A \vee A, A \vdash A}{A \vee A, A \vdash A \wedge A}\wedge_I}{A \vee A, A \vdash A}\wedge_{El}}{A \vee A \vdash A}\vee_E$$

and then we obtain

$$\dfrac{A \vee A \vdash A \vee A \qquad A \vee A, A \vdash A \qquad A \vee A, A \vdash A}{A \vee A \vdash A}\vee_E$$

For handling all connectors and quantifiers, we end up with the many

rewrite rules of Figure 4.1. For the corresponding LAMBDAPI file, see Section 8.10.

Fortunely, it has been proved that this rewrite system terminates and is confluent [Pra71, Gir87, GLT88, vdP96]. Hence, in $\lambda\Pi$ modulo these rewriting rules, every term has a unique normal form.

Note that commutative cuts are not translated to redexes in the shallow encoding. Indeed, the proof tree above is represented in $\Sigma_{ND}^{\epsilon,full}$ as:

$$\wedge_{El}\ A\ A\ (\vee_E\ A\ A\ h_{A\vee A}\ (A\wedge A)\ (\lambda a, \wedge_I\ A\ A\ a\ a)\ (\lambda a, \wedge_I\ A\ A\ a\ a))$$

which reduces to:

$$h_{A\vee A}\ (A\wedge A)\ (\lambda a, \lambda\_, \lambda\pi, \pi\ a\ a)\ (\lambda a, \lambda\_, \lambda\pi, \pi\ a\ a)$$

And we cannot use the shallow embedding $\Sigma_{ND}^{\epsilon,full}$ with the rules of 4.1 since these connectors and quantifiers are already defined.

Thanks to these new rules, one can prove the subformula property.

Proposition 4.1.1
For all $\mathcal{FOL}$-environments, for all π in NF in Σ_{CC}, for all C, if $\Gamma \vdash \pi : \epsilon C$, let ϖ be a subterm of π such that we have $\Gamma, \Gamma' \vdash \varpi : \epsilon D$ while checking that $\Gamma \vdash \pi : \epsilon C$, then D a subterm of Γ or C.

Proof. By Theorem 2.1.1, we have $\psi(\Gamma_2) \vdash_{ND} \psi(C)$, and since π is in NF, so is the proof in Natural Deduction with regard to standard and commuting cuts. Therefore, it enjoys the subformula property [Pra71] in Natural Deduction : all the formulas appearing in the proof derivation of $\psi(\Gamma_2) \vdash_{ND} \psi(C)$ are subformulas of $\psi(\Gamma)$ or $\psi(C)$. Since $\Gamma, \Gamma' \vdash \varpi : \epsilon D$ corresponds to a subproof $\psi(\Gamma_2), \psi(\Gamma_2') \vdash_{ND} \psi(D)$, we get that $\psi(D)$ is a subformula of $\psi(\Gamma_2)$ or $\psi(C)$, which in turn means that D is a subformula of Γ or C. □

4.2 Classical logic

For getting the subformula property in classical logic, we must give rules for the commutative cuts for $nnpp$ too. For instance, we can build the following derivation in Natural Deduction :

$$\dfrac{\dfrac{\dfrac{\dfrac{A, \neg(A\wedge A) \vdash \neg(A\wedge A) \qquad \dfrac{A, \neg(A\wedge A) \vdash A \qquad A, \neg(A\wedge A) \vdash A}{A, \neg(A\wedge A) \vdash A\wedge A}\wedge_I}{A, \neg(A\wedge A) \vdash \bot}\neg_E}{A \vdash \neg(\neg(A\wedge A))}\neg_I}{A \vdash A\wedge A}nnpp}{A \vdash A}\wedge_{El}$$

in which the formula $A \wedge A$ is not a subformula of A.

This proof tree is represented by the $\lambda\Pi/\equiv$ term

$$\wedge_{El}\ A\ A(nnpp\ (A \wedge A)(\lambda h : \epsilon\ (\neg(A \wedge A)), h\ (\wedge_I\ A\ A\ a\ a)))$$

in a context where $a : \epsilon A$, which cannot be reduced.

To solve this problem, we need to add the following (left-linear) rules on $nnpp$ (see the corresponding LAMBDAPI file in Section 8.11):

$$\begin{aligned}
\vee_E\ a\ b(nnpp\ _\ \pi)\ c\ \pi_{ac}\ \pi_{bc} &\hookrightarrow nnpp\ c(\lambda\ h_1,\ \pi\ (\lambda\ h_2,\ h_1(\vee_E\ a\ b\ h_2\ c\ \pi_{ac}\ \pi_{bc})))\\
=_E\ t\ v(nnpp\ _\ \pi)\ c\ \pi_{ct} &\hookrightarrow nnpp\ (c\ v)(\lambda\ h_1,\ \pi\ (\lambda\ h_2,\ h_1\ (=_E\ t\ v\ h_2\ c\ \pi_{ct})))\\
\wedge_{El}\ a\ b\ (nnpp\ _\ \pi) &\hookrightarrow nnpp\ a\ (\lambda\ h_1,\ \pi\ (\lambda\ h_2,\ h_1\ (\wedge_{El}\ a\ b\ h_2)))\\
\wedge_{Er}\ a\ b\ (nnpp\ _\ \pi) &\hookrightarrow nnpp\ b\ (\lambda\ h_1,\ \pi\ (\lambda\ h_2,\ h_1\ (\wedge_{Er}\ a\ b\ h_2)))\\
\exists_E\ p\ (nnpp\ _\ \pi)\ c\ \pi_{pc} &\hookrightarrow nnpp\ c\ (\lambda\ h_1,\ \pi\ (\lambda\ h_2,\ h_1\ (\exists_E\ p\ h_2\ c\ \pi_{pc})))\\
nnpp\ c\ (\vee_E\ a\ b\ \pi_{a\vee b}\ _\ \pi_{ac}\ \pi_{bc}) &\hookrightarrow \vee_E\ a\ b\ \pi_{a\vee b}\ c\ (\lambda\ \pi_a,\ nnpp\ c\ (\pi_{ac}\ \pi_a))\\
&\qquad (\lambda\ \pi_b,\ nnpp\ c\ (\pi_{bc}\ \pi_b))\\
nnpp\ c\ (\exists_E\ p\ \pi_{exp}\ _\ \pi_{pt}) &\hookrightarrow \exists_E\ p\ \pi_{exp}\ c\ (\lambda\ x\ \pi_p,\ nnpp\ c\ (\pi_{pt}\ x\ \pi_p))\\
nnpp\ p\ (nnpp\ _\ \pi) &\hookrightarrow nnpp\ p\ (\lambda\ \pi_{\neg p},\ \pi\ (\lambda\ \pi_{\neg\neg p},\ \pi_{\neg\neg p}\ \pi_{\neg p}))\\
nnpp\ (_\ \Rightarrow\ b)\ \pi_r\ q &\hookrightarrow nnpp\ b\ (\lambda\ h_1,\ \pi_r\ (\lambda\ h_2,\ h_1\ (h_2\ q)))\\
nnpp\ (\forall\ p)\ \pi\ t &\hookrightarrow nnpp\ (p\ t)\ (\lambda\ h_1 :\ \epsilon\ (\neg\ (p\ t)),\ \pi\\
&\qquad (\lambda\ h_2 :\ \epsilon\ (\forall\ p),\ h_1(h_2\ t)))
\end{aligned}$$

Using these rules, the proof term above reduces to

$$\begin{aligned}
&\wedge_{El}\ \ A\ A(nnpp\ (A \wedge A)(\lambda h : \epsilon\ (\neg(A \wedge A)), h\ (\wedge_I\ A\ A\ a\ a)))\\
\hookrightarrow\ &nnpp\ A(\lambda h_1 : \epsilon\ (\neg A), (\lambda h : \epsilon\ (\neg(A \wedge A)), h\ (\wedge_I\ A\ A\ a\ a))\ (\lambda h_2 : \epsilon(A \wedge A), h_1\ (\wedge_{El}\ h_2)))\\
\hookrightarrow\ &nnpp\ A(\lambda h_1 : \epsilon\ (\neg A), (\lambda h_2 : \epsilon(A \wedge A), h_1\ (\wedge_{El}\ h_2))\ (\wedge_I\ A\ A\ a\ a))\\
\hookrightarrow\ &nnpp\ A(\lambda h_1 : \epsilon\ (\neg A), h_1\ (\wedge_{El}\ (\wedge_I\ A\ A\ a\ a)))\\
\hookrightarrow\ &nnpp\ A(\lambda h_1 : \epsilon\ (\neg A), h_1\ a)
\end{aligned}$$

Remark 4.2.1
We may try to transform a classical proof that uses nnpp into an intuitionistic proof by applying the following rules, using higher-order pattern matching:

$$\begin{gathered}
nnpp\ a\ (\lambda h_1, h_1\ (nnpp\ a\ (\lambda h_2, K[h_1, h_2]))) \hookrightarrow nnpp\ a\ (\lambda h_1, K[h_1, h_1])\\
nnpp\ a\ (\lambda h_1, h_1\ K[]) \hookrightarrow\ K
\end{gathered}$$

Using these rules, the example above is rewritten to a.

Figure 4.1: Rewriting rules for handling commutative cuts in intuitionistic logic

$\Sigma_{CC} = \Sigma^{\epsilon}_{ND}+$

$\vee_E\ a\ \_\ (\vee_{Il}\ a\ \_\ \pi_a)\ \_\ \pi_{aP}\ \_$	$\hookrightarrow \pi_{aP}\ \pi_a$
$\vee_E\ \_\ b\ (\vee_{Ir}\ \_\ b\ \pi_b)\ \_\ \_\ \pi_{bP}$	$\hookrightarrow \pi_{bP}\ \pi_b$
$\vee_E\ a\ b\ \pi_{a\vee b}\ (\_ \Rightarrow q)\ \pi_{apq}\ \pi_{bpq}\ \pi_p$	$\hookrightarrow \vee_E\ a\ b\ \pi_{a\vee b}\ q\ (\lambda\pi_a, \pi_{apq}\ \pi_a\ \pi_p)\ (\lambda\pi_b, \pi_{bpq}\ \pi_b\ \pi_p)$
$\vee_E\ \pi_{a\vee b}\ (\forall a)\ \pi_{apq}\ \pi_{bpq}\ t$	$\hookrightarrow \vee_E\ \pi_{a\vee b}(a\ t)(\lambda\pi_a, \pi_{apq}\ \pi_a\ t)(\lambda\pi_b, \pi_{bpq}\ \pi_b\ t)$
$\vee_E\ c\ d\ (\vee_E\ a\ b\ \pi_{a\vee b}\ \_\ \pi_{a_{cord}}\ \pi_{b_{cord}})\ e\ \pi_{ce}\ \pi_{de}$	$\hookrightarrow \vee_E\ a\ b\ \pi_{a\vee b}\ e\ (\lambda\pi_a, \vee_E\ c\ d\ (\pi_{a_{cord}}\ \pi_a)\ e\ \pi_{ce}\ \pi_{de})$ $(\lambda\pi_b, \vee_E\ c\ d\ (\pi_{b_{cord}}\ \pi_b)\ e\ \pi_{ce}\ \pi_{de})$
$\vee_E\ a\ b\ (\exists_E\ p\ \pi_x(\ a \vee\ b)\ \pi_{pt})\ e\ \pi_{ce}\ \pi_{de}$	$\hookrightarrow \exists_E\ p\ \pi_x\ e\ (\lambda x\ \pi_p, \vee_E\ a\ b\ (\pi_{pt}\ x\ \pi_p)\ e\ \pi_{ce}\ \pi_{de})$
$\vee_E\ \_\ \_\ (\bot_E\ \pi_\bot\ (\_ \vee \_))\ P\ \_\ \_$	$\hookrightarrow \bot_E\ \pi_\bot\ P$
$=_E\ \ t\ v\ (=_I\ \_)\ \_\ pt$	$\hookrightarrow pt$
$=_E\ \ t\ v\ (\vee_E\ a\ b\ \pi_{a\vee b}\ \_\ \pi_{atv}\ \pi_{btv})\ P\ \pi_{Pt}$	$\hookrightarrow \vee_E\ a\ b\ \pi_{a\vee b}\ (P\ v)\ (\lambda\pi_a, =_E\ \ t\ v\ (\pi_{atv}\ \pi_a)\ P\ \pi_{Pt})$ $(\lambda\pi_b, =_E\ \ t\ v\ (\pi_{btv}\ \pi_b)\ P\ \pi_{Pt})$
$=_E\ \ t\ v\ (\exists_E\ p\ \pi_x\ \_\ \pi_{pt})\ P\ \pi_{Pt}$	$\hookrightarrow \exists_E\ p\ \pi_x\ (P\ v)\ (\lambda x\ \pi_p, =_E\ \ t\ v\ (\pi_{pt}\ x\ \pi_p)\ P\ \pi_{Pt})$
$=_E\ \ t\ v\ (\bot_E\ \pi_\bot\ \_)\ P\_$	$\hookrightarrow \bot_E\ \pi_\bot\ (P\ v)$
$\wedge_{El}\ a\ \_\ (\wedge_I\ a\ \_\ \pi_a\ \_)$	$\hookrightarrow \pi_a$
$\wedge_{El}\ c\ d\ (\vee_E\ a\ b\ \pi_{a\vee b}\ \_\ \pi_{at}\ \pi_{bt})$	$\hookrightarrow \vee_E\ a\ b\ \pi_{a\vee b}\ c\ (\lambda\pi_a, \wedge_{El}\ c\ d$ $(\pi_{at}\ \pi_a))(\lambda\pi_b, \wedge_{El}\ c\ d\ (\pi_{bt}\ \pi_b))$
$\wedge_{El}\ c\ d\ (\exists_E\ p\ \pi_x\ \_\ \pi_{pt})$	$\hookrightarrow \exists_E\ p\ \pi_x\ c\ (\lambda x\ \pi_p, \wedge_{El}\ c\ d\ (\pi_{pt}\ x\ \pi_p))$
$\wedge_{El}\ a\ \_\ (\bot_E\ \pi_\bot\ \_)$	$\hookrightarrow \bot_E\ \pi_\bot\ a$
$\wedge_{Er}\ \_\ b\ (\wedge_I\ \_\ b\ \_\ \pi_b)$	$\hookrightarrow \pi_b$
$\wedge_{Er}\ c\ d\ (\vee_E\ a\ b\ \pi_{a\vee b}\ \_\ \pi_{at}\ \pi_{bt})$	$\hookrightarrow \vee_E\ a\ b\ \pi_{a\vee b}\ d\ (\lambda\pi_a, \wedge_{Er}\ c\ d$ $(\pi_{at}\ \pi_a))(\lambda\pi_b, \wedge_{Er}\ c\ d\ (\pi_{bt}\ \pi_b))$
$\wedge_{Er}\ c\ d\ (\exists_E\ p\ \pi_x\ \_\ \pi_{pt})$	$\hookrightarrow \exists_E\ p\ \pi_x\ d\ (\lambda x\ \pi_p, \wedge_{Er}\ c\ d\ (\pi_{pt}\ x\ \pi_p))$
$\wedge_{Er}\ \_\ b\ (\bot_E\ \pi_\bot\ \_)$	$\hookrightarrow \bot_E\ \pi_\bot\ b$
$\exists_E\ p\ (\exists_I\ p\ t\ pt)\ \_\ \pi_x$	$\hookrightarrow \pi_x\ t\ pt$
$\exists_E\ p(\vee_E\ a\ b\ \pi_{a\vee b}\ \_\ \pi_{at}\ \pi_{bt})\ P\ \pi_{xP}$	$\hookrightarrow \vee_E\ a\ b\ \pi_{a\vee b}\ P\ (\lambda\pi_a, \exists_E\ p(\pi_{at}\ \pi_a)\ P\ \pi_{xP})$ $(\lambda\pi_b, \exists_E\ p\ (\pi_{bt}\ \pi_b)\ P\ \pi_{xP})$
$\exists_E\ p\ \pi_x\ (\_ \Rightarrow d)\ \pi_{pt}\ pc$	$\hookrightarrow \exists_E\ p\ \pi_x\ d\ (\lambda x\pi_p,\ \pi_{pt}\ x\ \pi_p\ pc)$
$\exists_E\ p(\exists_E\ q\ \pi_x\ \_\ \pi_{pt})\ P\ \pi_{xP}$	$\hookrightarrow \exists_E\ q\ \pi_x\ P\ (\lambda x\ \pi_p, \exists_E\ p\ (\pi_{pt}\ x\ \pi_p)\ P\ \pi_{xP})$
$\exists_E\ p\ \pi_x\ (\forall b)\ \pi_{pt}\ t$	$\hookrightarrow \exists_E\ p\ \pi_x\ (b\ t)(\lambda x\ \pi_p, \pi_{pt}\ x\ \pi_p\ t)$
$\exists_E\ \_\ (\bot_E\ \pi_\bot\ \_)\ P\ \_$	$\hookrightarrow \bot_E\ \pi_\bot\ P$
$\bot_E\ (\bot_E\ \pi_\bot\ \_)\ P$	$\hookrightarrow \bot_E\ \pi_\bot\ P$
$\bot_E\ \pi_\bot\ (\_ \Rightarrow b)\ \_$	$\hookrightarrow \bot_E\ \pi_\bot\ b$
$\bot_E\ \pi_\bot\ (\forall a)\ t$	$\hookrightarrow \bot_E\ \pi_\bot\ (a\ t)$
$\bot_E\ (\vee_E\ a\ b\ \pi_{a\vee b}\ \_\ \pi_{a\bot}\ \pi_{b\bot})\ P$	$\hookrightarrow \vee_E\ a\ b\ \pi_{a\vee b}\ P\ (\lambda\pi_a, \bot_E\ (\pi_{a\bot}\ \pi_a)\ P)$ $(\lambda\pi_b, \bot_E\ (\pi_{b\bot}\ \pi_b)\ P)$
$\bot_E\ (\exists_E\ p\ \pi_x\ \_\ \pi_{pt})\ P$	$\hookrightarrow \exists_E\ p\ \pi_x\ P\ (\lambda x\ \pi_p, \bot_E\ (\pi_{pt}\ x\ \pi_p)\ P)$

Chapter 5

De-Skolemization

When reconstructing proofs, some steps cannot be proved easily, such as the Skolemization step, where a prover introduces a new symbol, called a Skolem symbol, in order to eliminate an existential quantifier from the given formula, e.g., the formula $\forall x \exists y, x + y = 0$ is transformed into $\forall x, x + fx = 0$. Then the prover produces a proof of the new formula, which therefore contains the new symbol f. Following an unpublished work by Gilles Dowek and Benjamin Werner on deskolemization in natural deduction [DW05], we show in this chapter a way to eliminate this new symbol and get a proof of the original formula from a proof of the skolemized formula in the $\lambda\Pi$-calculus modulo the deep encoding of first-order logic and natural deduction in Σ_{ND} from Section 1.4 plus the rewrite rules for commutative cuts of the previous section.

5.1 Skolem theorem

Skolemization is the transformation of a $\mathcal{FOL}$ formula by eliminating the existential quantifier and replacing the variable linked to that quantifier with a new function symbol called the SKOLEM symbol. It first puts the formula in prenex normal form:

Definition 5.1.1 (Prenex normal form)
A $\mathcal{FOL}$ formula is in prenex normal form (PNF) if it has the form $\Xi x_1 \dots \Xi x_n F$ where Ξ is one of the quantifiers $\{\forall, \exists\}$ and F is a formula that contains no quantifiers.

We define the function PNF as follows:

$$
\begin{array}{lll}
PNF(\neg\exists xA) & = & \forall xPNF(\neg A) \\
PNF(\neg\forall xA) & = & \exists xPNF(\neg A) \\
PNF((\forall xA) \wedge B) & = & \forall xPNF(A \wedge B) \ \textit{if}\ x \notin FV(B) \\
PNF((\forall xA) \vee B) & = & \forall xPNF(A \vee B) \ \textit{if}\ x \notin FV(B) \\
PNF((\exists xA) \wedge B) & = & \exists xPNF(A \wedge B) \ \textit{if}\ x \notin FV(B) \\
PNF((\exists xA) \vee B) & = & \exists xPNF(A \vee B) \ \textit{if}\ x \notin FV(B) \\
PNF(A \wedge (\forall xB)) & = & \forall xPNF(A \wedge B) \ \textit{if}\ x \notin FV(A) \\
PNF(A \vee (\forall xB)) & = & \forall xPNF(A \vee B) \ \textit{if}\ x \notin FV(A) \\
PNF(A \wedge (\exists xB)) & = & \exists xPNF(A \wedge B) \ \textit{if}\ x \notin FV(A) \\
PNF(A \vee (\exists xB)) & = & \exists xPNF(A \vee B) \ \textit{if}\ x \notin FV(A) \\
PNF((\forall xA) \Rightarrow B) & = & \exists xPNF(A \Rightarrow B) \ \textit{if}\ x \notin FV(B) \\
PNF((\exists xA) \Rightarrow B) & = & \forall xPNF(A \Rightarrow B) \ \textit{if}\ x \notin FV(B) \\
PNF(A \Rightarrow (\forall xB)) & = & \forall xPNF(A \Rightarrow B) \ \textit{if}\ x \notin FV(A) \\
PNF(A \Rightarrow (\exists xB)) & = & \exists xPNF(A \Rightarrow B) \ \textit{if}\ x \notin FV(A) \\
PNF(A) & = & A \ \textit{otherwise (if A contains no quantifiers)}
\end{array}
$$

Definition 5.1.2 (Skolemization)
The function SK that skolemizes a $\mathcal{FOL}$ formula is defined as follows:
$SK(F) = SK_{pnf}(PNF(F))$ *where:*

$$
\begin{array}{lll}
SK_{pnf}(\forall\bar{x}\exists yA) & = & SK_{pnf}(\forall\bar{x}[f\ \bar{x}/y]A) \ \textit{where } f \textit{ is a new function symbol} \\
SK_{pnf}(F) & = & F \ \textit{if } F \textit{ contains no existential quantifiers}
\end{array}
$$

Example 5.1.3
- $\forall x\exists y, x = y$ *is transformed to* $\forall x.x = f\ x$.
- $\forall z\exists y_1\forall x_1\forall x_2\exists y_2, (x_1 = y_1) \Rightarrow (y_2 = x_2)$ *is transformed to* $\forall z\forall x_1\forall x_2, (x_1 = f_1\ z) \Rightarrow (f_2\ z\ x_1\ x_2 = x_2)$.

We recall the properties of PNF and skolemization:

Proposition 5.1.4 (Equivalence of prenex normal form)

- *The formula* $F \Leftrightarrow PNF(F)$ *is valid in classical logic.*
- *Given a context* Γ *and* $\mathcal{FOL}$ *formulas A and F.* $\Gamma, \forall\bar{x}\exists yA \vdash F$ *is provable if and only if* $\Gamma, \forall\bar{x}[f\ \bar{x}/y]A \vdash F$ *is provable, where f a new function symbol.*

5.2 Algorithm

Let A be the φ translation of a first-order formula. Assume that $\forall\bar{x}\exists yA$ is skolemized into $\forall\bar{x}[f\bar{x}/y]A$. Then, let B be the translation of a formula not containing f (e.g. $\bot$).

We now present an algorithm that transforms a $\lambda\Pi/\equiv$ term π of type ϵB in the typing environment $\Gamma, a_{sk} : \hat{a}_{sk}$ where $\hat{a}_{sk} = \epsilon(\forall \bar{x}[f\bar{x}/y]A)$

$$\Gamma, a_{sk} : \epsilon(\forall \bar{x}[f\bar{x}/y]A) \vdash \pi : \epsilon B$$

into a $\lambda\Pi/\equiv$ term π' of the same type ϵB but in the typing environment $\Gamma, a : \hat{a}$ where $\hat{a} = \epsilon(\forall \bar{x} \exists y A)$.

$$\Gamma, a : \epsilon(\forall \bar{x} \exists y A) \vdash \pi : \epsilon B$$

Definition 5.2.1
$\{f\bar{u} \to z\}$ is the substitution of $f\ \bar{u}$ by a variable z. It is defined as follows:

$$\begin{array}{lcl}
\{f\bar{u} \to z\}\Gamma & = & \alpha_1 : \{f\bar{u} \to z\}A_1, \ldots, \alpha_n : \{f\bar{u} \to z\}A_n \\
 & & \textit{where } \Gamma = \alpha_1 : A_1, \ldots, \alpha_n : A_n. \\
\{f\bar{u} \to z\}(f\ \bar{u}) & = & z \\
\{f\bar{u} \to z\}(x) & = & x \textit{ if } x \textit{ is a variable.} \\
\{f\bar{u} \to z\}(\Pi(x : u), v) & = & \Pi(x : \{f\bar{u} \to z\}(u)), \{f\bar{u} \to z\}(v) \\
\{f\bar{u} \to z\}(\lambda(x : u), v) & = & \lambda(x : \{f\bar{u} \to z\}(u), \{f\bar{u} \to z\}(v)) \\
\{f\bar{u} \to z\}(uv) & = & \{f\bar{u} \to z\}(u)\ \{f\bar{u} \to z\}(v) \textit{ if } uv \neq f\ \bar{u} \\
\{f\bar{u} \to z\}KIND & = & KIND \\
\{f\bar{u} \to z\}TYPE & = & TYPE
\end{array}$$

Definition 5.2.2 (Elimination of hypothesis)
Given a context Γ that does not contain y, $\mathcal{FOL}$-terms $\bar{u}$ arguments of f typable in Γ, a term B of type $Prop$, and a term π of type ϵB in $\Gamma, a_{sku} : \epsilon([\bar{u}/x][f\ \bar{x}/y]A)$, let $E(\Gamma, \bar{u}, B, \pi) =$
$\exists_E\ (\lambda y : \iota, [\bar{u}/x]A)\ (a\ \bar{u})\ B\ (\lambda y : \iota, \lambda a_{sku} : \epsilon([\bar{u}/x]A), \{f\ \bar{u} \to y\}\pi)$

Definition 5.2.3 ($size_f$)
Let $size_f$ be the function that returns the number of occurrences of a symbol f in a term t.

Definition 5.2.4 (f-term)
A term t is an f-term if it has the form $f\bar{u}$.

Definition 5.2.5 (Δ)
Given a context Γ and a term B, let

$$\Delta_{\Gamma,B} = \alpha_1 : \epsilon[u\bar{1}/\bar{x}][f\bar{x}/y]A, \ldots \alpha_n : \epsilon[u\bar{n}/\bar{x}][f\bar{x}/y]A$$

where $fu\bar{1} \ldots fu\bar{n}$ are all the f-terms of Γ and B.

Example 5.2.6
For a context $\Gamma = \beta : \epsilon(P(f\ a\ b)); \gamma : \epsilon(\top \Rightarrow P(f\ c\ c))$ *and a proposition* $B = P(x) \Rightarrow P(f\ a\ c)$ $\Delta_{\Gamma,B}$ *is*

$$\alpha_1 : \epsilon([a/x_1][b/x_2][f\ x_1\ x_2/y]A),$$
$$\alpha_2 : \epsilon([c/x_1][c/x_2][f\ x_1\ x_2/y]A),$$
$$\alpha_3 : \epsilon([a/x_1][c/x_2][f\ x_1\ x_2/y]A).$$

Definition 5.2.7 (Total instance)
A total instance of $\forall \bar{x}[f\bar{x}/y]A$ *is a term of the form* $[\bar{u}/\bar{x}][f\bar{x}/y]A$.

Definition 5.2.8 (Algorithm)
We define the main recursive function ψ *that takes as arguments:*

- Γ*: the context of the proof that we want to transform,*
- *B: a type in normal form (NF) wrt to the encoding presented in Section 4.2,*
- π*: the proof term of type B in NF,*

and returns a new proof term π' *defined as follows:*
$\psi(\Gamma; B; \pi) =$

- *If* $B \equiv \epsilon C$ *then*
  - *If C is equivalent to a total instance of* $\forall \bar{x}[f\bar{x}/y]A$ *then:*
    return the variable α *such that the type of* α *in* $\Delta_{\Gamma;B}$ *is equivalent to B.*
  - *Else if* π *is an abstraction* $\lambda x : t, u$ *then:*
    by inversion, B, which is in NF, must be of the form $\Pi x : t, U$
    return $\lambda x : t, \psi(\Gamma, x : t; U; u)$
  - *Else* π *is of the form* $h\ a_1\ \cdots\ a_n$ *where h is not an application.*
    By inversions, h has a type in NF of the form $\Pi x_1 : V_1, \ldots, \Pi x_n : V_n, W$.
    Let $a'_i = \psi(\Gamma; [a_1/x_1] \ldots [a_{i-1}/x_{i-1}]V_i; a_i)$.
    Let $\Delta' = \bigcup_{i=1}^{n} \Delta_{\Gamma;[a_1/x_1]\ldots[a_{i-1}/x_{i-1}]V_i}$. *We order the elements of* Δ' *that are not in* $\Delta_{\Gamma;B}$ *such that they are* $\gamma_1 : \epsilon([\overline{w_1}/\bar{x}][f\bar{x}/y]A), \ldots, \gamma_m : \epsilon([\overline{w_m}/\bar{x}][f\bar{x}/y]A)$, *where* $size_f(w_1) \leq \ldots \leq size_f(w_m)$.
    Let $\Gamma_j = \Gamma, a : \forall \bar{x} \exists y A, \Delta_{\Gamma;B}, \gamma_1 : \epsilon([\overline{w_1}/\bar{x}][f\bar{x}/y]A), \ldots, \gamma_i : \epsilon([\overline{w_j}/\bar{x}][f\bar{x}/y]A)$.
    Define recursively $\pi_m = h\ a'_1\ \cdots\ a'_n$ *and* $\pi_j = E(\Gamma_j, \overline{w_{j+1}}, C, \pi_{j+1})$.
    Return π_0.
- *Else:*
  return π

5.3 Correctness

Definition 5.3.1 (Γ-frozen f-terms)
An f-term t is a frozen term wrt a context Γ if t has the form $f\bar{u}$ and $\bar{u}$ contains no variables other than those of Γ.

Definition 5.3.2 (Γ-frozen term)
A term t is frozen wrt a context Γ if all f-terms of t are Γ-frozen.

Definition 5.3.3 (Frozen typing environments)

- *The empty environment is frozen.*
- *If Γ is frozen and all f-terms in B are Γ-frozen, then $\Gamma, x : B$ is frozen.*

Definition 5.3.4 (*size*)
Let $size(\Gamma, \pi) = size(\Gamma) + size(\pi)$ where:

- $size([]) = 0$
- $size(\Gamma, x : B) = size(\Gamma) + size(B)$
- $size(x) = size(g) = size(sort) = 1$ *if x is a variable and g a function symbol.*
- $size(u\ v) = size(\lambda x : u, v) = size(\Pi x : u, v) = 1 + size(u) + size(v)$

Lemma 5.3.5 (Substitution lemma)
For all $\pi, \Gamma, A, \bar{u}, z$, if Γ is frozen and in NF, π and A are Γ-frozen and in NF, $\Gamma \vdash \pi : A$ is provable with $z \notin FV(\pi, \Gamma, A, \bar{u})$, then $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \{f\bar{u} \to z\}\pi : \{f\bar{u} \to z\}A$ is provable.

Proof. We proceed by induction on the size of (Γ, π) (IH_1).
First note that $z : \iota, \{f\bar{u} \to z\}\Gamma$ is WF:

- $\Gamma = []$. Immediate.
- $\Gamma = H, x : B$. By inversion, $H \vdash B : sort$. By IH_1, $z : \iota, \{f\bar{u} \to z\}H \vdash \{f\bar{u} \to z\}B : sort$. Therefore, $z : \iota, \{f\bar{u} \to z\}\Gamma$ is WF.

Since π is in NF, there are only 5 cases to consider:

- $\pi = f\ \bar{u}$. Immediate.
- $\pi = TYPE$. $A = KIND$. Immediate.

- $\pi = \Pi x : B, C$:
  By inversion:

$$\frac{\Gamma \vdash B : TYPE \qquad \Gamma, x : B \vdash C : sort \qquad A \equiv sort}{\Gamma \vdash \Pi x : B, C : A}$$

  By IH_1: $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \{f\bar{u} \to z\}B : TYPE$. Since B is Γ-frozen, $\Gamma, x : B$ is frozen.
  Since C is Γ-frozen, C is $(\Gamma, x : B)$-frozen. Hence, by IH_1, $z : \iota, \{f\bar{u} \to z\}\Gamma, x : \{f\bar{u} \to z\}B \vdash \{f\bar{u} \to z\}C : sort$. Therefore, by $(prod)$, $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \Pi x : \{f\bar{u} \to z\}B, \{f\bar{u} \to z\}C : sort$.

- $\pi = \lambda x : B, u$:
  By inversion:

$$\frac{\Gamma, x : B \vdash u : C \qquad \Gamma \vdash \Pi x : B, C : sort}{\Gamma \vdash \lambda x : B, u : A}$$

  with $\Pi x : B, C \hookrightarrow^* A$ since A is in NF. Hence $A = \Pi x : B, C'$ where C' is the NF of C. Since B is Γ-frozen, $\Gamma, x : B$ is frozen. Since A is Γ-frozen, C' is Γ-frozen and thus $(\Gamma, x : B)$-frozen. By IH_1, $z : \iota, \{f\bar{u} \to z\}\Gamma, x : \{f\bar{u} \to z\}B \vdash \{f\bar{u} \to z\}u : \{f\bar{u} \to z\}C'$. Therefore, by (abs), $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \lambda x : \{f\bar{u} \to z\}B, \{f\bar{u} \to z\}u : \Pi x : \{f\bar{u} \to z\}B, \{f\bar{u} \to z\}C' = \{f\bar{u} \to z\}A$.

- $\pi = h\ w_1 \ldots w_n \neq f\ \bar{u}$ with h a variable or a function symbol:
  By inversion, there are $B, B_1, \ldots, B_n, C_1, \ldots, C_n$ such that:

  - $\Gamma \vdash h : B$
  - if $n > 0$, $B \equiv \Pi x_1 : B_1, C_1$
  - $\forall i \in \{1, \ldots, n\}, \Gamma \vdash w_i : B_i$
  - $\forall i \in \{1, \ldots, n\}, \Gamma \vdash h\ w_1..wi : [w_i/x_i]C_i$
  - $\forall i \in \{1, \ldots, n-1\}, C_i[w_i/x_i] \equiv \Pi x_{i+1} : B_{i+1}, C_{i+1}$
  - if $n > 0$ then $[w_n/x_n]C_n \equiv A$

  Let B' be the NF of B. If $h = x$, then $x : B' \in \Gamma$ (since Γ is in NF) and B' is Γ-frozen. If $h = g$, then B' is frozen since $f \notin type(g)$. Hence, by IH_1, $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \{f\bar{u} \to z\}h : \{f\bar{u} \to z\}B'$.
  If $n = 0$, then we are done ($B' = A$). So assume that $n > 0$.
  Let B'_i be the NF of B_i, and C'_i be the NF of C_i. We prove that, $\forall i \in$

$\{1,..,n\}, z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \{f\bar{u} \to z\}w_i : \{f\bar{u} \to z\}B'_i$ and $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash h\ w_1 \ldots w_i : \{f\bar{u} \to z\}[w_i/x_i]C'_i$ and C'_i is Γ-frozen, by induction on i (IH_2).

- $i = 1$:
  Since $B \equiv \Pi x_1 : B_1, C_1$, $B' = \Pi x_1 : B'_1, C'_1$. By IH_1, $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \{f\bar{u} \to z\}w_1 : \{f\bar{u} \to z\}B'_1$. Since $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash h : \Pi x_1 : \{f\bar{u} \to z\}B'_1, \{f\bar{u} \to z\}C'_1$, by (app), $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash h\{f\bar{u} \to z\}w1 : \{f\bar{u} \to z\}[\{f\bar{u} \to z\}x_1/x_1]C'_1$. Since B' is Γ-frozen, C'_1 is Γ-frozen. Hence, $\{f\bar{u} \to z\}[\{f\bar{u} \to z\}w_1/x_1]C'_1 = \{f\bar{u} \to z\}[w_1/x_1]C'_1$.
- $1 \leq i < n$:
  By IH_2, $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash h\{f\bar{u} \to z\}w_1 \ldots \{f\bar{u} \to z\}w_i : \{f\bar{u} \to z\}[w_i/x_i]C'_i$ and $[w_i/x_i]C'_i$ is Γ-frozen. By IH_1, $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \{f\bar{u} \to z\}w_{i+1} : \{f\bar{u} \to z\}B'_{i+1}$. Since $[w_i/x_i]C_i \equiv \Pi x_{i+1} : B_{i+1}, C_{i+1}$, $[w_i/x_i]C'_i \equiv \Pi x_{i+1} : B'_{i+1}, C'_{i+1}$. Since C'_i and w_i are Γ-frozen, $[w_i/x_i]C'_i$ and C'_{i+1} are Γ-frozen. Hence $\{f\bar{u} \to z\}[w_i/x_i]C'_i \equiv \Pi x_{i+1} : \{f\bar{u} \to z\}Bi', \{f\bar{u} \to z\}C'_{i+1}$. Therefore, by (app), $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash h\{f\bar{u} \to z\}w_1 \ldots \{f\bar{u} \to z\}w_{i+1} : [\{f\bar{u} \to z\}w_{i+1}/x_{i+1}]\{f\bar{u} \to z\}C'_{i+1}$. Since C'_{i+1} is Γ-frozen, $[\{f\bar{u} \to z\}w_{i+1}/x_{i+1}]\{f\bar{u} \to z\}C'_{i+1} = \{f\bar{u} \to z\}[w_{i+1}/x_{i+1}]C'_{i+1}$.

So $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \{f\bar{u} \to z\}\pi : \{f\bar{u} \to z\}[w_n/x_n]C'_n$ and C'_n is Γ-frozen. Since $[w_n/x_n]C_n \equiv A$ in NF, $C'_n[w_n/x_n] \hookrightarrow^* A$. Since C'_n and w_n are Γ-frozen, $[w_n/x_n]C'_n$ is Γ-frozen and $\{f\bar{u} \to z\}[w_n/x_n]C'_n \hookrightarrow^* \{f\bar{u} \to z\}A$. Therefore $z : \iota, \{f\bar{u} \to z\}\Gamma \vdash \{f\bar{u} \to z\}\pi : \{f\bar{u} \to z\}A$.

□

Proposition 5.3.6
Let Γ be a context that does not declare the variable y, B the translation of a $\mathcal{FOL}$ formula and $\bar{u}$ the translation of $\mathcal{FOL}$-arguments of f where $f\ \bar{u} \notin \Gamma, B$ and all f-terms are Γ-frozen. Assume that Γ contains $a : \epsilon(\forall \bar{x} \exists y A)$. If $\Gamma, a_{sku} : \epsilon([\bar{u}/\bar{x}][f\bar{x}/y]A) \vdash \pi : \epsilon B$ is provable then $\Gamma \vdash E(\Gamma, \bar{u}, B, \pi) : \epsilon B$ is provable.

Proof. If $\Gamma, a_{sku} : \epsilon([\bar{u}/\bar{x}][f\bar{x}/y]A) \vdash \pi : \epsilon B$ then by Lemma 5.3.5 we have $y : \iota, \{f\bar{u} \to y\}\Gamma, a_{sku} : \{f\bar{u} \to y\}\epsilon([\bar{u}/\bar{x}][f\bar{x}/y]A) \vdash \{f\bar{u} \to y\}\pi : \{f\bar{u} \to y\}\epsilon\ B$ which is equal to $y : \iota, \Gamma, a_{sku} : \epsilon([\bar{u}/\bar{x}]A) \vdash \{f\bar{u} \to y\}\pi : \epsilon B$ since $f\ \bar{u} \notin \Gamma, B$. By Lemma 1.3.9, we have $\Gamma, y : \iota, a_{sku} : \epsilon([\bar{u}/\bar{x}]A) \vdash \{f\bar{u} \to y\}\pi : \epsilon B$. Hence, $\Gamma \vdash \lambda y : \iota, \lambda a_{sku} : \epsilon([\bar{u}/x]A), \{f\ \bar{u} \to y\}\pi) : \Pi y : \iota, \epsilon([\bar{u}/x]A) \to \epsilon B$. We have $\Gamma \vdash a\ \bar{u} : \epsilon(\exists\ (\lambda y : \iota, [\bar{u}/x]A))$ because a :

$\epsilon(\forall \bar{x} \exists y A)$ belongs to Γ. Since $(\lambda y : \iota, [\bar{u}/x]A)\ y$ is convertible to $[\bar{u}/x]A$, we can therefore check that $\Gamma \vdash \exists_E\ (\lambda y : \iota, [\bar{u}/x]A)\ (a\ \bar{u})\ B\ (\lambda y : \iota, \lambda a_{sku} : \epsilon([\bar{u}/x]A), \{f\ \bar{u} \to y\}\pi) : \epsilon B$ □

Lemma 5.3.7
If $\Pi x_1 : A_1, \ldots \Pi x_n : A_n, A$ is the type of a constant of the signature Σ_{ND}, then forall i, A_i is a $\mathcal{FOL}$-type, and if A_i is an ϵ-term, then $x_i \notin FV(\Pi x_{i+1} : A_{i+1}, \ldots \Pi x_n : A_n, A)$.

Proof. By case on the type of the symbols of Σ_{ND}. The only non-trivial cases are:

- $\vee_E$, where $\epsilon p \to \epsilon x$ is an ϵ-term since it is convertible with $\epsilon(p \Rightarrow x)$ (and similarly for $\epsilon q \to \epsilon x$);
- $\Rightarrow_I$ where $\epsilon p \to \epsilon q$ is convertible with $\epsilon(p \Rightarrow q)$;
- $\forall_I$ where $\Pi x : \iota, \epsilon(p\ x)$ is convertible with $\epsilon(\forall\ (\lambda x : \iota, p\ x))$;
- $\exists_E$ where $\Pi x : \iota, \epsilon(p\ x) \to \epsilon P$ is convertible with $\epsilon(\forall\ (\lambda x : \iota, (p\ x \Rightarrow P)))$.

□

Lemma 5.3.8
If $\Pi x : A, B$ is an ϵ-term in NF, then A is either ι or an ϵ-term, and B is an ϵ-term. Moreover, if A is an ϵ-term, $x \notin FV(B)$.

Proof. By definition, there exists some C such that $\epsilon C \to^* \Pi x : A, B$. We therefore have $\epsilon C \to^*_\beta \epsilon C_1 \to_R \Pi x : A_1, B_1 \to^* \Pi x : A, B$. We have two cases depending on the rewriting rule that is used to rewrite ϵC_1 :

- C_1 is $D_1 \Rightarrow E_1$, then $\epsilon C_1 \to \epsilon D_1 \to \epsilon E_1$. By product compatibility, A is convertible to ϵD_1. Hence, A is an ϵ-term. Similarly, B is convertible to ϵE_1, thus it is an ϵ-term. Furthermore, since $x \notin FV(\epsilon E_1)$, we have $x \notin FV(B)$.
- C_1 is $\forall\ D_1$, then $\epsilon C_1 \to \Pi x : \iota, \epsilon(D_1 x)$. By product compatibility, A is convertible to ι. Hence, A is ι. Similarly, B is convertible to $\epsilon(D_1 x)$, thus it is an ϵ-term.

□

Corollary 5.3.9

If $\Pi x_1 : A_1, \ldots \Pi x_n : A_n, A$ *is the normal form of an* ϵ*-term, then forall* i, A_i *is a* $\mathcal{FOL}$*-type, and if* A_i *is an* ϵ*-term, then* $x_i \notin FV(\Pi x_{i+1} : A_{i+1}, \ldots \Pi x_n : A_n, A)$.

Lemma 5.3.10
If $\Gamma, a_{sk} : \hat{a}_{sk} \vdash h\ a_1\ \cdots\ a_n : \epsilon C$ *where* $h\ a_1\ \cdots\ a_n$ *is in normal form,* h *is a symbol of* Σ_{ND} *or a variable declared in* Γ, Γ *is a frozen* $\mathcal{FOL}$*-environment and* C *is* Γ*-frozen,*

if $\Gamma, a_{sk} : \hat{a}_{sk} \vdash a_i : \epsilon D$*, then* D *is not* $\forall x, E$ *where* E *contains an* f*-term which is not* Γ*-frozen.*

Proof. By case on h.

If h is a variable of Γ, since $\Gamma, a_{sk} : \hat{a}_{sk} \vdash a_i : \epsilon D$ we must have $\Gamma, a_{sk} : \hat{a}_{sk} \vdash h\ a_1\ \cdots\ a_{i-1} : \epsilon(D \Rightarrow F)$ for some F. By Proposition 4.1.1, $D \Rightarrow F$ must a subformula of $\Gamma, \hat{a}_{sk}, C$. If it is a subformula of Γ, C, then it cannot contain an f-term that is not frozen. Because $D \Rightarrow F$ does not start with $\forall$, it can only be a subformula of $\hat{a}_{sk}$ if it is a total instance of it, in which case it cannot contain an f-term that is not Γ-frozen. Hence, D does not contain such a term neither.

Consider the cases where h is a symbol.

It cannot be $\top_I$ or $=_I$ because no argument can have a type of the form ϵD. It cannot be $\Rightarrow_I$, $\Rightarrow_E$, $\forall_I$, nor $\forall_E$ because they can be rewritten in Σ^{ϵ}_{ND} which is included in Σ_{CC}.

If it is $\wedge_I$, $\vee_{Il}$, $\vee_{Ir}$, or $\exists_I$, if D contains an f-term which is not Γ-frozen, then so would C, hence a contradiction.

If it is $\wedge_{El}$, i cannot be 1 or 2 because the type of a_1 and a_2 must be $Prop$. i cannot be 3 because it must be $a_1 \wedge a_2$ and not $\forall x, E$. If i is 4, it means that a_1 is $P \Rightarrow Q$ for some P, Q of type $Prop$. As for the case where h is a variable, $P \Rightarrow Q$ is a subformula of $\Gamma, \hat{a}_{sk}, C$, so it cannot contain an f-term that is not frozen. This can be generalized if $i > 4$.

This is similar for $\wedge_{Er}$.

If h is $\vee_E$, then a_1 and a_2 must be subformulas of Γ, C. As above, i cannot be 1,2,3,4. It cannot be 5 or 6 because in that case D is of the form $a_i \Rightarrow a_4$ and not $\forall x, E$. If $i > 6$, then a_4 must be of the form $P \Rightarrow Q$ or $\forall x, R$. In both case, this would mean that $h\ a_1\ \cdots\ a_n$ is not in normal form because of the rules

$$\vee_E\ a\ b\ \pi_{a \vee b}\ (\_ \Rightarrow q)\ \pi_{apq}\ \pi_{bpq}\ \pi_p \hookrightarrow \vee_E\ a\ b\ \pi_{a \vee b}\ q\ (\lambda \pi_a, \pi_{apq}\ \pi_a\ \pi_p)\ (\lambda \pi_b, \pi_{bpq}\ \pi_b\ \pi_p)$$

$$\vee_E\ \pi_{a \vee b}\ (\forall x, a\ x)\ \pi_{apq}\ \pi_{bpq}\ t \hookrightarrow \vee_E\ \pi_{a \vee b}(a\ t)(\lambda \pi_a, \pi_{apq}\ \pi_a\ t)(\lambda \pi_b, \pi_{bpq}\ \pi_b\ t)$$

The reasoning is similar if h is $\exists_E$, with the following difference. If a_3 is C, then the type of a_4 is convertible to $\epsilon(\forall x, (a_1\ x) \Rightarrow C)$. Because of

the subformula property, $\exists a_1$ must be a subformula of $\Gamma, \hat{a}_{sk}, C$, so that its f-terms must be Γ-frozen as in the case where h is a variable. Hence, $(a_1\ x) \Rightarrow C$ does not contain f-terms that are not Γ-frozen. If a_3 is not C, then n must be strictly greater than 3, and a_3 must be of the form $P \Rightarrow Q$ or $\forall x, R$. In both case, this would mean that $h\ a_1\ \cdots\ a_n$ is not in normal form.

This is the same for $=_E$: for $i = 5$, $\epsilon(a_4\ a_1)$ cannot contain f-terms that are not Γ-frozen because $a_1 = a_2$ is a subformula of $\Gamma, \hat{a}_{sk}, C$. □

Theorem 5.3.11
If Γ is a $\mathcal{FOL}$-environment that is frozen, B is an ϵ-term Γ-frozen in NF, π is a term in NF, and $\Gamma, a_{sk} : \hat{a}_{sk} \vdash \pi : B$, then $\Gamma, a : \hat{a}, \Delta_{\Gamma;B} \vdash \psi(\Gamma; B; \pi) : B$.

Proof. Since B is an ϵ-term, it is convertible to ϵC for some C.

First consider the case where π is an abstraction $\lambda x : t, u$. Wlog, we can assume that $x \notin FV(a : \hat{a}, \Delta_{\Gamma,x:t;U})$. By inversion, B, which is in NF, is of the form $\Pi x : t, U$, and we have $\Gamma, a_{sk} : \hat{a}_{sk}, x : t \vdash u : U$. By Lemma 5.3.8, t is ι or an ϵ-term. Therefore, $a_{sk} \notin FV(t)$ and by Lemma 1.3.9, we have $\Gamma, x : t, a_{sk} : \hat{a}_{sk} \vdash u : U$. Furthermore, $\Gamma, x : t$ is a $\mathcal{FOL}$-environment. Since B is Γ-frozen, then $\Gamma, x : t$ is frozen and U is $\Gamma, x : t$-frozen. By Lemma 5.3.8 again, U is an ϵ-term. We can therefore apply the induction hypothesis to get $\Gamma, x : t, a : \hat{a}, \Delta_{\Gamma,x:t;U} \vdash \psi(\Gamma; U; u) : U$. By Lemma 1.3.9, $\Gamma, a : \hat{a}, \Delta_{\Gamma,x:t;U}, x : t \vdash \psi(\Gamma; U; u) : U$. Hence, $\Gamma, a : \hat{a}, \Delta_{\Gamma,x:t;U} \vdash \lambda x : t, \psi(\Gamma; U; u) : \Pi x : t, U$. By definition, $\Delta_{\Gamma,x:t;U}$ is the same as $\Gamma; \Pi x : t, U = \Delta_{\Gamma;B}$. By definition, $\psi(\Gamma; B; \pi) = \lambda x : t, \psi(\Gamma; U; u)$. Hence, $\Gamma, a : \hat{a}, \Delta_{\Gamma;B} \vdash \psi(\Gamma; B; \pi) : B$.

Otherwise, let π be $h\ a_1\ \cdots\ a_n$ where h is not an application. Since π is in normal form, h cannot be an abstraction. It cannot be $TYPE$, $KIND$, or a product, because such terms can only be typed by a sort and not an ϵ-term.

In the case where h is a function symbol of Σ^{ϵ}_{ND}, it has to be one of the symbols of Σ_{ND}, otherwise $h\ a_1\ \cdots\ a_n$ cannot be of type ϵC. Let $\Pi x_1 : V_1, \ldots \Pi x_n : V_n, W$ be the type of h. By uniqueness of types, ϵC is convertible with $[a_1/x_1, \ldots, a_n/x_n]W$. Let V'_i denote the normal form of $[a_1/x_1] \ldots [a_{i-1}/x_{i-1}]V_i$. We therefore have $\Gamma, a_{sk} : \hat{a}_{sk} \vdash a_i : V'_i$. Let us prove that for all i, we have $\Gamma, a : \hat{a}, \Delta_{\Gamma;V_i} \vdash \psi(\Gamma; V'_i; a_i) : V'_i$. By Lemma 5.3.7, each V_i is a $\mathcal{FOL}$-type. Therefore, each V'_i is a $\mathcal{FOL}$-type too. If it is not an ϵ-term, then $\psi(\Gamma; V'_i; a_i) = a_i$. Futhermore, a_{sk} cannot be used in a term of type $\iota \to \cdots \to \iota \to \iota$ or $\iota \to \cdots \to \iota \to Prop$. Thus $\Gamma \vdash \psi(\Gamma; V'_i; a_i) : V'_i$ and by weakening $\Gamma, a : \hat{a}, \Delta_{\Gamma;V_i} \vdash \psi(\Gamma; V'_i; a_i) : V'_i$. If V'_i is an ϵ-term, to apply the induction hypothesis, it remains to prove that V'_i is Γ-frozen. By Proposition 4.1.1, if V'_i is convertible to ϵD for some D, then D must be a subformula of $\Gamma, a_{sk} : \hat{a}_{sk}$ or C. If it is a subformula of $\hat{a}_{sk}$, then by Lemma 5.3.10,

D must be a total instance of $\hat{a}_{sk}$. Hence, its f-term are Γ-frozen. If it is a subformula of Γ or C, then all f-terms in V_i' are also f-terms in Γ or B, and since Γ is frozen and B is Γ-frozen, so is V_i'. By induction hypothesis, we therefore have $\Gamma, a : \hat{a}, \Delta_{\Gamma;V_i} \vdash \psi(\Gamma; V_i'; a_i) : V_i'$.

Let $\Delta' = \bigcup_{i=1}^{n} \Delta_{\Gamma;V_i'}$ ordered as in the algorithm. Let $a_i' = \psi(\Gamma; V_i'; a_i)$.

Let us prove by induction that $\Gamma, a : \hat{a}, \Delta' \vdash h\ a_1' \cdots a_i' : \Pi x_{i+1} : [a_1/x_1, \ldots, a_i/x_i]V_{i+1}, \ldots \Pi x_n : [a_1/x_1, \ldots, a_i/x_i]V_n, [a_1/x_1, \ldots, a_i/x_i]W$. It is trivial for $i = 0$. Assume by induction hypothesis that $\Gamma, a : \hat{a}, \Delta' \vdash h\ a_1' \cdots a_i' : \Pi x_{i+1} : [a_1/x_1, \ldots, a_i/x_i]V_{i+1}, \ldots \Pi x_n : [a_1/x_1, \ldots, a_i/x_i]V_n, [a_1/x_1, \ldots, a_i/x_i]W$. We know that $\Gamma, a : \hat{a}, \Delta' \vdash a_{i+1}' : V_{i+1}'$. If V_{i+1}' is not an ϵ-term, then $a_{i+1}' = a_{i+1}$. If it is an ϵ-term, then by Lemma 5.3.7, $x_{i+1} \notin FV(\Pi x_{i+2} : V_{i+2}, \ldots \Pi x_n : V_n, W)$. In both cases,

$$
\begin{aligned}
&[a_{i+1}'/x_{i+1}](\Pi x_{i+2} : [a_1/x_1, \ldots, a_i/x_i]V_{i+2}, \ldots \\
&\qquad \Pi x_n : [a_1/x_1, \ldots, a_i/x_i]V_n, [a_1/x_1, \ldots, a_i/x_i]W) \\
=&[a_{i+1}/x_{i+1}](\Pi x_{i+2} : [a_1/x_1, \ldots, a_i/x_i]V_{i+2}, \ldots \\
&\qquad \Pi x_n : [a_1/x_1, \ldots, a_i/x_i]V_n, [a_1/x_1, \ldots, a_i/x_i]W) \\
=&\Pi x_{i+2} : [a_1/x_1, \ldots, a_i/x_i, a_{i+1}/x_{i+1}]V_{i+2}, \ldots \\
&\qquad \Pi x_n : [a_1/x_1, \ldots, a_i/x_i, a_{i+1}/x_{i+1}]V_n, [a_1/x_1, \ldots, a_i/x_i, a_{i+1}/x_{i+1}]W
\end{aligned}
$$

Hence $\Gamma, a : \hat{a}, \Delta' \vdash h\ a_1' \cdots a_{i+1}' : \Pi x_{i+2} : [a_1/x_1, \ldots, a_{i+1}/x_{i+1}]V_{i+1}, \ldots \Pi x_n : [a_1/x_1, \ldots, a_{i+1}/x_{i+1}]V_n, [a_1/x_1, \ldots, a_{i+1}/x_{i+1}]W$.

Therefore, we have $\Gamma, a : \hat{a}, \Delta' \vdash h\ a_1' \cdots a_n' : [a_1/x_1, \ldots, a_n/x_n]W$. Since ϵC is convertible with $[a_1/x_1, \ldots, a_n/x_n]W$ we have $\Gamma, a : \hat{a}, \Delta' \vdash h\ a_1' \cdots a_n' : \epsilon C$.

Define Γ_j and π_j as in the algorithm. Let us prove by reverse induction that for all $0 \leq j \leq m$, $\Gamma_j \vdash \pi_j : \epsilon C$. Since $\pi_m = h\ a_1' \cdots a_n'$ and $\Gamma_m = \Gamma, a : \hat{a}, \Delta_{\Gamma;B}, \gamma_1 : \epsilon([\overline{w_1}/\bar{x}][f\bar{x}/y]A), \ldots, \gamma_m : \epsilon([\overline{w_m}/\bar{x}][f\bar{x}/y]A) = \Gamma, a : \hat{a}, \Delta'$, we have just proved that $\Gamma_m \vdash \pi_m : \epsilon C$. Assume by induction hypothesis that $\Gamma_j \vdash \pi_j : \epsilon C$. We know that $\Gamma_j = \Gamma_{j-1}, \gamma_j : \epsilon([\overline{w_j}/\bar{x}][f\bar{x}/y]A)$. Since $\epsilon([\overline{w_j}/\bar{x}][f\bar{x}/y]A)$ is not in $\Delta_{\Gamma;B}$, by definition $f\ \overline{w_j} \notin \Gamma, B$. Neither is it in $\hat{a}$. Because $size_f(w_k) \leq size_f(w_j)$ for all $k < j$, it is not in $\gamma_1 : \epsilon([\overline{w_1}/\bar{x}][f\bar{x}/y]A), \ldots, \gamma_{j-1} : \epsilon([\overline{w_{j-1}}/\bar{x}][f\bar{x}/y]A)$. Hence, it is not in Γ_{j-1}. By Proposition 5.3.6, $\Gamma_{j-1} \vdash E(\Gamma_{j-1}, \overline{w_j}, C, \pi_j) : \epsilon C$, so that by definition of π_{j-1} we have $\Gamma_{j-1} \vdash \pi_{j-1} : \epsilon C$.

Consequently, $\Gamma_0 \vdash \pi_0 : \epsilon C$. In other words, $\Gamma, a : \hat{a}, \Delta_{\Gamma;B} \vdash \psi(\Gamma; B; \pi) : \epsilon C$. By conversion, $\Gamma, a : \hat{a}, \Delta_{\Gamma;B} \vdash \psi(\Gamma; B; \pi) : B$.

In the case where h is a variable, its type must be an ϵ-term. If $h = a_{sk}$,

then n must be the size of $\bar{x}$, otherwise it would contradict the fact that B, which is convertible to $\epsilon([a_1/x_1, \ldots, a_n/x_n][f\bar{x}/y]A)$, is Γ-frozen. So C is a total instance of $\forall\bar{x}[f\bar{x}/y]A$. By definition of $\Delta_{\Gamma;B}$, this context contains a declaration $\alpha : \epsilon([\bar{a}/\bar{x}][f\bar{x}/y]A)$. Hence, by conversion $\Gamma, a : \hat{a}, \Delta_{\Gamma;B} \vdash \alpha : B$.

If $h \neq a_{sk}$, we can apply the same reasoning as in the function symbol case. The only difference is that we have to apply Corollary 5.3.9 instead of Lemma 5.3.7. □

5.4 SKonverto

The algorithm presented in this chapter is implemented in a tool named SKONVERTO[1]. The tool uses the encoding presented in Section 1.4 and produces a LAMBDAPI proof. In order to use this tool, we need to provide it some information about the proof that we want to transform. We will use the builtin mechanism of LAMBDAPI to provide this information since the tool shares the same parser and features of LAMBDAPI:

```
// The declaration of symbols used in the proof
constant symbol a : ι;
constant symbol p : ι → ι → Prop;
constant symbol s : ι → ι;
constant symbol f : ι → ι;
// The information transferred to SKonverto
builtin "Skolem"  ≔ f; // skolem symbol
builtin "Axiom"   ≔ A; // axiom using the skolem symbol
builtin "Formula" ≔ F; // formula of the proof that
                       // we want to transform
```

The axiom `A` is declared in the same file:

```
    symbol A : ϵ (∀ X, ∃ Y, (p X (s Y)));
```

Moreover, the proof that we want to transform is declared in the same file as well:

```
symbol F
 (ax_tran : ϵ (∀ X1, ∀ X2, ∀ X3, ((p X1 X2))
        ⇒ (((p X2 X3)) ⇒ ((p X1 X3)))))
 (ax_step : ϵ (∀ X1, (p X1 (s (f X1)))))
 (ax_congr : ϵ (∀ X1, ∀ X2, ((p X1 X2))
        ⇒ ((p (s X1) (s X2)))))
 (ax_goal : ϵ (¬ (∃ X4, ((p a (s (s X4))))))))
```

[1] Available at `https://github.com/elhaddadyacine/SKonverto`

```
  : ϵ ⊥
:= ax_goal (∃I (λ X4, p a (s (s X4))) (f (f a))
   (ax_tran a (s (f a)) (s (s (f (f a))))
     (ax_step a)
     (ax_congr (f a) (s (f (f a))) (ax_step (f a)))));
```

This proof will be de-skolemized and transformed into a proof without the occurrences of the symbol f that appear in red:

```
    symbol F
     (ax_tran : ϵ (∀ X1, ∀ X2, ∀ X3, ((p X1 X2))
            ⇒ (((p X2 X3)) ⇒ ((p X1 X3)))))
     (ax_congr : ϵ (∀ X1, ∀ X2, ((p X1 X2))
            ⇒ ((p (s X1) (s X2)))))
     (ax_goal : ϵ (¬ (∃ X4, ((p a (s (s X4)))))))
      : ϵ ⊥
    := ax_goal (λ r h, ∃E (λ z, p a (s z)) (A a) r
        (λ z α1, ∃E (λ z0, p z (s z0)) (A z) r
        (λ z0 α2, h z0 (ax_tran a (s z) (s (s z0)) α1
            (ax_congr z (s z0) α2)))));
```

Remark 5.4.1
Notice that the axiom `ax_step` *is removed from the transformed proof and replaced by the axiom* `A` *where the existential quantifier is not eliminated yet. Thus the axiom* `A` *is not skolemized and does not contain the Skolem symbol* f

Chapter 6

Related Works

We hereafter discuss various related works.

6.1 SledgeHammer

SledgeHammer [BN10] is a tool created for the proof assistant Isabelle. it is used to discharge some goals in Isabelle. It calls **E**, VAMPIRE, SPASS [WDF+09] and SMT Solvers to solve the current goal, and if one of the provers responds, then Sledgehammer calls the Isabelle internal prover Metis to reconstruct a proof inside Isabelle by using information obtained from the called prover. The information collected from the provers are mainly the axioms used and how the proof was built. Metis uses this information to find a proof. This means that, if the Metis calculus changes, SledgeHammer needs to be changed too since it relies on it. In contrast, EKSTRAKTO does not rely on the way *ZenonModulo* or *ArchSAT* and only uses their output to prove the subgoals as presented in Chapter 3.

6.2 CoqHammer

CoqHammer [CK18] is a tool made for Coq to discharge goals. It is split in three parts:

- Axiom selection: The tool can learn from proofs that are already done and could provide a set of axioms that are susceptible to be used in one of the proofs of the current goal.
- Translation of goals: CoqHammer translates its goals from the Cal-

culus of Inductive Constructions, which is the logic of Coq, into the logic of the prover used with CoqHammer.

- Proof reconstruction: The approach used by CoqHammer to reconstruct proofs is similar to SledgeHammer. However, CoqHammer does not use any external prover, it uses the tactic language of Coq to perform this reconstruction. It mainly uses the Coq tactic **sauto**, a super version of **auto**, which uses a specific proof search procedure.

6.3 SMTCoq

SMTCoq [AFG+11] is a plugin implemented inside the proof assistant Coq that verifies proofs generated by SAT and SMT solvers [BBP11]. It can be used as a tactic inside Coq to discharge some goals. The advantage of SMTCoq is its modularity in the sense that it can use various provers : veriT, CVC4 and zChaff [MMZ+01]. These provers are not compatible with each other and do not share the same input and output formats (zChaff for example accepts only boolean formulas and does not produce SMT-Lib format as veriT and CVC4). Since SMTCoq checks the proofs generated by SAT and SMT solvers, it only checks unsatisfiability proofs.

The link between Coq and SMTCoq is done by tactics, where each tactic added to Coq by SMTCoq calls a specific prover like CVC4 or zChaff to solve the current goal of the Coq environment. If the external prover finds an unsatisfiability proof, then SMTCoq will translate this proof from SAT/SMT format to Coq in order to apply it on the current proof state.

6.4 Proof reconstruction in veriT

veriT [BdODF09] is an SMT Solver that uses the SMT-Lib [BFT16] format to represent its proofs. An approach is introduced in [FS19] to reconstruct veriT proofs in Isabelle/HOL. The approach is similar to the one used in EKSTRAKTO since it treats each step of the SMT-Lib format as an Isabelle theorem with some exceptions (e.g. Skolemeziation steps). The main difference resides in the format used, which is TSTP for EKSTRAKTO and SMT-Lib for veriT. Another distinction is the subproofs generation which is managed internally in veriT, and delegated to a prover in the case of EKSTRAKTO. Finally, EKSTRAKTO does not handle Skolemization and assume that the TSTP proof taken as input does not contain Skolem symbols since the original problem has already been Skolemized, and the proof produced

by EKSTRAKTO will be transformed (De-Skolemized) with SKONVERTO, the tool presented in Chapter 5.

6.5 Hol(y) Hammer

HOL(y)Hammer [KU15] is a tool designed for HOL4 [SN08] and HOL-Light [Har09]. It helps the proof assistant to select a subset of axioms that the proof assistant provide. This set of axioms is susceptible to contain some axioms that could be applied to the current goal. The tool also translates the goal and the axioms to the logic used in the ATPs, as it is done in SledgeHammer. However, HOL(y)Hammer uses machine learning to select axioms from the proof assistant library, which is not the case of SledgeHammer.

6.6 TacticToe

TacticToe [GKU+20] is a tactical prover[1] used in the HOL4 theorem prover. It uses machine learning techniques to provide the user with a suitable tactic that can be used to discharge the current goal. It is split in two parts: learning and proving. It learns/trains from a provided human proofs database and tries to predict a set of tactics. The provided tactics may fail to prove the current goal, so a proof search is added to the tool. This tool can be used with Metis [Hur] to help it select its axioms before launching its proof search. It is not the case of EKSTRAKTO since it directly uses the axioms found in the TSTP file.

6.7 TSTP derivation checker

GDV [SUT06] is a derivation checker for the TSTP format. It checks if the DAG (directed acyclic graph) of a TSTP file is correct by verifying that every node of the DAG has a correct derivation or not. It is very similar to EKSTRAKTO on one side but differs in two important aspects:

- First, the GDV tool uses an internal prover to check the validity of every inference present in the TSTP file, which is not the case of EKSTRAKTO since it uses any prover that supports LAMBDAPI (*Zenon-Modulo* and *ArchSAT*).

[1] Automated tool that uses tactical language as proof calculus

- Second, the tool does not provide any witness for the validity of the TSTP and rely only on the prover used to prove the steps. However, EKSTRAKTO provides a proof in LAMBDAPI that can be translated and cross-checked by various proof assistants.

6.8 TESC proof format

TESC (Theory Extensible Sequent Calculus) [Bae20] is a low-level proof format for first-order automated provers. The tools around this format permit to check the output of provers that provide TSTP format. It can be seen as a witness format for the TSTP format. It is based on sequent calculus and supports Skolemization steps. However, the tools around this format differ from EKSTRAKTO in how they handle Skolemization. As presented in [Bae20], this format allows the use of the axiom of choice, which facilitates to prove Skolemization steps. This is not the case of LAMBDAPI, since it does not have this feature. We could add the axiom of choice to LAMBDAPI and enjoy this feature, but the aim of LAMBDAPI is to be as small as possible to have a safer trusted base. Another difference resides in the proof format itself since many translators have already been developed to translate LAMBDAPI proofs to other proof assistants and gain more insurance.

6.9 Certifying Why3 transformations

Why3 is a platform that calls ATPs to prove subgoals. Before calling a prover, the Why3 platform performs some transformations on the formula that represents the goal. These transformations are done at various levels. They could arrive just before calling the provers since a translation of the formula is necessary to be accepted by the input language of the provers. These transformations also could appear inside the Why3 platform to simplify the goal given to the provers. However, these transformations should be justified. Otherwise, we may end up with a formula that is not equivalent to the original one, and the provers will provide an answer that corresponds to a different goal. In [GKMP], the authors define a mechanism to certify these transformations using DEDUKTI or LAMBDAPI as a witness format. The approach used is similar to the one in EKSTRAKTO in the sense that every transformation performed by Why3 is declared as a hole that needs to be certified.

6.10 ProofCert

ProofCert is a project that aims at building a universal proof checker for theorem provers. The goal is to define theorems and proofs of various theorem provers that produces proof objects (proofs with explicit information) in the same framework. It is based on a set of inference rules, that is a sequent calculus for classical logic. The choice of this proof system is motivated by its minimal number of inference rules which leads to have a good trust base for this proof checker. It is somewhat similar to the LOGIPEDIA project [DT19] which is a large library for proof systems that uses DEDUKTI as a framework to translate proofs from one system into another.

6.11 Certifying Skolemization

In [CMM19], the authors show a way to check proofs containing Skolemization steps. The approach proposed is very similar to what is presented in Chapter 5 and does not use the axiom of choice like the solution described in Section 6.8. It is a direct approach to certify proofs coming from automated provers by using a mapping that goes from Skolem symbols to a set of variables called eigenvariables. However, the approach proposed diverges with what is presented in this thesis on two aspects:

- The system used to check proofs containing Skolem symbols is the sequent calculus and not natural deduction.
- Our approach transforms the proof containing Skolem symbols into a new proof without these Skolem symbols, i.e., we do not change the proof checker to check proofs with Skolem symbols.

6.12 LFSC

LFSC (Logical Framework with Side Conditions) [SOR+12] is an extension of LF [HHP93b] which is the same as the $\lambda\Pi$-calculus. It is a proof checker designed to check SMT proofs. It lets the user add rules that define its proof system. It is similar to DEDUKTI but for SMT solvers. The use of side conditions offers a good level of expressivity and computation, making him fast for type-checking and also for covering many encodings of theories that SMT format accepts.

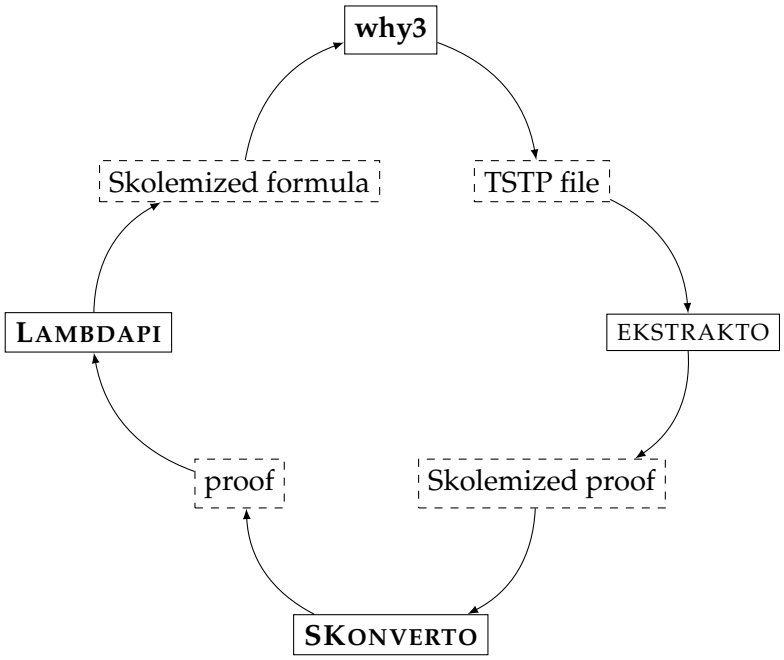
Chapter 7

Future Work

7.1 Workflow

The tools presented in this thesis are developed so that they can be used with each other. We will show here a practical use case where we start with a formula in LAMBDAPI, and we get a proof of that formula inside LAMBDAPI:

- We translate the given formula into first-order logic by using the translation presented in Chapter 2.
- We skolemize the formula in order to avoid having skolemization steps in the future.
- We give the skolemized formula to Why3 (by using the `why3` tactic) and wait for a prover to find a proof and return it in TSTP format.
- We use EKSTRAKTO on the TSTP file generated by the prover and we reconstruct a proof in LAMBDAPI using *ZenonModulo* and *ArchSAT*.
- The reconstructed proof represents a proof of the skolemized version of the original formula. We then use SKONVERTO to remove the Skolem symbol and get a proof of the original formula.



As shown in this diagram, the proof that we obtain could be a proof of another formula since we may alter the original formula to perform every step (e.g. Why3 could transform the given formula before calling the prover). Thus, we need to justify every transformation done to the formula to have an applicable proof. A solution is presented in [GKMP] where the authors show that the chain of transformations that we obtain at the end will be presented as a term in DEDUKTI or LAMBDAPI.

7.2 Arithmetic and Polymorphism

The tactic presented in Chapter 2 translates $\lambda\Pi/\equiv$ formulas into first-order formulas. The tactic could be extended to manage arithmetic since the Why3 platform language handles it. However, an adjustment is required to the proof of correctness of the extended translation. The tactic could also be extended to accept polymorphism by changing the encoding. Rather than having only one generic type, which is ι, we can add a new symbol τ that takes as an argument a type code and return a type. The new encoding for

first-order logic will be changed as follows:

$$
\begin{aligned}
&\Sigma^{Poly}_{\mathcal{FOL}} = \\
&\quad Type : TYPE, \\
&\qquad \tau : Type \to TYPE, \\
&\qquad \iota : Type, \\
&\qquad f : \Pi\alpha_1, \ldots, \Pi\alpha_m, \tau\ t_1 \to \ldots \to \tau\ t_n \to \tau\ t_{n+1}, \\
&\quad Prop : TYPE, \\
&\qquad P : \Pi\alpha_1, \ldots, \Pi\alpha_m, \tau\ t_1 \to \ldots \to \tau\ t_n \to Prop, \\
&\qquad \top : Prop, \\
&\qquad \bot : Prop, \\
&\qquad \neg : Prop \to Prop, \\
&\qquad \wedge : Prop \to Prop \to Prop, \\
&\qquad \vee : Prop \to Prop \to Prop, \\
&\qquad \Rightarrow : Prop \to Prop \to Prop, \\
&\qquad \Leftrightarrow : Prop \to Prop \to Prop, \\
&\qquad \forall : \Pi\alpha, (\tau\ \alpha \to Prop) \to Prop, \\
&\qquad \exists : \Pi\alpha, (\tau\ \alpha \to Prop) \to Prop, \\
&\qquad = : \Pi\alpha, \tau\ \alpha \to \tau\ \alpha \to Prop, \\
&\qquad \epsilon : Prop \to TYPE
\end{aligned}
$$

where $t_1, \ldots, t_{n+1}$ are terms of type *Type* which could contain the variables $\alpha_1, \ldots, \alpha_m$ (i.e., for every $i \in \{1, \ldots, n+1\}$ then $\alpha_1, \ldots, \alpha_m \vdash_{\Sigma^{Poly}_{fol}} t_i : Type$).

7.3 More steps in a TSTP file

As described in our experiments 3.2, we faced a problem with the reconstruction of VAMPIRE TSTP files due to the definition introduced by VAMPIRE. We could avoid this problem by adding a new step to prove by *Zenon-Modulo* or *ArchSAT*. This is an example of a TSTP file containing a new definition:

HWV045-2.p

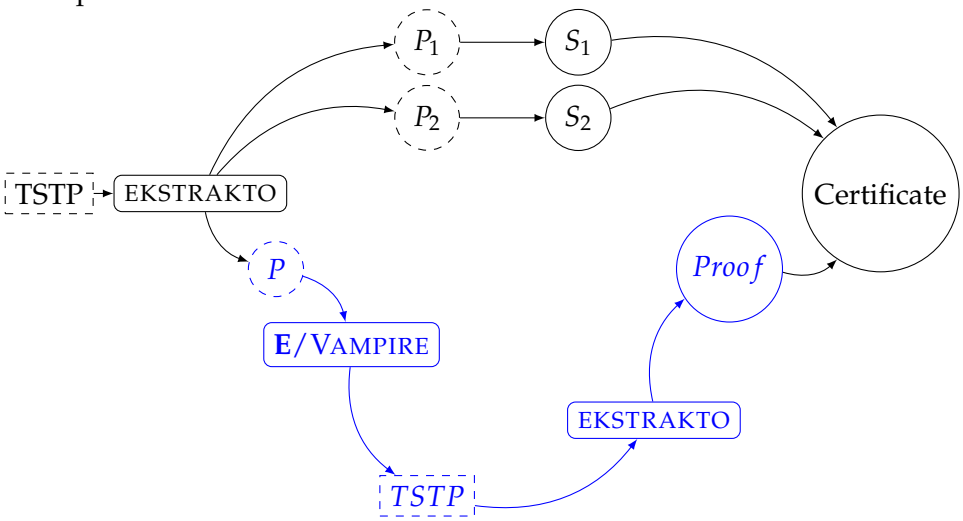
```
fof(f119619,plain,(
    spl0_5174 <=> v124(constB179)),
```

```
        introduced(avatar_definition,
            [new_symbols(naming,[spl0_5174])])).
```

From this definition, we can add the formula `v124(constB179)` as a step to prove by adding the axiom `spl0_5174 : Prop := v124(constB179)` to our signature file.

7.4 Proving steps with high-level provers

EKSTRAKTO reconstructs proofs from TSTP files by declaring each step as a TPTP problem. However, *ZenonModulo* and *ArchSAT* could fail to prove this TPTP problem. A solution could be the re-use of a high-level prover such as **E** or VAMPIRE to prove this step and re-use EKSTRAKTO again to have the proof in LAMBDAPI:



The blue part of the diagram represents the solution proposed. This solution could only work if the TSTP file of the new step contains more information than the original TSTP file. Otherwise, we will always have the same step to prove. We can bypass this by giving the prover more or fewer axioms to prove the step and assuming that it will find a different proof that could be reconstructed by EKSTRAKTO. Another idea is to tell the prover not to use the axioms to prove the current step, but this solution requires changing the prover if it does not provide this feature.

Chapter 8

Appendices

8.1 File `fol.lp` for $\Sigma_{\mathcal{FOL}}$

```
// encoding of intuitionistic first-order logic

constant symbol κ : TYPE;

constant symbol Prop : TYPE;
symbol ⊤ : Prop;
symbol ⊥ : Prop;
symbol ∧ : Prop → Prop → Prop; notation ∧ infix right 10;
symbol ∨  : Prop → Prop → Prop; notation ∨ infix right 9;
symbol ⇒ : Prop → Prop → Prop; notation ⇒ infix right 5;
symbol = : κ → κ → Prop; notation = infix 12;
symbol ∀ : (κ → Prop) → Prop; notation ∀ quantifier;
symbol ∃ : (κ → Prop) → Prop; notation ∃ quantifier;

symbol ¬ p ≔ p ⇒ ⊥; notation ¬ prefix 20;
symbol ⇔ p q ≔ (p ⇒ q)∧(q ⇒ p); notation ⇔ infix left 10;
symbol ≠ t u ≔ ¬ (t = u); notation ≠ infix 12;

symbol ϵ : Prop → TYPE;
```

8.2 File `nd.lp` for Σ_{ND}

```
// deep encoding of natural deduction proofs

require open logic.fol;

symbol ⊥E : ϵ ⊥ → Π r, ϵ r;
```

```
symbol ⊤I : ϵ ⊤;

symbol ∧I p q : ϵ p → ϵ q → ϵ (p ∧ q);
symbol ∧El p q : ϵ (p ∧ q) → ϵ p;
symbol ∧Er p q : ϵ (p ∧ q) → ϵ q;

symbol ∨Il p q : ϵ p → ϵ (p ∨ q);
symbol ∨Ir p q : ϵ q → ϵ (p ∨ q);
symbol ∨E p q :
  ϵ (p ∨ q) → Π r, (ϵ p → ϵ r) → (ϵ q → ϵ r) → ϵ r;

symbol ⇒I p q : (ϵ p → ϵ q) → ϵ (p ⇒ q);
symbol ⇒E p q : ϵ (p ⇒ q) → ϵ p → ϵ q;

symbol =I t : ϵ (t = t);
symbol =E t u : ϵ (t = u) → Π p, ϵ (p t) → ϵ (p u);

symbol ∀I p : (Π x, ϵ (p x)) → ϵ (∀ p);
symbol ∀E p x : ϵ (∀ p) → ϵ (p x);

symbol ∃I p x : ϵ (p x) → ϵ (∃ p);
symbol ∃E p : ϵ (∃ p) → Π r, (Π x, ϵ (p x) → ϵ r) → ϵ r;
```

8.3 File `classic.lp` for classical logic

```
// classical logic

require open logic.fol;

symbol nnpp p : ϵ (¬ ¬ p) → ϵ p;
```

8.4 File `nd_eps.lp` for Σ^{ϵ}_{ND}

```
// semi-shallow embedding of natural deduction proofs

require open logic.fol logic.nd;

rule ϵ ($p ⇒ $q) ↪ ϵ $p → ϵ $q;
rule ϵ (∀ $p) ↪ Π x, ϵ ($p x);

rule ⇒I ↪ λ p q π, π;
rule ⇒E ↪ λ p q π, π;
```

```
rule ∀I ↪ λ p π, π;
rule ∀E ↪ λ p x π, π x;
```

8.5 File nd_eps_full.lp for $\Sigma_{ND}^{\epsilon,full}$

```
// shallow embedding of natural deduction proofs

require open logic.fol logic.nd logic.nd_eps;

rule ϵ ⊥ ↪ Π r, ϵ r;
rule ϵ ⊤ ↪ Π r, ϵ r → ϵ r;
rule ϵ ($p ∨ $q) ↪ Π r, ϵ (($p ⇒ r) ⇒ ($q ⇒ r) ⇒ r);
rule ϵ ($t = $u) ↪ Π r, ϵ (r $t ⇒ r $u);
rule ϵ ($p ∧ $q) ↪ Π r, ϵ (($p ⇒ $q ⇒ r) ⇒ r);
rule ϵ (∃ $p) ↪ Π r, ϵ ((∀ (λ x, $p x ⇒ r)) ⇒ r);

rule ⊤I ↪ λ p πp, πp;

rule ⊥E ↪ λ π⊥ p, π⊥ p;

rule ∧I ↪ λ p q πp πq r πp⇒q⇒r, πp⇒q⇒r πp πq;
rule ∧El ↪ λ p q πp∧q, πp∧q p (λ x _, x);
rule ∧Er ↪ λ p q πp∧q, πp∧q q (λ _ x, x);

rule ∨Il ↪ λ p q πp r πp⇒r _, πp⇒r πp;
rule ∨Ir ↪ λ p q πq r _ πq⇒r, πq⇒r πq;
rule ∨E ↪ λ p q π, π;

rule =I ↪ λ t, λ r, λ π, π;
rule =E ↪ λ t u πt=u p πpt, πt=u p πpt;

rule ∃I ↪ λ p t πpt r h, h t πpt;
rule ∃E ↪ λ p π∃p, π∃p;
```

8.6 File ll.lp for Σ_{LL}

```
// deep encoding of Zenon LL proofs

require open logic.fol;

symbol R⊥ : ϵ ⊥ → ϵ ⊥;
symbol R¬⊤ : ϵ (¬ ⊤) → ϵ ⊥;
symbol Rax p : ϵ p → ϵ (¬ p) → ϵ ⊥;
symbol R≠ t : ϵ (¬ (t = t)) → ϵ ⊥;
```

```
symbol R= t u : ϵ (t = u) → ϵ (¬ (u = t)) → ϵ ⊥;
symbol Rcut p : (ϵ p → ϵ ⊥) → (ϵ (¬ p) → ϵ ⊥) → ϵ ⊥;
symbol R¬¬ p : (ϵ p → ϵ ⊥) → ϵ (¬ (¬ p)) → ϵ ⊥;
symbol R∧ p q : (ϵ p → ϵ q → ϵ ⊥) → ϵ (p ∧ q) → ϵ ⊥;
symbol R∨ p q :(ϵ p → ϵ ⊥) → (ϵ q → ϵ ⊥) → ϵ(p ∨ q) → ϵ ⊥;
symbol R⇏ p q :
  (ϵ (¬ p) → ϵ ⊥) → (ϵ q → ϵ ⊥) → ϵ (p ⇒ q) → ϵ ⊥;
symbol R⇔ p q : (ϵ (¬ p) → ϵ (¬ q) → ϵ ⊥)
  → (ϵ p → ϵ q → ϵ ⊥) → ϵ (p ⇔ q) → ϵ ⊥;
symbol R¬∧ p q :
  (ϵ (¬ p) → ϵ ⊥) → (ϵ (¬ q) → ϵ ⊥) → ϵ (¬ (p ∧ q)) → ϵ ⊥;
symbol R¬∨ p q :
  (ϵ (¬ p) → ϵ (¬ q) → ϵ ⊥) → ϵ (¬ (p ∨ q)) → ϵ ⊥;
symbol R¬⇏ p q :(ϵ p → ϵ (¬ q) → ϵ ⊥) → ϵ (¬ (p ⇒ q)) → ϵ
⊥ ;
symbol R¬⇔ p q :
  (ϵ (¬ p) → ϵ q → ϵ ⊥) → (ϵ p → ϵ (¬ q) → ϵ ⊥)
  → ϵ (¬ (p ⇔ q)) → ϵ ⊥;
symbol R∃ p : (Π x, ϵ (p x) → ϵ ⊥) → ϵ (∃ p) → ϵ ⊥;
symbol R∀ p t : (ϵ (p t) → ϵ ⊥) → ϵ (∀ p) → ϵ ⊥;
symbol R¬∃ p t : (ϵ (¬ (p t)) → ϵ ⊥) → ϵ (¬ (∃ p)) → ϵ ⊥;
symbol R¬∀ p : (Π x, ϵ (¬ (p x))→ ϵ ⊥) → ϵ(¬ (∀ p)) → ϵ ⊥;
symbol Rσ p t u :
  (ϵ (¬ (t = u)) → ϵ ⊥) → (ϵ (p u) → ϵ ⊥) → ϵ (p t) → ϵ ⊥;
symbol R≡r p t u :
  (ϵ (p u) → ϵ ⊥) → ϵ (p t) → ϵ (t = u) → ϵ ⊥;
symbol R≡l p t u :
  (ϵ (p u) → ϵ ⊥) → ϵ (p t) → ϵ (u = t) → ϵ ⊥;
```

8.7 File nd_eps_aux.lp for $\Sigma_{ND}^{\epsilon,aux}$

```
// auxiliary meta-theorems of first-order logic

require open logic.fol logic.nd logic.nd_eps;

opaque symbol Lcontraposition p q
: ϵ (p ⇒ q) → ϵ (¬ q ⇒ ¬ p) ≔ λ h1 h2 h3, h2 (h1 h3);

opaque symbol L⇔1 p q
: (ϵ q → ϵ p) → (ϵ p → ϵ (¬ q)) → ϵ q → ϵ (¬ q)
≔ λ h5 h2 hq, h2 (h5 hq);

opaque symbol L⇔2 p q
: (ϵ q → ϵ p) → (ϵ p → ϵ (¬ q)) → ϵ (¬ q)
```

```
≔ λ h5 h2 hq, L⇔1 p q h5 h2 hq hq;

opaque symbol L⇔3 p q
: (ϵ q → ϵ p) → (ϵ p → ϵ (¬ q)) → (ϵ p → ϵ q) → ϵ (¬ p)
≔ λ h5 h2 h4, Lcontraposition p q h4 (L⇔2 p q h5 h2);

opaque symbol L⇔4 p q
: (ϵ q → ϵ p) → (ϵ p → ϵ (¬ q)) → (ϵ p → ϵ q)
  → (ϵ (¬ p) → ϵ (¬ ¬ q)) → ϵ (¬ ¬ q)
≔ λ h5 h2 h4 h1, h1 (L⇔3 p q h5 h2 h4);

opaque symbol L¬∨1 p q : ϵ p → ϵ (p ∨ q) ≔ ∨Il p q;

opaque symbol L¬∨2 p q : ϵ q → ϵ (p ∨ q) ≔ ∨Ir p q;

opaque symbol L¬∨3 p q : ϵ (¬ (p ∨ q)) → ϵ (¬ p)
≔ λ h2, Lcontraposition p (p ∨ q) (L¬∨1 p q) h2;

opaque symbol L¬∨4 p q : ϵ (¬ (p ∨ q)) → ϵ (¬ q)
≔ λ h2, Lcontraposition q (p ∨ q) (L¬∨2 p q) h2;

opaque symbol L¬⇒1 p q : ϵ (¬ (p ⇒ q)) → ϵ (¬ q)
≔ λ h2 h3, h2 (λ _, h3);

opaque symbol L¬⇒2 p q
: (ϵ p → ϵ (¬ ¬ q)) → ϵ (¬ (p ⇒ q)) → ϵ (¬ p)
≔ λ h1 h2 h3, h1 h3 (L¬⇒1 p q h2);

opaque symbol L¬⇒3 p q : ϵ (¬ p) → ϵ (p ⇒ q)
≔ λ h3 h4, ⊥E (h3 h4) q;

opaque symbol L¬⇔1 p q
: (ϵ p → ϵ (¬ ¬ q)) → ϵ (¬ (p ⇔ q)) → ϵ (¬ p) ≔ λ h2 h3 hp,
h2 hp (λ hq, h3 (∧I (p ⇒ q) (q ⇒ p) (λ _, hq) (λ _, hp)));
```

8.8 File `ll_nd.lp` for Σ^{LL}_{ND}

```
// translation of Zenon LL proofs into natural deduction proofs

require open logic.fol logic.ll logic.nd logic.nd_eps
             logic.nd_eps_aux logic.classic;

rule R⊥ ↪ λ x, x;
```

```
rule R¬⊤ ↪ λ h, h ⊤I;

rule Rax ↪ λ p h π¬p, π¬p h;

rule R≠ ↪ λ t h1, h1 (=I t);

rule R= ↪ λ t u h1 h2, h2 (=E t u h1 (λ x, x = t) (=I t));

rule Rcut ↪ λ p h1 h2, h2 h1;

rule R¬¬ ↪ λ p h1 h2, h2 h1;

rule R∧ ↪ λ p q h1 h2, h1 (∧El p q h2) (∧Er p q h2);

rule R∨ ↪ λ p q h1 h2 h3, ∨E p q h3 ⊥ h1 h2;

rule R⇒ ↪ λ p q h1 h2 h3, h1 (Lcontraposition p q h3 h2);

rule R⇔ ↪ λ p q h1 h2 h3,L⇔4 p q (∧Er (p ⇒ q) (q ⇒ p) h3)
  h2 (∧El (p ⇒ q) (q ⇒ p) h3) h1
  (L⇔2 p q (∧Er (p ⇒ q) (q ⇒ p) h3) h2);

rule R¬∧ ↪ λ p q h1 h2 h3,
  h1 (λ h5, h2 (λ h6, h3 (∧I p q h5 h6)));

rule R¬∨ ↪ λ p q h1 h2, h1 (L¬∨3 p q h2) (L¬∨4 p q h2);

rule R¬⇒ ↪ λ p q h1 h2,
  h2 (λ h3, ⊥E (h1 h3 (λ h4, h2 (λ _, h4))) q);

rule R¬⇔ ↪ λ p q h1 h2 h3,
  (λ hnp, h3 (∧I (p ⇒ q) (q ⇒ p) (λ hp, ⊥E (hnp hp) q)
  (λ hq, ⊥E (h1 hnp hq) p))) (L¬⇔1 p q h2 h3);

rule R∃ ↪ λ p h1 h2, ∃E p h2 ⊥ h1;

rule R∀ ↪ λ p t h1 h2, h1 (h2 t);

rule R¬∃ ↪ λ p t h1 h2, h1 (λ h4, h2 (∃I p t h4));

rule R¬∀ ↪ λ p h1 h2, h2 (λ t, nnpp (p t) (h1 t));

rule Rσ ↪ λ p t u h1 h2 h3, h1 (λ h4, h2 (=E t u h4 p h3));
```

8.9 File `cc.lp` for commutative cuts

```
require open logic.fol logic.nd logic.nd_eps;

rule ∨E $a _ (∨Il $a _ $pa) _ $h _ ↪ $h $pa;
rule ∨E _ $b (∨Ir _ $b $pb) _ _ $h ↪ $h $pb;
rule ∨E $a $b $paorb (_ ⇒ $q) $papq $pbpq $pp ↪
    ∨E $a $b $paorb $q
        (λ pa, $papq pa $pp) (λ pb, $pbpq pb $pp);
rule ∨E $a $b $paorb (∀ $r) $papq $pbpq $t ↪
    ∨E $a $b $paorb ($r $t)
        (λ pa, $papq pa $t) (λ pb, $pbpq pb $t);
rule ∨E $c $d (∨E $a $b $paorb _ $pacord $pbcord)
    $e $pce $pde ↪ ∨E $a $b $paorb $e
            (λ pa, ∨E $c $d ($pacord pa) $e $pce $pde)
            (λ pb, ∨E $c $d ($pbcord pb) $e $pce $pde);
rule ∨E $a $b (∃E $p $pexp _ $ppt) $e $pce $pde ↪
    ∃E $p $pexp $e
        (λ x pp, ∨E $a $b ($ppt x pp) $e $pce $pde);

rule =E $t $t (=I _) _ $pt ↪ $pt;
rule =E $t $v (∨E $a $b $paorb _ $patv $pbtv) $P $pPt
   ↪ ∨E $a $b $paorb ($P $v)
        (λ pa, =E $t $v ($patv pa) $P $pPt)
        (λ pb, =E $t $v ($pbtv pb) $P $pPt);
rule =E $t $v (∃E $p $pexp _ $ppt) $P $pPt ↪
    ∃E $p $pexp ($P $v)
        (λ x pp, =E $t $v ($ppt x pp) $P $pPt);

rule ∧El $a _ (∧I $a _ $pa _) ↪ $pa;
rule ∧El $c $d (∨E $a $b $paorb _ $pat $pbt) ↪
    ∨E $a $b $paorb $c (λ pa, ∧El $c $d ($pat pa))
        (λ pb, ∧El $c $d ($pbt pb));
rule ∧El $c $d (∃E $p $pexp _ $ppt) ↪
    ∃E $p $pexp $c (λ x pp, ∧El $c $d ($ppt x pp));

rule ∧Er _ $b (∧I _ $b _ $pb) ↪ $pb;
rule ∧Er $c $d (∨E $a $b $paorb _ $pat $pbt) ↪
    ∨E $a $b $paorb $d (λ pa, ∧Er $c $d ($pat pa))
        (λ pb, ∧Er $c $d ($pbt pb));
rule ∧Er $c $d (∃E $p $pexp _ $ppt) ↪
    ∃E $p $pexp $d (λ x pp, ∧Er $c $d ($ppt x pp));

rule ∃E $p (∃I $p $t $pt) _ $pxpxP ↪ $pxpxP $t $pt;
```

```
rule ∃E $p (∨E $a $b $paorb _ $pat $pbt) $P $pxpP ↪
    ∨E $a $b $paorb $P (λ pa, ∃E $p ($pat pa) $P $pxpP)
                       (λ pb, ∃E $p ($pbt pb) $P $pxpP);
rule ∃E $p $pexp (_ ⇒ $d) $ppt $pc ↪
    ∃E $p $pexp $d (λ x pp, $ppt x pp $pc);
rule ∃E $p (∃E $q $pexp _ $ppt) $P $pxpP ↪
    ∃E $q $pexp $P (λ x pp, ∃E $p ($ppt x pp) $P $pxpP);
rule ∃E $p $pexp (∀ $b) $ppt $t ↪
    ∃E $p $pexp ($b $t) (λ x pp, $ppt x pp $t);
```

8.10 File `icc.lp` for intuitionistic commutative cuts

```
require open logic.fol logic.nd logic.nd_eps logic.cc;

rule ∨E _ _ (⊥E $pbot _) $P _ _ ↪ ⊥E $pbot $P;

rule =E _ $v (⊥E $pbot _) $P _ ↪ ⊥E $pbot ($P $v);

rule ∧El $a _ (⊥E $pbot _) ↪ ⊥E $pbot $a;

rule ∧Er _ $b (⊥E $pbot _) ↪ ⊥E $pbot $b;

rule ∃E _ (⊥E $pbot _) $P _ ↪ ⊥E $pbot $P;

rule ⊥E (⊥E $pbot _) $P ↪ ⊥E $pbot $P;
rule ⊥E $pbot (_ ⇒ $b) _ ↪ ⊥E $pbot $b;
rule ⊥E $pbot (∀ $a) $t ↪ ⊥E $pbot ($a $t);
rule ⊥E (∨E $a $b $paorb _ $pabot $pbbot) $P ↪
    ∨E $a $b $paorb $P (λ pa, ⊥E ($pabot pa) $P)
        (λ pb, ⊥E ($pbbot pb) $P);
rule ⊥E (∃E $p $pexp _ $ppt) $P ↪
    ∃E $p $pexp $P (λ x pp, ⊥E ($ppt x pp) $P);
```

8.11 File `ccc.lp` for classical commutative cuts

```
require open logic.fol logic.nd logic.nd_eps logic.classic
  logic.cc;

rule ∨E $a $b (nnpp _ $h) $c $i $j ↪
  nnpp $c (λ x, $h (λ y, x (∨E $a $b y $c $i $j)));

rule =E $t $v (nnpp _ $h) $c $i ↪
  nnpp ($c $v) (λ x, $h (λ y, x (=E $t $v y $c $i)));
```

```
rule ∧El $a $b (nnpp _ $h) ↪
  nnpp $a (λ x, $h (λ y, x (∧El $a $b y)));

rule ∧Er $a $b (nnpp _ $h) ↪
  nnpp $b (λ x, $h (λ y, x (∧Er $a $b y)));

rule ∃E $p (nnpp _ $h) $c $i ↪
  nnpp $c (λ x, $h (λ y, x (∃E $p y $c $i)));

rule nnpp $c (∨E $a $b $h _ $i $j) ↪
    ∨E $a $b $h $c (λ x, nnpp $c ($i x))
      (λ y, nnpp $c ($j y));

rule nnpp $c (∃E $p $h _ $i) ↪
    ∃E $p $h $c (λ x y, nnpp $c ($i x y));

rule nnpp $p (nnpp _ $h) ↪
  nnpp $p (λ x, $h (λ y, y x));

rule nnpp (_ ⇒ $b) $h $q ↪
  nnpp $b (λ x, $h (λ y, x (y $q)));

rule nnpp (∀ $p) $h $t ↪
  nnpp ($p $t) (λ x, $h (λ y, x (y $t)));

// additional rules for removing nnpp

rule nnpp $a[] (λ x, x (nnpp $a[] (λ y, $K[x;y]))) ↪
  nnpp $a[] (λ x, $K[x;x]);

rule nnpp _ (λ x, x $K[]) ↪ $K[];
```

Bibliography

[ABC+] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, 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.

[AC15] A. Assaf and R. Cauderlier. Mixing HOL and Coq in Dedukti. In *Proceedings of the 4th International Workshop on Proof eXchange for Theorem Proving*, Electronic Proceedings in Theoretical Computer Science 186, 2015.

[ADJL16] A. Assaf, G. Dowek, J.-P. Jouannaud, and J. Liu. Encoding Proofs in Dedukti: the case of Coq proofs, 2016. Presented at the First International Workshop on Hammers for Type Theories (HaTT).

[AFG+11] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Thery, and Benjamin Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In Jouannaud, Jean-Pierre, Shao, and Zhong, editors, *CPP - Certified Programs and Proofs - First International Conference - 2011*, volume 7086 of *Lecture notes in computer science - LNCS*, pages 135–150, Kenting, Taiwan, December 2011. Springer.

[Ass15] A. Assaf. *A framework for defining computational higher-order logics*. PhD thesis, École Polytechnique, France, 2015.

[Bae20] Seulkee Baek. The tesc proof format for first-order atps (extended abstract). 2020.

[BB12] M. Boespflug and G. Burel. CoqInE: translating the calculus of inductive constructions into the lambda-Pi-calculus mod-

ulo. In *Proceedings of the 2nd International Workshop on Proof eXchange for Theorem Proving*, CEUR Workshop Proceedings 878, 2012.

[BBP11] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending sledgehammer with SMT solvers. In *Lecture Notes in Computer Science*, pages 116–130. Springer Berlin Heidelberg, 2011.

[BCD18] Guillaume Bury, Simon Cruanes, and David Delahaye. SMT Solving Modulo Tableau and Rewriting Theories. July 2018.

[BDD07] Richard Bonichon, David Delahaye, and Damien Doligez. Zenon : An extensible automated theorem prover producing checkable proofs. In *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, pages 151–165, 2007.

[BDG+21] F. Blanqui, G. Dowek, E. Grienenberger, G. Hondet, and F. Thiré. Some axioms for mathematics. In *Proceedings of the 6th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics ?, 2021.

[BdODF09] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. veriT: An open, trustable and efficient SMT-solver. In *Automated Deduction – CADE-22*, pages 151–156. Springer Berlin Heidelberg, 2009.

[BFMP11] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Why3: Shepherd your herd of provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*, pages 53–64, Wrocław, Poland, August 2011. `https://hal.inria.fr/hal-00790310`.

[BFT16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[Bie08] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2–4):75–97, May 2008.

[Bla01] Frédéric Blanqui. *Théorie des types et réécriture*. Theses, Université Paris Sud - Paris XI, September 2001. english version: http://hal.inria.fr/inria-00105525/.

[Bla05] F. Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.

[Bla20] F. Blanqui. Type Safety of Rewrite Rules in Dependent Types. In *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 167, 2020.

[BN10] Sascha Boehme and Tobias Nipkow. Sledgehammer: Judgement day. In *Automated Reasoning*, pages 107–121. Springer Berlin Heidelberg, 2010.

[Boe11] M. Boespflug. *Conception d'un noyau de vérification de preuves pour le lambda-Pi-calcul modulo*. PhD thesis, École Polytechnique, France, 2011.

[Bro12] Chad E. Brown. Satallax: An automatic higher-order prover. In *Automated Reasoning*, pages 111–117. Springer Berlin Heidelberg, 2012.

[Bur11] Guillaume Burel. Experimenting with deduction modulo. In Viorica Sofronie-Stokkermans and Nikolaj BjÃ¸rner, editors, *CADE 2011*, volume 6803 of *Lecture Notes in Artificial Intelligence*, pages 162–176. Springer, 2011.

[Cau16] R. Cauderlier. *Object-oriented mechanisms for interoperability between proof systems*. PhD thesis, CNAM, Paris, France, 2016.

[CD] R. Cauderlier and C. Dubois. FoCaLiZe and Dedukti to the rescue for proof interoperability. In *ITP17*.

[CD07] D. Cousineau and G. Dowek. Embedding pure type systems in the lambda-Pi-calculus modulo. In *Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 4583, 2007.

[CH86] T. Coquand and Gérard Huet. The calculus of constructions. Technical Report RR-0530, INRIA, May 1986.

[CH15] Raphaël Cauderlier and Pierre Halmagrand. Checking Zenon Modulo Proofs in Dedukti. In *Fourth Workshop on Proof eXchange for Theorem Proving (PxTP)*, Berlin, Germany, August 2015.

[CK18] Łukasz Czajka and Cezary Kaliszyk. Hammer for coq: Automation for dependent type theory. *Journal of Automated Reasoning*, 61(1-4):423–453, February 2018.

[CMM19] Kaustuv Chaudhuri, Matteo Manighetti, and Dale Miller. A proof-theoretic approach to certifying skolemization. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs - CPP 2019*. ACM Press, 2019.

[Cru15] Simon Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond*. Theses, École polytechnique, September 2015.

[dB70] N. G. de Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In *Proceedings of the 1968 Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, 1970.

[DDG+13] David Delahaye, Damien Doligez, Frédéric Gilbert, Pierre Halmagrand, and Olivier Hermant. Zenon modulo: When achilles outruns the tortoise using deduction modulo. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 274–290, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer Berlin Heidelberg, 2008.

[dMKA+15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *Automated Deduction - CADE-25*, pages 378–388. Springer International Publishing, 2015.

[Dor11] Alexis Dorra. Équivalence de Curry-Howard entre le lambda-Pi-calcul et la logique intuitionniste. Rapport de stage de L3, École Polytechnique, 2011.

[Dow17] G. Dowek. Models and Termination of Proof Reduction in the $\lambda\Pi$-Calculus Modulo Theory. In *Proceedings of the 44th International Colloquium on Automata, Languages and Programming*, Leibniz International Proceedings in Informatics 80, 2017.

[DRK+14] Morgan Deters, Andrew Reynolds, Tim King, Clark W. Barrett, and Cesare Tinelli. A tour of CVC4: how it works, and how to use it. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, page 7. IEEE, 2014.

[DT19] Gilles Dowek and François Thiré. Logipedia: a multi-system encyclopedia of formal proofs. 2019.

[Dut14] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, July 2014.

[DW05] G. Dowek and B. Werner. A constructive proof of skolem theorem forconstructive logic. `http://www.lsv.fr/~dowek/Publi/skolem.pdf`, 2005. Draft.

[Fér21] G. Férey. *Higher-Order Confluence and Universe Embedding in the Logical Framework*. PhD thesis, Université Paris-Saclay, France, 2021.

[FS19] Mathias Fleury and Hans-Jörg Schurr. Reconstructing veriT proofs in isabelle/HOL. *Electronic Proceedings in Theoretical Computer Science*, 301:36–50, August 2019.

[Gen34] Gerhard Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934.

[Gen20] G. Genestier. Encoding Agda Programs Using Rewriting. In *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 167, 2020.

[Gir87] J.-Y. Girard. *Proof Theory and Logical Complexity*, volume I of *Studies in Proof Theory*. Bibliopolis, 1987.

[GKMP] Quentin Garchery, Chantal Keller, Claude Marché, and Andrei Paskevich. Des transformations logiques passent leur certicat.

[GKSS08] Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Chapter 2 satisfiability solvers. In *Handbook of Knowledge Representation*, pages 89–134. Elsevier, 2008.

[GKU+20] Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. TacticToe: Learning to prove with tactics. *Journal of Automated Reasoning*, 65(2):257–286, August 2020.

[GLT88] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and types*. Cambridge University Press, 1988.

[Har09] John Harrison. HOL light: An overview. In *Lecture Notes in Computer Science*, pages 60–66. Springer Berlin Heidelberg, 2009.

[HB20] G. Hondet and F. Blanqui. The New Rewriting Engine of Dedukti. In *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 167, 2020.

[HBB19] Mohamed Yacine El Haddad, Guillaume Burel, and Frédéric Blanqui. EKSTRAKTO a tool to reconstruct dedukti proofs from TSTP files (extended abstract). *Electronic Proceedings in Theoretical Computer Science*, 301:27–35, August 2019.

[HHP93a] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

[HHP93b] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM (JACM)*, 40(1):143–184, January 1993.

[HS08] J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press, USA, 2 edition, 2008.

[Hur] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. pages 56–68.

[KEH+09] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: formal verification of an OS Kernel. In *Proceedings of the 22nd ACM Symposium on Operating System Principles*, 2009.

[Klo80] J. W. Klop. *Combinatory reduction systems*. PhD thesis, Utrecht Universiteit, NL, 1980. Published as Mathematical Center Tract 129.

[KU15] Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, 2015.

[KV13] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *Computer Aided Verification*, pages 1–35. Springer Berlin Heidelberg, 2013.

[KvOvR93] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

[Ler09] X. Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4):363–446, 2009.

[Mil91] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff. In *Proceedings of the 38th conference on Design automation - DAC '01*. ACM Press, 2001.

[Nip21] Tobias Nipkow. Programming and Proving in Isabelle/HOL. Technical report, 2021.

[OS70] Sam Owre and Natarajan Shankar. The formal semantics of pvs. 02 1970.

[Pau94] L. Paulson. *Isabelle: a generic theorem prover*. Number 828 in Lecture Notes in Computer Science. Springer, 1994.

[Pra71] D. Prawitz. Ideas and results in proof theory. In J. Fenstad, ed-

itor, *Proceedings of the 2nd Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 235–307. North-Holland, 1971.

[PS] Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In *Lecture Notes in Computer Science*, pages 232–245. Springer Berlin Heidelberg.

[RPS+19] T. Ringer, K. Palmskog, I. Sergey, M. Gligoric, and Z. Tatlock. QED at Large: A Survey of Engineering of Formally Verified Software. *Foundations and Trends in Programming Languages*, 5(2-3):102–281, 2019.

[SA09] Laurent Simon and Gilles Audemard. Predicting Learnt Clauses Quality in Modern SAT Solver. In *Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*, Pasadena, United States, July 2009.

[Sai15] R. Saillard. *Type checking in the Lambda-Pi-calculus modulo: theory and practice*. PhD thesis, Mines ParisTech, France, 2015.

[Sch13] Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.

[Sha96] N. Shankar. PVS: combining specification, proof checking, and model checking. In *Proceedings of the 1st International Conference on Formal Methods in Computer-Aided Design*, Lecture Notes in Computer Science 1166, 1996.

[SN08] Konrad Slind and Michael Norrish. A brief overview of HOL4. In *Lecture Notes in Computer Science*, pages 28–32. Springer Berlin Heidelberg, 2008.

[SOR+12] Aaron Stump, Duckki Oe, Andrew Reynolds, Liana Hadarean, and Cesare Tinelli. SMT proof checking using a logical framework. *Formal Methods in System Design*, 42(1):91–118, July 2012.

[SUT06] GEOFF SUTCLIFFE. SEMANTIC DERIVATION VERIFICATION: TECHNIQUES AND IMPLEMENTATION. *International Journal on Artificial Intelligence Tools*, 15(06):1053–1070, December 2006.

[Sut17] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[Sut18] Geoff Sutcliffe. The 9th IJCAR automated theorem proving system competition - CASC-J9. *AI Commun.*, 31(6):495–507, 2018.

[Thi18] F. Thiré. Sharing a Library between Proof Assistants: Reaching out to the HOL Family. In *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, Electronic Proceedings in Theoretical Computer Science 274, 2018.

[Thi19] F. Thiré. Cumulative types systems and levels. `https://hal.archives-ouvertes.fr/hal-02150179`, 2019. Presented at LFMTP'19.

[Thi20] F. Thiré. *Interoperability between proof systems using the Dedukti logical framework*. PhD thesis, Université Paris-Saclay, France, 2020.

[vdP96] J. van de Pol. *Termination of higher-order rewrite systems*. PhD thesis, Utrecht Universiteit, NL, 1996.

[vO94] V. van Oostrom. *Confluence for abstract and higher-order rewriting*. PhD thesis, Vrije Universiteit Amsterdam, NL, 1994.

[WDF+09] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In *Automated Deduction – CADE-22*, pages 140–145. Springer Berlin Heidelberg, 2009.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Integrating Automated Theorem Provers in Proof Assistants

Keywords: Proof assistant, Automated deduction, Proof reconstruction, Type theory, Skolemization

Abstract: Lambdapi is a proof assistant that allows users to construct a proof of a given theorem in a universal language based on the lambda-pi-calculus. The goal of this thesis is to add more automation to Lambdapi to gain more time and effort for the users. This thesis presents three contributions associated with the integration of automated provers in proof assistants. The first contribution consists of the implementation of a tactic that calls automated provers from Lambdapi by using an external platform called Why3. Usually, automated provers do not generate a complete certificate of a given formula, thus, the second contribution presented in this thesis is the reconstruction in Lambdapi of proofs generated by first-order automated provers implemented in a tool called Ekstrakto. Finally, automated provers often perform some transformations on the formula that they are trying to solve. Among these transformations, we can find Skolemization steps. The last contribution is devoted to the certification of Skolemization steps performed by the automated provers in order to have a complete reconstruction. This has been implemented in a tool called Skonverto.

Title: Utiliser des démonstrateurs automatiques dans un assistant à la preuve

Résumé: Lambdapi est un assistant de preuve qui permet à l'utilisateur la construction d'une preuve d'un théorème donné dans un langage universel basé sur le lambda-pi-calcul. Le but de cette thèse est de rajouter de l'automatisation à Lambdapi pour faire gagner du temps à l'utilisateur. Cette thèse présente trois contributions liées à l'intégration des démonstrateurs automatiques dans les assistants de preuve. La première contribution consiste en l'implémentation d'une tactique qui fait appel au démonstrateurs automatiques depuis Lambdapi à travers une plateforme tiers appelé Why3. Généralement, les démonstrateurs automatiques ne génèrent pas un certificat de preuve complet, d'où la deuxième contribution présentée dans cette thèse: la reconstruction de preuves générées par les démonstrateurs automatiques du premier ordre dans Lambdapi implémenté dans un outil appelé Ekstrakto. Enfin, ces démonstrateurs peuvent parfois effectuer des modifications sur la formule qu'ils sont en train de prouver. Le dernier résultat de la thèse est consacré à la certification des étapes de Skolemisation faites par les démonstrateurs automatiques. Un algorithme est présenté, montré correct et implémenté dans l'outil Skonverto.

Mots clés: Assistants à la preuve, Reconstruction de preuves, Démonstration automatique, Théorie des types, Skolemisation

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France