Development and Verification of Probability Logics and Logical Frameworks

Petar Maksimovic

UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

UNIVERSITÉ DE NICE SOPHIA ANTIPOLIS

ECOLE DOCTORALE DE SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

PETAR MAKSIMOVIĆ

DEVELOPMENT AND VERIFICATION OF PROBABILITY LOGICS AND LOGICAL FRAMEWORKS

DOCTORAL DISSERTATION

CO-TUTELLE DOCTORAL STUDIES BETWEEN
THE UNIVERSITY OF NOVI SAD AND
THE UNIVERSITY OF NICE SOPHIA ANTIPOLIS

JURY:

PRESIDENT:	**ZORAN MARKOVIĆ**	RESEARCH PROFESSOR, MATHEMATICAL INSTITUTE SANU
REVIEWERS:	**PIERRE LESCANNE**	PROFESSOR EMERITUS, ENS LYON
	HUGO HERBELIN	RESEARCH DIRECTOR, INRIA, TEAM $\pi\rho^2$
EXAMINERS:	**FURIO HONSELL**	FULL PROFESSOR, UNIVERSITY OF UDINE
	MARINA LENISA	ASSOCIATE PROFESSOR, UNIVERSITY OF UDINE
	ZORAN OGNJANOVIĆ	RESEARCH PROFESSOR, MATHEMATICAL INSTITUTE SANU
ADVISORS:	**SILVIA GHILEZAN**	FULL PROFESSOR, UNIVERSITY OF NOVI SAD
	LUIGI LIQUORI	RESEARCH DIRECTOR, INRIA, TEAM LOGNET
INVITED:	**JOVANKA PANTOVIĆ**	FULL PROFESSOR, UNIVERSITY OF NOVI SAD
	JOËLLE DESPEYROUX	RESEARCHER, I3S, TEAM MDSC

NOVI SAD/SOPHIA ANTIPOLIS, 2013.

In loving memory of my father and of my grandfather.
You will always live on in my heart.

For my mother, who has stood by me through
all these years with the strength and courage
that I can only pray to have someday myself.

Without you, none of this would have been possible.

Acknowledgments

Five years ago, I have had the good fortune to have been offered what then were one of the very first co-tutelle doctoral studies between Serbia and France. At first, reluctantly, I turned the offer down, but after a little soul searching, ultimately accepted. That turned out to be the best decision of my life. During the course of my studies, I have had the privilege of working together with some of the finest researchers in my field of expertise and of experiencing first-hand the magic that is science. It was a once-in-a-lifetime experience. However, as professionally fulfilling as I have found that to be, it has no choice but to stand eclipsed by the wonderful people that I have encountered in the process, lifelong friendships and connections that I have witnessed both form and dissolve, and the multi-cultural exchange of thoughts and ideas on anything and everything, through which I have grown immensely as a person. With the defense of my thesis, this beautiful chapter of my life arrives to a close, and the time comes for me to share my gratitude with the people who helped it to be the amazing journey that it truly was.

First of all, I would like to thank my three de facto advisers - professors Luigi Liquori, Zoran Ognjanović, and Silvia Ghilezan. Thank you Luigi, for your optimism, enthusiasm, dedication, and your endless flow of ideas. Thank you Zoran, for your directions, insights, and meticulousness. Thank you Silvia, for your calmness, persistence, and unyielding support. Thank you all for gifting me with this incredible, life-changing experience. I am indebted to you forever.

In May and June 2011, I have had the opportunity of being a visiting researcher at the University of Udine, and the privilege of collaborating with professors Furio Honsell, Marina Lenisa, and Ivan Scagnetto, on what was then an idea off the top of our heads, but what has since evolved to be the keystone of my thesis. Your methodicalness, precision, and drive have helped me improve as a researcher, and understand and appreciate the growing world of typed λ-calculi ever more so. Thank you all so much for that.

I would also like to thank professor Pierre Lescanne and professor Hugo Herbelin, for doing me the honour of being rapporteurs for my thesis, as well as professor Zoran Marković and professor Jovanka Pantović, for graciously accepting to be part of the jury.

Finally, my deepest gratitude goes to my family and to my friends, old and new alike. You have been a never-ending source of love and support, patience and wittiness, in good times and bad. I am so grateful for having you in my life. Thank you.

Belgrade, Novi Sad, Udine, and Sophia Antipolis, 2008-2013.

Abstract

The research for this thesis has followed two main paths: the one of probability logics and the other of type systems and logical frameworks, bringing them together through interactive theorem proving. With the development of computer technology and the need to capture real-world dynamics, situations, and problems, reasoning under uncertainty has become one of the more important research topics of today, and one of the tools for formalizing this kind of knowledge are probability logics. Given that probability logics, serving as decision-making or decision-support systems, often form a basis for expert systems that find their application in fields such as game theory or medicine, their correct functioning is of great importance, and formal verification of their properties would add an additional level of security to the design process. On the other hand, in the field of logical frameworks and interactive theorem proving, attention has been directed towards a more natural way of encoding formal systems where derivation rules are subject to side conditions which are either rather difficult or impossible to encode naively, in the Edinburgh Logical Framework LF or any other type-theory based Logical Framework, due to their inherent limitations, or to the fact that the formal systems in question need to access the derivation context, or the structure of the derivation itself, or other structures and mechanisms not available at the object level.

The first part of the thesis deals with the development and formal verification of probability logics. First, we introduce a Probability Logic with Conditional Operators - LPCP, its syntax, semantics, and a sound and strongly-complete axiomatic system, featuring an infinitary inference rule. We prove the obtained formalism decidable, and extend it so as to represent evidence, making it the first propositional axiomatization of reasoning about evidence.

Next, we show how to encode probability logics $LPP_1^{\mathbb{Q}}$ and $LPP_2^{\mathbb{Q}}$ in the Proof Assistant Coq. Both of these logics extend classical logic with modal-like probability operators, and both feature an infinitary inference rule. $LPP_1^{\mathbb{Q}}$ allows iterations of probability operators, while $LPP_2^{\mathbb{Q}}$ does not. We proceed to formally verify their key properties - soundness, strong completeness, and non-compactness. In this way, we formally justify the use of probabilistic SAT-solvers for the checking of consistency-related questions.

In the second part of the thesis, we present $\mathsf{LF}_{\mathcal{P}}$ - a Logical Framework with External Predicates, by introducing a mechanism for locking and unlocking types and terms into LF, allowing the use of external oracles. We prove that $\mathsf{LF}_{\mathcal{P}}$ satisfies all of the main meta-theoretical properties (strong normalization, confluence, subject reduction, decidability of type checking). We develop a corresponding canonical framework, allowing for easy proofs of encoding adequacy. We provide a number of encodings - the simple untyped λ-calculus with a Call-by-Value reduction strategy, the Design-by-Contract paradigm, a small imperative language with Hoare Logic, Modal Logics in Hilbert and Natural Deduction style, and Non-Commutative Linear Logic (encoded for the first time in an LF-like framework), illustrating that in $\mathsf{LF}_{\mathcal{P}}$ we can encode side-conditions on the application of rules elegantly, and achieve a separation between derivation and computation, resulting in cleaner and more readable proofs.

We believe that the results presented in this thesis can serve as a foundation for fruitful future research. On the one hand, the obtained formal correctness proofs add an additional level of security when it comes to the construction of expert systems constructed using the verified logics, and pave way for further formal verification of other probability logics. On the other hand, there is room for further improvement, extensions, and deeper analysis of the $\mathsf{LF}_{\mathcal{P}}$ framework, as well as the building of a prototype interactive theorem prover based on $\mathsf{LF}_{\mathcal{P}}$ and discovering its place in the world of proof assistants.

Contents

List of Figures

Chapter 1

Introduction

Contents

Context. This Ph.D. thesis has been accomplished in co-tutelle between the Faculty of Technical Sciences, University of Novi Sad, Serbia, and the University of Nice Sophia Antipolis, France. It is one of the outcomes of a pilot Ph.D. program in Information Technology, developed within the Tempus DEUKS (Doctoral School towards European Knowledge Society) Project JEP-41099-2006. The research for this thesis has been performed, over a period of three and a half years, in the following institutions:

- Mathematical Institute of the Serbian Academy of Sciences and Arts, Belgrade, Serbia and the Faculty of Technical Sciences, University of Novi Sad, Serbia (under supervision of Research Professor Zoran Ognjanović and Professor Silvia Ghilezan).
- INRIA Sophia Antipolis Méditerranée, France (under supervision of INRIA Research Director Luigi Liquori).
- Università di Udine, Italy (in collaboration with Professors Furio Honsell, Marina Lenisa, and Ivan Scagnetto).

Motivation and Research Directions. The research for this thesis has followed two main paths: the one of probability logics and the other of type systems and logical frameworks, bringing them together through interactive theorem proving.

With the development of computer technology and the need to capture real-world dynamics, situations, and problems, reasoning under uncertainty has become one of the more important research topics of today. One instrument for formalizing this kind of reasoning are probability logics. In their various forms, they constitute a framework for encoding probability-related statements, and offer a possibility and methodology for deducing conclusions from such statements, in a manner analogous to that present in classical or intuitionistic logic, using axioms and inference rules. Probability logics, serving as decision-making or decision-support systems, often form a basis for expert systems that find their application in fields such as game theory or medicine. As such, their correct functioning is of great importance, and formal verification of their properties appears as a natural step for one to take. To the author's knowledge, the subject of probability

logics has not yet been approached in that context. In this thesis, probability logics have been treated from two perspectives - that of development, and that of verification. First, we present a new probability logic with conditional probability operators (LPCP), its syntax, semantics, a corresponding strongly-complete axiomatic system, and show how this logic can be used to represent evidence (Chapter 2). Next, we encode in Coq and provide a formal verification of the soundness, strong completeness and non-compactness for two rationally-valued probability logics, each with their own specificities (Chapter 3 and Chapter 4). The proofs for these two logics share a common outline, which can be adapted and re-used for proving properties of other probability and modal-like logics.

The Edinburgh Logical Framework or LF is one of the better-known logical frameworks today. After being introduced in 1993, it has found widespread usage in theoretical and applied research. It is one of the systems of Barendregt's λ-cube [Barendregt 1992], and constitutes the basis of several proof assistants. However, it does have its limitations. For instance, there are formal systems where derivation rules are subject to *side conditions* which are either rather difficult or impossible to encode naively, in LF or any other type-theory based Logical Framework, due to their inherent limitations, or to the fact that the formal systems in question need to access the derivation context, or the structure of the derivation itself, or other structures and mechanisms not available at the object level (modal and program logics, for one). Bearing this in mind, together with the idea of developing a framework in which pre- and post-conditions could be encoded naturally, we have developed $\mathsf{LF}_{\mathcal{P}}$ - an extension of LF with external predicates. These predicates act as *locks* in the flow of reduction, blocking it until a condition has been (possibly externally) verified to be true. In this way, not only can we encode the aforementioned systems with greater ease, but we also manage to, in a sense, separate derivation and computation, by "outsourcing" the latter to an external verifier. We introduce $\mathsf{LF}_{\mathcal{P}}$ (Chapter 5) present its type system (Chapter 6) prove all of the important meta-theoretical properties (Chapter 7), construct an appropriate canonical framework $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ (Chapter 8), and present a series of encodings illustrating the features and benefits of $\mathsf{LF}_{\mathcal{P}}$ (Chapter 9).

Contributions of the Thesis. In a nutshell, the original contributions of this thesis are threefold:

1. A new probability logic with conditional probability operators (LPCP) with its corresponding strongly-complete axiomatic system, which can be used to represent evidence,
2. Formal verification of the soundness, strong completeness and non-compactness for two rationally-valued probability logics, in the proof assistant Coq, and
3. $\mathsf{LF}_{\mathcal{P}}$ - a Logical Framework with dependent types, extending the Edinburgh Logical Framework LF with calls to external oracles. The main novelty of this framework are locked types, which block the flow of reduction until a given condition holds, allowing substantially easier encodings of systems with side conditions and an elegant separation between derivation and computation.

Publications. This thesis relies on the following published conference and journal papers:

1. Petar Maksimović, Dragan Doder, Bojan Marinković and Aleksandar Perović. *A Logic with a Conditional Probability Operator*. In Kata Balogh, editor, Proceedings of the 13$^{\text{th}}$ ESSLLI Student Session, pp. 105-114, 2008, Hamburg, Germany.
2. Dragan Doder, Bojan Marinković, Petar Maksimović, Aleksandar Perović. *A Logic with Conditional Probability Operators*. Publication de l'Institut Mathématique, N.S. 87(101), pp. 85-96, 2010.

3. Petar Maksimović. *Formal Verification of Key Properties for Several Probability Logics in the Proof Assistant Coq.* First National Conference on Probability Logics and their Applications, Book of Abstracts, p. 25, Belgrade, Serbia. 29-30 September, 2011.

4. Petar Maksimović. *Probability Logics in Coq.* TYPES 2013: Types for Proofs and Programs, Book of Abstracts, pp. 60-61, 19$^{\text{th}}$ TYPES Meeting, April 22-26, 2013, Toulouse, France.

5. Furio Honsell, Marina Lenisa, Luigi Liquori, Petar Maksimović, Ivan Scagnetto. $\mathsf{LF}_{\mathcal{P}}$ - *A Logical Framework with External Predicates.* In Proceedings of LFMTP 2012: The Seventh International Workshop on Logical Frameworks and Meta-languages, Theory and Practice, pp. 13-22, Copenhagen, Denmark, 08-09 September, 2012.

6. Furio Honsell, Marina Lenisa, Luigi Liquori, Petar Maksimović, Ivan Scagnetto. *An Open Logical Framework.* Journal of Logic and Computation, 2013, to appear.

1.1 Probability in Logic

In this section, we provide a historical overview of the development of probability logics, reflecting on the ideas of the most important researchers in this field. The following text was adapted, in the better part, from [Ognjanović 2009].

The first one to consider incorporating the notions of probability and logic together was the famous German mathematician and philosopher Gottfried Wilhelm Leibnitz. As he searched for a universal basis for all sciences and sought to establish logic as a generalized mathematical calculus, he considered probability logic as a tool for estimation of uncertainty, and defined probability as a *measure of knowledge*. In a number of his essays [Leibnitz 1665, Leibnitz 1669, Leibnitz 1765], Leibnitz suggested that the tools which have been developed for analyzing games of chance should be directed towards the development of a new kind of logic addressing degrees of probability and that this logic could then facilitate rational decision-making on apparently conflicting claims. He distinguished between two calculi. The first one, *forward calculus*, was concerned with estimating the probability of an event if the probabilities of its conditions are known, whereas the second one, *reverse calculus*, dealt with estimations of probabilities of causes, once the probability of their consequence is known. Although most of Leibnitz's logical works were published long after his death (in the early 1900s), he did have a number of followers, the most important of whom, when it comes to probabilistic logic, were the brothers Jacobus and Johann Bernoulli, Thomas Bayes, Pierre Simon de Laplace, Bernard Bolzano, Augustus De Morgan, George Boole, John Venn, Charles S. Peirce, etc.

Jacobus Bernoulli was the first who made advance along Leibnitz's ideas in his unfinished work [Bernoulli 1713, Part IV, Chapter III]. By making use of Huygen's notion of expectation, *i.e.* the value of a gamble in games of chance, he offered a procedure for determining numerical degrees of certainty of conjectures produced by arguments. He used the word *argument* to represent statements, as well as the implication relation between premises and conclusions. He divided arguments into categories according to whether the premises and the argumentation from premises to conclusions are contingent or necessary. For example, if an argument were to exist contingently (*i.e.* it is true in cases when $b > 0$, and is not true in cases when $c > 0$) and implies a conclusion necessarily, then such an argument would establish $\frac{b}{b+c}$ as the certainty of the conclusion. Bernoulli also discussed the question of computing the degree of certainty when there was more than one argument for the same conclusion.

In [Bayes 1764], T. Bayes first presented a result involving conditional probability. In modern notation, he considered the problem of finding the conditional probability $P(A|B)$ where A is the proposition "$P(E) \in [a, b]$", and B is the proposition "an event E occurred p and did not

occur q times in $p+q$ independent trials". A. De Morgan devoted a chapter of [de Morgan 1847] to probability inference, offering a defense for the numerical probabilistic approach as a part of logic. Instead of giving a systematic treatment of the field, he described several problems and tried to apply logical concepts to them. It is interesting that in his work, De Morgan made some mistakes, mostly due to his ignoring of (in)dependence of events.

The calculus presented by G. Boole in [Boole 1847, Boole 1854] led to a rapid development of mathematical logic. Boole sought to make his system the basis of a logical calculus as well as a more general method to be applied in probability theory. The most general problem (originally called the "general problem in the theory of probability") Boole claimed that he could solve, concerned an arbitrary set of logical functions $\{f_1(x_1,\ldots,x_m),\ \ldots,\ f_k(x_1,\ldots,x_m),\ F(x_1,\ldots,x_m)\}$ and the corresponding probabilities $p_1 = P(f_1(x_1,\ldots,x_m)),\ \ldots,\ p_k = P(f_k(x_1,\ldots,x_m))$, and asked for $P(F(x_1,\ldots,x_m))$ in terms of $p_1,\ldots,p_k$. He elaborated on the relation between the logic of classical connectives and the formal probability properties of compound events using the following assumptions. He restricted disjunctions to the exclusive ones, and believed that any compound proposition can be expressed in terms of, maybe ideal, simple and independent components. Thus, the probability of an or-compound is equal to the sum of the components, whereas the probability of an and-compound is equal to the product of the components. In such a way, it was possible to convert logical functions of events into a system of algebraic functions of the corresponding probabilities. Boole attempted to solve such systems using a procedure equivalent to Fourier-Motzkin elimination. His procedure, although not entirely successful, provided an important basis for probabilistic inferences. In [Hailperin 1984, Hailperin 1986] a rationale and a correction for the Boole's procedure were given using the linear programming approach.

In the 1870's, J. Venn developed the idea of extending the frequency of occurrence concept of probability to logic. Venn thought that probability logic is the logic of sequence of statements. A single element sequence of this type attributes to the given proposition one of two values 0 or 1, whereas an infinite sequence attributes any real number which lies in the interval [0, 1]. Some of the traditional logicians were dissatisfied with the inclusion of the induction in the definition of the concept of probability, but the others continued to work in that direction.

During the first half of the twentieth century, there were at least three directions in the development of theory of probability. The researchers who belonged to the first one, Richard von Miss and Hans Reichenbach, for example, regarded probability as a relative frequency and derived rules of the theory from that interpretation. The second approach was characterized by the development of formal calculus of probability. Some of the corresponding authors were Georg Bohlmann [Bohlmann 1901], Sergei Natanovich Bernstein [Bernstein 1917], and Émil Borel [Borel 1924, Borel 1925]. These investigations culminated in A. N. Kolmogorov's axiomatization of probability [Kolmogorov 1933]. Finally, some of the researchers, like John M. Keynes [Keynes 1921], Hans Reichenbach [Reichenbach 1935, Reichenbach 1949], and Rudolf Carnap [Carnap 1950, Carnap 1952] continued Boole's approach connecting probability and logic.

In spite of the works of these researchers, the main streams of development of logic and probability theory were almost separated during the second half of XX century. Namely, in the last quarter of the nineteenth century, independently of the algebraic approach, there was a development of mathematical logic inspired by the need of giving axiomatic foundations of mathematics. The main representative of that effort was Gottlob Frege. He tried to explain the fundamental logical relationships between the concepts and propositions of mathematics. Truth-values, as special kinds of abstract values, were described by Frege according to whom every proposition is a name for truth or falsity. It is clear that, according to Frege, the truth values had a special status that had nothing to do with probabilities. That approach culminated with Kurt Gödel's proof of completeness for first order logic [Gödel 1929]. Since those works, first order logic played the central role in the logical community for many years, and only in the late 1970s a wider interest in probability logics reappeared.

The most important advancement in probability logic, after the works of Leibnitz and Boole, was made by H. Jerome Keisler. The purpose of his famous paper [Keisler 1977] was to introduce a model-theoretic approach into probability theory, using non-standard analysis.

Keisler introduced several probability quantifiers, such as $Px > r$, with the intended meaning that the formula $(Px > r)\phi(x)$ holds if the set $\{x : \phi(x)\}$ has probability greater than r. A recursive axiomatization for these logics (the main of which is denoted by L_{AP}) was given by D. Hoover [Hoover 1978], using admissible and countable fragments of infinitary predicate logic (but without resorting to two standard quantifiers - $\forall$ and $\exists$). In the following years, Keisler and Hoover made very important contributions in the field. They proved completeness theorems for various kinds of models (probability, graded, analytic, hyperfinite etc.) and many other model-theoretical theorems. The development of probability model theory has engendered the need for the study of logics with greater expressive power than that of the logic L_{AP}. The logic L_{AI}, introduced in [Keisler 1985] as an equivalent of the logic L_{AP}, allows the user to express many properties of random variables in an easier way. In this logic, the integral quantifiers $\int \ldots dx$ are used in place of the quantifiers $Px > r$.

The logic L_{AI} is not rich enough to express probabilistic notions involving conditional expectations of random variables with respect to σ-algebras, such as martingales, Markov process, Brownian motion, stopping time, optional stochastic process, etc. These properties can be naturally expressed in a language with both integral quantifiers and conditional expectation operators. The logics L_{AE} and L_{Aad}, introduced by Keisler in [Keisler 1985], are appropriate for the study of random variables and stochastic processes. The model theory of these logics has been developed further by Rodenhausen in [Rodenhausen 1982], Fajardo in [Fajardo 1985], Keisler in [Keisler 1986a, Keisler 1986b], Hoover in [Hoover 1987], and Rašković in [Rašković 1985, Rašković 1986, Rašković 1988].

Since the middle of the 1980s, the interest in probabilistic logics began to grow, mainly because of the development of many fields of application of reasoning about uncertain knowledge: economics, artificial intelligence, computer science, medicine, philosophy, etc. Many researchers attempted to combine probability-based and logic-based approaches to knowledge representation. In a logical framework for modeling uncertainty, probabilities express degrees of belief. For example, one can say that "probability that Homer wrote the Iliad is at most one half", thus expressing one's disbelief into the truthfulness of that statement. The first in line of those papers is [Nilsson 1986] (a revision of which can be found in [Nilsson 1993]) which resulted from the development of an expert system in medicine, where Nilsson tried to construct a logic with probabilistic operators as a well-founded framework for uncertain reasoning. Sentences of this logic provided information on probabilities, and he was able to express a probabilistic generalization of modus ponens as "if α holds with probability s, and β follows from α with probability t, then the probability of β is r". In this way, Nillson gave a procedure for probabilistic entailment that, given probabilities of premises, could calculate bounds on probabilities of the derived sentences. Nillson's approach was semantic in nature and it motivated some authors to provide axiomatizations and decision procedures for the logic. In the same year, Gaifman published a paper [Gaifman 1986] that studied higher-order probabilities and connections with modal logics.

In [Fagin 1990], Fagin, Halpern and Megiddo presented a propositional logic with real-valued probabilities in which higher level probabilities were not allowed (similar to $LPP_2^{\mathbb{Q}}$, presented in Chapter 3). The language of that logic allowed basic probabilistic formulas of the form

$$\sum_{i=1}^{n} a_i w(\alpha_i) \geq s,$$

where a_i and s are rational numbers, α_i are classical propositional formulas, and $w(\alpha_i)$ denotes the probability that α_i holds. Probabilistic formulas are treated as boolean combinations of

basic probabilistic formulas. A finitary axiomatic system for the logic was given, and since the compactness theorem does not hold for their logic, the authors were able to prove simple completeness only, using the tools of the theory of real closed fields. The papers [Fagin 1994, Halpern 1991], by the same authors, introduced a probabilistic extension of the modal logic of knowledge, which is similar to $LPP_1^{\mathbb{Q}}$, presented in Chapter 4. First-order probability logics were discussed in [Abadi 1994, Halpern 1990]. It was shown that the set of valid formulas of the considered logic is not recursively enumerable, and thus, no finitary axiomatization is possible.

Finally, we will give mention to the probability logics that have been developed, over the last 20 years, by a group of researchers at the Mathematical Institute of the Serbian Academy of Sciences and Arts, led by Miodrag Rašković, himself a student of H. Jerome Keisler. The approach taken there extends classical (propositional or first order) calculus with operators that provide information on probability, whereas the truth values of the formulas can still only be either true or false. In this way, one is able to construct statements of the form $P_{\geq s}\alpha$ with the intended meaning "the probability that α holds is at least s".

These probability operators behave like modal operators and the corresponding semantics consist in special types of Kripke models (involving possible worlds) with the addition of probability measures defined on algebras of subsets of these worlds. One of the main proof-theoretical problems with the approach in question is providing an axiom system which would be strongly complete ("every consistent set of formulas has a model", in contrast to the simple completeness "every consistent formula has a model"). This problem arises from the inherent non-compactness of such systems, namely the possibility to define an inconsistent infinite set of formulas, every finite subset of which is consistent (*e.g.* $\{\neg P_{=0}\alpha\} \cup \{P_{<\frac{1}{n}}\alpha \mid n \in \mathbb{N}\}$). These logics include, but are not limited to the following ones:

- LPP_1, a probability logic which is built atop classical propositional logic, with iterations of probability operators and real-valued probability functions [Ognjanović 1999a, Ognjanović 2000] (a variant of this logic, with rationally-valued probability functions, has been formally verified in Chapter 4,
- $LPP_1^{Fr(n)}$, similar to LPP_1, but with probability functions restricted to have the range $\{0, \frac{1}{n}, \ldots, \frac{n-1}{n}, 1\}$ [Ognjanović 1996, Ognjanović 1999a, Ognjanović 2000],
- LPP_1^{LTL}, a probability logic similar to LPP_1, but with discrete linear-time temporal logic LTL as the underlying logic [Ognjanović 1998, Ognjanović 1999a, Ognjanović 2006],
- LPP_2 and $LPP_2^{Fr(n)}$, probability logics similar to LPP_1 and $LPP_1^{Fr(n)}$, but without iterations of probability operators [Ognjanović 1999a, Ognjanović 2000, Rašković 1993] (a variant of this logic, with rationally-valued probability functions, has been formally verified in Chapter 3,
- $LPP_{2,\preceq}$, a probability logic similar to LPP_2, but allowing reasoning about qualitative probabilities [Ognjanović 2008],
- LPP_2^I, a probability logic similar to LPP_1, but with propositional intuitionistic logic as the underlying logic [Marković 2003a, Marković 2003b, Marković 2004],
- $LFOP_1$, $LFOP_1^{Fr(n)}$, and $LFOP_2$, first-order counterparts of LPP_1, $LPP_1^{Fr(n)}$, and LPP_2 [Ognjanović 2000, Rašković 1999],
- $\mathcal{L}_{\mathbb{Q}_p}$, a probability logic with probabilities in the field $\mathbb{Q}_p$ of p-adic numbers [Ilić-Stepić 2012],
- $LFOCP$ and $LFPOIC^=$, first-order condidional probability logics without and with iterations [Milošević 2012, Milošević 2013].

Briefly on the relationship between probability and fuzzy logics. Another method for reasoning under uncertainty are fuzzy logics. Just as probability logics, fuzzy logics also fall into the category of weighted logics, i.e. logics in which more than two truth values can be assigned to formulas (usually values from the real unit interval [0, 1]). In this way, we can apply a fine granulation atop the classical bipolar concept of truth. The key difference between fuzzy and probability logics, in the propositional case, is the mechanism through which the truth value of a formula is calculated once the truth values of its propositional letters are known. In probability logics, we use the principle of additivity, and have that the value of the formula $\varphi(p_1, \ldots, p_n)$ is uniquely calculated based on the values of finite conjunctions of distinct propositional letters from the set $\{p_1, \ldots, p_n\}$. There, every classical tautology has the maximal possible measure. On the other hand, fuzzy logics make use of the principle of truth functionality, where each connective is assigned its corresponding truth function. t-norms are assigned to conjunctions, and the dual s-norms (or t-conorms) are assigned to disjunctions. The set of valid formulas in fuzzy logics is a proper subset of the set of classical tautologies.

1.2 A Word on λ-calculi

λ-calculus is a formal system invented in the 1920s, whose aim was to describe the most basic properties of functional abstraction, application and substitution in a very general setting. [1]

The history of λ-calculus can be divided into three main periods: the first period comprised several years of intensive and very fruitful initial study in the 1920s and 1930s, and one of its most notable results is the first proof that predicate logic is undecidable. Next ensued a second period of almost 30 years of relative stagnation, featuring specialized results such as completeness, cut-elimination and standardization theorems that found many uses later. Finally, in the late 1960s, came an increase of activity motivated by advances in higher-order function theory, newly-found connections with programming languages, and new technical discoveries. The main contributions of the third period, from the 1960s onward, include constructions and analyses of models, development of polymorphic type systems, deep analyses of the reduction process, and many others.

With the aim of developing a foundation for logic which would be more natural than the type theory of Russell or the set theory of Zermello, Alonzo Church came to invent the first λ-calculus in 1928 [Church 1932]. This calculus featured abstraction $\lambda x[M]$ and application $\{F\}(X)$, notation that has, over the years, evolved into the more familiar $\lambda x.M$ and $(F\,X)$. The system from [Church 1932] was a type-free logic with unrestricted quantification, with the law of excluded middle excluded, and explicit formal rules of λ-conversion included. However, it also featured a contradiction which was discovered almost immediately upon publication. The system was revised a year later [Church 1933], and in the revised paper Church stated a hope that Gödel's (then) recent incompleteness theorems for the system of [Russell and Whitehead 1913] would not extend to his own revised system, and that a finitary proof of the consistency of this system might be found. In this revised paper, Church first introduced the λ-terms representing positive integers, today known as the *Church numerals*.

In the subsequent works of Church, Kleene and Rosser, the developments on Church's 1933 logic and the underlying pure λ-calculus resulted in somewhat surprising discoveries: that the 1933 logic is, in fact, inconsistent, and that the pure λ-calculus could have a number of intriguing uses. Confluence of the λ-calculus was proven [Church and Rosser 1936], ensuring the consistency of the pure system, and λ-definable numerical functions were proven equivalent to both Herbrand-Gödel recursive functions [Kleene 1936, Church 1936] and Turing-computable

[1] The historical account of the development of the λ-calculus presented in this section has mostly been adapted from [Cardone 2005].

functions [Turing 1937], leading Church to his conjecture that λ-definability exactly captured the informal concept of effective calculability, today known as *Church's Thesis*.

Numerous advances and discoveries concerning λ-calculi have been made in the subsequent years and are still being made today, including the connection between λ-calculi, logics and programming languages. Many systems have been constructed on top of the pure λ-calculus, and to describe them all, or even representatively, would constitute a book-worthy effort. However, since this thesis relies on typed λ-calculi, we will take a moment to present them in more detail. A typed λ-calculus is essentially an extension of the pure λ-calculus with types. Here, types are usually objects of a syntactic nature which are assigned to lambda terms, following a set of rules which we refer to as a typing system. There are many such formalisms, and the exact nature of a type varies from one to the other. We will provide a brief overview of some of these systems, starting from the most basic example: the simply-typed λ-calculus. For more detailed information on this and many other λ-calculi with types, we kindly refer the reader to [Barendregt 2013].

1.2.1 The Simply-Typed Lambda Calculus

The simply-typed λ-calculus $\lambda_{\to}$ is defined in the following way:

- We need to fix a family of base types $\mathcal{B}$. The types in $\mathcal{B}$ are sometimes also called type constants or atomic types. Examples of base types could be `nat` or `bool`, and could, for instance, serve to representing integers or boolean values in a programming language.
- Given $\mathcal{B}$, the syntax of types is given by $\tau ::= b \mid \tau \to \tau$, where $b \in \mathcal{B}$. Intuitively, we can understand the type $\sigma \to \tau$ as the set of functions which require an input that is of type σ and produce an output that is of type τ. Some examples of valid types would be `nat` or $\texttt{nat} \to \texttt{nat}$ or $(\texttt{bool} \to \texttt{nat}) \to \texttt{bool}$.
- Another item we need to fix is a family of term constants $\mathcal{C}$, where each $c \in \mathcal{C}$ is assigned its base type $b \in \mathcal{B}$. For instance, if there would be a type constant `nat` representing natural numbers, the appropriate term constants could be `0`, `1`, etc. Term constants form what we call a *typing signature*.
- Given $\mathcal{C}$, the syntax of terms is given by $M ::= c \mid x \mid \lambda x{:}\tau.M \mid M\,M$, where $c \in \mathcal{C}$ - a term can either be a term constant, a variable, a λ-abstraction or an application.

However, we can *a priori* decide only the types for term constants, whereas the typing assignments for all of the other terms are handled by the rules of the typing system. We also need a typing context Γ, which contains variables and their respective types, such as, for instance $x{:}\sigma$, $y{:}\tau$, etc. The rules of the typing system of $\lambda_{\to}$ are the following:

1. If the variable x has type σ in the context Γ, then x is of type σ:

$$\frac{}{\Gamma, x{:}\sigma, \Gamma' \vdash x : \sigma}$$

2. A constant term has its base type in any context:

$$\frac{c \text{ is a constant of type } T}{\Gamma \vdash c : T}$$

3. If, in a context where x has type σ, we have that M is of type τ, then, in the same context without x, $\lambda x{:}\sigma.M$ is of type $\sigma \to \tau$:

$$\frac{\Gamma, x{:}\sigma \vdash M : \tau}{\Gamma \vdash \lambda x{:}\sigma.M : \sigma \to \tau}$$

4. If, in a certain context, M is of type $\sigma \to \tau$ and in that same context N is of type σ, then in that same context $M\,N$ is of type τ:

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}$$

Using this system, we can arrive to certain conclusions. Let us assume, for instance, that we have base types σ, τ, and ρ. Then, with the context $\{x{:}\sigma\}$, we could derive:

1. $x{:}\sigma \vdash x{:}\sigma$ by rule 2.
2. $\vdash \lambda x{:}\sigma.x : \sigma \to \sigma$ from 1, by rule 3.

or, with the context $\{x{:}\sigma,\ y{:}\tau\}$, we could derive:

1. $x{:}\sigma,\ y{:}\sigma \vdash x{:}\sigma$ by rule 2.
2. $x{:}\sigma \vdash \lambda y{:}\tau.x : \tau \to \sigma$ from 1, by rule 3.
3. $\vdash \lambda x{:}\sigma.\lambda y{:}\tau.x : \sigma{\to}(\tau{\to}\sigma)$ from 2, by rule 3.

or, with the context $\{x{:}\sigma{\to}(\tau{\to}\rho),\ y{:}\sigma{\to}\tau,\ z{:}\sigma\}$, we could derive:

1. $x : \sigma{\to}(\tau{\to}\rho),\ y : \sigma{\to}\tau,\ z{:}\sigma \vdash x : \sigma{\to}(\tau{\to}\rho)$ by rule 2.
2. $x : \sigma{\to}(\tau{\to}\rho),\ y : \sigma{\to}\tau,\ z{:}\sigma \vdash y : \sigma{\to}\tau$ by rule 2.
3. $x : \sigma{\to}(\tau{\to}\rho),\ y : \sigma{\to}\tau,\ z{:}\sigma \vdash z{:}\sigma$ by rule 2.
4. $x : \sigma{\to}(\tau{\to}\rho),\ y : \sigma{\to}\tau,\ z{:}\sigma \vdash xz : \tau{\to}\rho$ from 1 and 3, by rule 4.
5. $x : \sigma{\to}(\tau{\to}\rho),\ y : \sigma{\to}\tau,\ z{:}\sigma \vdash yz : \tau$ from 2 and 3, by rule 4.
6. $x : \sigma{\to}(\tau{\to}\rho),\ y : \sigma{\to}\tau,\ z{:}\sigma \vdash xz\,(yz) : \rho$ from 4 and 5, by rule 4.
7. $x : \sigma{\to}(\tau{\to}\rho),\ y : \sigma{\to}\tau \vdash \lambda z{:}\sigma.xz\,(yz) : \sigma{\to}\rho$ from 6, by rule 3.
8. $x : \sigma{\to}(\tau{\to}\rho) \vdash \lambda y{:}\sigma{\to}\tau.\lambda z{:}\sigma.xz\,(yz) : (\sigma{\to}\tau){\to}(\sigma{\to}\rho)$ from 7, by rule 3.
9. $\vdash \lambda x{:}\sigma{\to}(\tau{\to}\rho).\lambda y{:}\sigma{\to}\tau.\lambda z{:}\sigma.xz\,(yz) : (\sigma{\to}(\tau{\to}\rho)){\to}((\sigma{\to}\tau){\to}(\sigma{\to}\rho))$ from 8, by rule 3.

The reader can notice that the types of the final terms in the previous examples resemble tautologies from intuitionistic (as well as classical) propositional logic. As we will see later, this is not a coincidence.

1.2.1.1 β-reduction and Meta-theoretical Properties

Apart from the typing system, λ-calculi feature a reduction mechanism capturing the idea of function application. This mechanism is called β-reduction, is denoted by $\to_\beta$, and functions in the following manner:

$$(\lambda x{:}\sigma.M)\,N \to_\beta M[N/x]$$

where $M[N/x]$ denotes the term M in which every free occurrence of the variable x has been substituted with the term N. For instance, the term $(\lambda x{:}\sigma.x)\,M$ reduces to $x[M/x] \equiv M$, and we can see that $\lambda x{:}\sigma.x$ behaves as the identity function. Terms of the form $(\lambda x{:}\sigma.M)\,N$ are called *β-redexes*, and we say that a term M is in *normal form* if it contains no β-redexes.

If one were to view λ-calculi as idealized functional programming languages, one could think of β-reduction as a computational step. These steps can be performed in sequence and can either terminate, when one of the normal forms of the term is reached, or can be performed indefinitely, depending on the term and calculus in question. Naturally, the preference would be for this sequence of reductions to terminate, and is expressed through the following property:

Strong Normalization. A term calculus is *strongly normalizing* if every sequence of reductions of every term of the calculus eventually terminates to a normal form. A term calculus satisfying this property can be viewed as a programming language in which every program terminates, which is a very useful property, but renders the calculus Turing-incomplete, meaning that there are computable functions that cannot be defined within the calculus.

As there can be several β-redexes at a time within one term, a natural question one could ask is "Does the order in which reductions are performed impact the final result?". The property addressing this issue is:

Confluence. If, for a given term calculus, the answer to this question is "No.", then the calculus is *confluent*. Using the symbol $\twoheadrightarrow_\beta$ to denote the reflexive and transitive closure of $\rightarrow_\beta$, confluence can be expressed more formally as follows:

If $M \twoheadrightarrow_\beta M'$ and $M \twoheadrightarrow_\beta M''$, then there exists an M''', such that $M' \twoheadrightarrow_\beta M'''$ and $M'' \twoheadrightarrow_\beta M'''$,

for every term M of the λ-calculus in question. In confluent calculi, all reduction sequence can be made to meet at a common expression. However, this does not mean that they *must* meet at this expression. Also, confluence implies uniqueness of normal forms, meaning that if a term M reduces to a term M' in normal form, then this normal form is unique. In other words, if there exists an M'' in normal form, such that M reduces to M'', then M' and M'' must be equal.

There are several more important properties that a λ-calculus would be desired to satisfy:

Subject Reduction. This property states that typing is preserved under β-reduction of the subject, *i.e.* if $\Gamma \vdash M : \sigma$, and $M \rightarrow_\beta M'$, then also $\Gamma \vdash M' : \sigma$.

Decidability of Type Checking. In a given signature, and for a given context Γ, term M and type σ, it is decidable whether or not $\Gamma \vdash M : \sigma$ is derivable.

Decidability of Type Inhabitation. In a given signature, and for a given context Γ and type σ, it is decidable whether or not there exists a term M, such that $\Gamma \vdash M : \sigma$ is derivable.

The simply-typed λ-calculus satisfies all of these properties, with the decidability of type inhabitation being PSPACE-complete. However, it is fairly restrictive, as we can neither have full recursion nor construct more expressive types which we could think of, for instance, polymorphic types or types depending on other types or terms.

1.2.2 The Lambda Cube

Apart from the simply-typed λ-calculus, there are many other more complex formalisms relying on assigning types to λ-terms. Some of these include *polymorphic types* (terms depending on types), *higher-order types* (types depending on types), *dependent types* (types depending on terms), *intersection types* (terms with multiple types), etc. An elegant way to subsume some of these systems is Barendregt's λ-cube [Barendregt 1992], presented in Figure 1.1.

How does the λ-cube actually work, and what is the mechanism which connects these eight λ-calculi? First, we select two constants, called *sorts*, and name them $*$ and $\Box$. In the literature and, perhaps, more informatively, they are often known as Type and Kind. If we let $\mathcal{S} = \{*, \Box\}$, and $s, s_1, s_2 \in \mathcal{S}$, we can express the eight systems via a set of general axioms and rules (Figure 1.2) and rules specific to each of the systems (Figure 1.3). Essentially, by instantiating the (s_1, s_2) specific rule with elements of $\mathcal{S}$ and adding these instantiations to the general axioms and rules, we can arrive to each of these eight systems.

One very interesting thing which can be observed here is the apparent "modularity" of this segment of typed lambda calculi. Starting from the core rules, we can arrive to many different systems and levels of expressiveness, each of them important in its own right, by adding instances

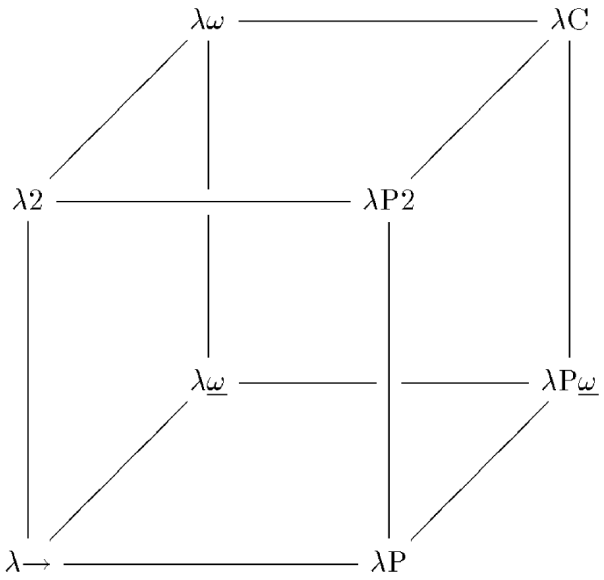


Figure 1.1: The λ-cube

of one single rule schema, akin to simply plugging in an add-on module, which we can also combine together. Let us examine it in more detail.

The $(*,*)$ instantiation of (s_1, s_2) gives us the simply-typed λ-calculus $\lambda_\rightarrow$, and places us in the bottom-left corner of the cube. From there, we can move in the following three directions:

1. **Upward**: by adding the $(\Box, *)$ rule, we arrive at the system λ2, which is also known as the second-order typed λ-calculus or Girard's System F [Girard 1972], where we can have terms with polymorphic types, such as $\Pi\alpha{:}*.(\alpha \rightarrow \alpha)$, arriving at terms depending on types, which would not be possible in $\lambda_\rightarrow$. Also, since here we can derive $\vdash \Pi\alpha{:}*.(\alpha \rightarrow \alpha) : *$, we effectively bring impredicativity (self-referencing) back in the game whenever we introduce type polymorphism (the four systems on the top side of the cube).

2. **Forward**: by adding the $(\Box, \Box)$ rule, we arrive at the system $\lambda\underline{\omega}$, where type constructors such as $\lambda\alpha{:}*.\alpha \rightarrow \alpha$ (types depending on types) can be formed and used to create new higher-order constructors and types, which cannot be used in $\lambda_\rightarrow$ or λ2. The systems allowing higher-order constructions are on the back side of the cube.

3. **Sideways**: by adding the $(*, \Box)$ rule, we arrive at the system λP, where we have dependent types, or types depending on terms. Here, we can, for instance, capture the notion of predicates, and have derivations such as $A{:}* \vdash (A \rightarrow *) : \Box$ (meaning that $A \rightarrow *$ is the kind of predicates on A), which is not possible in any of the previous systems.

The remaining three systems ($\lambda 2P$, $\lambda\omega$ and λC) are combinations of the four which we have presented here, with λC, which is also known as the Calculus of Constructions, being the most expressive one and combining all of the demonstrated features.

One may view typed λ-calculi as an interesting and amusing theoretical exercise. However, perhaps somewhat unexpectedly, they share a fundamental connection with mathematical logic and have very important applications in computer science. Typed lambda calculi are, in fact, in the foundations of programming languages, and form the base of typed functional programming languages such as Haskell and ML, as well as, albeit more indirectly, typed imperative programming languages.

$$\text{(axiom)} \quad <> \vdash * : \Box;$$

$$\text{(start rule)} \quad \frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}, \; x \notin \Gamma;$$

$$\text{(weakening rule)} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x{:}C \vdash A : B}, \; x \notin \Gamma;$$

$$\text{(application rule)} \quad \frac{\Gamma \vdash F : (\Pi x{:}A.B) \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]};$$

$$\text{(abstraction rule)} \quad \frac{\Gamma, x{:}A \vdash b : B \quad \Gamma \vdash (\Pi x{:}A.B) : s}{\Gamma \vdash (\lambda x{:}A.b) : (\Pi x{:}A.B)};$$

$$\text{(conversion rule)} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}.$$

Figure 1.2: The general axioms and rules for the λ-cube

The specific rule

$$(s_1, s_2) \text{ rule} \quad \frac{\Gamma \vdash A : s_1, \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.B) : s_2}$$

$$(*,*)\ \frac{\Gamma \vdash A : * \ , \quad \Gamma, x{:}A \vdash B : *}{\Gamma \vdash (\Pi x{:}A.B) : *} \qquad (\Box,*)\ \frac{\Gamma \vdash A : \Box, \quad \Gamma, x{:}A \vdash B : *}{\Gamma \vdash (\Pi x{:}A.B) : *}$$

$$(*,\Box)\ \frac{\Gamma \vdash A : * \ , \quad \Gamma, x{:}A \vdash B : \Box}{\Gamma \vdash (\Pi x{:}A.B) : \Box} \qquad (\Box,\Box)\ \frac{\Gamma \vdash A : \Box, \quad \Gamma, x{:}A \vdash B : \Box}{\Gamma \vdash (\Pi x{:}A.B) : \Box}$$

System	Set of specific rules			
$\lambda\!\rightarrow$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
λP2	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
λP$\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
λC	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

Figure 1.3: The specific axioms and rules for the λ-cube

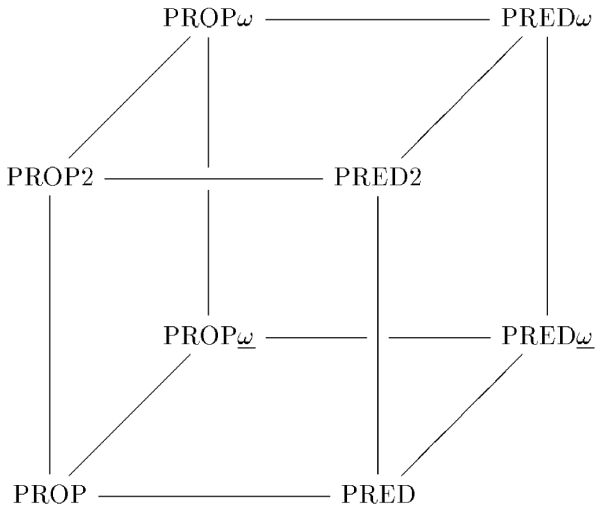
1.2.3 The Curry-Howard Correspondence

It was first noticed, by Haskell Curry [Curry 1934], that the types of some of the combinators ($\boldsymbol{K}$ and $\boldsymbol{S}$, for example) in the simply-typed λ-calculus could be seen as axiom schemata for intuitionistic implicational logic. Some twenty years later [Curry 1958], he concluded that Hilbert-style deduction systems correspond, in some fragment, to the typed fragment of combinatory logic, while Howard [Howard 1980] was the first to state that intuitionistic natural deduction can directly be interpreted as a typed variant of the λ-calculus. This observation is known as the *Curry-Howard correspondence*. Essentially, it tells us that two formalisms which we need

not have expected to be similar - lambda calculi (as models of computation) and mathematical logics (as proof systems) - are structurally the same.

Through the Curry-Howard correspondence, we arrive at the important notion of *propositions-as-types*, meaning that propositions of a proof system correspond to types within a typed λ-calculus. In that case, if the proposition is provable in the proof system, the corresponding type will be inhabited in the λ-calculus, and the term inhabiting it can be seen as its proof. Even further, this term can be seen as an executable program.

The logics corresponding to the systems of the λ-cube and the relationships between them, in a format similar to the λ-cube (called the logic cube), are shown in Figure 1.4. All of the logics of the logic cube are intuitionistic.



PROP	Propositional logic (PPL)	PROP2	Second-order PPL
PROP$\underline{\omega}$	Weakly higher-order PPL	PROPω	Higher-order PPL
PRED	Predicate logic (PRL)	PRED2	Second-order PRL
PRED$\underline{\omega}$	Weakly higher-order PRL	PREDω	Higher-order PRL

Figure 1.4: The Logic Cube

1.3 Interactive Theorem Proving and the Proof Assistant Coq

An *interactive theorem prover* or a *proof assistant* is a software tool that allows the user to describe within it concepts such as mathematical theories or programming languages, formulate a certain problem or state certain properties with respect to that concept, and specify a solution or verify that that property holds. The proving process is guided by the user and assisted by the proof assistant, in the sense that the user provides the steps and the proof assistant verifies that these steps are correct. The steps themselves mostly mimic pen-and-paper reasoning, and involve concepts such as proof by induction or contradiction, case analysis, application of already proven theorems, and many more. While parts of the proof within a proof assistant can be automated, it is the user who concentrates on the creative details and truly guides the proving process, separating clearly proof assistants from *automated theorem provers*, whose task is to autonomously construct the entire proof.

The proofs produced using automated and interactive theorem provers have a higher degree of certainty when compared to pen-and-paper proofs, because they rely on a *trusted core* - a

collection of code on which the theorem prover is based, that is small enough to be manually verifiable, that is declared correct, and upon which all of the subsequent inferences are made. This higher degree of certainty is required in fields and contexts (such as medicine, aviation, robotics and security) where software faults could introduce a life-threatening risk.

Currently, there is a number of interactive theorem provers available to researchers, all essentially defined by their respective underlying formalisms. The Curry-Howard correspondence has paved way for the development of several of them, while the others rely on set theory and higher-order logic. Here is a non-exhaustive list of the most widely used proof assistants today:

- **ACL2 - A Computational Logic for Applicative Common Lisp** [Kaufmann 2000] is a software system consisting of a programming language, an extensible theory in a first-order logic, and a mechanical (capable of working in both interactive and automatic modes) theorem prover, in the Boyer-Moore tradition [Boyer 1995]. It is designed to support automated reasoning in inductive logical theories, mostly for the purpose of software and hardware verification.

- **Agda** [The Agda development team 2013] is a proof assistant for developing constructive proofs as well as a functional programming language with dependent types. It is based on the idea of the Curry-Howard correspondence. It has a certain degree of support for tactics and proofs are dominantly written in functional programming style. The language has ordinary programming constructs such as data types, pattern matching, records, let expressions and modules, and a Haskell-like syntax.

- **Coq** [The Coq development team 2013, Bertot 2004] is based on an extension of λC known as the Calculus of Inductive Constructions [Coquand 1988, Paulin-Mohring 1996], and is, therefore, at the very "top" of the λ-cube when it comes to expressiveness. It allows the expression of mathematical assertions, is capable of mechanically checking proofs of these assertions, allows the use of various tactics for the finding of these formal proofs, and can extract a certified program from the constructive proof of its formal specification. It has been developed over the past two decades in the French National Institute for Research in Computer Science and Control (INRIA). Coq is written in a typed functional language called Objective CaML, an extension of the core Caml language with a fully-fledged object-oriented layer, also developed in INRIA.

- **HOL** [Gordon 2013] (Higher Order Logic) denotes a family of interactive theorem proving systems sharing similar (higher-order) logics and implementation strategies. Systems in this family are implemented as a library in some programming language. This library implements an abstract data type of proven theorems so that new objects of this type can only be created using the functions in the library which correspond to inference rules in higher-order logic. As long as these functions are correctly implemented, all theorems proven in the system must be valid. The latest system from this family is HOL4 [The Hol4 development team 2013].

- **Isabelle** [The Isabelle development team 2013] is an interactive theorem prover, successor of the HOL theorem prover. It is written in Standard ML, and is based on a small logical core guaranteeing logical correctness. Isabelle is generic: it provides a meta-logic (a weak type theory), which is used to encode object logics like First-order logic (FOL), Higher-order logic (HOL) or Zermelo-Fraenkel set theory (ZFC). Isabelle's main proof method is a higher-order version of resolution, based on higher-order unification. Though interactive, Isabelle also features efficient automatic reasoning tools, such as a term rewriting engine and a tableaux prover, as well as various decision procedures.

- **PVS** [The PVS development team 2013] is a mechanized environment for formal specification and verification. PVS consists of a specification language, a number of predefined theories, a type checker, an interactive theorem prover that supports the use of several decision procedures and a symbolic model checker, various utilities including a code generator and a random tester, documentation, formalized libraries, and examples that illustrate different methods of using the system in several application areas. It is based on a kernel consisting of an extension of Church's type theory with dependent types, and is fundamentally a classical typed higher-order logic.

- **Twelf** [The Twelf development team 2013] is language used to specify, implement, and prove properties of deductive systems such as programming languages and logics. It features dependent types, relies on the Curry-Howard correspondence and is based on the λP calculus and the LF logical framework.

Let us now take a more detailed look into the inner workings of Coq, the proof assistant relevant for this thesis. Coq offers a variety of tools and mechanisms for encoding and verification:

- **All the tools of λC.** As the basis of Coq is the Calculus of Inductive Constructions, which subsumes λC, we have immediately at our disposal dependent types, polymorphism, and higher-order constructions. All of the objects in Coq have a type, and even types have their own types, which we call *sorts*. In Coq, there are two sorts: Type and Prop, the former applicable to data, and the latter for logical propositions.

- **Inductive types.** Using inductive types, *i.e.* by defining types through a case-by-case description of the type, with possibly making use of previously constructed terms of that type, it is possible to naturally encode notions such as natural numbers, integers, lists, binary trees, propositional formulas, and virtually any structure inductive in nature. For instance, we would encode natural numbers as

```
Inductive nat : Type = 0 : nat | S : nat -> nat.
```

- **Induction.** As a mechanism for proving assertions over inductive types, structural induction is one of the built-in mechanisms of Coq. Furthermore, it is possible to define a custom induction principle, which, of course, has to first be proven correct.

- **Proof tactics.** There is a number of methods for proof simplification, or tactics, readily available. Some of these tactics involve simple manipulations on the syntactic and semantic levels of higher-order intuitionistic logic and equality (`intros`, `apply` or `reflexivity`), whereas others involve more complicated mechanisms (`induction` or `inversion`) or the use of computational tools (`omega`).

- **Records.** Coq also provides support for records, which are essentially collections of objects. Using records, we can, for instance, elegantly exploit dependent types to express side conditions of a certain type. We will illustrate this on the following example:

```
Record typeRestriction (U : Type) (W : U -> Prop) : Type :=
    mkTypeRes {origU :> U; inW : W origU}.
```

  where, starting from a given type `U` and its subset `W` we have constructed a new type `typeRestriction U W` which contains only the elements of `U` which are in `W`. There are three things we can notice about this definition: it is parametric over `U` and `W`, it features dependent types (since the type of `W` depends on the term `U`, and the type of the

condition `inW` depends on the term `origU`), and it also features another of Coq's useful mechanisms, which is *coercion*, denoted by `:>`, which tells the system that objects of type `typeRestriction U W` can be treated, if necessary, as objects of type `U`.

- **Infinite objects.** In addition to inductive types, Coq also offers co-inductive types, which allow reasoning about infinite objects while still remaining in the finite confines of a computer. Some of the objects we can construct in this manner are streams, lazy lists and lazy binary trees, whereas one of the notions we can capture using co-inductive types is bisimilarity.

- **Modules.** Coq provides pre-formalized collections of facts, lemmas, and theorems regarding various topics, such as set theory, number theory and linear algebra, all organized in importable modules. The user can also define his own modules, which can be compiled and re-used across developments. As the underlying logic of Coq is intuitionistic, there exists a module (`Classical`) which adds the double negation elimination axiom to the axiom pool, making it possible to reason in classical terms.

- **Program extraction.** Coq has the possibility to extract certified and efficient functional programs from either Coq functions or Coq proofs of specifications. The output language of these programs can be Objective CaML, Haskell or Scheme. One of the most significant, and one of the currently most complex examples of program extraction is the CompCert C certified compiler of C-light (a large subset of the C language), intended for compilation of life-critical and mission-critical software [Leroy 2009a] [Leroy 2009b].

- **CoqIDE.** Last, but not least, Coq comes with a graphical user interface called CoqIDE, which is highly interactive, giving the user an overview of the current proof code, the state of the current proof, and the options at his disposal.

So far, many important and fundamental theorems have already been proven in Coq, such as the denumerability of rational numbers, the non-denumerability of the continuum, Gödel's Incompleteness Theorem, the Cayley-Hamilton theorem. Stirling's formula, etc. However, as the most important ones we could single out the Four Color Theorem (stating that, given any separation of a plane into contiguous regions, producing a map, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color) [Gonthier 2004] or, more recently, the Feit-Thompson Theorem (stating that every finite group of odd order is solvable) [Gonthier 2012].

In conclusion, we present a very simple illustrative proof in Coq, showing that the well-known formula $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$ is indeed a tautology. The claim is stated as a `Lemma`, the name of which is `example`, and the syntax of the formulation is quite intuitive. Parentheses are written mostly for the benefit of the reader, as most of them are not needed, due to the built-in associativity of the intuitionistic implication. `Prop` is the sort for propositions.

```
Lemma example : forall A B C : Prop, (A -> (B -> C)) -> ((A -> B) -> (A -> C)).

Proof.              // The proof context is empty at this point, and our goal is stated in the format
                       encountered after the name of the lemma.

intros A B C.       // The proposition has to hold for all A, B and C of sort Prop (boolean values - true
                       and false). Hence, we can pull them into the context as A : Prop, B : Prop, and
                       C : Prop, leaving us with the formula without the leading forall part as the goal.

intros Habc Hab Ha. // We usually prove an implication of the form A -> B by assuming to have A as an
                       assumption, and then attempting  to prove B, given that assumption. Here, we have
                       several implication which we can pull back into the context as assumptions -
                       Habc : A -> (B -> C) is the first one, followed by Hab : A -> B, and Ha : A,
                       leaving only C in the goal.
```

```
apply Habc.          // Also, if we have to prove a goal C, and we have an assumption of the form A -> C,
                        it would be sufficient to prove A. Here, given that we have an assumption
                        A -> (B -> C), it would be sufficient to prove A and to prove B. This splits the
                        goal in two - one for A, one for B.

assumption.          // We already have A in the assumptions.

apply Hab.           // Similarly as before, our goal is B, and we already have A -> B in the hypotheses.
                        Therefore, to prove our goal, it would be sufficient to prove A.

assumption.          // We already have A in the assumptions.

Qed.                 // All the goals have been proven.
```

1.4 Probability, λ-calculus, and Formal Verification

Several works on the treatment of probability in λ-calculi, as well as formal verification of algorithms and programs involving probability and/or randomness can be found in the literature.

In [Park 2006], the authors have presented a probabilistic language, called $\lambda_\circ$, which uniformly supports all types of probability distributions - discrete, continuous, and even those not belonging to either group. Its mathematical basis are sampling functions, *i.e.* , mappings from the unit interval $(0, 1]$ to probability domains. They have also described the implementation of $\lambda_\circ$ as an extension of Objective CAML and demonstrated its practicality with three applications in robotics, with experiments carried out with real robots.

In [Hurd 2002], where the proof assistant of choice was HOL, the author starts with an extensive formalization of measure theory and probability spaces. Next, using that formalization, the author proceeds to define the notion of a probabilistic program that terminates with probability 1, and shows, both in theory and by example, how probabilistic programs can be specified and formally verified in that setting.

In [Hurd 2005], the authors present a mechanization of the quantitative logic for the guarded command language pGCL [Morgan 1999], using the HOL theorem prover. pGCL is a language which allows for the expression of both probabilistic and demonic (arbitrary) choices, and it is suitable for reasoning about distributed random algorithms. The authors show that the mechanized theory supports the creation of an automatic proof tool which takes as input an annotated pGCL program and its partial-correctness specification, and derives from that a sufficient set of verification conditions.

Finally, in [Audebaud 2009], the authors rely on [Hurd 2002] and [Park 2006] to be able to reason in a formal setting (the proof assistant Coq) about properties of randomized algorithms, namely the validity of general rules for estimating the probability that a randomized algorithm satisfies a specified property.

In this thesis, the formalization of the mathematics surrounding probability theory has been executed on a smaller scale than in [Hurd 2002], on a need-to-have basis (confined to finitely additive measures), while probabilistic programs are not addressed, and the emphasis in put on meta-theoretic properties of probability logics.

Part I

Probability Logics

Chapter 2

LPCP - A Probability Logic with Conditional Probability Operators

Contents

LPCP is a probability logic with multiple conditional probability operators, based on classical logic. Here, we present its syntax and semantics, together with a corresponding strongly complete axiomatization. LPCP is the outcome of a research effort which has followed the lines of investigation presented in [Maksimović 2008, Ognjanović 2009, Fagin 1990, Lukasiewicz 2002, Rašković 2004, Ognjanović 2005] on the formal development of probability logics, in which probability-related statements can be expressed using probabilistic operators which specify bounds on probabilities of a propositional formulas. The work presented here originated from [Maksimović 2008], which has been presented at the 13^{th} ESSLLI Student Session in Hamburg, in 2008, and which developed into a journal publication [Doder 2010], published in the Publications de l'Institut Mathematique, in 2010.

As mentioned in the previous paragraph, in LPCP we have multiple conditional probability operators $CP_i, i \in \mathcal{I}$, where $\mathcal{I}$ denotes a finite non-empty set of indexes. These operators take as input two classical formulas α and β, and produce as output the conditional probability that α holds if β holds. One interpretation which could be given to these operators is that each of them represents an agent which has his own independent assessment of the conditional probability of an event. For instance, we would be able to formally write the statement "The conditional probability of an event α given an event β viewed by agent i is at least the sum of conditional probabilities of α given γ viewed by agent j and twice γ given α viewed by agent k." as

$$CP_i(\alpha, \beta) \geqslant CP_j(\alpha, \gamma) + \underline{2} \cdot CP_k(\gamma, \alpha).$$

We provide the syntax and semantics of LPCP, accompanied with a strongly complete axiomatization. We also prove that the developed logic is decidable, and show that it can be used to represent evidence, solving the problem of propositional axiomatization of reasoning about evidence, which has been presented in [Halpern 2006].

Before we examine the logic in detail, we will address a minor issue which is well-known in probability theory. In the classical Kolmogorovian sense, the conditional event "α given β" can

be considered only in the case when $P(\beta) > 0$, and for such a conditional event, we calculate its probability in the following way:

$$P(\alpha|\beta) = P(\alpha \wedge \beta) \cdot P(\beta)^{-1}. \tag{2.1}$$

As the $^{-1}$ operator is not total, since 0 does not have an inverse, this approach could introduce difficulties into the formal construction of probabilistic formulas, and it would be better if $P(\alpha|\beta)$ were to be a well-defined term, regardless of the α, β, and the possible value of $P(\beta)$.

We can reach an elegant solution to the situation by adopting the convention that $^{-1}$ is a total operation, so that we can extend Kolmogorov's definition of conditional probability onto all of the events: $P(\alpha|\beta) = P(\alpha \wedge \beta)P(\beta)^{-1}$. In particular, if $P(\beta) = 0$, then $P(\alpha \wedge \beta) = 0$, and

$$P(\alpha|\beta) = P(\alpha \wedge \beta)P(\beta)^{-1} = 0 \cdot P(\beta)^{-1} = 0.$$

If we look closely at this last equation, we can observe that the actual value of 0^{-1} is essentially irrelevant for the computation of $P(\alpha|\beta)$, and that in the case when $P(\beta) = 0$ the conditional probability defined as above does indeed behave correctly. For the sake of simplicity, we will take 0^{-1} to be equal to 1.

The remainder of this chapter is organized as follows: in Section 2.1, the syntax of the logic LPCP is presented and the class of measurable probabilistic models is described. Section 2.2 contains the corresponding axiomatization and introduces the notion of deduction. A proof of the strong completeness theorem for LPCP is presented in Section 2.4, whereas the decidability of the logic is analyzed in Section 2.5. Representing evidence in LPCP is discussed in Section Section 2.7, and the concluding remarks are in Section 2.8.

2.1 Syntax and Semantics of LPCP

Let $Var = \{p_n \mid n < \omega\}$ be the set of propositional variables. The corresponding set of all propositional formulas over Var will be denoted by For_C, and is defined as the smallest set which satisfies the following conditions:

1. it contains all of the propositional variables, *i.e.* $Var \subset For_C$,
2. it is closed under the following rules of construction: if α and β are propositional formulas, then $\neg\alpha$ and $\alpha \wedge \beta$ are propositional formulas.

Here, we will take $\neg$ and $\wedge$ to be the basic propositional connectives, and we define the remaining connectives in the usual manner for classical propositional logic. Propositional formulas will be denoted by α, β and γ, possibly with indexes. Next, let $\mathcal{I}$ be a finite non-empty set of indexes.

Definition 1. *The set $Term$ of all probabilistic terms is recursively defined as follows:*

- $Term(0) = \{\underline{s} \mid s \in \mathbb{Q}\} \cup \{CP_i(\alpha, \beta) \mid \alpha, \beta \in For_C,\ i \in \mathcal{I}\}$.
- $Term(n+1) = \{\mathtt{f}, (\mathtt{f} + \mathtt{g}), (\underline{s} \cdot \mathtt{g}), (-\mathtt{f}) \mid \mathtt{f}, \mathtt{g} \in Term(n),\ s \in \mathbb{Q}\}$.
- $Term = \bigcup_{n=0}^{\infty} Term(n)$.

Probabilistic terms will be denoted by $\mathtt{f}, \mathtt{g}$ and $\mathtt{h}$, possibly with indexes. To simplify our notation, we introduce the following convention: $\mathtt{f} + \mathtt{g}$ will represent $(\mathtt{f} + \mathtt{g})$, $\mathtt{f} + \mathtt{g} + \mathtt{h}$ will represent $((\mathtt{f} + \mathtt{g}) + \mathtt{h})$. For $n > 3$, $\sum_{i=1}^{n} \mathtt{f}_i$ will represent $((\cdots((\mathtt{f}_1 + \mathtt{f}_2) + \mathtt{f}_3) + \cdots) + \mathtt{f}_n)$. Similarly, $-\mathtt{f}$ will represent $(-\mathtt{f})$ and $\mathtt{f} - \mathtt{g}$ will represent $(\mathtt{f} + (-\mathtt{g}))$. The intended meaning

behind probabilistic terms is for them to represent linear combinations of rationally-weighted conditional probabilities, in the sense that each term can be rewritten (using the axioms and inference rules from the next Section, which essentially correspond to the axioms of rational numbers) into an expression of the form

$$\left(\sum_{i=1}^{n} \underline{s_i} \cdot CP_i(\alpha_i, \beta_i)\right) + \underline{s}. \tag{2.2}$$

Finally, if α and β are propositional formulas, and $i \in \mathcal{I}$, then we will have the probabilistic term $CP_i(\alpha, \beta)$ to be read as "the conditional probability of α given β viewed by agent i". To further simplify the notation, we will be writing $P_i(\alpha)$ instead of $CP_i(\alpha, \top)$, where $\top$ denotes an arbitrary tautology instance.

Definition 2. *A basic probabilistic formula is any formula of the form* $\mathtt{f} \geqslant \underline{0}$. *The set of all probabilistic formulas is defined as the smallest set which satisfies the following conditions:*

1. *it contains all of the basic propositional formulas,*
2. *it is closed under the following rules of construction: if* $\mathtt{f}$ *and* $\mathtt{g}$ *are propositional formulas, then* $\neg\mathtt{f}$ *and* $\mathtt{f} \wedge \mathtt{g}$ *are propositional formulas.*

As in the case of propositional formulas, illustrated above, here we also treat $\neg$ and $\wedge$ as the primitive connectives, while all of the other connectives are introduced in the usual way, as in classical propositional logic. Probabilistic formulas will be denoted by ϕ, ψ and θ, possibly with indexes, while the set of all probabilistic formulas will be denoted by For_P. Furthermore, we define the following abbreviations:

1. $\mathtt{f} \leqslant \underline{0}$ represents $-\mathtt{f} \geqslant \underline{0}$.
2. $\mathtt{f} > \underline{0}$ represents $\neg(\mathtt{f} \leqslant \underline{0})$.
3. $\mathtt{f} < \underline{0}$ represents $\neg(\mathtt{f} \geqslant \underline{0})$.
4. $\mathtt{f} = \underline{0}$ represents $\mathtt{f} \leqslant \underline{0} \ \wedge \ \mathtt{f} \geqslant \underline{0}$.
5. $\mathtt{f} \neq \underline{0}$ represents $\neg(\mathtt{f} = \underline{0})$.
6. $\mathtt{f} \geqslant \mathtt{g}$ represents $\mathtt{f} - \mathtt{g} \geqslant \underline{0}$.

We define $\mathtt{f} \leqslant \mathtt{g}$, $\mathtt{f} > \mathtt{g}$, $\mathtt{f} < \mathtt{g}$, $\mathtt{f} = \mathtt{g}$ and $\mathtt{f} \neq \mathtt{g}$ in a similar way. By "formula", we will be referring to either a propositional formula or a probabilistic formula. We do not allow for the mixing of propositional and probabilistic formulas, nor for the nesting of probability operators CP_i. Therefore, it is not possible to have formulas of the form $\alpha \wedge CP_i(\beta, \gamma) \geq \underline{0}$ or formulas such as $CP_i(CP_j(\alpha, \beta), \gamma)$ in LPCP. Formulas will be denoted by Φ, Ψ and Θ, possibly with indexes. The set of all of the formulas will be denoted by For.

Next, we proceed to the semantics of LPCP. We define the notion of a model as a special kind of Kripke model. Namely, *a model* M is any quadruple $\langle W, H, \{\mu_i \mid i \in \mathcal{I}\}, v\rangle$ such that:

- W is a non-empty set. As usual, we will be referring to its elements as worlds.
- H is an algebra of sets over W.
- for each $i \in \mathcal{I}$, $\mu_i : H \longrightarrow [0, 1]$ is a finitely additive probability measure.
- $v : For_C \times W \longrightarrow \{0, 1\}$ is a truth assignment compatible with $\neg$ and $\wedge$. That is, $v(\neg\alpha, w) = 1 - v(\alpha, w)$ and $v(\alpha \wedge \beta, w) = v(\alpha, w) \cdot v(\beta, w)$.

For a given model M, we will denote by $[\alpha]_M$ the set of all $w \in W$ such that $v(\alpha, w) = 1$. If the context is clear, we will write $[\alpha]$ instead of $[\alpha]_M$. We say that M is a *measurable model* if $[\alpha] \in H$ for all $\alpha \in For_C$.

Definition 3. *Let $M = \langle W, H, \{\mu_i \mid i \in \mathcal{I}\}, v\rangle$ be any measurable model. We define the satisfiability relation $\models$ recursively as follows:*

1. $M \models \alpha$ *if* $v(\alpha, w) = 1$ *for all* $w \in W$.
2. $M \models \mathtt{f} \geqslant \underline{0}$ *if* $\mathtt{f}^M \geqslant 0$, *where* $\mathtt{f}^M$ *is recursively defined in the following way:*
   - $\underline{s}^M = s$.
   - $CP_i(\alpha, \beta)^M = \mu_i([\alpha \wedge \beta]) \cdot \mu_i([\beta])^{-1}$, *for any* $i \in \mathcal{I}$.
   - $(\mathtt{f} + \mathtt{g})^M = \mathtt{f}^M + \mathtt{g}^M$.
   - $(\underline{s} \cdot \mathtt{g})^M = s \cdot \mathtt{g}^M$.
   - $(-\mathtt{f})^M = -(\mathtt{f}^M)$.
3. $M \models \neg\phi$ *if* $M \not\models \phi$.
4. $M \models \phi \wedge \psi$ *if* $M \models \phi$ *and* $M \models \psi$.

A formula Φ is *satisfiable* if there exists a measurable model $M = \langle W, H, \{\mu_i \mid i \in \mathcal{I}\}, v\rangle$ such that $M \models \Phi$; Φ is *valid* if it is satisfied in every measurable model. We say that the set T of formulas is *satisfiable* if there is a measurable model M such that $M \models \Phi$ for all $\Phi \in T$. The reader can notice that the last two clauses of Definition 3 guarantee the validity of each tautology instance.

2.2 An Axiomatization of LPCP

In this section, we will introduce the axioms and inference rules for LPCP. The set of axioms of our axiomatic system, which we denote by AX_{LPCP}, is divided into three groups: axioms for propositional reasoning, axioms for probabilistic reasoning, and arithmetical axioms.

Axioms for propositional reasoning:

A1a. $\tau(p_1, \ldots, p_n)$ - all tautologies instantiated with propositional formulas

A1b. $\tau(\Phi_1, \ldots, \Phi_n)$ - all tautologies instantiated with probabilistic formulas

Axioms for probabilistic reasoning ($i \in \mathcal{I}$):

A2. $P_i(\alpha) \geqslant \underline{0}$.	Non-negativity of probability
A3. $P_i(\top) = \underline{1}$.	Something true will always occur
A4. $P_i(\bot) = \underline{0}$.	Something false will never occur
A5. $P_i(\alpha \leftrightarrow \beta) = \underline{1} \rightarrow P_i(\alpha) = P_i(\beta)$.	Equivalent formulas are equally probable
A6. $P_i(\alpha \vee \beta) = P_i(\alpha) + P_i(\beta) - P_i(\alpha \wedge \beta)$.	The inclusion–exclusion principle
A7. $P(\beta) = 0 \rightarrow CP(\alpha, \beta) = 0$;	Disallowing division by zero

Capturing conditional probability:

A8. $(P_i(\alpha \wedge \beta) \geq \underline{r} \wedge P_i(\beta) \leq \underline{s}) \rightarrow CP_i(\alpha, \beta) \geq \underline{r \cdot s^{-1}}$, $s \neq 0$.
A9. $(P_i(\alpha \wedge \beta) > \underline{r} \wedge P_i(\beta) \leq \underline{s}) \rightarrow CP_i(\alpha, \beta) > \underline{r \cdot s^{-1}}$, $s \neq 0$.
A10. $(P_i(\alpha \wedge \beta) \geq \underline{r} \wedge P_i(\beta) < \underline{s}) \rightarrow CP_i(\alpha, \beta) > \underline{r \cdot s^{-1}}$, $s \neq 0$.
A11. $(P_i(\alpha \wedge \beta) \leq \underline{r} \wedge P_i(\beta) \geq \underline{s}) \rightarrow CP_i(\alpha, \beta) \leq \underline{r \cdot s^{-1}}$, $s \neq 0$.
A12. $(P_i(\alpha \wedge \beta) < \underline{r} \wedge P_i(\beta) \geq \underline{s}) \rightarrow CP_i(\alpha, \beta) < \underline{r \cdot s^{-1}}$, $s \neq 0$.
A13. $(P_i(\alpha \wedge \beta) \leq \underline{r} \wedge P_i(\beta) > \underline{s}) \rightarrow CP_i(\alpha, \beta) < \underline{r \cdot s^{-1}}$, $s \neq 0$.

Arithmetical axioms:

A14. $\underline{r} > \underline{s}$, whenever $r > s$.	Compatibility of $>$ with $>$ of $\mathbb{Q}$
A15. $\underline{r} \geqslant \underline{s}$, whenever $r \geqslant s$.	Compatibility of $\geq$ with $\geq$ of $\mathbb{Q}$
A16. $\underline{s} \cdot \underline{r} = \underline{sr}$.	Compatibility of $\cdot$ with $\cdot$ of $\mathbb{Q}$, part 1
A17. $\underline{s} + \underline{r} = \underline{s + r}$.	Compatibility of $+$ with $+$ of $\mathbb{Q}$
A18. $\mathtt{f} + \mathtt{g} = \mathtt{g} + \mathtt{f}$.	Commutativity of $+$
A19. $(\mathtt{f} + \mathtt{g}) + \mathtt{h} = \mathtt{f} + (\mathtt{g} + \mathtt{h})$.	Associativity of $+$
A20. $\mathtt{f} + \underline{0} = \mathtt{f}$.	0 is the neutral for $+$
A21. $\mathtt{f} - \mathtt{f} = \underline{0}$.	Inverse terms for $+$
A22. $(\underline{r} \cdot \mathtt{f}) + (\underline{s} \cdot \mathtt{f}) = \underline{r + s} \cdot \mathtt{f}$.	Distributivity of $\cdot$ with $+$
A23. $\underline{s} \cdot (\mathtt{f} + \mathtt{g}) = (\underline{s} \cdot \mathtt{f}) + (\underline{s} \cdot \mathtt{g})$.	Distributivity of $+$ with $\cdot$
A24. $\underline{r} \cdot (\underline{s} \cdot \mathtt{f}) = \underline{r \cdot s} \cdot \mathtt{f}$.	Compatibility of $\cdot$ with $\cdot$ of $\mathbb{Q}$, part 2
A25. $\underline{1} \cdot \mathtt{f} = \mathtt{f}$.	Neutrality of $\underline{1}$
A26. $\mathtt{f} \geqslant \mathtt{g} \ \vee \ \mathtt{g} \geqslant \mathtt{f}$.	Totality of $\geq$
A27. $(\mathtt{f} \geqslant \mathtt{g} \ \wedge \ \mathtt{g} \geqslant \mathtt{h}) \rightarrow \mathtt{f} \geqslant \mathtt{h}$.	Transitivity of $\geq$
A28. $\mathtt{f} \geqslant \mathtt{g} \ \rightarrow \ \mathtt{f} + \mathtt{h} \geqslant \mathtt{g} + \mathtt{h}$.	Compatibility of $\geq$ with $+$
A29. $(\mathtt{f} \geqslant \mathtt{g} \ \wedge \ \underline{s} > \underline{0}) \ \rightarrow \underline{s} \cdot \mathtt{f} \geqslant \underline{s} \cdot \mathtt{g}$.	Compatibility of $\geq$ with $\cdot$
A30. $\mathtt{f} = \mathtt{g} \rightarrow (\phi(\ldots, \mathtt{f}, \ldots) \rightarrow \phi(\ldots, \mathtt{g}, \ldots))$.	Contextual closure for terms

Inference rules:

R1. From α and $\alpha \rightarrow \beta$ infer β; from ϕ and $\phi \rightarrow \psi$ infer ψ.

R2. From α infer $P_i(\alpha) = \underline{1}$, for all $i \in \mathcal{I}$.

R3. From the set of premises $\{\phi \ \rightarrow \ \mathtt{f} \geqslant \underline{-n^{-1}} \mid n = 1, 2, 3, \ldots\}$ infer $\phi \rightarrow \mathtt{f} \geqslant \underline{0}$.

Let us briefly comment on the axioms and inference rules. The axioms A2 through A6 provide the required properties of probability, the axioms A7 through A13 capture Equation 2.1, using the fact that $\mathbb{Q}$ is dense in $\mathbb{R}$, while the axioms A14 through A30 provide the properties required for computation. As for the inference rules, R1 is modus ponens for classical and probabilistic formulas, R2 resembles necessitation, while the infinitary rule R3 ensures that non-Archimedean probabilities are not permitted. Using the rule R3, we are able to syntactically deal with the inherent non-compactness[1] of LPCP and achieve strong completness. Also, we can see that there exist two levels of formulas - propositional and probabilistic, and that these levels are separate, in the sense that it is possible to cross from the propositional to the probabilistic level using the inference rule R2, but there is no mechanism to provide a crossing in the opposite direction.

Definition 4 (Derivability). *A formula Φ is* derivable *from a set of formulas T[2] ($T \vdash \Phi$) if there is an at most countable sequence of formulas $\Phi_0, \Phi_1, \ldots, \Phi$, such that every Φ_i is an axiom or a formula from the set T, or it is derived from the preceding formulas by an inference rule. Particularly, a formula Φ is a* theorem *($\vdash \Phi$) if it is deducible from the empty set of hypotheses.*

[1] In LPCP, it is possible to construct a set of formulas that is not satisfiable, but whose every finite subset is satisfiable. An example of this set is the set $\{\phi \rightarrow \mathtt{f} > 0\} \cup \{\phi \ \rightarrow \ \mathtt{f} < \underline{-n^{-1}} \mid n = 1, 2, 3, \ldots\}$.

[2] We refer to formulas in T as assumptions or hypotheses.

Definition 5 (Consistent and Inconsistent Sets). *A set of formulas T is* consistent *if there is at least one formula from For_C and at least one formula from For_P that are not deducible from T. Otherwise, T is* inconsistent.

Definition 6 (Maximally Consistent Sets). *A consistent set T of sentences is said to be* maximally consistent *if for every $\alpha \in For_C$, it holds that if $T \vdash \alpha$ then $\alpha \in T$ and $(P_i(\alpha) = \underline{1}) \in T$, for all $i \in \mathcal{I}$, and for every $\phi \in For_P$, either $\phi \in T$ or $\neg\phi \in T$.*

Definition 7 (Deductively Closed Sets). *A set T is* deductively closed *if for every $\Phi \in For$, if $T \vdash \Phi$, then $\Phi \in T$.*

With the presented axioms and inference rules, one can notice that the length of the inference may be any successor ordinal lesser than the first uncountable ordinal ω_1, *i.e.* that we can have infinite (countable) proofs in LPCP.

2.3 Soundness of LPCP

The first meta-theoretical result we will prove about our logic will be the soundness theorem, which follows easily from the formulation of the axiom system and the inference rules.

Theorem 1 (Soundness). *If Φ is a theorem of AX_{LPCP}, then it is valid.*

Proof. Essentially, we need to prove that the instances of all of the axioms of AX_{LPCP} are valid, and that applications of the inference rules preserve validity. As the axiom schemes are checked to be valid in the usual manner, we will focus on the application of the inference rules:

R1. Let us assume that the classical formulas α and $\alpha \to \beta$ are valid. This means that α and $\alpha \to \beta \equiv \neg(\alpha \wedge \neg\beta)$ are true in every world $w \in W$ of every model $M = \langle W, H, \{\mu_i | i \in \mathcal{I}\}, v\rangle$. However, since we have that $\neg(\alpha \wedge \neg\beta)$ is true, then it must be that β is true, which concludes this part of the proof. The part for the modus ponens for probabilistic formulas is proven similarly.

R2. Let us assume that the classical formula α is valid. This means that α is true in every world $w \in W$ of every model $M = \langle W, H, \{\mu_i | i \in \mathcal{I}\}, v\rangle$. Therefore, we have that $[\alpha] = W$, meaning that we have that $\mu_i([\alpha]) = \mu_i(W) = 1$, for all $i \in \mathcal{I}$. This, in turn, means that $M \models P_i(\alpha) = \underline{1}$, which concludes this part of the proof.

R3. Let us assume that the set of probabilistic formulas $\{\phi \to \mathtt{f} \geq \underline{-n^{-1}} \mid n = 1, 2, 3, \ldots\}$ is valid, *i.e.* that for every model $M = \langle W, H, \{\mu_i | i \in \mathcal{I}\}, v\rangle$, it holds that $M \models \{\phi \to \mathtt{f} \geq \underline{-n^{-1}} \mid n = 1, 2, 3, \ldots\}$. Now, if $M \not\models \phi$, then $M \models \phi \to \mathtt{f} \geq \underline{0}$ immediately, as the antecedent of the (classical) implication is false. Therefore, let us assume that $M \models \phi$. Then, given the definition of implication via conjunction and negation, we have that $M \models \{\mathtt{f} \geq \underline{-n^{-1}} \mid n = 1, 2, 3, \ldots\}$, and we need to prove that $M \models \mathtt{f} \geq \underline{0}$. Given the definition of satisfiability, we have that $\{\mathtt{f}^M \geq \frac{1}{n} \mid n = 1, 2, 3, \ldots\}$, and we need to prove that $\mathtt{f}^M \geq 0$, which follows from the axiomatization of real numbers. □

2.4 Strong Completeness of LPCP

In this section, we will prove that the proposed axiomatization is strongly complete with respect to the class of all measurable models. As is quite common, the first step to take is to prove the Deduction Theorem.

Theorem 2 (Deduction theorem). *Let T be an arbitrary set of formulas, and let $\Phi, \Psi \in For_C$ or $\Phi, \Psi \in For_P$. Then, it holds that $T \vdash \Phi \to \Psi$ iff $T \cup \{\Phi\} \vdash \Psi$.*

Proof. First, let us assume that $T \vdash \Phi \to \Psi$. From this, regardless of whether Φ and Ψ are classical or probabilistic, we have that $T \cup \{\Phi\} \vdash \Phi \to \Psi$. However, we also have that $T \cup \{\Phi\} \vdash \Phi$. Therefore, using the appropriate modus ponens (inference rule R1), we obtain that $T \cup \{\Phi\} \vdash \Psi$.

Conversely, let $T \cup \{\Phi\} \vdash \Psi$. We proceed by structural induction on this inference with the goal of proving $T \vdash \Phi \to \Psi$. In case when the last applied rule was R1, the proof proceeds just as for classical logic. Next, let the last applied rule be the inference rule R2. In that case, we have that Ψ is of the form $P_i(\alpha) = \underline{1}$, for some $i \in \mathcal{I}$, which was obtained from $T \cup \{\Phi\} \vdash \alpha$. However, as the classical and probabilistic formulas are separate, the derivation of classical formulas does not depend on probabilistic formulas, and we have that $T \vdash \alpha$. From there, using the inference rule R2, we obtain that $T \vdash P_i(\alpha) = \underline{1}$. From here, using the tautology $\phi_0 \to (\phi_1 \to \phi_0)$ and the axiom A1b, we obtain the desired $T \vdash \Phi \to (P_i(\alpha) = \underline{1})$.

Finally, let the last applied rule be R3. Then, Ψ is of the form $\phi \to \mathtt{f} \geqslant \underline{0}$ and we have, by the induction hypothesis, that $T \vdash \Phi \to (\phi \to \mathtt{f} \geqslant \underline{-n^{-1}})$ for all n. Since the formula $(\phi_0 \to (\phi_1 \to \phi_2)) \leftrightarrow ((\phi_0 \wedge \phi_1) \to \phi_2)$ is a tautology, we also have that $T \vdash (\Phi \wedge \phi) \to \mathtt{f} \geqslant \underline{-n^{-1}}$, for all n, with the help of the axiom A1b. Now, by the inference rule R3, we have that $T \vdash (\Phi \wedge \phi) \to \mathtt{f} \geqslant \underline{0}$, and by using the same tautology as above, that $T \vdash \Phi \to \Psi$. With this proof, the reader can appreciate why the '$\phi \to$' "prefix" in needed in the rule R3 – without it, we could not have used the tautology required for the crucial application of the rule R3. □

We will use the result of the next technical lemma in the construction of a maximally consistent extension of a consistent set of formulas.

Lemma 1. *Let T be a consistent set of formulas. If $T \cup \{\phi \to \mathtt{f} \geqslant \underline{0}\}$ is inconsistent, then there exists a positive integer n such that $T \cup \{\phi \to \mathtt{f} < \underline{-n^{-1}}\}$ is consistent.*

Proof. Let us assume the converse, *i.e.* that $T \cup \{\phi \to \mathtt{f} < \underline{-n^{-1}}\}$ is inconsistent for all positive integers n. Therefore, we have that, for all positive integers n, $T \cup \{\phi \to \mathtt{f} < \underline{-n^{-1}}\} \vdash \bot$. From this, using the Deduction Theorem, we have that

$$T \vdash \phi \to \mathtt{f} \geqslant \underline{-n^{-1}},$$

for all positive integers n. Now, using the inference rule R3, we have that $T \vdash \phi \to \mathtt{f} \geqslant \underline{0}$, and so we have that T must be inconsistent, which is in contradiction with our original assumption, therefore completing our proof. □

Definition 8. *Let T be a consistent set of formulas, and let us have an enumeration of probabilistic formulas, i.e. let $For_P = \{\phi_i \mid i \in \omega\}$[3]. We define a completion T^* of T as follows:*

1. $T_0 = T \cup \{\alpha \in For_C \mid T \vdash \alpha\} \cup \{P_i(\alpha) = \underline{1} \mid T \vdash \alpha,\ i \in \mathcal{I}\}$.
2. *If $T_i \cup \{\phi_i\}$ is consistent, then $T_{i+1} = T_i \cup \{\phi_i\}$.*
3. *If $T_i \cup \{\phi_i\}$ is inconsistent, then:*
   (a) *If ϕ_i is of the form $\psi \to \mathtt{f} \geqslant \underline{0}$, then $T_{i+1} = T_i \cup \{\psi \to \mathtt{f} < \underline{-n^{-1}}\}$, where n is a positive integer such that T_{i+1} is consistent*[4].
   (b) *Otherwise, $T_{i+1} = T_i$.*

[3] This is possible, as probabilistic formulas are countable by construction.
[4] The existence of such an n is provided by Lemma 1.

4. $T^* = \bigcup_{n=0}^{\infty} T_n$.

We can prove that each T_i is consistent by construction. In the next theorem, we will prove that T^* is maximally consistent and deductively closed.

Theorem 3. *Let T be a consistent set of formulas, and let T^* be constructed as above. Then:*

1. *For each $\phi \in For_P$, either $\phi \in T^*$ or $\neg\phi \in T^*$.*
2. *T^* is consistent, i.e.*
   (a) *There is a formula $\alpha \in For_C$, such that $\alpha \notin T^*$.*
   (b) *There is a formula $\phi \in For_P$, such that $\phi \notin T^*$.*
3. *T^* is deductively closed, i.e. if $T^* \vdash \Phi$ then $\Phi \in T^*$.*

Proof.

1. Let us assume the opposite: let ϕ be a probabilistic formula such that both $\phi \in T^*$ and $\neg\phi \in T^*$, or neither $\phi \in T^*$ nor $\neg\phi \in T^*$. Without loss of generality, we can assume that, in the above enumeration, $\phi \equiv \phi_i$ and $\neg\phi \equiv \phi_j$, and let $k = \max(i, j)$. Then, by construction, in the first case, we have that $T_{k+1} \vdash \phi$, and $T_{k+1} \vdash \neg\phi$. However, then we also have that $T_{k+1} \vdash \phi \wedge \neg\phi$, *i.e.* that T_{k+1} is inconsistent, which is not possible. In the second case, by construction, we have that $T_{k+1} \cup \{\phi\} \vdash \bot$, and that $T_{k+1} \cup \{\neg\phi\} \vdash \bot$. From this, using the Deduction Theorem and the elimination of double negation, we again obtain that $T_{k+1} \vdash \phi$, and $T_{k+1} \vdash \neg\phi$, and from that, we have that $T_{k+1} \vdash \phi \wedge \neg\phi$, *i.e.* that T_{k+1} is inconsistent, which is still not possible, and we have our claim.

2. (a) Since there are no classical formulas added during the formation of T^*, all of the classical formulas which are in T^* are also in T_0. Therefore, if it were to hold that all classical α are in T^*, then all classical α would also have to be in T_0, making it inconsistent by definition, which is not the case. Therefore, there must be a classical formula α, such that $\alpha \notin T$.
   (b) This is a direct consequence of item 1.

   At this point, taking into consideration the manner in which T_0 is constructed, we have effectively proven that T^* is maximally consistent.

3. We will proceed by structural induction on the derivation of $T^* \vdash \Phi$.
   (a) Let Φ be an instance of an axiom. Then, if $\Phi \in For_C$ we have that $\Phi \in T_0 \subseteq T^*$. Otherwise, $\Phi \in For_P$, and there is a non-negative integer i such that $\Phi \equiv \phi_i$ of the above enumeration. Then, since $\vdash \Phi$, it must be that $T_i \vdash \phi_i$ and so $\Phi \in T_{i+1} \subseteq T^*$.
   (b) Let $T^* \vdash \Phi \equiv \beta$ be obtained from $T^* \vdash \alpha$ and $T^* \vdash \alpha \to \beta$ by the inference rule R1. Then, by the construction of T^*, we have that $\alpha \to \beta \in T$ and $\alpha \in T$, and from there, by the construction of T_0, we have that $\beta \in T_0 \subseteq T^*$.
   (c) Let $T^* \vdash \Phi \equiv \phi$ be obtained from $T^* \vdash \psi$ and $T^* \vdash \psi \to \phi$ by the inference rule R1, and let $\psi \equiv \phi_i$, $\psi \equiv \phi_j$, and $\phi \to \psi \equiv \phi_k$ in the above enumeration. Then, ϕ is a deductive consequence of each T_l, where $l \geqslant \max(i, k) + 1$. Next, let $\neg\phi \equiv \phi_m$. If $\phi_m \in T_{m+1}$, then $\neg\phi$ is a deductive consequence of each T_n, where $n \geqslant m + 1$. Therefore, for every $n \geqslant \max(i, k, m) + 1$, we would have that $T_n \vdash \phi \wedge \neg\phi$, which is a contradiction with the consistency of T_n. Thus, it must be that $\neg\phi \notin T^*$. However, given item 1, it must then be that $\phi \in T^*$.

(d) Let $T^* \vdash \Phi \equiv P_i(\alpha) = \underline{1}$, for some $i \in \mathcal{I}$ be obtained from $T^* \vdash \alpha$ by the inference rule R2. However, this means that $T \vdash \alpha$ and, by the construction of T_0, we have that $P_i(\alpha) = \underline{1} \in T_0 \subseteq T^*$ for all $i \in \mathcal{I}$.

(e) Let $T^* \vdash \Phi \equiv \phi \rightarrow \mathtt{f} \geqslant \underline{0}$ be obtained from $T^* \vdash \phi \rightarrow \mathtt{f} \geqslant \underline{-n^{-1}}$, for all $n \geq 0$. By the induction hypothesis, we then have that $\phi \rightarrow \mathtt{f} \geqslant \underline{-n^{-1}} \in T^*$, for all $n \geq 0$. Next, let $\phi \rightarrow \mathtt{f} \geqslant \underline{0} = \phi_i$ in the above enumeration, and let us assume that $\phi_i \notin T^*$. Then, $T_i \cup \{\phi_i\}$ must be inconsistent, which means that, for some positive integer n, $\phi \rightarrow \mathtt{f} < \underline{-n^{-1}} \in T_{i+1}$, according to item 3a of Definition 8. Next, let $\phi \rightarrow \mathtt{f} \geq \underline{-n^{-1}} \equiv \phi_j$ in the above enumeration. Then, for all $k \geq \max(i, j) + 1$, we have that $T_k \vdash \phi \rightarrow \mathtt{f} < \underline{-n^{-1}}$ and $T_k \vdash \phi \rightarrow \mathtt{f} \geq \underline{-n^{-1}}$, which is in contradiction with the consistency of T_k. Therefore, it must be that $\Phi \in T$, and we have our claim.

□

Next, for a given consistent set of formulas T and its completion to a maximally consistent set T^*, we proceed to define a *canonical model* $M^* = \langle W, H, \{\mu_i | i \in \mathcal{I}\}, v\rangle$ as follows:

1. W is the set of all functions $w : For_C \longrightarrow \{0, 1\}$ with the following properties:
   - w is compatible with $\neg$ and $\wedge$.
   - $w(\alpha) = 1$ for each $\alpha \in T^*$.
2. $v : For_C \times W \longrightarrow \{0, 1\}$ is defined by $v(\alpha, w) = 1$ iff $w(\alpha) = 1$.
3. $H = \{[\alpha] \mid \alpha \in For_C\}$.
4. $\mu_i : H \longrightarrow [0, 1]$ is defined by $\mu_i([\alpha]) = \sup\{s \in [0, 1] \cap \mathbb{Q} \mid T^* \vdash P_i(\alpha) \geqslant \underline{s}\}$, for all $i \in \mathcal{I}$.

Lemma 2. *M^* is a measurable model.*

Proof. Here, we need to prove that H is an algebra of sets and that each μ_i is indeed a finitely additive probability measure. Given the properties of the measure, we have that $[\alpha] \cap [\beta] = [\alpha \wedge \beta]$, $[\alpha] \cup [\beta] = [\alpha \vee \beta]$ and $H \setminus [\alpha] = [\neg\alpha]$, from which we obtain that H is an algebra of sets. Concerning the measures μ_i, the non-negativity requirement ($\mu_i([\alpha]) \geq 0$) is the consequence of axiom A2 and the definition of μ_i, while the requirement that $\mu_i(W) = 1$ follows directly from the axiom A3, since $W = [\top]$. Now, we will turn our attention to the proof of finite additivity. Let $\mu_i([\alpha]) = a$, $\mu_i([\beta]) = b$ and $\mu_i([\alpha \wedge \beta]) = c$. What we would like to prove is that

$$\mu_i([\alpha \vee \beta]) = a + b - c.$$

Since the set of rational numbers $\mathbb{Q}$ is dense in $\mathbb{R}$, we can choose an increasing sequence $\underline{a}_0 < \underline{a}_1 < \underline{a}_2 < \cdots$ and a decreasing sequence $\overline{a}_0 > \overline{a}_1 > \overline{a}_2 > \cdots$ in $\mathbb{Q}$ such that $\lim \underline{a}_n = \lim \overline{a}_n = a$. Using the definition of μ_i and item 1 of Theorem 3, we obtain that $T^* \vdash P_i(\alpha) \geqslant \underline{a}_n$ and that $T^* \vdash P_i(\phi) < \overline{a}_n$, for all positive integers n. In the same way, we can also choose increasing sequences $(\underline{b}_n)_{n\in\omega}$ and $(\underline{c}_n)_{n\in\omega}$, and decreasing sequences $(\overline{b}_n)_{n\in\omega}$ and $(\overline{c}_n)_{n\in\omega}$ in $\mathbb{Q}$, such that $\lim \underline{b}_n = \lim \overline{b}_n = b$ and $\lim \underline{c}_n = \lim \overline{c}_n = c$. For these sequences, we have that $T^* \vdash P_i(\beta) \geqslant \underline{b}_n \wedge P_i(\beta) < \overline{b}_n$ and $T^* \vdash P_i(\alpha \wedge \beta) \geqslant \underline{c}_n \wedge P_i(\alpha \wedge \beta) < \overline{c}_n$. From here, using the arithmetical axioms, we have that

$$T^* \vdash P_i(\alpha) + P_i(\beta) - P_i(\alpha \wedge \beta) \geqslant \underline{a}_n + \underline{b}_n - \overline{c}_n$$

and

$$T^* \vdash P_i(\alpha) + P_i(\beta) - P_i(\alpha \wedge \beta) < \overline{a}_n + \overline{b}_n - \underline{c}_n$$

for all positive integers n. Using the axioms A6 and A30, we obtain that $T^* \vdash P_i(\alpha \vee \beta) \geqslant \underline{a}_n + \underline{b}_n - \overline{c}_n$ and that $T^* \vdash P_i(\alpha \vee \beta) < \overline{a}_n + \overline{b}_n - \underline{c}_n$, for all n. Finally, from the fact that

$$\mu_i([\alpha \vee \beta]) = \sup\{r \in \mathbb{Q} \mid T^* \vdash P_i(\alpha \vee \beta) \geqslant r\}$$

and $\lim \underline{a}_n + \underline{b}_n - \overline{c}_n = \lim \overline{a}_n + \overline{b}_n - \underline{c}_n = a + b - c$, we obtain that $\mu_i([\alpha \vee \beta]) = a + b - c$. $\square$

Now, we are ready to prove the main theorem:

Theorem 4 (Strong Completeness)**.** *Every consistent set of formulas has a measurable model.*

Proof. Let T be a consistent set of formulas. We can extend it to a maximally consistent set T^*, and define a canonical model M^*, as above. We will now prove that $M^* \models \Phi$ iff $\Phi \in T^*$ by induction on the complexity of the formulas. First, we prove that if $M^* \models \Phi$, then $\Phi \in T^*$:

1. As the base case, let $M^* \models \alpha$. By the completeness of classical propositional logic, we have that $T^* \vdash \alpha$, and $\alpha \in T^*$.

2. Let $M^* \models \mathtt{f} \geqslant \underline{0}$, *i.e.* $f^{M^*} \geq 0$. If $\mathtt{f} \geqslant \underline{0} \notin T^*$, from the construction of T^*, there exists a positive integer n, such that $\mathtt{f} < \underline{-n^{-1}} \in T^*$. But then, by the construction of M^*, we would have that $\mathtt{f}^{M^*} < 0$, which is a contradiction. Therefore, we have that $\mathtt{f} \geqslant \underline{0} \in T^*$.

3. Let $M^* \models \neg\phi$. Then, by definition of $\models$, we get that $M^* \not\models \phi$. By the induction hypothesis, we have that $\phi \notin T^*$, and finally, by item 1 of Theorem 3, we have that $\neg\phi \in T^*$.

4. Finally, let $M^* \models \phi \wedge \psi$. Then, by definition of $\models$, we have that $M^* \models \phi$ and $M^* \models \psi$. By the induction hypothesis, we get that $\phi, \psi \in T^*$, and finally, by item 3 of Theorem 3, we have that $\phi \wedge \psi \in T^*$.

Here, it is important to notice that that $T^* \vdash CP_i(\alpha, \beta) \geq \underline{r}$ implies that $\mu_i([\alpha_i \wedge \beta_i])\mu_i([\beta_i])^{-1} \geq r$. Indeed, if it were that $\mu_i([\alpha_i \wedge \beta_i])\mu_i([\beta_i])^{-1} < r$, there would exist $a, b \in \mathbb{Q}$, such that $a > \mu_i([\alpha_i \wedge \beta_i])$, $b \leq \mu_i([\beta_i])$, and $\mu_i([\alpha_i \wedge \beta_i])\mu_i([\beta_i])^{-1} < \frac{a}{b} < r$. Using the previously proven direction, we would have that $T^* \vdash P_i(\alpha \wedge \beta) < \underline{a}$ and $T^* \vdash P_i(\beta) \geq \underline{b}$, and hence, by A12, that $T^* \vdash CP_i(\alpha, \beta) < \underline{a \cdot b^{-1}}$, which would be a contradiction. Similarly, we can show that $T^* \vdash CP_i(\alpha, \beta) \leq \underline{r}$ implies that $\mu_i([\alpha_i \wedge \beta_i])\mu_i([\beta_i])^{-1} \leq r$. As a consequence, we have that $\mu_i([\alpha_i]|[\beta_i]) = \sup\{r \in \mathbb{Q} \mid T^* \vdash CP_{n_i}(\alpha, \beta) \geqslant \underline{r}\}$. Next, we proceed to prove the opposite direction: if $\Phi \in T^*$, then $M^* \models \Phi$:

1. As the base case, let $\Phi = \alpha \in For_C$. If $\alpha \in T^*$, *i.e.* $T^* \vdash \alpha$, then, by definition of M^*, $M^* \models \alpha$.

2. Let us suppose that $\mathtt{f} \geqslant \underline{0} \in T^*$. Then, using the axioms A16–A19, A22–A25 and A30, we can prove that

$$T^* \vdash \mathtt{f} = \underline{s} + \sum_{i=1}^{m} \underline{s_i} \cdot CP_{n_i}(\alpha_i, \beta_i) \text{ and } T^* \vdash \underline{s} + \sum_{i=1}^{m} \underline{s_i} \cdot CP_{n_i}(\alpha_i, \beta_i) \geqslant \underline{0},$$

   for some $s, s_i \in \mathbb{Q}$ and some $\alpha_i, \beta_i \in For_C, n_i \in \mathcal{I}$. This is not surprising, given the intended design of terms (cf. Equation 2.2). Moreover, given the axioms A7, A16, A20 and A30, we can effectively cancel out all of the subterms of $\mathtt{f}$ in which we have conditional probabilities $CP_i(\alpha, \beta)$ with $P_i(\beta) = 0$, and, therefore, may assume that $T^* \vdash P_{n_i}(\beta_i) > \underline{0}$. Next, let $a_i = \mu_{n_i}([\alpha_i \wedge \beta_i])$ and $b_i = \mu_{n_i}([\beta_i])$. We need to prove that

$$s + \sum_{i=1}^{m} s_i \cdot a_i \cdot b_i^{-1} \geqslant 0. \tag{2.3}$$

Therefore, we can choose increasing sequences $(c^{inc}_{i,k})_{k\in\omega}$ and decreasing sequences $(c^{dec}_{i,k})_{k\in\omega}$ in $\mathbb{Q}$, such that $\lim c^{inc}_{i,k} = \lim c^{dec}_{i,k} = a_i b_i^{-1}$, for $i \in \{1,\dots,m\}$. Hence, we have that $T^* \vdash CP_{n_i}(\alpha,\beta) \geqslant \underline{c^{inc}_{i,k}} \wedge CP_{n_i}(\alpha,\beta) < \underline{c^{dec}_{i,k}}$, for $i \in \{1,\dots,m\}$ and $k \in \omega$.

Without loss of generality, we can assume that $T^* \vdash \underline{s_i} \geqslant \underline{0}$, for $1 \leqslant i \leqslant l$, and $T^* \vdash \underline{s_i} < \underline{0}$, for $l < i \leqslant m$. Then, using the arithmetical axioms, we obtain that

$$T^* \vdash \underline{s} + \sum_{i=1}^{l} \underline{s_i} \cdot \underline{c^{inc}_{i,k}} + \sum_{i=l+1}^{m} \underline{s_i} \cdot \underline{c^{dec}_{i,k}} \leqslant \underline{s} + \sum_{i=1}^{m} \underline{s_i} \cdot CP_{n_i}(\alpha_i,\beta_i)$$

and

$$T^* \vdash \underline{s} + \sum_{i=1}^{l} \underline{s_i} \cdot \underline{c^{dec}_{i,k}} + \sum_{i=l+1}^{m} \underline{s_i} \cdot \underline{c^{inc}_{i,k}} \geqslant \underline{s} + \sum_{i=1}^{m} \underline{s_i} \cdot CP_{n_i}(\alpha_i,\beta_i)$$

for all k. Consequently, we have that

$$s + \sum_{i=1}^{m} s_i \cdot a_i \cdot b_i^{-1} = \sup\{r \in \mathbb{Q} \mid T^* \vdash \underline{s} + \sum_{i=1}^{m} \underline{s_i} \cdot CP_{n_i}(\alpha_i,\beta_i) \geqslant \underline{r}\}.$$

Now, Equation 2.3 follows from $T^* \vdash \underline{s} + \sum_{i=1}^{m} \underline{s_i} \cdot CP_{n_i}(\alpha_i,\beta_i) \geqslant \underline{0}$.

3. Let $\neg\phi \in T^*$. Then, by item 1 of Theorem 3, we have that $\phi \notin T^*$. Next, by the induction hypothesis, we have that $M^* \not\models \phi$, and finally, by the definition of satisfiability, we obtain that $M^* \models \neg\phi$.

4. Finally, let $\Phi = \phi \wedge \psi \in T^*$. Then, by item 3 of Theorem 3, we have that ϕ, $\psi \in T^*$. Next, by the induction hypothesis, we have that $M^* \models \phi$ and $M^* \models \psi$, and finally, by the definition of satisfiability, we obtain that $M^* \models \phi \wedge \psi$.

□

2.5 Decidability of LPCP

Theorem 5. *Satisfiability of probabilistic formulas is decidable.*

Proof. Up to equivalence, every probabilistic formula is a finite disjunction of finite conjunctions of literals, where a literal is either a basic probabilistic formula, or a negation of a basic probabilistic formula. Thus, it would be sufficient to show the decidability of the satisfiability problem for formulas of the form

$$\bigwedge_i \mathtt{f}_i \geqslant \underline{0} \ \wedge \ \bigwedge_j \mathtt{g}_j < \underline{0}. \tag{2.4}$$

Suppose that $p_1,\dots,p_n$ are all of the propositional letters appearing in Equation 2.4. Let $A_1,\dots,A_{2^n}$ be all of the formulas of the form

$$\pm p_1 \wedge \cdots \wedge \pm p_n,$$

where $+p = p$ and $-p = \neg p$. We can see that A_i are pairwise-disjoint and form a partition of $\top$. Furthermore, for each classical formula α appearing in Equation 2.4, there exists a unique set $I_\alpha \subseteq \{1,\dots,2^n\}$ such that

$$\alpha \leftrightarrow \bigvee_{i\in I_\alpha} A_i$$

is a tautology. Now, given Equation 2.2, we can equivalently rewrite Equation 2.4 as

$$\bigwedge_i \sum_{i'} \underline{s_{ii'}} CP_{n_{i'}}\Big(\bigvee_{k\in I_{\alpha_{ii'}}} A_k, \bigvee_{l\in I_{\beta_{ii'}}} A_l\Big) \geqslant \underline{r}_i \ \wedge \ \bigwedge_j \sum_{j'} \underline{s_{jj'}} CP_{n_{j'}}\Big(\bigvee_{k\in I_{\alpha_{jj'}}} A_k, \bigvee_{l\in I_{\beta_{jj'}}} A_l\Big) < \underline{r}_j.$$

Let the set $\{i_1, \ldots, i_m\} \subseteq \mathcal{I}$ be the set of all of the different conditional probability operator indexes which appear in Equation 2.4, and let $\sigma_i(x_{(1,i_1)}, \ldots, x_{(2^n,i_1)}, \ldots, x_{(1,i_m)}, \ldots, x_{(2^n,i_m)})$, $\delta_j(x_{(1,i_1)}, \ldots, x_{(2^n,i_1)}, \ldots, x_{(1,i_m)}, \ldots, x_{(2^n,i_m)})$ be the formulas

$$\sum_{i'} s_{ii'} \cdot \Big(\sum_{k\in I_{\alpha_{ii'}}\cap I_{\beta_{ii'}}} x_{(k,n_{i'})}\Big) \cdot \Big(\sum_{l\in I_{\beta_{ii'}}} x_{(l,n_{i'})}\Big)^{-1} \geqslant r_i$$

and

$$\sum_{j'} s_{jj'} \cdot \Big(\sum_{k\in I_{\alpha_{jj'}}\cap I_{\beta_{jj'}}} x_{(k,n_{j'})}\Big) \cdot \Big(\sum_{l\in I_{\beta_{jj'}}} x_{(l,n_{j'})}\Big)^{-1} < r_j.$$

Furthermore, let $\chi(x_{(1,i_1)}, \ldots, x_{(2^n,i_1)}, \ldots, x_{(1,i_m)}, \ldots, x_{(2^n,i_m)})$ be the formula

$$\bigwedge_{h\in\{1,\ldots,m\}} x_{(1,i_h)} + \cdots x_{(2^n,i_h)} = 1 \wedge \bigwedge_{k,h} x_{(k,i_h)} \geqslant 0.$$

Then, we can notice that Equation 2.4 is satisfiable *if and only if* the sentence

$$\exists x_{(1,i_1)} \ldots \exists x_{(2^n,i_1)} \ldots \exists x_{(1,i_m)} \ldots \exists x_{(2^n,i_m)} \Big(\bigwedge_i \sigma_i(\bar{x}) \ \wedge \ \bigwedge_j \delta_j(\bar{x}) \wedge \ \chi(\bar{x})\Big) \tag{2.5}$$

is satisfied in the ordered field of reals. Formally, the first order language of fields does not contain "$^{-1}$" and "$-$". However, both of these functions are easily definable. For instance, "$^{-1}$" can be defined by the following formula:

$$\varphi(x, y) : (x = 0 \to y = 1) \wedge (x \neq 0 \to xy = 1),$$

since the first order sentence $(\forall x)(\exists_1 y)\varphi(x, y)$ is satisfied in every field. In particular, the formula $\psi(\ldots, x^{-1}, \ldots)$ holds if and only if the formula $\psi(\ldots, y, \ldots) \wedge \varphi(x, y)$ holds. Therefore, Equation 2.5 can be seen as a first-order formula belonging to the language of ordered fields L_{OF} – definitions by extensions, see [Shoenfield 1967]. Based on results presented in [Marker 2002], we have that the satisfiability of sentences of L_{OF} in the ordered field of reals is decidable.

It should be noted here that our choice to extend the operation $^{-1}$ so that $0^{-1} = 1$ does not affect the aforementioned reasoning, as the conditional probability of α given β is equal to 0 *iff* the probability of $\alpha \wedge \beta$ is equal to 0, regardless of the value of the probability of β. □

Now, let us suppose that there is only one conditional probability operator, *i.e.* that there exists only one agent. It should be noted that this logic can be embedded into the logic described in [Fagin 1990], which has a PSPACE containment for the decision procedure. Also, the rewriting of formulas from our logic into that logic can be accomplished in linear time:

$$CP(\alpha, \beta) \text{ is equivalent to } \tfrac{\omega(\alpha\wedge\beta)}{\omega(\beta)},$$

which can be represented easily in [Fagin 1990]. Furthermore, the generalization of the logic from [Fagin 1990] to a multi-agent case is straightforward. Therefore, we conclude that our logic is also decidable in PSPACE.

2.6 An Interlude: Reasoning about Evidence

Imagine the following scenario: we have two coins, one of which is fair (with a head on one side and a tail on the other, with the probability of 50% for each of the outcomes), while the other one is extremely biased (e.g. with a probability of only 1% for the tail to appear on top). We also have the results of a series of tosses originating from one of the coins, but we do not know which. Furthermore, we have no information whatsoever regarding prior probability assigned to the choice of the coin, and we would like to make some sort of probabilistic inference as to which coin has actually been tossed only from the results of the tosses. Then, we can view these results as *evidence* in favor of or against one of the coins, and it is precisely this notion that we would like to model. The model should capture our intuition about the matter in question. For instance, if we were to have a series of 50 tosses with only three tails, we could induce that the coin in question is the biased one, while if we were to have a series of 50 tosses with 24 heads and 26 tails, we could induce that the coin in question is the fair one.

The topic of evidence has been discussed comprehensively in philosophical literature, namely philosophy of science and, more precisely, confirmation theory, where the main topic of discussion is the extent to which experimental evidence lends credence to various scientific theories [Good 1950, Popper 1959, Carnap 1962, Milne 1996].

There are three things which a framework capturing evidence should incorporate: probabilistic outcomes (such as the outcome of the actual coin toss), non-probabilistic outcomes (such as the choice of the coin toss), and, mainly, the change in likelihood of a hypothesis caused by the previous probabilistic outcomes. While there were many research efforts which have successfully been able to capture the first two requirements [Halpern 1993, de Alfaro 1997, He 1995], it was not until [Halpern 2006] that the third one was also realized.

2.7 Representing Evidence with LPCP

In [Halpern 2006], Halpern and Pucella have presented a first-order logic for reasoning about evidence. It includes propositional formulas on hypotheses $\mathcal{H}$, observations $\mathcal{O}$, probabilities P_1 and P_2 of formulas before and after the observation, the evidence $E(o,h)$ provided by the observation o for the hypothesis h, and quantification by real-valued variables. They have posed an open question whether it would be possible to axiomatize their logic without using quantification. Intuitively, the evidence function e represents the "weight" with which an observation leads to the fulfillment of a hypothesis. Also, it was shown that evidence can be seen as a function which maps prior probability P_1 to posterior probability P_2, using Dempster's Rule of Combination. For more details, we refer the reader to [Halpern 2006]. In this section, we will show how evidence can be represented using LPCP. We will introduce the following modifications:

1. There is a finite number of propositional letters divided into two categories: $Var = \mathcal{H} \cup \mathcal{O}$, where $\mathcal{H} = \{h_1, \ldots, h_m\}$ are used to denote hypotheses, $\mathcal{O} = \{o_1, \ldots, o_n\}$ are used to denote observations, and $\mathcal{H} \cap \mathcal{O} = \emptyset$.

2. There are only two conditional probability operators – CP_1 and CP_2, which will be interpreted as prior and posterior conditional probabilities, respectively;

3. there is an additional syntactic object – $E(o,h)$, where $o \in \mathcal{O}$, $h \in \mathcal{H}$.

4. The definition of $Term(0)$ is adjusted accordingly to:

$$Term(0) = \{\underline{s} \mid s \in \mathbb{Q}\} \cup \{CP_i(\alpha, \beta) \mid \alpha, \beta \in For_C, i \in \mathcal{I}\} \cup \{E(o,h) | o \in \mathcal{O}, h \in \mathcal{H}\}.$$

5. The definition of a model is extended to be a sextuple $\langle W, H, \{\mu_i | i \in \mathcal{I}\}, v, e\rangle$, where we have $e : (\{[o]|o \in \mathcal{O}\} \times [\{[h]|h \in \mathcal{H}\}]) \to [0,1]$ by the formula

$$e([o],[h]) = \frac{\mu_1([o \wedge h]) \cdot \mu_1([h])^{-1}}{\sum_{k=1}^{m} \mu_1([o \wedge h_k]) \cdot \mu_1([h_k])^{-1}},$$

for $o \in \mathcal{O}, h \in \mathcal{H}$.

6. The definition of satisfiability is extended to include:

$$E(o,h)^M = (CP_1(o,h)^M) \cdot \left(\sum_{k=1}^{m} CP_1(o,h_k)^M \right)^{-1},$$

for $o \in \mathcal{O}, h \in \mathcal{H}$.

7. There are nine additional axioms:

A31. $(\bigvee_{i=1}^{m} h_i) \wedge (\bigwedge_{i \neq j} (h_i \to \neg h_j))$.

A32. $(\bigvee_{i=1}^{n} o_i) \wedge (\bigwedge_{i \neq j} (o_i \to \neg o_j))$.

A33. $\bigwedge_{i=1}^{n} (CP_1(o_i,h_1) + \ldots + CP_1(o_i,h_m) > 0)$.

A34. $(CP_1(o_i,h_j) \geqslant r \wedge CP_1(o_i,h_1) + \ldots + CP_1(o_i,h_m) \leqslant s) \to E(o_i,h_j) \geqslant r \cdot s^{-1}$.

A35. $(CP_1(o_i,h_j) \geqslant r \wedge CP_1(o_i,h_1) + \ldots + CP_1(o_i,h_m) < s) \to E(o_i,h_j) > r \cdot s^{-1}$.

A36. $(CP_1(o_i,h_j) > r \wedge CP_1(o_i,h_1) + \ldots + CP_1(o_i,h_m) \leqslant s) \to E(o_i,h_j) > r \cdot s^{-1}$.

A37. $(CP_1(o_i,h_j) \leqslant r \wedge CP_1(o_i,h_1) + \ldots + CP_1(o_i,h_m) \geqslant s) \to E(o_i,h_j) \leqslant r \cdot s^{-1}$.

A38. $(CP_1(o_i,h_j) \leqslant r \wedge CP_1(o_i,h_1) + \ldots + CP_1(o_i,h_m) > s) \to E(o_i,h_j) < r \cdot s^{-1}$.

A39. $(CP_1(o_i,h_j) < r \wedge CP_1(o_i,h_1) + \ldots + CP_1(o_i,h_m) \geqslant s) \to E(o_i,h_j) < r \cdot s^{-1}$,

8. There is one additional inference rule:

$$\text{R4. } \frac{o_i}{P_2(h_j) = CP_1(h_j|o_i)}, i \in \{1,\ldots,n\}, j \in \{1,\ldots,m\}.$$

It can be shown, very similarly to the already laid-out proofs, that the logic with these modifications in place is also strongly complete and decidable in PSPACE. In this way, we have solved the problem of propositional axiomatization of reasoning about evidence, which was presented in [Halpern 2006].

2.8 Summary

In this chapter, we have introduced a sound and strongly-complete axiomatic system for LPCP- a probability logic with conditional probability operators $CP_i, i \in \mathcal{I}$, which allows for linear combinations and comparative statements. As it was noticed in [van der Hoek 1997], it is not possible to provide a finitary strongly complete axiomatization for such a system. In our case, the strong completeness was made possible by adding an infinitary rule of inference.

The formalism which we have obtained is fairly expressive and allows for the representation of uncertain knowledge, where uncertainty is modeled by probability formulas. We have shown it to be decidable, and also that it can be extended in order to be used to represent evidence, making it the first propositional axiomatization of reasoning about evidence.

Chapter 3

Formal Verification of Key Properties for $LPP_2^{\mathbb{Q}}$ - a Rationally-valued Probability Logic without Iterations - in the Proof Assistant Coq

Contents

In this chapter, we introduce $LPP_2^{\mathbb{Q}}$ - a probability logic that extends classical logic with probabilistic operators of the form $P_{\geq r}\alpha$, with the meaning that the probability that the classical formula α holds is at least r. This logic does not allow iterations of probability operators. We present its syntax and semantics, as well as a strongly complete axiomatic system. We formally prove its main meta-theoretical properties (soundness, strong and simple completeness, and non-compactness) using the proof assistant Coq.

3.1 Syntax of $LPP_2^{\mathbb{Q}}$

In the logic $LPP_2^{\mathbb{Q}}$, there exist two types of formulas: classical and probabilistic. For the classical formulas, we require a denumerably infinite set of propositional letters: $\phi = \{p_0, p_1, \ldots\}$, and two propositional connectives: $\neg_c$ and $\rightarrow_c$. The set of all classical formulas For_C is then defined as the smallest set that satisfies the following conditions:

- it contains all of the propositional letters, and
- is closed under the following formation rules: if $\alpha, \beta \in For_C$, then $\neg_c\alpha, \alpha \rightarrow_c \beta \in For_C$.

These are, in fact, formulas as they are commonly defined in classical propositional logic and, as we will see in more detail later on, classical propositional logic will serve us as a base logic underneath the probabilistic level. In the encoding of classical formulas (represented by the inductive type `forC`), the propositional letters are represented by Coq's built-in type for natural

numbers `nat`, while the connectives $\neg_c$ and $\rightarrow_c$ are, respectively, represented by the constructors `NegC : forC -> forC` and `ImpC : forC -> forC -> forC`.

Next, let us denote by $\mathbb{Q}_{[0,1]}$ the set of all rational numbers belonging to the unit interval. For the probabilistic formulas, we require a denumerably infinite set of probabilistic operators $P_{\geq s}, s \in \mathbb{Q}_{[0,1]}$, and two propositional connectives: $\neg_p$ and $\rightarrow_p$. A basic probabilistic formula is an expression of the form $P_{\geq s}\alpha$, where $s \in \mathbb{Q}_{[0,1]}$, and $\alpha \in For_C$. The set of all probabilistic formulas For_P is then defined as the smallest set satisfying the following conditions:

- it contains all of the basic probabilistic formulas, and
- is closed under the following formation rules: if $A, B \in For_P$, then $\neg_p A, A \rightarrow_p B \in For_P$.

Here, the intended meaning of the probabilistic operators, using the example of $P_{\geq s}\alpha$, for $s \in \mathbb{Q}_{[0,1]}$, and a classical formula α, is that the probability that α holds is at least s. With this, we have constructed a new layer of formulas on top of classical logic, and within this layer we are able to assign probabilities to the classical formulas themselves. As we inspect the formation rules, we can notice that it is neither possible to combine classical and probabilistic formulas via a binary connective, nor is it possible to iterate the probabilistic operators. Therefore, in this logic, we cannot obtain formulas such as $p_0 \rightarrow P_{\geq \frac{1}{2}} p_1$ or $P_{\geq \frac{1}{2}}(P_{\geq \frac{1}{3}} p_0)$.

To be able to encode probabilistic formulas in Coq, we first require a type which could play the role of a rational number from the unit interval. This has been accomplished using `Record` types, by imposing appropriate side conditions on the already present type `Qc`, the obtained record type is called `Q01`, and coerces to `Qc`. The probabilistic formulas are then represented by the inductive type `forP`, the connectives $\neg_p$ and $\rightarrow_p$ are, respectively, represented by its constructors `NegP` and `ImpP` (handled analogously to their classical counterparts), and, in particular, the constructor for probabilistic operators $P_{\geq s}$ is of the form:

```
| Pge :  Q01 -> forC -> forP
```

Finally, the set of all formulas $For_{LPP_2^{\mathbb{Q}}}$ (represented by the inductive type `FOR`) is defined as the union of all classical and probabilistic formulas:

$$For_{LPP_2^{\mathbb{Q}}} = For_C \cup For_P.$$

```
Inductive FOR : Type :=
  | Clas : forC -> FOR
  | Prob : forP -> FOR.
```

Hereinafter, when we are not addressing an encoding of a concept in Coq in detail, we will specify the label of the encoding in brackets, immediately following the theoretical explanation. The remaining classical and probabilistic propositional connectives $\vee_c$ (`OrC`), $\wedge_c$ (`AndC`), $\leftrightarrow_c$ (`EquC`), and $\vee_p$ (`OrP`), $\wedge_p$ (`AndP`), $\leftrightarrow_p$ (`EquP`) are defined in the usual way, while, additionally, for the probabilistic operators, we introduce the following abbreviations:

(`Plt`) $P_{<s}\alpha$ for $\neg_p P_{\geq s}\alpha$.
(`Pgt`) $P_{>s}\alpha$ for $\neg_p P_{\leq s}\alpha$.
(`Pne`) $P_{\neq s}\alpha$ for $\neg_p P_{=s}\alpha$.
(`FalP`) $\bot_p$ for $\neg_p(A \rightarrow_p A)$.
(`Ple`) $P_{\leq s}\alpha$ for $P_{\geq 1-s}\neg_c\alpha$.
(`Peq`) $P_{=s}\alpha$ for $P_{\geq s}\alpha \wedge_p P_{\leq s}\alpha$.
(`FalC`) $\bot_c$ for $\neg_c(\alpha \rightarrow_c \alpha)$.

where we have that $s \in \mathbb{Q}_{[0,1]}$, $\alpha \in For_C$, and $A \in For_P$. We should note that `FalC` and `FalP` are parameterized by α and A, respectively.

3.2 Semantics of $LPP_2^{\mathbb{Q}}$

The semantics for the logic $LPP_2^{\mathbb{Q}}$ is based on the possible–world approach, akin to many modal and modal-like logics, bearing in mind that for us it is imperative to correctly capture the notion of probability expressed in the syntax.

Definition 9. *An $LPP_2^{\mathbb{Q}}$-model is a structure $M = \langle W, H, M, v\rangle$, where*

- *W is a non-empty set of objects we will refer to as worlds.*
- *H is an algebra of subsets on W.*
- *μ is a finitely additive probability measure $\mu : H \to \mathbb{Q}_{[0,1]}$.*
- *v is a valuation function $v : W \times \phi \to \{true, false\}$, assigning truth-values to each propositional letter in each of the worlds. It is extended to all classical formulas as usual.*

In this definition, we illustrate to what precisely the term "rationally-valued" in the title of the chapter and the description of the logic refers to, as the range of the measure, *i.e.* the set of values from which we are aiming to assign probabilities to classical formulas, is the set of all rational numbers belonging to the unit interval.

We will provide a little more detailed look into the encoding of the semantics and the representation of models in Coq. For representing sets, we have used Coq's `Ensemble` library, which treats sets of type `U` as functions from `U` to `Prop`, Coq's sort for propositions. First, a single world $w \in W$ is represented by:

```
Record ElemW (U : Type) (W : Ensemble U) : Type :=
    mkElemW {origU :> U; inW : In _ W origU}.
```

In this manner, and with use of Coq's dependent types, we are able to capture the property of being a member of a subset within the type, which simplifies dealing with worlds. Now, with `Algebra` and `MeasureQc` being shorthand, respectively, for a set of sets of a given type `U`, and a function `U → Qc`, and `AOS` and `FAM_Qc` being predicates which check, respectively, that a given set of sets is indeed an algebra of subsets (*i.e.* that it is not empty and that it is closed under complement and union), and that a given function is indeed a probability measure on a given algebra of subsets (*i.e.* that it is non-negative, that the empty set measures 0, that the entire set measures 1, and that the measure itself is finitely additive), a model is represented by:

```
Record Model_Simp (U : Type) (W : Ensemble U) : Type :=
    mkMSimp {MC_Worlds    : Ensemble (ElemW U W);
             MC_Algebra   : Algebra U (ElemW U W);
             MC_Measure   : MeasureQc U (ElemW U W);
             MC_Valuation : (ElemW U W) -> nat -> Prop;
             MC_ElemWS_Cd : inhabited (ElemW U W);
             MC_Worlds_Cd : MC_Worlds = Full_set (ElemW U W);
             MC_Is_Model  : (AOS U MC_Algebra) ∧ (FAM_Qc U MC_Measure MC_Algebra)}.
```

Therefore, apart from the four basic constituents, we have also embedded into the model the conditions which ensure that our set of worlds is not empty, that the set of worlds encompasses all of the elements of the appropriate `ElemW` type, and that the algebra of subsets and the measure meet their respective requirements. In this way, we ensure that we cannot construct a model which does not meet the required criteria. Moreover, the worlds are parameterized by a type `U`, allowing for different choices of sets representing worlds, depending on the proof scenario. Here, we make use of the Coq predicate `inhabited`

```
Inductive inhabited (A : Type) : Prop := inhabits : A -> inhabited A
```

that, if `A` is thought of as a type, states that there exists an element of type `A`. Also, note the use of the definition `Full_set` from the `Ensembles` library

```
Inductive Full_set (U : Type) : Ensemble U := Full_intro : ∀x:U, Full_set U x.
```

that describes the set containing all elements of type `U`. Later on, we will also be using the `Empty_set` definition, also from the `Ensembles` library

```
Inductive Empty_set (U : Type) : Ensemble U := .
```

that describes the empty set of elements of type `U`. Let us proceed, after this brief detour into formalization, with the theoretical development of the semantics. For a given $LPP_2^{\mathbb{Q}}$-model M and a given formula α, we will denote the set of all of the worlds (`worlds`) in which α is true – $\{w|v(w,\alpha) = true\}$ by $[\alpha]_M$. If the model in question is clear from the context, we will omit the subscript and write only $[\alpha]$ instead.

Definition 10. *An $LPP_2^{\mathbb{Q}}$-model M is* measurable *if $[\alpha]_M \in H$, for all $\alpha \in For_C$.*

We will onward focus on the class of all measurable models (`Model_Meas`), which we will denote by $LPP_{2,Meas}^{\mathbb{Q}}$. Measurable models are encoded in Coq by adding the measurability condition into the definition of `Model_Simp`.

Definition 11. *The satisfiability relation $\models \subseteq LPP_{2,Meas}^{\mathbb{Q}} \times For_{LPP_2^{\mathbb{Q}}}$ satisfies the following conditions, for every $LPP_2^{\mathbb{Q}}$-measurable model $M = \langle W, H, M, v\rangle$ and every formula Φ:*

- *if $\Phi \in For_C$, $M \models \Phi$ iff $v(w, \Phi) = true$, for all $w \in W$.*
  Classical formulas are satisfied *iff* they are true in **all** of the worlds.
- *if $\Phi \in For_P$, and is of the form $P_{\geq s}\alpha$, $M \models \Phi$ iff $\mu([\alpha]_M) \geq s$.*
  This is how we semantically capture the notion of probability: a probabilistic formula $P_{\geq s}\alpha$ is satisfied *iff* the set of worlds in which α holds has the measure at least equal to s.
- *if $\Phi \in For_P$, and is of the form $\neg_p A$, $M \models \Phi$ iff $M \not\models A$.*
- *if $\Phi \in For_P$, and is of the form $A \rightarrow_p B$, $M \models \Phi$ iff $M \not\models A$ or $M \models B$.*

In Coq, we represent the satisfiability of one classical formula with the function `modelsC`, the satisfiability of one probabilistic formula with the function `modelsP`, the overall satisfiability of one formula with the function `models`, and the overall satisfiability of a set of formulas with the function `modelsSet`. It should be noted that the satisfiability of the negation and implication has been encoded using Coq's built-in connectives, which operate in an intuitionistic setting. However, as we make use of theorems from Coq's `Classical` library throughout the proofs, we *de facto* have that they behave in a classical manner. Also, since we have delegated the evaluation of negation and implication to Coq, and didn't encode it explicitly, we can say that the encoding of $LPP_2^{\mathbb{Q}}$ presented here is a *shallow encoding*.

Definition 12. *A formula Φ is* satisfiable *if there exists an $LPP_2^{\mathbb{Q}}$-measurable model M such that $M \models \Phi$; Φ is* valid (`Valid`) *if $M \models \Phi$, for all $LPP_2^{\mathbb{Q}}$-measurable models M; a set of formulas T is* satisfiable (`Satisfiable`) *if there exists an $LPP_2^{\mathbb{Q}}$-measurable model M such that $M \models \Phi$, for all $\Phi \in T$.*

3.3 A Complete Axiomatization

The set of valid formulas can be characterized by the axiom schemata presented in Figure 3.1, while the appropriate inference rules are presented in Figure 3.2. We denote this axiomatic system by $Ax_{LPP_2^{\mathbb{Q}}}$, and for each axiom schema we provide a brief description of its semantics. The main task is, once again, to have a set of axiom schemata which would allow us to easily syntactically deal with the properties of probability and the measure function.

ACT. All instances of classical propositional tautologies for classical formulas.

APT. All instances of classical propositional tautologies for probabilistic formulas.

AP1. $P_{\geq 0}\alpha$.
The probability of a formula being true is non-negative.

AP2. $P_{\leq r}\alpha \rightarrow_p P_{<s}\alpha$, for $s > r$.
Compatibility of the probability with $\leq$ and $<$, standard property of the measure function.

AP3. $P_{<r}\alpha \rightarrow_p P_{\leq r}\alpha$.
Weakening of $<$ into $\leq$, standard property of the measure function.

AP4. $P_{\geq r}\alpha \rightarrow_p (P_{\geq s}\beta \rightarrow_p (P_{\geq 1}\neg_c(\alpha \wedge_c \beta) \rightarrow_p P_{\geq r+s}(\alpha \vee_c \beta)))$, for $r + s \leq 1$.
If the probability that α and β hold are, respectively, at least r and s, and if there are no worlds in which α and β both hold, then the probability that $\alpha \vee \beta$ holds is at least $r + s$. This axiom schema effectively corresponds to the inclusion-exclusion principle for two sets, when their intersection is empty.

AP5. $P_{\leq r}\alpha \rightarrow_p (P_{<s}\beta \rightarrow_p P_{<r+s}(\alpha \vee_c \beta))$, for $r + s \leq 1$.
If the probability that α holds is at most r and the probability that β holds is less than s, then the probability that $\alpha \vee \beta$ holds is less than $r+s$. Semantically, this is a consequence of the finite additivity of the measure function.

AP6. $P_{\geq 1}(\alpha \rightarrow_c \beta) \rightarrow_p (P_{\geq r}\alpha \rightarrow_p P_{\geq r}\beta)$.
If $\alpha \rightarrow_c \beta$ is true in all of the worlds, then the probability that α holds is at least equal to the probability that β holds.

Figure 3.1: $LPP_2^{\mathbb{Q}}$ Axiom schemata

Modus Ponens for classical formulas: from α and $\alpha \rightarrow_c \beta$, infer β.

Modus Ponens for probabilistic formulas: from A and $A \rightarrow_p B$, infer B.

Probabilistic Necessitation: from α, infer $P_{\geq 1}\alpha$.

Domain Enforcement: from $A \rightarrow_p P_{\neq r}\alpha$, for all $r \in \mathbb{Q}_{[0,1]}$, infer $A \rightarrow \bot_p$.

Figure 3.2: $LPP_2^{\mathbb{Q}}$ Inference rules

Let us examine this axiomatic system in more detail. Clearly, in its current form it is not minimal, as all instances of the classical propositional tautologies are taken as axiom schemata, in contrast to, for example, taking three Hilbert-style axioms. Also, due to the axiom (ACT) and the modus ponens for classical formulas inference rule, we observe that classical logic is a sublogic of $LPP_2^{\mathbb{Q}}$. We also notice that the process of proving consists of two parts (one

of which may be empty): the first part deals only with classical, while the second deals only with probabilistic formulas. The only means of switching from one part to the other is the probabilistic necessitation inference rule, lifting a classical formula into a probabilistic one, with no existing means of switching back from the probabilistic to the classical level. Hence, the logic $LPP_2^{\mathbb{Q}}$ is a conservative extension of classical logic. The domain enforcement rule serves for preserving strong completeness, given the inherent non-compactness of the system, and will be discussed in detail later in this chapter. Next, we introduce several standard syntactic notions:

Definition 13. *A formula Φ is* derivable (`Derivable`) *from a given set of formulas (premises) T (denoted by $T \vdash \Phi$) if there exists a finite sequence of formulas $\Phi_0, \ldots, \Phi_k, \Phi$, such that each Φ_i is either in the set T, is an instance of one of the axiom schemata, or is obtained from the preceding formulas by using one of the inference rules. We call such a sequence a* proof *of Φ from T. A formula Φ is a* theorem *(denoted by $\vdash \Phi$) if it is derivable from the empty set of formulas* (`isTheorem`).

Equivalently, we could define a proof of $T \vdash \Phi$ as a tree of formulas in whose root is the formula Φ, in whose leaves are formulas either from the set T or instances of axiom schemata, and where all of the non-leaves are obtained from previously existing nodes by one of the inference rules (`proof_term`).

In Coq, the axioms are encoded straightforwardly, as definitions, while the inference rules are embedded into the structure of the proofs, which we have encoded using the tree-structure approach. To illustrate, here we show the definition of axiom (AP3), and the part of the proof structure corresponding to the use of the modus ponens for classical formulas inference rule:

```
Definition AxAP07 (A : forC) (r s : Q01) (HCond : s > r) : FOR :=
    Prob (ImpP (Ple r A) (Plt s A)).
```

```
| dbyIRMPc : ∀ (T : Ensemble FOR) (A B : forC) (pA pAB : proof_term),
      derives T pA (Clas A) → derives T pAB (Clas (ImpC A B)) →
      derives T (dIRMPc pA pAB) (Clas B).
```

Here, it is important to note that, for the domain enforcement rule, there are *countably many* premises required for its application. This countable collection of premises is handled in Coq as a function from natural numbers to proof terms `proofs : nat -> proof_term`), just as in [Honsell 1995]. This is further supported by introducing, as an axiom, a bijection between natural numbers and the $\mathbb{Q}_{[0,1]}$ interval (`f_nat_Q01 : nat -> Q01`). The approach of adopting such a function as an axiom was taken due to the well-known fact that such a bijection exists, and a formal proof of its existence would be out of scope of this development.

Definition 14. *A set of formulas T is consistent* (`Consistent`) *if there exists at least one classical formula α and at least one probabilistic formula A which are not derivable from it, and otherwise is inconsistent* (`Inconsistent_Alternate`). *Alternatively, a set of formulas T is inconsistent if $T \vdash \bot_c$ or $T \vdash \bot_p$* (`Inconsistent`). *A set of formulas T is maximally consistent* (`Max_Consistent`) *if it is consistent and the following holds:*

- *for each $\alpha \in For_C$: if $T \vdash \alpha$, then $\alpha \in T$ and $P_{\geq 1}\alpha \in T$.*
- *for each $A \in For_P$: either $A \in T$ or $\neg_p A \in T$.*

Before proceeding to the actual proofs of the meta-theoretical properties of $LPP_2^{\mathbb{Q}}$, we would like to note that all of the proofs laid out in this and the following chapter reflect the proving process as done in Coq, but are presented in human-readable form.

3.4 Key Properties of $Ax_{LPP_2^{\mathbb{Q}}}$

3.4.1 Soundness

Soundness of $Ax_{LPP_2^{\mathbb{Q}}}$ is proven in the standard manner. We show that all of the instances of the axiom schemata are valid formulas, and that the inference rules preserve validity. Here, we present the proof that the axiom schema $AP4$ is valid, as well as the proofs that some of the inference rules preserve validity, while the remaining cases are handled similarly.

Lemma 3 (Validity of $AP4$). *$P_{\geq r}\alpha \rightarrow_p (P_{\geq s}\beta \rightarrow_p (P_{\geq 1}\neg_c(\alpha \wedge_c \beta) \rightarrow_p P_{\geq r+s}(\alpha \vee_c \beta)))$ is valid for all $\alpha, \beta \in For_C$, $r, s \in \mathbb{Q}_{[0,1]}$, and $r + s \leq 1$* (`valid_AP04`).

Proof. Let $M = \langle W, H, \mu, v\rangle$ be a measurable model, and let us assume that $M \models P_{\geq r}\alpha$, $M \models P_{\geq s}\beta$, and $M \models P_{\geq 1}\neg_c(\alpha \wedge_c \beta)$, while our goal is to show that $M \models P_{\geq r+s}(\alpha \vee_c \beta)$. From our assumptions, we have that $[\alpha] \geq r$, $[\beta] \geq s$, and $[\neg_c(\alpha \wedge_c \beta)] = 1$ or, equivalently, given the properties of μ, that $[\alpha \wedge_c \beta] = 0$.

Next, we will take a moment to prove that $[\alpha \wedge_c \beta] = [\alpha] \cap [\beta]$. First, let $w \in [\alpha \wedge_c \beta]$. This means that $v(w, \alpha \wedge_c \beta) = true$, from which, given the truthtable of the connective $\wedge_c$, we obtain that $v(w, \alpha) = true$ and $v(w, \beta) = true$. Therefore, we have that $w \in [\alpha]$ and $w \in [\beta]$, and hence, that $w \in [\alpha] \cap [\beta]$. For the other direction, let $w \in [\alpha] \cap [\beta]$. Then, $v(w, \alpha) = true$ and $v(w, \beta) = true$. However, given the truthtable of the connective $\wedge_c$, we also have that $v(w, \alpha \wedge_c \beta) = true$, *i.e.* that $w \in [\alpha \wedge_c \beta]$. In a similar way, we prove that $[\alpha \vee_c \beta] = [\alpha] \cup [\beta]$.

Now, given the property of the measure that $\mu([\alpha] \cup [\beta]) = \mu([\alpha]) + \mu([\beta]) - \mu([\alpha] \cap [\beta])$ (`FAM_Qc_union_split`), we obtain that $\mu([\alpha] \cup [\beta]) \geq r + s$, and have our claim. □

Lemma 4 (Validity preservation of $Ax_{LPP_2^{\mathbb{Q}}}$ inference rules). *Let $\alpha, \beta \in For_C$, $A, B \in For_P$.*

1. *If α is valid, and $\alpha \rightarrow_c \beta$ is valid, then β is also valid* (`vp_MPc`).
2. *If A is valid, and $A \rightarrow_p B$ is valid, then B is also valid* (`vp_MPp`).
3. *If α is valid, then $P_{\geq 1}\alpha$ is also valid* (`vp_PN`).
4. *If $A \rightarrow_p P_{\neq r}\alpha$ is valid, for all $r \in \mathbb{Q}_{[0,1]}$, then $A \rightarrow_p \bot_p$ is also valid* (`vp_DE`).

Proof.

1. Let us suppose that, for a given model $M = \langle W, H, \mu, v\rangle \in LPP^{\mathbb{Q}}_{2,Meas}$, $\alpha, \beta \in For_C$, we have that:
$$M \models \alpha \text{ and } M \models \alpha \rightarrow_c \beta.$$
From this, given the definition of the satisfiability relation and the definition of implication for classical formulas, it must also be that $M \models \beta$, which concludes our proof.

2. This case is handled analogously to the first one.

3. Let us suppose that, for any given $M = \langle W, H, \mu, v\rangle \in LPP^{\mathbb{Q}}_{2,Meas}$, $\alpha \in For_C$, we have that $M \models \alpha$. From this, we have that $[\alpha]_M = W$, and by the definition of μ, we have that $\mu([\alpha]_M) = 1$, which means, by the definition of the satisfiability relation, that $M \models P_{\geq 1}\alpha$, concluding our proof.

4. Let us suppose that, for any given $M = \langle W, H, \mu, v\rangle \in LPP^{\mathbb{Q}}_{2,Meas}$, $A \in For_P$, $\alpha \in For_C$. we have that
$$M \models A \rightarrow_p P_{\neq r}\alpha, \text{ for all } r \in \mathbb{Q}_{[0,1]}. \tag{3.1}$$

Our goal is to prove that $M \models A \rightarrow_p \bot_p$. If $M \not\models A$, the proof is immediate, given the definition of classical implication. Let us onward assume that $M \models A$. From this, Equation 3.1, and the definition of $\models$, we have that

$$\mu([\alpha]) \neq r, \text{ for all } r \in \mathbb{Q}_{[0,1]},$$

which is a contradiction with the definition of μ. Therefore, the case in which $M \models A$ is not possible, and we have our claim.

□

Therefore, we have effectively proven the soundness theorem:

Theorem 6 (Soundness of $LPP_2^{\mathbb{Q}}$). *If Φ is a theorem of $Ax_{LPP_2^{\mathbb{Q}}}$, then Φ is valid.*

```
Theorem LPP2_Q_Soundness : ∀ (Φ : FOR), isTheorem Φ → Valid Φ.
```

3.4.2 Completeness

3.4.2.1 The Deduction Theorem.

The first important meta-theoretical result on the road to proving strong completeness of $Ax_{LPP_2^{\mathbb{Q}}}$ is the Deduction Theorem. In order to prove it, we will need the following lemma:

Lemma 5. *Derivations of classical formulas do not depend on probabilistic formulas* (`Der_clas_P_indep`). *If T is a set of formulas, T' is a set of probabilistic formulas, and $\alpha \in For_C$, then*

$$T \cup T' \vdash \alpha \text{ implies } T \vdash \alpha.$$

Proof. By induction on the structure of the derivation of $T \cup T' \vdash \alpha$. □

Given the limitations of the syntax, we will have two Deduction Theorems, one for classical and one for probabilistic formulas. Here, we will state and prove only the one for probabilistic formulas, as the one for classical is proven just as in classical logic.

Theorem 7 (Deduction Theorem for Probabilistic Formulas). *Let T be a set of formulas and $A, B \in For_P$. Then, the following holds:*

$$T \cup \{A\} \vdash B \text{ if and only if } T \vdash A \rightarrow_p B.$$

```
Theorem LPP2_Q_Deduction_Theorem_Prob : ∀ (T : Ensemble FOR) (A B : forP),
   Derivable (Union FOR T (Singleton FOR (Prob A))) (Prob B) ↔
   Derivable T (Prob (ImpP A B)).
```

Proof. The right-to-left implication is proven as in classical logic. We will prove the left-to-right implication by induction on the structure of the derivation of $T \cup \{A\} \vdash B$. The cases in which B is obtained by using any of the axioms or the modus ponens for classical formulas inference rule are proven directly, by using the induction hypothesis, or as in classical logic.

Let $T \cup \{A\} \vdash B$ by the use of the probabilistic necessitation inference rule. In this case, B is of the form $P_{\geq 1}\alpha$, for some $\alpha \in For_C$, and we have that $T \cup \{A\} \vdash \alpha$. From this, by Lemma 5, we have that $T \vdash \alpha$. Now, using probabilistic necessitation, we obtain that $T \vdash B$, from which $T \vdash A \rightarrow_p B$ immediately follows.

Let $T \cup \{A\} \vdash B$ by the use of the domain enforcement inference rule. In this case, B is of the form $A' \rightarrow_p \bot_p$, for some $A' \in For_P$, and we have that, for all $r \in \mathbb{Q}_{[0,1]}$, $T \cup \{A\} \vdash A' \rightarrow_p P_{\neq r}\alpha$, for some $\alpha \in For_C$. By the induction hypothesis, we have that for all $r \in \mathbb{Q}_{[0,1]}$, $T \vdash A \rightarrow_p (A' \rightarrow_p P_{\neq r}\alpha)$. However, since $A \rightarrow_p (B \rightarrow_p C) \leftrightarrow (A \wedge_p B) \rightarrow_p C$ is a tautology (where $A, B, C \in For_P$), we also have that for all $r \in \mathbb{Q}_{[0,1]}$, $T \vdash (A \wedge_p A') \rightarrow_p P_{\neq r}\alpha$. From this, using the domain enforcement inference rule, we obtain that $T \vdash (A \wedge_p A') \rightarrow_p \bot_p$, and from that, using the abovementioned tautology again, we obtain that $T \vdash A \rightarrow_p B$. □

3.4.2.2 Maximally Consistent Extension.

The next step is to prove that, in $LPP_2^{\mathbb{Q}}$, a consistent set can always be extended to a maximally consistent set. For this, we will need the following auxiliary claims:

Lemma 6. *Derivability is not affected by adding theorems into the context, i.e. if $T \cup T' \vdash \Phi$, and every formula in T' is a theorem, then $T \vdash \Phi$, where $\Phi \in FOR$* (`Der_addThm`). *Consequently, consistency is not affected by adding theorems into the context* (`Consistency_addThm`).

Proof. The first part of the lemma is proven by induction on the structure of the derivation of $T \cup T' \vdash \Phi$, while the second part follows directly from the first one and the definition of consistency. □

Lemma 7. *For every consistent set of formulas T, and every probabilistic formula A, either $T \cup \{A\}$ is consistent or $T \cup \{\neg_p A\}$ is consistent* (`Consistency_xor`).

Proof. Let us assume, instead, that both $T \cup \{A\}$ and $T \cup \{\neg_p A\}$ are inconsistent. Then, it would have to be that $T \cup \{A\} \vdash \bot_c$ or $T \cup \{A\} \vdash \bot_p$. However, by Lemma 5, if $T \cup \{A\} \vdash \bot_c$ we would have that $T \vdash \bot_c$ as well, which is in contradiction with our assumption that T is consistent. Therefore, it must be that $T \cup \{A\} \vdash \bot_p$. Similarly, it must be that $T \cup \{\neg_p A\} \vdash \bot_p$. By the Deduction Theorem, we then have that $T \vdash A \to \bot_p$, and $T \vdash \neg_p A \to \bot_p$. However, given that $(A \to_p B) \to_p (\neg_p A \to_p B) \to_p B$ is a tautology (where $A, B \in For_P$), we also have that $T \vdash \bot_p$, which is in contradiction with our assumption that T is consistent, and concludes our proof. □

Lemma 8. *For all maximally consistent sets T, and for all probabilistic formulas A, B, the following holds:*

1. *Either $A \in T$ or $\neg_p A \in T$* (`MCons_A_NegA_p`).
2. *$A \in T$ if and only if $T \vdash A$* (`MCons_In_Der`).
3. *If $\neg_p A \in T$, then $A \notin T$* (`MCons_NegP`).
4. *If $A \wedge_p B \in T$, then $A \in T$ and $B \in T$* (`MCons_AndP`).
5. *If $A \vee_p B \in T$, then $A \in T$ or $B \in T$* (`MCons_OrP`).

Proof. The proof follows from the definition of maximal consistency and the axiomatic system, using several classical tautologies concerning the relationships between conjunction, disjunction and implication. □

The next lemma provides us with an important insight into the connection between maximal consistency and classical formulas.

Lemma 9. *Let T be a maximally consistent set. Then, for every $\alpha \in For_C$, there exists a unique $q(\alpha) \in \mathbb{Q}_{[0,1]}$, such that $T \vdash P_{=q(\alpha)}\alpha$.* (`unique_existence_M`)

Proof. We will split the proof into two parts. In the first part, we will prove that, for a given $\alpha \in For_C$, a $q \in \mathbb{Q}_{[0,1]}$ satisfying the conditions of the Lemma exists, and in the second part, we will prove that such a q has to be unique.

Let us assume that there is no $q \in \mathbb{Q}_{[0,1]}$, such that $T \vdash P_{=q}\alpha$. This means (using Lemma 8, item 1) that, for all $q \in \mathbb{Q}_{[0,1]}$, we have that $T \vdash P_{\neq q}\alpha$. However, using axiom AP1, the tautology $P_{\neq q}\alpha \to (P_{\geq 0}\alpha \to P_{\neq q}\alpha)$, and the modus ponens for probabilistic formulas inference rule, we obtain that for all $q \in \mathbb{Q}_{[0,1]}$, it holds that $T \vdash P_{\geq 0}\alpha \to P_{\neq q}\alpha$. From this, using the domain

enforcement inference rule, we obtain that $T \vdash P_{\geq 0}\alpha \rightarrow \bot_p$. Finally, from this and (as $P_{\geq 0}\alpha$ is an axiom) $T \vdash P_{\geq 0}\alpha$, and using the modus ponens for probabilistic formulas inference rule, we obtain that $T \vdash \bot_p$, which is a contradition with the assumed consistency of T. Therefore, there exists a $q \in \mathbb{Q}_{[0,1]}$, such that $T \vdash P_{=q}\alpha$.

Next, let us assume that there are $q_1, q_2 \in \mathbb{Q}_{[0,1]}$, such that $T \vdash P_{=q_1}\alpha$ and $T \vdash P_{=q_2}\alpha$. If $q_1 = q_2$, we have the desired claim. Next, let us suppose that $q_1 < q_2$. Then, given that $T \vdash P_{\leq q_1}\alpha$ (from the definition of $P_=$) and using the axiom AP2, we obtain that $T \vdash P_{<q_2}\alpha$. However, from the definition of $P_=$, we also have that $T \vdash P_{\geq q_2}\alpha$, which gives us a contradiction. The case when $q_1 > q_2$ is handled analogously, and we finally obtain our claim. □

Onward, we assume to have a consistent set of formulas T. If we denote the classical formulas in T by T_{Clas}, and the probabilistic formulas in T by T_{Prob}, we have that $T = T_{Clas} \cup T_{Prob}$. We define the set of classical consequences of T, denoted by $CC(T)$ (`ClasDCons`), as

$$CC(T) = \{\alpha \mid \alpha \in For_C,\ T \vdash \alpha\},$$

and its image through the $P_{\geq 1}$ operator (`ExtendPge1`) as

$$PN(T) = \{P_{\geq 1}\alpha \mid \alpha \in CC(T)\}.$$

Next, we define a primary extension of T, denoted by T_0 (`makeT0`), from which we will be constructing the maximally consistent extension of T, as:

$$T_0 = T \cup CC(T) \cup PN(T),$$

and prove that the following properties hold:

Lemma 10. *Let T be a consistent set of formulas, and T_0 its primary extension. Then:*

1. *For any classical formula α, if $T_0 \vdash \alpha$ then $\alpha \in T_0$* (`clas_in_T0`).
2. *For any probabilistic formula A, if $T_0 \vdash A$ then $T \vdash A$* (`prob_in_T0`).
3. *T_0 is consistent* (`T0_cons`).

Proof. The first two parts are proven using induction on the structure of the derivation of $T_0 \vdash \alpha$ and $T_0 \vdash A$, respectively. As for the third part, let us assume that T_0 is inconsistent. Since T_0 is inconsistent, it must be that either $T_0 \vdash \bot_c$ or $T_0 \vdash \bot_p$. Let first $T_0 \vdash \bot_c$. Given the first part of this lemma, we then have that $\bot_c \in T_0$, meaning that either $\bot_c \in T$ or $\bot_c \in CC(T)$. However, in both of these cases we would have that $T \vdash \bot_c$, which is in contradiction with the assumed consistency of T. Next, let $T_0 \vdash \bot_p$. Given the second part of this lemma, we have that $T \vdash \bot_p$, which is again in contradiction with the assumed consistency of T. Therefore, T_0 must be consistent. □

The next lemma, similar in style to Lemma 9, provides another insight into the functioning of probabilistic operators and their compatibility with consistency:

Lemma 11. *For every consistent set of formulas T, a probabilistic formula A, and a classical formula α, if $T \cup \{A\}$ is consistent, then there exists a rational number $p(\alpha) \in \mathbb{Q}_{[0,1]}$ such that $T \cup \{A\} \cup \{P_{=p(\alpha)}\alpha\}$ is consistent.* (`C_Peq_existence`)

Proof. Let us assume, to the contrary, that for all $r \in \mathbb{Q}_{[0,1]}$, $T \cup \{A\} \cup \{P_{=r}\alpha\}$ is inconsistent, i.e. that $T \cup \{A\} \cup \{P_{=r}\alpha\} \vdash \bot_p$ ($T \cup \{A\} \cup \{P_{=r}\alpha\} \vdash \bot_c$ is impossible, as then, given Lemma 5, T would be inconsistent). By applying the Deduction Theorem, we have that, for all $r \in \mathbb{Q}_{[0,1]}$, $T \cup \{A\} \vdash P_{=r}\alpha \rightarrow \bot_p$. However, this is equivalent to having that, for all $r \in \mathbb{Q}_{[0,1]}$, $T \cup \{A\} \vdash$

$P_{\neq r}\alpha$, which is, in turn, equivalent (by applying the Deduction Theorem) to having that, for all $r \in \mathbb{Q}_{[0,1]}$, $T \vdash A \rightarrow_p P_{\neq r}\alpha$. However, then, by the domain enforcement inference rule, we have that $T \vdash \bot_p$, which is a contradiction with the consistency of T. □

Next, we will be needing to establish an enumeration of classical formulas $\alpha_0, \alpha_1, \alpha_2, \ldots$ (`f_enum_c : nat -> forC`), and an enumeration of probabilistic formulas $A_0, A_1, A_2, \ldots$ (`f_enum_p : nat -> forP`). Again, as in the case of the countability of the $\mathbb{Q}_{[0,1]}$ interval, we take the existence of these functions as axioms in Coq. We define a sequence of sets T_n (`Tn_const`), for $n \geq 1$, as follows:

$$T_{n+1} = \begin{cases} T_n \cup \{A_n\} \cup \{P_{=p(\alpha_n)}\alpha_n\} & \text{if } T_n \cup \{A_n\} \text{ is consistent,} \\ T_n \cup \{\neg_p A_n\} \cup \{P_{=p(\neg_c \alpha_n)}\neg_c \alpha_n\} & \text{otherwise.} \end{cases}$$

The proof of the following lemma is based of the previously proven lemmas:

Lemma 12. *The following properties concerning the sequence T_n hold:*

1. *For all $n \geq 0$, $T_n \subseteq T_{n+1}$* (`Tn_increasing`).
2. *For all $n \geq 0$, T_n is consistent* (`Tn_all_consistent`).
3. *For all classical formulas α, if $\alpha \in T_n$ then $\alpha \in T_0$* (`Tn_clas_origin`).

Next, we define the maximally consistent extension T^* of T, as

$$T^* = \bigcup_{i=1}^{\infty} T_i.$$

```
Definition MC_Extension (T : Ensemble FOR) : Ensemble FOR :=
    (fun A : FOR ⇒ ∃ k : nat, In FOR (Tn_const T k) A.)
```

As can be seen, this extension is naturally defined in Coq as a function in `Prop`, which is true only for the formulas that are contained in at least one of the extensions of T, which has been captured elegantly by the existential quantifier. For T^* defined in this way, we prove that the following statements hold:

Lemma 13 (Properties of T^*)**.**

1. *For all $A \in For_P$, either $A \in T^*$ or $\neg_p A \in T^*$* (`MCE_max`).
2. *$\bot_p \notin T^*$* (`MCE_nFalP`).
3. *$T \subseteq T^*$* (`T_in_MCE`).
4. *If $T^* \vdash \alpha$, then $T_0 \vdash \alpha$, for $\alpha \in For_c$* (`MCE_clas`).
5. *If $T^* \vdash A$, then $A \in T^*$, for $A \in For_p$* (`MCE_prob`).

Proof.

1. Directly, by construction of T^*.
2. By contradiction. If $\bot_p \in T^*$, then, by definition of T^*, there exists an integer k, such that $\bot_p \in T_k$, which is a contradiction with the previously proven consistency of T_k.
3. Directly, as $T \subseteq T_0$, and $T_0 \subseteq T^*$.

4. By induction on the structure of the derivation of $T^* \vdash \alpha$. First, if $\alpha \in T^*$, then, by definition of T^*, there exists an integer k, such that $\alpha \in T_k$. Then, from Lemma 12, we obtain that $\alpha \in T_0$, from which $T_0 \vdash \alpha$ follows immediately. Next, if α is an instance of a tautology, it automatically follows that $T_0 \vdash \alpha$. Finally, let us assume that $T^* \vdash \alpha$ has been obtained from $T^* \vdash \beta$ and $T^* \vdash \beta \rightarrow_c \alpha$, using the modus ponens for classical formulas inference rule. By the induction hypothesis, we then have that $T_0 \vdash \beta$ and that $T_0 \vdash \beta \rightarrow_c \alpha$, from which we directly obtain $T^* \rightarrow \alpha$ using the modus ponens for classical formulas inference rule.

5. By induction on the structure of the derivation of $T^* \vdash A$. The cases where A is an instance of a tautology are proven directly, as well as the cases when A is an instance of any of the other axioms. Next, let us assume that $T^* \vdash A$ has been obtained from $T^* \vdash B$ and $T^* \vdash B \rightarrow_p A$, using the modus ponens for probabilistic formulas inference rule. By the induction hypothesis, we then have that $B \in T^*$ and that $B \rightarrow_p A \in T^*$. Let i, j, and k be the values assigned to B, $B \rightarrow_p A$, and A, respectively, in the enumeration of probabilistic formulas we have used to construct T^*, and let $m = \max(i+1, j+1, k+1)$. Then, it must hold that B, $B \rightarrow_p A$, and either A or $\neg_p A$ are are in the set T_m. However, if $\neg_p A \in T_m$, then T_m would be inconsistent, as both A and $\neg_p A$ would be derivable from it. Therefore, it must be that $A \in T_m$, and, consequently, that $A \in T^*$. The case when A has been obtained using the probabilistic necessitation inference rule is handled perhaps surprisingly easily, as then, by construction, it holds that $A \in T_0$. Finally, let $T^* \vdash A$ be obtained by using the domain enforcement inference rule. Then, $A \equiv A' \rightarrow_p \bot_p$, and we have that $T^* \vdash A' \rightarrow_p P_{\neq r}\alpha$, for all $r \in \mathbb{Q}_{[0,1]}$, and some $\alpha \in For_C$, $A' \in For_P$. By construction, we have that $P_{=p(\alpha)}\alpha \in T^*$, for some integer $p(\alpha)$. However, by the induction hypothesis, we have that $A' \rightarrow_p P_{\neq r}\alpha \in T^*$, for all $r \in \mathbb{Q}_{[0,1]}$, including $r = p(\alpha)$. Therefore, it must be that $A' \notin T^*$ and $\neg_p A' \in T^*$, because we would be able to find an inconsistent set in the sequence T_n otherwise. Now, let i and j be the values assigned to $\neg_p A'$ and $A' \rightarrow_p \bot_p$ in the enumeration of probabilistic formulas we have used to construct T^*, and let $m = \max(i+1, j+1)$. Then, it must hold that $\neg_p A'$ and either $A' \rightarrow_p \bot_p$ or $\neg_p(A' \rightarrow_p \bot_p)$ are in the set T_m. However, given the tautology $\neg_p A \rightarrow_p (A \rightarrow_p B$, if $\neg_p(A' \rightarrow_p \bot_p) \in T_m$, then T_m would be inconsistent, and we have our claim.

□

As a direct consequence of the previous Lemma, we obtain the main result of this subsection:

Theorem 8. *T^* is a maximally consistent set of formulas.*

3.4.2.3 The Canonical Model.

Now, we can turn to the construction of a canonical model M^* for a given consistent set of formulas T. We proceed to define each of the required model components:

Worlds. Let w be a valuation function mapping the set of propositional letters ϕ to $\{true, false\}$ (`nat -> Prop`), and let us denote its standard extension to all classical formulas by w_{ext}. The set of worlds W (`Worlds`) will be the set of all valuation functions, such that their standard extensions satisfy every classical formula which is a consequence of T:

$$W = \{w \mid w : \phi \rightarrow \{true, false\}, \{w_{ext}(\alpha) = 1 \mid \alpha \in For_C, T \vdash \alpha\}\}.$$

Algebra of Subsets. We will define H (`H`) in the following way:

$$H = \{[\alpha] \mid \alpha \in For_C\},$$

where $[\alpha]$, as earlier, denotes the set of worlds from W in which a classical formula α is satisfied. It is easily proven that H, defined in this way, satisfies the requirements to be an algebra of subsets over W: it contains the empty set (`H_empty_set_in_H`), it is closed under complement (`H_complement_in_H`), and is closed under finite union (`H_union_in_H`).

Measure. The measure should map the algebra of subsets H into the rational unit interval $\mathbb{Q}_{[0,1]}$. Since the elements of H are essentially determined by classical formulas, we will define μ as follows, using notation from Lemma 9:

$$\mu([\alpha]) = q(\alpha), \text{ where } T^* \vdash P_{=q(\alpha)}\alpha.$$

Lemma 9 states that a unique $q(\alpha)$ exists for each $\alpha \in For_C$, ensuring that the measure is well-defined. The following claims needed to be proven with regard to the properties of μ, and have mostly consisted of untangling a number of technical details related to properties of sets and of rational numbers:

- If $[\beta] \subseteq [\alpha]$, then $T \vdash \alpha \to_c \beta$ (`M_included_derivable`)[1].
- Measure and negation: $\mu([\alpha]) = 1 - \mu([\neg_c \alpha])$ (`M_negation`).
- Non-negativity: $\mu([\alpha]) \geq 0$, for all $\alpha \in For_C$ (`M_FAM_nonneg`).
- Measure and the empty set: $\mu(\emptyset) = 0$ (`M_FAM_empty_set`).
- Measure and the full set: $\mu(W) = 1$ (`M_FAM_Full_set`).
- Additivity: μ is finitely additive. (`M_FAM_additive`).

Valuation. The valuation function $v : W \times \phi \to \{true, false\}$ (`v`) is defined naturally as $v(w, p_i) = w(p_i)$.

In this way, we have obtained a model $M^* = \langle W, H, \mu, v\rangle$ (`Const_Model_Simp`), and the remaining claim that we need to prove is that it is measurable.

Lemma 14. *M^* is a measurable model* (`Construction_is_Measurable_Model`).

Proof. We should prove that for $\alpha \in For_C$, it holds that $[\alpha]_{M^*} \in H$, which follows immediately, given the construction of H. □

3.4.2.4 Strong Completeness.

With the canonical model in place, we need to prove several more auxiliary lemmas on the road to strong completeness. Let T be a consistent set of formulas, T^* denote the maximally consistent extension of T, and M^* denote the canonical model constructed from T^*.

Lemma 15. *$\alpha \in T^*$ if and only if $M^* \models \alpha$, for all $\alpha \in For_C$* (`Cpt_iff_c`).

Proof. Let $M^* \models \alpha$. We will prove that $\alpha \in T_0$, i.e. (by definition of T_0) that $T \vdash \alpha$, which would be sufficient for our claim. However, we obtain this immediately from the strong completeness of classical logic.

For the other direction, let $\alpha \in T^*$. By construction of T^*, we have that $T \vdash \alpha$. On the other hand, we need to prove that for all $w \in W$, $v(w, \alpha) = true$. However, directly by construction of M^*, we have that $v(w, \alpha) = w_{ext}(\alpha) = true$, because the worlds of the canonical model are specifically chosen so that all of the consequences of T are true in all of the worlds. □

[1] In the proof of this statement, we use strong completeness of classical logic as an axiom in Coq (`LPP2_Q_Classical_Strong_Completeness_models_T`).

Lemma 16. *$A \in T^*$ if and only if $M^* \models A$, for all $A \in For_P$* (`Cpt_iff_p`).

Proof. By structural induction on A. First, let $A = P_{\geq r}\alpha$, and let $P_{\geq r}\alpha \in T^*$. From Lemma 9 and item 2 of Lemma 8, we obtain that there exists a unique $q(\alpha) \in \mathbb{Q}_{[0,1]}$, such that $P_{=q(\alpha)}\alpha \in T^*$, while, by Theorem 8, we obtain that $q(\alpha) \leq r$. Now, given the definition of μ in the canonical model and the definition of satisfiability, we directly obtain that $M^* \models P_{\geq r}\alpha$. For the other direction, let $M^* \models P_{\geq r}\alpha$. Then, by construction of the canonical model and the definition of satisfiability, we have that r is greater than or equal to the aforementioned $q(\alpha)$, obtained from Lemma 9. From here, since T^* is maximally consistent, it has to be that $T^* \vdash P_{\geq r}\alpha$.

Next, let $A = \neg_p B$. Using the induction hypothesis, our goal amounts to $B \notin T^* \leftrightarrow \neg_p B \in T^*$, which is easily obtained from item 1 of Lemma 13.

Finally, let $A = A' \rightarrow_p B'$. Using the induction hypothesis, our goal amounts to $A' \notin T^* \vee B' \in T^* \leftrightarrow (A' \rightarrow_p B') \in T^*$. If $A' \notin T^*$ or $B' \in T^*$, then, by using the tautologies $\neg_p A \rightarrow_p (A \rightarrow B)$ and $B \rightarrow_p (A \rightarrow_p B)$, respectively, we obtain that $T \vdash A' \rightarrow_p B'$, and, by item 5 of Lemma 13, we obtain that $A' \rightarrow_p B' \in T^*$. The other direction is proven easily using item 1 of Lemma 13. □

From the previous two lemmas, we have

Lemma 17. *$\Phi \in T^*$ if and only if $M^* \models \Phi$, for all $\Phi \in For$* (`Cpt_iff`).

We also need the following claim:

Lemma 18. *Let T be a set of formulas and $M = \langle W, H, \mu, v\rangle$ be a measurable model, and let us assume that $M \models T$. Then, it holds that for all $\Phi \in For$, if $T \vdash \Phi$, then $M \models \Phi$* (`Strong_Completeness_Consistent_T_rl`).

Proof. By structural induction on the derivation of $T \vdash \Phi$. The cases in which $\Phi \in T$ and in which Φ is an instance of an axiom or obtained by the modus ponens inference rules follow directly from the soundness of the system (Theorem 6) and the induction hypothesis (IH). Next, let $T \vdash \Phi$ be obtained by the probabilistic necessitation inference rule. Then, we have that $\Phi \equiv P_{\geq 1}\alpha$, for some $\alpha \in For_C$, and that $T \vdash \Phi$ was obtained from $T \vdash \alpha$. By the IH, we have that $M \models \alpha$, which means that α is true in every world, i.e. that $[\alpha] = W$, i.e. that $\mu([\alpha]) = 1$. This, by definition of satisfiability, means that $M \models P_{\geq 1}\alpha$, and we have our claim. Finally, let $T \vdash \Phi$ be obtained by the domain enforcement inference rule. Then, we have that $\Phi \equiv A \rightarrow_p \bot_p$, and that $T \vdash \Phi$ was obtained from $T \vdash A \rightarrow_p P_{\neq r}\alpha$, for all $r \in \mathbb{Q}_{[0,1]}$, and some $\alpha \in For_C$, $A \in For_P$. By the IH, we have that $M \models A \rightarrow_p P_{\neq r}\alpha$, for all $r \in \mathbb{Q}_{[0,1]}$. However, $M \models A$ is not possible, since then we would have $M \models \bot_p$ as well, which cannot be, by definition of satisfiability and the measure. Therefore, it must be that $M \not\models A$, *i.e.* that $M \vdash \neg_p A$, from which, using classical propositional tautologies, we obtain the desired $M \models A \rightarrow_p \bot_p$. □

Finally, we can prove the main result of this chapter:

Theorem 9 (Strong Completeness of $LPP_2^{\mathbb{Q}}$)**.** *A set of formulas T is $LPP_2^{\mathbb{Q}}$-consistent if and only if it is $LPP_{2,Meas}^{\mathbb{Q}}$-satisfiable* (`LPP2_Q_Strong_Completeness`).

```
Theorem LPP2_Q_Strong_Completeness : ∀ T : Ensemble FOR,
                                     Consistent T <-> Satisfiable T.
```

Proof. First, let T be consistent. We show that $M^* \models T$, i.e. that $M^* \models \Phi$, for every formula $\Phi \in T$. By Lemma 17, this is true if and only if $\Phi \in T^*$. However, this is immediate, as every formula Φ which is in T is also, by construction, in T_0, which, in turn, is a subset of T^*.

Next, let T be satisfiable. Then, there exists a model $M = \langle W, H, \mu, v\rangle$ such that $M \models \Phi$, for all $\Phi \in T$. Then, by Lemma 18, we obtain that M models all formulas derivable from T. If T were to be inconsistent, then we would have that $M \models \bot_c$ or $M \models \bot_p$, which is not possible by definition of satisfiability, and T must be consistent. □

3.4.2.5 Simple Completeness of $LPP_2^{\mathbb{Q}}$.

The Simple Completeness Theorem follows naturally from the Strong Completeness Theorem, with help from the following lemma:

Lemma 19. *If a set of classical formulas T is consistent in the sense of classical logic, then it is also $LPP_2^{\mathbb{Q}}$-consistent* (`Cpt_ConsClas_Cons`).

Proof. Directly, given strong completeness of both classical logic and $LPP_2^{\mathbb{Q}}$. □

Now, we can proceed to the Simple Completeness Theorem:

Theorem 10 (Simple Completeness of $LPP_2^{\mathbb{Q}}$.)**.** *If a formula Φ is $LPP_{2,Meas}^{\mathbb{Q}}$-valid, then it is a theorem of $LPP_2^{\mathbb{Q}}$.* (`LPP2_Q_Simple_Completeness`)

```
Theorem LPP2_Q_Simple_Completeness : ∀ Φ : FOR, Valid Φ -> isTheorem Φ.
```

Proof. We have two cases to consider. First, let $\Phi \equiv \alpha$ be a classical formula. If α is $LPP_{2,Meas}^{\mathbb{Q}}$-valid, then $\neg_c\alpha$ is not satisfiable, and by the Strong Completeness Theorem, $\{\neg_c\alpha\}$ is inconsistent, i.e. $\{\neg_c\alpha\} \vdash \bot_c$ or $\{\neg_c\alpha\} \vdash \bot_p$. If $\{\neg_c\alpha\} \vdash \bot_c$, then we obtain $\vdash \alpha$ from the Deduction Theorem and the elimination of double negation. If $\{\neg_c\alpha\} \vdash \bot_p$, and $\{\neg_c\alpha\} \nvdash \bot_c$, then we have that $\{\neg_c\alpha\}$ is classically consistent and also, by the previous Lemma, also $LPP_2^{\mathbb{Q}}$-consistent, which is a contradiction.

Second, let $\Phi \equiv A$ be a probabilistic formula. As in the previous case, if A is $LPP_{2,Meas}^{\mathbb{Q}}$-valid, then $\neg_p A$ is not satisfiable and by the Strong Completeness Theorem, $\{\neg_p A\}$ is inconsistent, i.e. $\{\neg_p A\} \vdash \bot_c$ or $\{\neg_p A\} \vdash \bot_p$. If $\{\neg_p A\} \vdash \bot_p$, then we again obtain $\vdash A$ from the Deduction Theorem and the elimination of double negation. On the other hand, if $\{\neg_p A\} \vdash \bot_c$, then since classical derivations do not depend on probabilistic formulas, it would be that $\emptyset \vdash \bot_c$, which is a contradiction with the consistency of $\emptyset$, guaranteed by the Strong Completeness Theorem. □

3.4.3 Non-Compactness

Let us consider the set $T = \{P_{\neq 0}\alpha\} \cup \{P_{<\frac{1}{n}}\alpha |\ n \in \mathbb{N}^+\}$. Using the domain enforcement inference rule, we obtain that T is inconsistent (`Cpc_Incons_T`). By the strong completeness theorem, T is not satisfiable. However, every finite subset of T is consistent (`Cpc_Finite_Sat_T`), and, therefore, satisfiable. This gives us the non-compactness theorem for $LPP_2^{\mathbb{Q}}$:

Theorem 11 (Non-compactness of $LPP_2^{\mathbb{Q}}$)**.** *There exists a set of $LPP_2^{\mathbb{Q}}$-formulas which is unsatisfiable, but whose every finite subset is satisfiable.*

```
Theorem LPP2_Q_NonCompactness : ∃ T : Ensemble FOR,
(∀ T' : Ensemble FOR, Finite _ T' → Included _ T' T → Satisfiable T') ∧
                                ¬ Satisfiable T.
```

3.4.4 Concerning Decidability

By using a method similar to that shown for LPCP in Chapter 2, we can prove that the satisfiability of probabilistic formulas for $LPP_2^{\mathbb{Q}}$ is decidable, and that it is NP-complete. However, due to the perceived complexity of encoding the actual decision procedure as well as the required mathematical background in Coq, it was decided to keep the formal proof of decidability and, possibly, extraction of a certified probabilistic SAT-solver, as a future research goal.

3.5 Summary

In this chapter, we have presented $LPP_2^{\mathbb{Q}}$ - a rationally-valued probability logic without iterations of probability operators. We have introduced a sound and strongly-complete axiomatic system for $LPP_2^{\mathbb{Q}}$, and formally proven in Coq the main meta-theoretic properties of $LPP_2^{\mathbb{Q}}$ – soundness, strong and simple completeness, and non-compactness. Similarly to LPCP, due to the inherent non-compactness of the system, an infinitary inference rule had to be introduced into the system.

Chapter 4

Formal Verification of Key Properties for $LPP_1^{\mathbb{Q}}$ - a Rationally-valued Probability Logic with Iterations - in the Proof Assistant Coq

Contents

In this chapter, we address the logic $LPP_1^{\mathbb{Q}}$ - a probability logic similar to $LPP_2^{\mathbb{Q}}$, with the main difference being that iterations of probability operators are allowed. We formally prove its main meta-theoretical properties in the proof assistant Coq, and discuss the differences in syntax, semantics and proof technology between $LPP_2^{\mathbb{Q}}$ and $LPP_1^{\mathbb{Q}}$.

4.1 Syntax of $LPP_1^{\mathbb{Q}}$

The syntax of $LPP_1^{\mathbb{Q}}$ is somewhat more uniform than that of $LPP_2^{\mathbb{Q}}$, shown in the previous chapter. We also require a denumerably infinite set of propositional letters: $\phi = \{p_0, p_1, \ldots\}$ (`nat`), and two propositional connectives: $\neg$ (`Neg : FOR -> FOR`)[1] and $\rightarrow$ (`Imp : FOR -> FOR -> FOR`), where `FOR` denotes the type of $LPP_1^{\mathbb{Q}}$ formulas in Coq. Let us again denote by $\mathbb{Q}_{[0,1]}$ (`Q01`) the set of all rational numbers from the unit interval, encoded in Coq just as for $LPP_2^{\mathbb{Q}}$, and let us have a denumerably infinite set of probabilistic operators $P_{\geq s}, s \in \mathbb{Q}_{[0,1]}$ (`Pge : Q01 -> FOR -> FOR`). Then, the set of all formulas $For_{LPP_1^{\mathbb{Q}}}$ (`FOR`) of $LPP_1^{\mathbb{Q}}$ is defined as the smallest set that satisfies the following conditions:

- it contains all of the propositional letters.
- if $\alpha \in For$, $s \in \mathbb{Q}_{[0,1]}$, then $P_{\geq s}\alpha \in For$.
- if α, then $\neg\alpha \in For$.

[1] Just as in the previous chapter, when we are not addressing an encoding of a concept in Coq in detail, we will be specifying the label of the encoding in brackets, immediately following the theoretical explanation.

- if $\alpha, \beta \in For$, then $\alpha \to \beta \in For$.

Immediately we can see the main syntactic difference between $LPP_1^{\mathbb{Q}}$ and $LPP_2^{\mathbb{Q}}$: while in $LPP_2^{\mathbb{Q}}$ we have a separation of formulas into classical and probabilistic, in $LPP_1^{\mathbb{Q}}$ all formulas are treated equally. The intended meaning of the probabilistic operators, using the example of $P_{\geq s}\alpha$, for $s \in \mathbb{Q}_{[0,1]}$, and a formula α, is still that the probability that a formula α holds is at least s. As we inspect the formation rules, we can notice that it is now possible to iterate the probabilistic operators. Therefore, in this logic, we can use formulas such as $P_{\geq \frac{1}{2}}(P_{\geq \frac{1}{3}} p_0)$, expressing the statement that "The probability that the probability of p_0 is at least $1/3$, is at least $1/2$". The remaining connectives $\vee$ (`Or`), $\wedge$ (`And`), and $\leftrightarrow$ (`EquC`) are defined in the usual way, while, additionally, for the probabilistic operators, we make use of the following abbreviations:

(`Plt`) $P_{<s}\alpha$ for $\neg P_{\geq s}\alpha$. (`Ple`) $P_{\leq s}\alpha$ for $P_{\geq 1-s}\neg\alpha$.
(`Pgt`) $P_{>s}\alpha$ for $\neg P_{\leq s}\alpha$. (`Peq`) $P_{=s}\alpha$ for $P_{\geq s}\alpha \wedge P_{\leq s}\alpha$.
(`Pne`) $P_{\neq s}\alpha$ for $\neg P_{=s}\alpha$. (`Fal`) $\bot$ for $\neg(\alpha \to \alpha)$.

where we have that $s \in \mathbb{Q}_{[0,1]}$, and $\alpha \in For$.

4.2 Semantics of $LPP_1^{\mathbb{Q}}$

The semantics for the logic $LPP_1^{\mathbb{Q}}$ is based on the possible–world approach, and is somewhat more complicated than that of $LPP_2^{\mathbb{Q}}$, as a result of the simplification of the syntax and the removal of the distinction between classical and probabilistic formulas.

Definition 15. *An $LPP_1^{\mathbb{Q}}$-model is a structure $M = \langle W, Prob, v\rangle$, where*

- *W is a non-empty set of objects we will refer to as worlds,*
- *$Prob$ is a probability assignment which assigns to each $w \in W$ a probability space $Prob(w) = \langle W(w), H(w), \mu(w)\rangle$, such that:*
  - *$W(w)$ is a non-empty subset of W.*
  - *$H(w)$ is an algebra of subsets on $W(w)$.*
  - *$\mu(w)$ is a finitely additive probability measure $\mu : H(w) \to \mathbb{Q}_{[0,1]}$.*
- *v is a valuation function $v : W \times \phi \to \{true, false\}$, assigning truth-values to each propositional letter in each of the worlds. It is extended to all classical formulas as usual.*

As we can see, this time each of the worlds is equipped with its own subset of worlds, algebra of subsets, and measure, in contrast to $LPP_2^{\mathbb{Q}}$, where there was only one "general" measure on a single algebra of subsets. This change in approach is necessary because of the possibility of iterations of probability operators. Again, the range of the measures, *i.e.* the set of values from which we are assigning probabilities to formulas is the set of all rational numbers from the unit interval. We will provide a detailed look into the encoding of the semantics and the representation of models in Coq, with emphasis on the differences from the previous chapter. First of all, the worlds and the concepts of algebras of subsets and measures are encoded in the same manner as in $LPP_2^{\mathbb{Q}}$, using record types (`ElemW U W`) capturing the property of an element being a member of a subset of `W`, a set of elements of type `U`, within the type. Probability spaces are encoded as follows:

```
Record ProbSpace (U : Type) (W : Ensemble U) : Type :=
    mkPSpace {gWp : Ensemble (ElemW U W);
              gHp         : Algebra (ElemW _ gWp);
              gMp         : MeasureQc (ElemW _ gWp);
              PS_Nes_Cond : gWp <> Empty_set _;
              PS_Alg_Cond : AOS _ gHp;
              PS_Mea_Cond : FAM_Qc _ gMp gHp}.
```

In this manner, we capture all of the necessary conditions – the non-emptiness of $W(w)$, $H(w)$ being an algebra of subsets over $W(w)$, and $\mu(w)$ being a finitely additive measure on $H(w)$ – during the construction of a probability space. Models are then naturally encoded as:

```
Record Model_Simp (U : Type) (W : Ensemble U) : Type :=
    mkMSimp {MC_Worlds    : Ensemble (ElemW U W);
             MC_PSpace    : (ElemW U W) -> (ProbSpace U W);
             MC_Valuation : (ElemW U W) -> nat -> Prop
             MC_ElemWS_Cd : inhabited (ElemW U W);
             MC_Worlds_Cd : MC_Worlds = Full_set (ElemW U W)}.
```

Therefore, apart from the three required elements, we have also embedded into the model the conditions which ensure that our set of worlds is not empty and that the set of worlds encompasses all of the elements of the appropriate `ElemW` type. Again, the worlds are parameterized by a type `U`, allowing for different choices of sets representing worlds, depending on the proof scenario. Next, we proceed to the definition of satisfiability:

Definition 16. *The satisfiability relation $\models$ satisfies the following conditions, for every $LPP_1^{\mathbb{Q}}$-model $M = \langle W, Prob, v\rangle$, every world $w \in W$, and every formula $\Phi \in For_{LPP_1^{\mathbb{Q}}}$:*

- *if $\Phi \in \phi$, then $M, w \models \Phi$ iff $v(w)(\Phi) = true$.*
  Propositional letters are satisfied in a world *iff* they are true in that world.
- *if Φ is of the form $\neg\alpha$, $M, w \models \Phi$ iff $M, w \not\models \alpha$.*
- *if Φ is of the form $\alpha \to \beta$, $M, w \models \Phi$ iff $M, w \not\models \alpha$ or $M, w \models \beta$.*
- *if Φ is of the form $P_{\geq s}\alpha$, $M, w \models \Phi$ iff $\mu(w)([\alpha]_{M,w}) \geq s$,*
  where $[\alpha]_{M,w}$ denotes the set $\{u \in W(w) | M, w \models \alpha\}$, *i.e.* the set of worlds in $W(w)$ in which α holds. This is analogous to the capturing of probability for $LPP_2^{\mathbb{Q}}$.

We will onward focus on the class of all measurable models (`Model_Meas`), which we will denote by $LPP_{1,Meas}^{\mathbb{Q}}$. Measurable models are defined as follows:

Definition 17. *An $LPP_1^{\mathbb{Q}}$-model M is* measurable *if $[\Phi]_{M,w} \in H(w)$, for all $\Phi \in For_{LPP_1^{\mathbb{Q}}}$.*

and are encoded in Coq by adding the measurability condition into the definition of `Model_Simp`. We represent satisfiability of a set of formulas with the function `modelsSet`. Again, we encode satisfiability of the classical negation and implication using Coq's built-in connectives, and make use of Coq's `Classical` library.

Definition 18. *A formula Φ is* satisfiable *if there exists an $LPP_1^{\mathbb{Q}}$-measurable model $M = \langle W, Prob, v\rangle$ and a world $w \in W$ such that $M, w \models \Phi$; Φ is* valid (`Valid`) *if $M, w \models \Phi$, for all $LPP_1^{\mathbb{Q}}$-measurable models $M = \langle W, Prob, v\rangle$ and all worlds $w \in W$ in these models; a set of formulas T is* satisfiable (`Satisfiable`) *if there exists an $LPP_1^{\mathbb{Q}}$-measurable model $M = \langle W, Prob, v\rangle$ and a world $w \in W$, such that $M, w \models \Phi$, for all $\Phi \in T$.*

4.3 A Complete Axiomatization

Interestingly, the set of valid formulas of $LPP_1^{\mathbb{Q}}$ can be characterized both with the same axiom schemata presented in Figure 3.1, and the inference rules as presented in Figure 3.2, with several purely syntactic modifications and only one fundamental difference. We denote this axiomatic system by $Ax_{LPP_1^{\mathbb{Q}}}$, and present it in Figure 4.1 and Figure 4.2.

APT. All instances of classical propositional tautologies.

AP1. $P_{\geq 0}\alpha$.

AP2. $P_{\leq r}\alpha \to P_{<s}\alpha$, for $s > r$.

AP3. $P_{<r}\alpha \to P_{\leq r}\alpha$.

AP4. $P_{\geq r}\alpha \to (P_{\geq s}\beta \to (P_{\geq 1}\neg_c(\alpha \wedge_c \beta) \to P_{\geq r+s}(\alpha \vee_c \beta)))$, for $r + s \leq 1$.

AP5. $P_{\leq r}\alpha \to (P_{<s}\beta \to P_{<r+s}(\alpha \vee_c \beta))$, for $r + s \leq 1$.

AP6. $P_{\geq 1}(\alpha \to \beta) \to (P_{\geq r}\alpha \to P_{\geq r}\beta)$.

Figure 4.1: $LPP_1^{\mathbb{Q}}$ Axiom schemata

Modus Ponens: from α and $\alpha \to \beta$, infer β,

Probabilistic Necessitation: if α is a theorem, infer $P_{\geq 1}\alpha$,

Domain Enforcement: from $\beta \to P_{\neq r}\alpha$, for all $r \in \mathbb{Q}_{[0,1]}$, infer $\beta \to \bot$.

Figure 4.2: $LPP_1^{\mathbb{Q}}$ Inference rules

As we can see, the axioms have remained the same as for $LPP_2^{\mathbb{Q}}$, with the doubled connectives $\neg_c$ and $\neg_p$, and $\to_c$ and $\to_p$ replaced by the unique $\neg$ and $\to$. The domain enforcement rule is once again present in order to preserve strong completeness, given the inherent non-compactness of the system. The standard syntactic notions are defined as follows:

Definition 19. *A formula Φ is* derivable (`Derivable`) *from a given set of formulas (premises) T (denoted by $T \vdash \Phi$) if there exists a finite sequence of formulas $\Phi_0, \ldots, \Phi_k, \Phi$, such that each Φ_i is either in the set T, is an instance of one of the axiom schemata, or is obtained from the preceding formulas by using one of the inference rules. We call such a sequence a* proof *of Φ from T. A formula Φ is a* theorem *(denoted by $\vdash \Phi$) if it is derivable from the empty set of formulas* (`isTheorem`).

Again, in Coq, we define a proof of $T \vdash \Phi$ as a tree of formulas with Φ as the root, with formulas either from the set T or instances of axiom schemata as leaves, and where all of the non-leaves are obtained from previously existing nodes by one of the inference rules (`proof_term`).

Observation 1. *Here, the reader might notice that in the Probabilistic Necessitation inference rule, technically, the notion of being a theorem has been used before it has been introduced, and its referencing may seem somewhat circular. This impreciseness is standard practice in pen-and-paper definitions of logics that require the side condition of "being a theorem", and it has been kept here only so that the reader could make a clear comparison between the axiomatic systems of $LPP_1^{\mathbb{Q}}$ and $LPP_2^{\mathbb{Q}}$. In order to be fully formally correct, the inference rules must be defined jointly with derivability, just as it was done in the Coq encoding.*

Just as $LPP_2^{\mathbb{Q}}$, $LPP_1^{\mathbb{Q}}$ also has an inference rule with a countable collection of premises, which has been handled analogously, by introducing a function from natural numbers to proof terms (`proofs : nat -> proof_term`), and we again make use of a bijection between natural numbers and the $\mathbb{Q}_{[0,1]}$ interval (`f_nat_Q01 : nat -> Q01`).

Definition 20. *A set of formulas T is consistent* (`Consistent`) *if there exists at least one formula α which are not derivable from it, and otherwise is inconsistent* (`Inconsistent_Alternate`). *Alternatively, a set of formulas T is inconsistent if $T \vdash \bot$* (`Inconsistent`). *A set of formulas T is maximally consistent* (`Max_Consistent`) *if it is consistent and for each formula, either it or its negation are in T ($\alpha \in T$ or $\neg\alpha \in T$).*

4.4 Key Properties of $Ax_{LPP_1^{\mathbb{Q}}}$

4.4.1 Soundness

Soundness of $Ax_{LPP_1^{\mathbb{Q}}}$ is proven just as that of $Ax_{LPP_2^{\mathbb{Q}}}$. We show that all of the instances of the axiom schemata are valid formulas, and that the inference rules preserve validity. Here, we present the proof that the axiom schema $AP4$ is valid, as well as the proofs that some of the inference rules preserve validity, while the remaining cases are handled similarly.

Lemma 20 (Validity of $AP4$)**.** *$P_{\geq r}\alpha \to (P_{\geq s}\beta \to (P_{\geq 1}\neg(\alpha \wedge \beta) \to P_{\geq r+s}(\alpha \vee \beta)))$ is valid for all $\alpha, \beta \in For_{LPP_1^{\mathbb{Q}}}$, $r, s \in \mathbb{Q}_{[0,1]}$, and $r + s \leq 1$* (`valid_AP04`).

Proof. Let $M = \langle W, Prob, v\rangle$ be a measurable model, $w \in W$, and let us assume that $M, w \models P_{\geq r}\alpha$, $M, w \models P_{\geq s}\beta$, and $M, w \models P_{\geq 1}\neg(\alpha \wedge \beta)$, while our goal is to show that $M, w \models P_{\geq r+s}(\alpha \vee \beta)$. From our assumptions, we have that $[\alpha]_{M,w} \geq r$, $[\beta]_{M,w} \geq s$, and $[\neg(\alpha \wedge_c \beta)]_{M,w} = 1$ or, equivalently, given the properties of $\mu(w)$, that $[\alpha \wedge \beta]_{M,w} = 0$. Just as in the previous chapter, it can be proven that $[\alpha \wedge_c \beta]_{M,w} = [\alpha]_{M,w} \cap [\beta]_{M,w}$ and $[\alpha \vee_c \beta]_{M,w} = [\alpha]_{M,w} \cup [\beta]_{M,w}$. Given the property of the measure that $\mu(w)([\alpha]_{M,w} \cup [\beta]_{M,w}) = \mu(w)([\alpha]_{M,w}) + \mu(w)([\beta]_{M,w}) - \mu(w)([\alpha]_{M,w} \cap [\beta]_{M,w})$ (`FAM_Qc_union_split`), we obtain that $\mu(w)([\alpha]_{M,w} \cup [\beta]_{M,w}) \geq r + s$, and have our claim. □

Lemma 21 (Validity preservation of $Ax_{LPP_1^{\mathbb{Q}}}$ inference rules)**.** *Let $\alpha, \beta \in For_{LPP_1^{\mathbb{Q}}}$.*

1. *If α is valid, and $\alpha \to \beta$ is valid, then β is also valid* (`vp_MP`).
2. *If α is valid, then $P_{\geq 1}\alpha$ is also valid* (`vp_PN`).
3. *If $\beta \to P_{\neq r}\alpha$ is valid, for all $r \in \mathbb{Q}_{[0,1]}$, then $\beta \to \bot$ is also valid* (`vp_DE`).

Proof.

1. Let us suppose that, for a given model $M = \langle W, Prob, v\rangle \in LPP^{\mathbb{Q}}_{1,Meas}$, $w \in W$, $\alpha, \beta \in For_{LPP_1^{\mathbb{Q}}}$, we have that:
$$M, w \models \alpha \text{ and } M, w \models \alpha \to \beta.$$
From this, given the definition of the satisfiability relation and the definition of implication, it must also be that $M, w \models \beta$, which concludes our proof.

2. Let us suppose that, for any given $M = \langle W, Prob, v\rangle \in LPP^{\mathbb{Q}}_{1,Meas}$, $w \in W$, $\alpha \in For_{LPP_1^{\mathbb{Q}}}$, we have that $M, w \models \alpha$. From this, we have that $[\alpha]_{M,w} = W$, and by the definition of $\mu(w)$, we have that $\mu([\alpha]_{M,w}) = 1$, which means, by the definition of the satisfiability relation, that $M, w \models P_{\geq 1}\alpha$, concluding our proof.

3. Let us suppose that, for any given $M = \langle W, Prob, v\rangle \in LPP^{\mathbb{Q}}_{1,Meas}$, $w \in W$, $\alpha, \beta \in For_{LPP_1^{\mathbb{Q}}}$, we have that

$$M, w \models \beta \to P_{\neq r}\alpha, \text{ for all } r \in \mathbb{Q}_{[0,1]}. \tag{4.1}$$

Our goal is to prove that $M, w \models \beta \to \bot$. If $M, w \not\models \beta$, the proof is immediate. Let us onward assume that $M, w \models A$. From this, Equation 4.1, and the definition of $\models$, we have that

$$\mu(w)([\alpha]) \neq r, \text{ for all } r \in \mathbb{Q}_{[0,1]},$$

which is a contradiction with the definition of $\mu(w)$. Therefore, the case in which $M \models \beta$ is not possible, and we have our claim.

□

Therefore, we have proven the soundness theorem:

Theorem 12 (Soundness of $LPP_1^{\mathbb{Q}}$). *If Φ is a theorem of $Ax_{LPP_1^{\mathbb{Q}}}$, then Φ is valid.*

```
Theorem LPP1_Q_Soundness : ∀ (Φ : FOR), isTheorem Φ → Valid Φ.
```

4.4.2 Completeness

Since the syntax of $LPP_1^{\mathbb{Q}}$ is almost identical to the syntax of $LPP_2^{\mathbb{Q}}$, the proofs of syntactic theorems for both logics follow the same path and are, in some cases, even identical. In the following subsections, we will only focus on proof segments which are different, while for the remaining segments and details, we refer the reader to Chapter 3.

4.4.2.1 The Deduction Theorem.

Just as in the previous chapter, the first theorem we will be proving on the road to strong completeness is the Deduction Theorem. This time, we do not need two versions, as there exists only one type of formulas.

Theorem 13 (Deduction Theorem for $LPP_1^{\mathbb{Q}}$). *Let T be a set of formulas and $\alpha, \beta \in For_{LPP_1^{\mathbb{Q}}}$. Then, the following holds:*

$$T \cup \{\alpha\} \vdash \beta \text{ if and only if } T \vdash \alpha \to_p \beta.$$

```
Theorem LPP1_Q_Deduction_Theorem : ∀ (T : Ensemble FOR) (A B : FOR),
   Derivable (Union FOR T (Singleton FOR A)) B ↔ Derivable T (Prob (Imp A B)).
```

Proof. The right-to-left implication is proven as in classical logic, while the left-to-right implication is proven by induction on the structure of the derivation of $T \cup \{\alpha\} \vdash \beta$, and is mostly identical to the proof of Theorem 7. Here, we will only present the case which is different, which is the case when $T \cup \{\beta\} \vdash \beta$ is obtained by the use of the probabilistic necessitation inference rule. In this case, β is of the form $P_{\geq 1}\gamma$, for some $\gamma \in For_{LPP_1^{\mathbb{Q}}}$, and we have that γ is a theorem. Therefore, we have that $T \vdash \gamma$. Now, using probabilistic necessitation, we obtain that $T \vdash \beta$, from which $T \vdash \alpha \to \beta$ follows. □

4.4.2.2 Maximally Consistent Extension.

The next step is to prove that we can always extend a consistent set of formulas to a maximally consistent set. We prove the following auxiliary claims, analogously to Lemma 6, Lemma 7, and Lemma 8:

Lemma 22. *Derivability is not affected by adding theorems into the context, i.e. if $T \cup T' \vdash F$, and every formula in T' is a theorem, then $T \vdash F$, where $F \in FOR$* (`Der_addThm`). *Consequently, consistency is not affected by adding theorems into the context* (`Consistency_addThm`).

Lemma 23. *For every consistent set of formulas T, and every formula α, either $T \cup \{\alpha\}$ is consistent or $T \cup \{\neg\alpha\}$ is consistent* (`Consistency_xor`).

Lemma 24. *For all maximally consistent sets T, and for all formulas α, β, the following holds:*

1. *Either $\alpha \in T$ or $\neg\alpha \in T$* (`MCons_A_NegA`).
2. *$\alpha \in T$ if and only if $T \vdash \alpha$* (`MCons_In_Der`).
3. *If $\neg\alpha \in T$, then $\alpha \notin T$* (`MCons_Neg`).
4. *If $\alpha \to \beta \in T$, then $\neg\alpha \in T$ or $\beta \in T$* (`MCons_Imp`).
5. *If $\alpha \leftrightarrow \beta \in T$, then $\alpha \in T$ if and only if $\beta \in T$* (`MCons_Equ`).
6. *If $\alpha \wedge \beta \in T$, then $\alpha \in T$ and $\beta \in T$* (`MCons_And`).
7. *If $\alpha \vee \beta \in T$, then $\alpha \in T$ or $\beta \in T$* (`MCons_Or`).

The following lemma, analogous to Lemma 11 both in formulation and in proof, provides an insight into the compatibility of probabilistic operators with consistency:

Lemma 25. *For every consistent set of formulas T, formulas α and β, if $T \cup \{\beta\}$ is consistent, then there exists a rational number $p(\alpha) \in \mathbb{Q}_{[0,1]}$ such that $T \cup \{\beta\} \cup \{P_{=p(\alpha)}\alpha\}$ is consistent.* (`Peq_existence`)

Hereinafter, we will assume to have a consistent set of formulas T. We denote $T_0 = T$, establish an enumeration of formulas $\alpha_0, \alpha_1, \alpha_2, \ldots$ (`f_enum : nat -> FOR`), and define a sequence of sets T_n (`Tn`), for $n \geq 1$, as follows:

$$T_{n+1} = \begin{cases} T_n \cup \{\alpha_n\} \cup \{P_{=p(\alpha_n)}\alpha_n\} & \text{if } T_n \cup \{\alpha_n\} \text{ is consistent,} \\ T_n \cup \{\neg\alpha_n\} \cup \{P_{=p(\neg\alpha_n)}\neg\alpha_n\} & \text{otherwise.} \end{cases}$$

The following lemma is proven using the previously proven lemmas:

Lemma 26. *The following properties concerning the sequence T_n hold:*

1. *For all $n \geq 0$, $T_n \subseteq T_{n+1}$* (`Tn_increasing`).
2. *For all $n \geq 0$, T_n is consistent* (`Tn_all_consistent`).

Next, we define the maximally consistent extension T^* of T, as

$$T^* = \bigcup_{i=1}^{\infty} T_i.$$

```
Definition MC_Extension (T : Ensemble FOR) : Ensemble FOR :=
        (fun A : FOR ⇒ ∃ k : nat, In FOR (Tn T k) A.)
```

Again, this extension is naturally defined in Coq as a function in `Prop`, which is true only for the formulas contained in at least one of the extensions of T. For T^* defined in this way, we prove that the following statements hold:

Lemma 27 (Properties of T^*)**.**

1. *For all $\alpha \in For_{LPP_1^{\mathbb{Q}}}$, either $\alpha \in T^*$ or $\neg\alpha \in T^*$* (`MCE_maximal`).
2. $\bot \notin T^*$ (`MCE_no_Fal`).
3. $T \subseteq T^*$ (`T_in_MCE`).
4. *If $T^* \vdash \alpha$, then $\alpha \in T^*$, for $\alpha \in For_{LPP_1^{\mathbb{Q}}}$* (`MCE_in`).

Proof.

1. Directly, by construction of T^*.
2. By contradiction. If $\bot \in T^*$, then, by definition of T^*, there exists an integer k, such that $\bot \in T_k$, which is a contradiction with the previously proven consistency of T_k.
3. Directly, as $T = T_0$, and $T_0 \subseteq T^*$.
4. By induction on the derivation of $T \vdash \alpha$. We will only show the case where A has been obtained using the probabilistic necessitation inference rule. Then, we have that α is of the form $P_{\geq 1}\beta$, and that β is a theorem. Next, let k be the index of the formula β in the enumeration of formulas. Then, since β is a theorem, and given Lemma 22, we have that $T_{n+1} = T_n \cup \beta$, and we have our claim.

□

As a direct consequence of the previous Lemma, we obtain the main result of this subsection:

Theorem 14. *T^* is a maximally consistent set of formulas.*

4.4.2.3 The Canonical Model.

Before we can proceed to the actual construction, we need several auxiliary lemmas:

Lemma 28. *Let T be a maximally consistent set. Then, for every $\alpha \in For_{LPP_1^{\mathbb{Q}}}$, there exists a unique $q(\alpha) \in \mathbb{Q}_{[0,1]}$, such that $T \vdash P_{=q(\alpha)}\alpha$.* (`unique_for_M`)

Proof. Analogous to the proof of Lemma 9. □

Lemma 29. *A formula α is in all maximally consistent sets if and only if α is a theorem.* (`in_all_MCons_thm`)

Proof. The right-to-left direction follows directly from the definition of maximal consistency. For the left-to-right direction, let us assume that for all maximally consistent sets T, it holds that $\alpha \in T$. First, we will prove that $T' = \{\neg\alpha\}$ is inconsistent. Let us assume, to the contrary, that it is consistent, and let T'^* be its maximally consistent extension. Then, we would have that both $\neg\alpha \in T'^*$ (by construction) and $\alpha \in T'^*$ (by assumption), which is not possible, as then T'^* would not be consistent. Now, we have that $\{\neg\alpha\}$ is inconsistent, *i.e.* that $\{\neg\alpha\} \vdash \bot$. Using the Deduction Theorem, we obtain that $\emptyset \vdash \neg\alpha \to \bot$ and from there, using classical propositional tautologies, we obtain that $\emptyset \vdash \alpha$. □

Now, we can construct a canonical model M^* for a given consistent set of formulas T. Now, we define each of the required model components as follows:

Worlds. The set of worlds W will be the set of all maximally consistent sets (`CM_Worlds`).

Valuation. The valuation function $v : W \times \phi \to \{true, false\}$ (`v`) is defined naturally as $v(w, p_i) = true$ if $p_i \in w$, and otherwise $false$ (`CM_Valuation`).

Probability Space. We will define the probability spaces for each world w as follows:

- **Subsets of Worlds.** For each $w \in W$, we take $W(w) = W$ (`CMWp`). This proves to be sufficient, and there is no need for having different subsets for different worlds.
- **Algebra of Subsets.** We will define $H(w)$ (`CMHp`) in the following way:

$$H(w) = \{[\alpha] = \{w \in W : \alpha \in w\} \mid \alpha \in For_{LPP_1^{\mathbb{Q}}}\}.$$

  It is easily proven that $H(w)$, defined in this way, satisfies the requirements to be an algebra of subsets over $W(w)$: it contains the empty set, it is closed under complement, and is closed under finite union. (`CM_Alg_Cond`)
- **Measure.** The measure should map the algebra of subsets $H(w)$ into the rational unit interval $\mathbb{Q}_{[0,1]}$. Since the elements of $H(w)$ are essentially determined by formulas, we will define $\mu(w)$ (`CMMp_to_Q01`) as follows, using notation from Lemma 28:

$$\mu(w)([\alpha]) = q(\alpha), \text{ where } T^* \vdash P_{=q(\alpha)}\alpha.$$

  Lemma 28 states that a unique $q(\alpha)$ exists for each $\alpha \in For_{LPP_1^{\mathbb{Q}}}$, ensuring that the measure is well-defined. Just as in the case of $LPP_2^{\mathbb{Q}}$, the following claims needed to be proven with regard to the properties of $\mu(w)$:
  - Measure and negation: $\mu(w)([\alpha]) = 1 - \mu(w)([\neg\alpha])$ (`CMMp_negation`).
  - Non-negativity: $\mu(w)([\alpha]) \geq 0$, for all $\alpha \in For_{LPP_1^{\mathbb{Q}}}$ (`CMMp_FAM_nonneg`).
  - Measure and the empty set: $\mu(\emptyset) = 0$ (`CMMp_FAM_empty_set`).
  - Measure and the full set: $\mu(W) = 1$ (`CMMp_FAM_Full_set`).
  - Additivity: μ is finitely additive (`CMMp_FAM_additive`).

In this way, we have obtained a model $M^* = \langle W, Prob, v\rangle$ (`CMModel_Simp`), and the remaining claim that we need to prove is that it is measurable.

Lemma 30. *M^* is a measurable model* (`CM_isMeas`).

Proof. We should prove that for $\alpha \in For_{LPP_1^{\mathbb{Q}}}$, and all $w \in W$, it holds that $[\alpha]_{M^*,w} \in H(w)$. However, this is true, as this was how $H(w)$ was constructed in the first place. □

4.4.2.4 Strong Completeness.

With the canonical model constructed, we require an additional lemma before we can tackle strong completeness:

Lemma 31. *Let T be a set of formulas, $M = \langle W, Prob, v\rangle$ be a measurable model, $w \in W$, and let us assume that $M, w \models T$. Then, it holds that for all $\alpha \in For_{LPP_1^{\mathbb{Q}}}$, if $T \vdash \alpha$, then $M, w \models \alpha$.*

Proof. By structural induction on the derivation of $T \vdash \alpha$. The cases in which $\alpha \in T$ and in which α is an instance of an axiom or obtained by the modus ponens inference rule follow directly from the soundness of the system (Theorem 12) and the induction hypothesis (IH). Next, let $T \vdash \alpha$ be obtained by the probabilistic necessitation inference rule. This means that $\alpha = P_{\geq 1}\beta$, for some $\beta \in For_{LPP_1^{\mathbb{Q}}}$, and that β is a theorem. However, by the Soundness theorem, β is then valid, *i.e.* $M, w \models \beta$ for all $w \in W$. Therefore, we have that $[\beta]_{M,w} = W(w)$, from which we obtain that $\mu(w)([\beta]_{M,w}) = 1$, and, consequently, that $M, w \models P_{\geq 1}\beta$, which is our claim. Finally, let $T \vdash \alpha$ be obtained by the domain enforcement inference rule. This means that $\alpha = \beta \to \bot$, and that this was obtained from $T \vdash \beta \to P_{\neq r}\gamma$, for all $r \in \mathbb{Q}_{[0,1]}$, and some $\beta, \gamma \in For_{LPP_1^{\mathbb{Q}}}$. By the IH, we have that $M \models \beta \to P_{\neq r}\gamma$, for all $r \in \mathbb{Q}_{[0,1]}$. However, $M \models \beta$ is not possible, as by definition of satisfiability and the measure, it would have to be that $M \models \bot$. Therefore, it must be that $M \not\models \beta$, from which we easily obtain the desired $M \models \beta \to \bot$. □

Finally, we can prove the main result of this chapter:

Theorem 15 (Strong Completeness of $LPP_1^{\mathbb{Q}}$)**.** *A set of formulas T is $LPP_1^{\mathbb{Q}}$-consistent if and only if it is $LPP_{1,Meas}^{\mathbb{Q}}$-satisfiable* (`LPP1_Q_Strong_Completeness`).

```
Theorem LPP2_Q_Strong_Completeness : ∀ T : Ensemble FOR,
                                     Consistent T <-> Satisfiable T.
```

Proof. First, let T be consistent. We need to prove that it is satisfiable, *i.e.* that there exists an $LPP_{1,Meas}^{\mathbb{Q}}$-model $M = \langle W, Prob, v\rangle$ and a world $w \in W$, such that $M, w \models \alpha$, for all formulas $\alpha \in T$. We will choose the model to be the canonical model M^*, while the corresponding world will be T^*, which is possible as the worlds in M^* are maximally consistent sets. We prove easily, by structural induction on α, that $M^*, T^* \models \alpha$ if and only if $\alpha \in T^*$. From there, we obtain that $M, w \models \alpha$, for all $\alpha \in T$, as $T \subseteq T^*$.

Next, let T be satisfiable. Then, there exists a model $M = \langle W, Prob, v\rangle$ and a world $w \in W$ such that $M, w \models \alpha$, for all $\alpha \in T$. Then, by Lemma 31, we obtain that M and w model all formulas derivable from T. If T were to be inconsistent, then we would have that $M, w \models \bot$ which is not possible by definition of satisfiability, and T must be consistent. □

4.4.2.5 Simple Completeness of $LPP_1^{\mathbb{Q}}$.

The Simple Completeness Theorem follows naturally from the Strong Completeness Theorem:

Theorem 16 (Simple Completeness of $LPP_1^{\mathbb{Q}}$.)**.** *If a formula Φ is $LPP_{1,Meas}^{\mathbb{Q}}$-valid, then it is a theorem of $LPP_1^{\mathbb{Q}}$.* (`LPP1_Q_Simple_Completeness`)

```
Theorem LPP1_Q_Simple_Completeness : ∀ Φ : FOR, Valid Φ -> isTheorem Φ.
```

Proof. If α is $LPP_{1,Meas}^{\mathbb{Q}}$-valid, then $\neg\alpha$ is not satisfiable and by the Strong Completeness Theorem, $\{\neg\alpha\}$ is inconsistent, *i.e.* $\{\neg\alpha\} \vdash \bot$. From there, we easily obtain $\vdash \alpha$ from the Deduction Theorem and the elimination of double negation. □

4.4.3 Non-Compactness

Let us consider the set $T = \{P_{\neq 0}\alpha\} \cup \{P_{<\frac{1}{n}}\alpha |\ n \in \mathbb{N}^+\}$. Using the domain enforcement inference rule, we obtain that T is inconsistent (`Cpc_Incons_T`). By the strong completeness theorem, T is not satisfiable. However, every finite subset of T is consistent (`Cpc_Finite_Sat_T`), and, therefore, satisfiable. This gives us the non-compactness theorem for $LPP_1^{\mathbb{Q}}$:

Theorem 17 (Non-compactness of $LPP_1^{\mathbb{Q}}$). *There exists a set of $LPP_1^{\mathbb{Q}}$-formulas which is unsatisfiable, but whose every finite subset is satisfiable.*

```
Theorem LPP1_Q_NonCompactness : ∃ T : Ensemble FOR,
(∀ T' : Ensemble FOR, Finite _ T' → Included _ T' T → Satisfiable T') ∧
                                ¬ Satisfiable T.
```

4.5 Summary

In this chapter, we have presented $LPP_1^{\mathbb{Q}}$ - a rationally-valued probability logic with iterations of probability operators. We have introduced a sound and strongly-complete axiomatic system for $LPP_1^{\mathbb{Q}}$, and formally proven in Coq the main meta-theoretic properties of $LPP_1^{\mathbb{Q}}$ – soundness, strong and simple completeness, and non-compactness. Similarly to LPCP and $LPP_2^{\mathbb{Q}}$, due to the inherent non-compactness of the system, an infinitary inference rule had to be introduced into the system. We have examined the similarities and differences between $LPP_1^{\mathbb{Q}}$ and $LPP_2^{\mathbb{Q}}$ and observed that due to the more uniform syntax, the proofs in $LPP_1^{\mathbb{Q}}$ are more elegant and somewhat more straightforward. Finally, the formalization of $LPP_1^{\mathbb{Q}}$ in Coq revealed an imprecision in the pen-and-paper definitions of the inference rules, and brought to light the need for the inference rules to be defined jointly with the notion of derivability.

Part II

A Logical Framework with External Predicates

Chapter 5

Introducing $\mathsf{LF}_{\mathcal{P}}$

Contents

The Edinburgh Logical Framework LF, presented in [Harper 1993], is a typing system featuring dependent types. It was first introduced as a *general meta-language for logics*, as well as a specification language for *generic proof-checking/proof-development environments*. In this thesis, we consider an extension of LF with *predicates*, which is accomplished by defining *locked type constructors*, which resemble $\diamond$-*modality constructors*, for constructing types of the shape $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, where $\mathcal{P}$ is predicate on typed judgements and the validity of which may be verified outside of the $\mathsf{LF}_{\mathcal{P}}$ framework, making it, in that sense, an *external predicate*.

Following the standard specification paradigm in Constructive Type Theory, we define locked types using *introduction*, *elimination*, and *equality rules*. We introduce a lock *constructor* for building objects $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]$ of type $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, via the *introduction rule* (O·Lock), presented below, and a corresponding unlock *destructor*, $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]$, and an *elimination rule* (O·Unlock) which allows for the elimination of the locked type constructor, under the condition that a specific predicate $\mathcal{P}$ is verified, possibly *externally*, on an appropriate *correct*, *i.e.* derivable, judgement.

$$\frac{\Gamma \vdash_\Sigma M : \rho \qquad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]} \qquad (\mathsf{O{\cdot}Lock})$$

$$\frac{\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \qquad \Gamma \vdash_\Sigma N : \sigma \qquad \mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)}{\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho} \qquad (\mathsf{O{\cdot}Unlock})$$

The *equality rule* for locked types amounts to a new form of reduction we refer to as *lock-reduction* ($\mathcal{L}$-reduction), $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\mathcal{L}} M$, which allows for the elimination of a *lock*, in the presence of an *unlock*. The $\mathcal{L}$-reduction combines with standard β-reduction into $\beta\mathcal{L}$-reduction.

$\mathsf{LF}_{\mathcal{P}}$ is parametric over a potentially unlimited[1] set of predicates $\mathcal{P}$, which are defined on derivable typing judgements of the form $\Gamma \vdash_\Sigma N : \sigma$. The syntax of $\mathsf{LF}_{\mathcal{P}}$ predicates is not specified, with the main idea being that their truth is to be verified via a *call* to an *external validation tool*; one can view this externalization as an *oracle call*. Thus, $\mathsf{LF}_{\mathcal{P}}$ allows for the invocation of external "modules" which, in principle, can be executed elsewhere, and whose successful verification can be acknowledged in the system via $\mathcal{L}$-reduction. Pragmatically, locked types allow for the factoring out of the complexity of derivations by delegating the {checking, verification, computation} of such predicates to an external proof engine or tool. The proof terms themselves do not contain explicit evidence for external predicates, but just record that

[1] The predicates need to satisfy certain requirements so that Subject Reduction of $\mathsf{LF}_{\mathcal{P}}$ holds, *cf.* Definition 21.

a verification {has to be (lock), has been successfully (unlock)} carried out. In this manner, we combine the reliability of formal proof systems based on constructive type theory with the efficiency of other computer tools, in the style of the *Poincaré Principle* [Barendregt 2002].

In this thesis, we develop the meta-theory of $\mathsf{LF}_{\mathcal{P}}$. Strong normalization and confluence are proven without any additional assumptions on predicates. For subject reduction, we require the predicates to be *well-behaved*, *i.e. closed under weakening, permutation*, *substitution*, and *$\beta\mathcal{L}$-reduction* in the arguments. $\mathsf{LF}_{\mathcal{P}}$ is decidable, if the external predicates are decidable. We also provide a *canonical* presentation of $\mathsf{LF}_{\mathcal{P}}$, in the style of [Watkins 2002, Harper 2007], based on a suitable extension of the notion of $\beta\eta$-long normal form. This allows for simple proofs of adequacy of the encodings.

In particular, we encode in $\mathsf{LF}_{\mathcal{P}}$ the call-by-value λ-calculus and discuss a possible extension which supports the *design-by-contract* paradigm. We provide smooth encodings of side conditions in the rules of Modal Logics, both in Hilbert and Natural Deduction styles, *cf.* [Avron 1998, Crary 2010]. We also encode sub-structural logics, namely non-commutative Linear Logic, *cf.* [Polakow 1999, Crary 2010]. We also illustrate how $\mathsf{LF}_{\mathcal{P}}$ can naturally support *program correctness* systems and Hoare-like logics. We also discuss how other systems can be embedded into $\mathsf{LF}_{\mathcal{P}}$ via locked types and provide pseudo-code for some of the used predicates.

As far as expressiveness is concerned, $\mathsf{LF}_{\mathcal{P}}$ is a stepping stone towards a general theory of *shallow vs deep encodings*, with our encodings being shallow by definition. Clearly, by Church's thesis, all external decidable predicates in $\mathsf{LF}_{\mathcal{P}}$ can be encoded, possibly with very deep encodings, in standard LF. It would be interesting to state in a precise categorical setting the relationship between such deep internal encodings and the encodings in $\mathsf{LF}_{\mathcal{P}}$.

$\mathsf{LF}_{\mathcal{P}}$ can also be viewed as a neat methodology for separating the logical-deductive contents from, on one hand, the verification of structural and syntactical properties, which are often needlessly cumbersome but ultimately computable, or, on the other hand, from more general means of validation.

5.1 A Brief Detour into Philosophy

Since Euclid first introduced the concept of *rigorous proof* and the *axiomatic/deductive method*, philosophers have been discussing the nature of mathematics, debating over whether it is essentially *analytic* or *synthetic*? Although we do not presume to give comments on such a fundamental philosophical issue, we do believe that the topics in this part of the thesis could be cast against that background, and we will, therefore, offer some comments in that direction.

Several possible, albeit partial modern readings of the synthetic vs. analytic opposition could be, in our view, those of *deduction from axioms* vs. *computation according to rules*, *proof checking* vs. *verification*, and proving *inhabitability* of judgements vs. *definitional equality* of types.

The mechanism of *locking and unlocking types* in the presence of *external oracles*, which we are introducing in $\mathsf{LF}_{\mathcal{P}}$, effectively opens up the Logical Framework to alternate means of providing evidence for judgements. In standard LF, there are only two ways of providing this evidence, namely discovering types to be inhabited or postulating that types are inhabited by introducing appropriate constants. The *locked/unlocked types* of $\mathsf{LF}_{\mathcal{P}}$ open the door to an intermediate level, one provided by external means, such as computation engines or automated theorem proving tools. However, among these, one could also think of graphical tools based on neural networks, or even intuitive visual arguments, as were used in ancient times for giving the first *demonstrations* of the Pythagoras' theorem, for instance. In a sense, $\mathsf{LF}_{\mathcal{P}}$, by allowing formal accommodation of any alternative proof method to pure axiomatic deduction, vindicates all of the "proof cultures" which have been used pragmatically in the history of mathematics, and not only in the Western tradition.

One natural objection that can be raised against the $\mathsf{LF}_{\mathcal{P}}$ framework is: "Alternative proof methods are not rigorous enough! We need to go through the pains of rigorous formalized proof checking in order to achieve the highest possible reliability of our certifications!". While this point is, of course, true, several points do need to be made.

First of all, absolute certainty is a myth, as it cannot be achieved. The *De Bruijn Principle* [Barendregt 2002, de Bruijn 1968] is usually invoked in this respect. It amounts to the request that the core of the proof checker be small and verifiable by hand. Alternate proof techniques certainly do not satisfy it in a strict sense. But alternate proof techniques, if properly recorded, are not useless and come at a somewhat intermediate level between rigorous encoding and blatant axioms. They can expedite verifications, as in the case of the *Poincaré's Principle* [Barendregt 2002], or *Deduction Modulo* [Dowek 2003], or make the proof more perspicuous and provide some intuition, as Schopenhauer advocated [Shopenhauer 2008]. On a lighter note, just recall the anecdote of a famous mathematician lecturing at a seminar, who, halfway through the proof, said: "And this trivially holds!" Just to say a few seconds later: "But is it really trivial here? Hmm...". And after about ten minutes of silence triumphantly exclaimed: "Yes, it is indeed trivial!". How do we encode such evidence? Should we just rule it out?

Anecdote aside, there is a far deeper reason, however, why a fundamentalist approach to certainty cannot be maintained that easily, and this has to do with the issue of *adequacy*. Contrast, for a moment, the process of proving a computation correct with respect to carrying out its verification by directly executing it. Consider, for example, that $1^1 * 2^2 * 3^3 = 108$. In the latter case, one would need to do some simple arithmetic, while in the former case, one would need to reify the rules for computing exponentials and products. Of course, using the *autarkic* approach explained in [Barendregt 2002] or reasoning by reflection as in [Chlipala 2013], one could internalize the needed arithmetics checking procedures (proving their correctness once and for all), while still preserving the de Bruijn principle and keeping proof terms small. However, our approach is more "schematic", in the sense that it creates room for "plugging-in" any verifier, without the need to specify which one and without the need to prove its soundness internally.

But what can truly guarantee that the formalization we have encoded is adequate, *i.e.* that it corresponds to our intended understanding of arithmetic? Enter the Münchhausen Trilemma in all its glory. The choices we have are:

- *Infinite regression*, proofs requiring more proofs, which appears because of the necessity to go ever further back. However, this isn't practically feasible and by its very nature does not provide any foundation. Just ask Achilles how his dialogue with the Tortoise went, if he ever gets out of it, that is [Carroll 1895]. Ok, how about then...
- *A Logical Circle* in the deduction, caused by the fact that one, in the need to found, falls back on statements which had already appeared before as requiring a foundation. Hmmm... this feels like building something ready to collapse at any moment. Ok then, we really have no choice but to...
- "Just do it". Break the search at a certain point and proclaim what the foundations are. Enter axioms. This is principally feasible, but involves the concept of "sufficient reason", which tends to vary from person to person. Where do we stop the search and declare that something must hold?

The moral is that the issue of proving that formal statements, such as *specifications*, *encodings*, and *proof obligations*, do indeed correspond to the intended meanings and pragmatic usages cannot ever be done completely internally to any system. Ultimately, we have to resort to some informal argument outside any possible De Bruijn Principle. And any such argument can, at best, increase our confidence in the correctness of our proof of the arithmetical computation. If one looks for a definitive proof of adequacy, one is led into an infinite regress. Fully

internalized arguments can rely on the De Bruijn Principle, but even the simplest application takes us outside the system. Ultimately, we have no choice but to "just do it".

One concluding comment. The traditional LF answer to the question "What is a Logic?" was: "A signature in LF". In $\mathsf{LF}_{\mathcal{P}}$, we can give the homologue answer, namely "A signature in $\mathsf{LF}_{\mathcal{P}}$", since external predicates can be read off the types occurring in the signatures themselves. But, we can also use this very definition to answer a far more intriguing question:

"What is a Proof Culture?".

5.2 Comparison with Related Work

The work presented in this part of the thesis relies on [Honsell 2013b, Honsell 2013a], and continues the research line of [Honsell 2007, Honsell 2008a], which present extensions of the original LF, where a notion of β-reduction *modulo* a predicate $\mathcal{P}$ is considered. The main idea of these works is that of *stuck-reductions* in terms and types in the setting of higher-order term rewriting systems, by Cirstea-Kirchner-Liquori [Cirstea 2001], later generalized to a framework of Pure Type Systems with Patterns [Barthe 2003]. This typing protocol was essential for the preservation of strong normalization of typable terms, as was proven in [Honsell 2007]. In [Honsell 2007, Honsell 2008a] the dependent function type is conditioned by a predicate, and we have a corresponding *conditioned* β-reduction, which fires when the predicate holds on a term or judgement. In $\mathsf{LF}_{\mathcal{P}}$, the predicates are external to the system and the verification of the validity of the predicate is part of the typing system. Standard β-reduction is recovered and combined with an *unconditioned* lock reduction. The move of having predicates as new type constructors rather than parameters of Π's and λ's allows $\mathsf{LF}_{\mathcal{P}}$ to be a mere *language extension* of standard LF. This simplifies the meta-theory, and provides a more modular approach.

The approach adopted here generalizes and subsumes, in an abstract way, other approaches in the literature which combine internal and external derivations. In many cases, it can express and incorporate these approaches. The relationship with the systems of [Cirstea 2001, Barthe 2003, Honsell 2007, Honsell 2008a], which combine derivation and computation, has been discussed above. Systems supporting the *Poincaré Principle* [Barendregt 2002], or *Deduction Modulo* [Dowek 2003], where derivation is separated from verification, can be directly incorporated in $\mathsf{LF}_{\mathcal{P}}$. Similarly, we can abstractly subsume the system presented in [Blanqui 2008], which addresses a specific instance of our problem: how to outsource the computation of a decision procedure in Type Theory in a sound and principled way via an abstract conversion rule. One other system which has a very similar goal with respect to $\mathsf{LF}_{\mathcal{P}}$ is presented in [Cousineau 2007], where a framework named $\lambda\Pi$-calculus modulo is introduced, extending the original LF with computation rules. The latter are realized by means of rewrite rules empowering the "traditional" conversion rule of LF (*i.e.* the congruence relation $\equiv_\beta$ is replaced by $\equiv_{\beta\mathcal{R}}$, where $\mathcal{R}$ denotes the set of rewrite rules introduced into the system). The authors then successfully encode all functional Pure Type Systems (PTS) into the $\lambda\Pi$-calculus modulo, proving the conservativity of their embedding under the termination hypothesis. The main difference between $\lambda\Pi$-calculus modulo and $\mathsf{LF}_{\mathcal{P}}$ lies in the fact that the latter features a simpler metatheory, because the reduction is closer to the standard β-reduction (at least in principle) and the external predicates are handled in a more controlled way by means of the lock/unlock mechanism. The direct consequence of this approach, from a practical point of view (when considering a possible implementation), is that we do not need to change the kernel of the original LF, but only *extend* it.

In [Virga 1999], an extension of the Edinburgh LF with an equational theory is proposed, opening the door to new ways of conversions among types within the framework. As a consequence, strong normalization and confluence properties remain valid only in a weaker form (namely, modulo the equivalence induced by the equational theory on types). In the second part

of the work, Higher-order Term Rewriting Systems (HTRS) with dependent types are introduced and used to generate equational theories, much like those analyzed in the first part. Of course, the rewriting rules of such an HTRS must adhere to some constraints, in order to guarantee the fundamental properties of the extended LF. For instance, it is forbidden to use a rewrite rule to rewrite the type of another rule, *i.e.* the rewriting must preserve well-typedness of expressions. According to the author, the benefit of the new system with respect to the original LF, is to overcome the inadequacies emerging when dealing with the encoding of object languages embodying notions of computations via equational rules. This work has served as a stepping stone to constraint-based extensions of the proof assistant Twelf, which are called *constraint domains* [Pfenning 2013]. These extensions provide a way for users to work easily with objects (such as rational numbers), the explicit formalization of which in Twelf would otherwise be quite lengthy or inefficient, but are still considered to be highly experimental.

The work presented here also has a bearing on proof irrelevance. In [Pfenning 1993], two terms inhabiting the same *proof irrelevant type* are set to be equal. However, when dealing with proof irrelevance in this way, a great amount of internal work is required, all of the relevant rules have to be explicitly specified in the signature, and the *irrelevant* terms need to be derived in the system anyway. With our approach, we move one step further, and do away completely with *irrelevant* terms in the system by delegating the task of building them to the external proof verifier. We limit ourselves to the recording, through a locked type, that one such evidence, possibly established elsewhere, needs to be provided, making our approach more modular.

In the present work, predicates are defined on derivable judgements, and hence may, in particular, inspect the signature and the context, which normal LF cannot. The ability to inspect the signature and the context is reminiscent of [Pientka 2008, Pientka 2010], although in that approach the inspection was layered upon LF, whereas in $\mathsf{LF}_{\mathcal{P}}$ it is integrated in the system. This integration is closer to the approach of [Licata 2009], but additional work is required in order to be able to compare their expressive powers precisely.

Another interesting framework, which adds a layer on top of LF is the Delphin system [Poswolsky 2009], providing a functional programming language allowing the user to encode, manipulate, and reason over dependent higher-order datatypes. However, in this case as well, the focus is placed on the computational level inside the framework, rather than on the capability of delegating the verification of predicates to an external oracle.

LF with Side Conditions (LFSC), presented in [Stump 2009], is more reminiscent of our approach as "it extends LF to allow side conditions to be expressed using a simple first-order functional programming language". Indeed, the author aims at factoring the verifications of (complicated) side-conditions out of the main proof. Such a task is delegated to the type checker, which runs the code associated with the side-condition, verifying that it yields the expected output. The proposed machinery is focused on providing improvements for solvers related to Satisfiability Modulo Theories (SMT).

5.3 Synopsis of the Upcoming Chapters.

In Chapter 6, we present the syntax of $\mathsf{LF}_{\mathcal{P}}$, its typing system, and the notion of $\beta\mathcal{L}$-reduction. In Chapter 7, we prove the main meta-theoretical properties of the system, and discuss the expressive power of $\mathsf{LF}_{\mathcal{P}}$. In Chapter 8, we present a canonical version of $\mathsf{LF}_{\mathcal{P}}$, and discuss the correspondence with the full $\mathsf{LF}_{\mathcal{P}}$ framework. In Chapter 9, we show how to encode into $\mathsf{LF}_{\mathcal{P}}$ the call-by-value λ-calculus, a minimal functional language following the *design-by-contract* programming paradigm, Modal Logics in Hilbert and natural deduction style, non-commutative linear logic, a small imperative programming language featuring Hoare Logic, and provide several concluding comments on the benefits of $\mathsf{LF}_{\mathcal{P}}$.

CHAPTER 6

The $\mathsf{LF}_{\mathcal{P}}$ System

Contents

The pseudo-syntax of the $\mathsf{LF}_{\mathcal{P}}$ system is presented in Figure 6.1. We have five syntactic categories: signatures, contexts, kinds, families or types, and objects. This pseudo-syntax is, essentially, that of LF (*cf.* [Harper 1993]), with the removal of abstraction in families, and the addition of a *lock constructor* ($\mathcal{L}^{\mathcal{P}}_{N,\sigma}[-]$) on families and objects, and a corresponding *lock destructor* ($\mathcal{U}^{\mathcal{P}}_{N,\sigma}[-]$) on objects. The environment in which derivations occur is, as usual, described by the signature and the context, where the former keeps track of types or kinds assigned, respectively, to constant families and objects, while the latter keeps track of families assigned to variables. Both the lock and unlock constructors are parametrized over a unary logical predicate $\mathcal{P}$, which is defined on derivable type judgements of the form $\Gamma \vdash_\Sigma N : \sigma$. The entire $\mathsf{LF}_{\mathcal{P}}$ system is parameterized over a finite set of such predicates, and as these predicates are external by nature, they are not formalized explicitly (more comments are provided in Section 7.4). However, these predicates are required to satisfy certain well-behavedness conditions, which will be presented in Chapter 7, in order to ensure subject reduction of the system. For the sake of notational completeness, the list of external predicates should also appear in the signature, but we will omit it so as to increase readability.

Notational conventions and auxiliary definitions. We will be using the following notation: $M, N, \ldots \in \mathcal{O}$ denote objects, $c, d, \ldots$ denote object constants, $x, y, z, \ldots$ denote object variables, $\sigma, \tau, \rho, \ldots \in \mathcal{F}$ denote types, $a, b, \ldots$ denote constant types, $K \in \mathcal{K}$ denotes kinds, $\Gamma \in \mathcal{C}$ denotes contexts, $\Sigma \in \mathcal{S}$ denotes signatures, and $\mathcal{P}$ denotes predicates. We refer to $\mathcal{L}$ as the *lock symbol*, and to $\mathcal{U}$ as the *unlock symbol*. We will be using T to denote any term of the calculus (kind, family, or object), where, in some cases, the syntactic category to which T can belong will be clear from the context. We suppose that, in the context $\Gamma, x{:}\sigma$, the variable x does not occur free in Γ or in σ. We will be working modulo α-conversion and Barendregt's

Σ	$\in$	$\mathcal{S}$	Σ	$::=$	$\emptyset \mid \Sigma, a{:}K \mid \Sigma, c{:}\sigma$	*Signatures*
Γ	$\in$	$\mathcal{C}$	Γ	$::=$	$\emptyset \mid \Gamma, x{:}\sigma$	*Contexts*
K	$\in$	$\mathcal{K}$	K	$::=$	$\mathsf{Type} \mid \Pi x{:}\sigma.K$	*Kinds*
σ, τ, ρ	$\in$	$\mathcal{F}$	σ	$::=$	$a \mid \Pi x{:}\sigma.\tau \mid \sigma\, N \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$	*Families (Types)*
M, N	$\in$	$\mathcal{O}$	M	$::=$	$c \mid x \mid \lambda x{:}\sigma.M \mid M\, N \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \mid \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]$	*Objects*

Figure 6.1: The pseudo-syntax of $\mathsf{LF}_{\mathcal{P}}$

variable convention. All of the symbols can appear indexed. Next, we proceed to specify the notions of the domain of a signature, the domain of a context, free and bound variables of a term, as well as substitution, in the light of the newly introduced constructors and destructors. The domain of a signature $\mathsf{Dom}(\Sigma)$ is defined as follows:

$$\begin{aligned}
\mathsf{Dom}(\emptyset) &= \emptyset \\
\mathsf{Dom}(\Sigma, a{:}K) &= \mathsf{Dom}(\Sigma) \cup \{a\} \\
\mathsf{Dom}(\Sigma, c{:}\sigma) &= \mathsf{Dom}(\Sigma) \cup \{c\}
\end{aligned}$$

the domain of a context $\mathsf{Dom}(\Gamma)$ as follows:

$$\begin{aligned}
\mathsf{Dom}(\emptyset) &= \emptyset \\
\mathsf{Dom}(\Gamma, x{:}\sigma) &= \mathsf{Dom}(\Gamma) \cup \{x\}
\end{aligned}$$

the free variables of a term $\mathsf{Fv}(T)$ as follows:

$$\begin{aligned}
\mathsf{Fv}(\mathsf{Type}) = \mathsf{Fv}(a) = \mathsf{Fv}(c) &= \emptyset \\
\mathsf{Fv}(x) &= \{x\} \\
\mathsf{Fv}(\Pi x{:}\sigma.T) &= (\mathsf{Fv}(\sigma) \cup \mathsf{Fv}(T)) \backslash \{x\} \\
\mathsf{Fv}(\lambda x{:}\sigma.T) &= (\mathsf{Fv}(\sigma) \cup \mathsf{Fv}(T)) \backslash \{x\} \\
\mathsf{Fv}(T\ N) &= \mathsf{Fv}(T) \cup \mathsf{Fv}(N) \\
\mathsf{Fv}(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T]) &= \mathsf{Fv}(N) \cup \mathsf{Fv}(\sigma) \cup \mathsf{Fv}(T) \\
\mathsf{Fv}(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[T]) &= \mathsf{Fv}(N) \cup \mathsf{Fv}(\sigma) \cup \mathsf{Fv}(T)
\end{aligned}$$

the bound variables of a term $\mathsf{Bv}(T)$ as follows:

$$\begin{aligned}
\mathsf{Bv}(\mathsf{Type}) = \mathsf{Bv}(a) = \mathsf{Bv}(c) = \mathsf{Bv}(x) &= \emptyset \\
\mathsf{Bv}(\Pi x{:}\sigma.T) &= \mathsf{Bv}(\sigma) \cup \mathsf{Bv}(T) \cup \{x\} \\
\mathsf{Bv}(\lambda x{:}\sigma.T) &= \mathsf{Bv}(\sigma) \cup \mathsf{Bv}(T) \cup \{x\} \\
\mathsf{Bv}(T\ N) &= \mathsf{Bv}(T) \cup \mathsf{Bv}(N) \\
\mathsf{Bv}(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T]) &= \mathsf{Bv}(N) \cup \mathsf{Bv}(\sigma) \cup \mathsf{Bv}(T) \\
\mathsf{Bv}(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[T]) &= \mathsf{Bv}(N) \cup \mathsf{Bv}(\sigma) \cup \mathsf{Bv}(T)
\end{aligned}$$

and, finally, substitution is defined as follows (here, we reiterate that $x \neq y$, and that we are working modulo Barendregt's variable condition):

$$\begin{aligned}
\mathsf{Type}[M/x] &= \mathsf{Type} \\
a[M/x] &= a \\
c[M/x] &= c \\
x[M/x] &= M \\
(\Pi y{:}\sigma.T)[M/x] &= \Pi y{:}\sigma[M/x].T[M/x] \\
(\lambda y{:}\sigma.T)[M/x] &= \lambda y{:}\sigma[M/x].T[M/x] \\
(T\ N)[M/x] &= T[M/x]\ N[M/x] \\
(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T])[M/x] &= \mathcal{L}^{\mathcal{P}}_{N[M/x],\sigma[M/x]}[T[M/x]] \\
(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[T])[M/x] &= \mathcal{U}^{\mathcal{P}}_{N[M/x],\sigma[M/x]}[T[M/x]]
\end{aligned}$$

6.1 The $\mathsf{LF}_{\mathcal{P}}$ Type System

The type system for $\mathsf{LF}_{\mathcal{P}}$, presented in detail below, proves judgements of the shape:

	Σ	sig	Σ is a valid signature
	$\vdash_\Sigma$	Γ	Γ is a valid context in Σ
Γ	$\vdash_\Sigma$	K	K is a kind in Γ and Σ
Γ	$\vdash_\Sigma$	$\sigma : K$	σ has kind K in Γ and Σ
Γ	$\vdash_\Sigma$	$M : \sigma$	M has type σ in Γ and Σ

Signatures, Contexts, Kinds. Signatures, contexts, and kinds of $\mathsf{LF}_{\mathcal{P}}$ are formed in the usual manner:

$$\frac{}{\emptyset\ \mathsf{sig}} \quad (\mathsf{S{\cdot}Empty})$$ An empty signature is a valid signature.

$$\frac{\Sigma\ \mathsf{sig} \quad \vdash_\Sigma K \quad a \notin \mathsf{Dom}(\Sigma)}{\Sigma, a{:}K\ \mathsf{sig}} \quad (\mathsf{S{\cdot}C{\cdot}Fam})$$ A valid signature Σ can be extended with a fresh family a, whose kind K is a kind in the empty context and signature Σ.

$$\frac{\Sigma\ \mathsf{sig} \quad \vdash_\Sigma \sigma{:}\mathsf{Type} \quad c \notin \mathsf{Dom}(\Sigma)}{\Sigma, c{:}\sigma\ \mathsf{sig}} \quad (\mathsf{S{\cdot}C{\cdot}Obj})$$ A valid signature Σ can be extended with a fresh object c, whose type σ has kind Type in the empty context and signature Σ.

$$\frac{\Sigma\ \mathsf{sig}}{\vdash_\Sigma \emptyset} \quad (\mathsf{C{\cdot}Empty})$$ An empty context is a valid context in any valid signature Σ.

$$\frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma \sigma{:}\mathsf{Type} \quad x \notin \mathsf{Dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}\sigma} \quad (\mathsf{C{\cdot}Var})$$ A valid context Γ in the signature Σ can be extended with a fresh variable x, whose type is of kind Type in Γ and Σ.

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathsf{Type}} \quad (\mathsf{K{\cdot}Type})$$ If Γ is a valid context in the signature Σ, then Type is a kind in Γ and Σ.

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.K} \quad (\mathsf{K{\cdot}Prod})$$ If K is a kind in the context $\Gamma, x{:}\sigma$ and signature Σ, then the dependent product $\Pi x{:}\sigma.K$ is a kind in Γ and Σ.

Families. As for the families, we have chosen to omit the abstraction rule, as we have found little practical use of it, compounded with the fact that the proofs of the properties of the system would have been slightly more complicated in its presence. On the other hand, we have introduced a rule allowing for the creation of locked types.

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_\Sigma a : K} \quad (\mathsf{F{\cdot}Const})$$ If Γ is a valid context in the signature Σ, then any family a of kind K belonging to Σ also has kind K in Γ and Σ.

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.\tau : \mathsf{Type}} \quad (\mathsf{F{\cdot}Prod})$$ If τ has kind Type in the context $\Gamma, x{:}\sigma$ and signature Σ, then the dependent product $\Pi x{:}\sigma.\tau$ has kind Type in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma \sigma : \Pi x{:}\tau.K \quad \Gamma \vdash_\Sigma N : \tau}{\sigma\, N : K[N/x]} \quad (\mathsf{F{\cdot}App})$$

If σ has kind $\Pi x{:}\tau.K$ in the context Γ and signature Σ, and N has type τ in Γ and Σ, then the application of N to σ has kind K, in which all occurrences of x have been substituted for N, in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma \rho : \mathsf{Type} \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}} \quad (\mathsf{F{\cdot}Lock})$$

If ρ has kind K in the context Γ and signature Σ, and N has type σ in Γ and Σ, then the type locking ρ with a predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N : \sigma$ has kind Type in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma \sigma : K \quad \Gamma \vdash_\Sigma K' \quad K{=}_{\beta\mathcal{L}}K'}{\Gamma \vdash_\Sigma \sigma : K'} \quad (\mathsf{F{\cdot}Conv})$$

If σ has kind K in the context Γ and signature Σ, and K is definitionally equal to K', which is a kind in Γ and Σ, then σ also has kind K' in Γ and Σ.

Objects. As for the object rules, we take those of LF in their entirety, and add one more rule addressing construction of locked terms, and one more rule addressing the unlocking of such locked terms.

$$\frac{\vdash_\Sigma \Gamma \quad c{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma c : \sigma} \quad (\mathsf{O{\cdot}Const})$$

If Γ is a valid context in the signature Σ, then any object c of type σ belonging to Σ also has type σ in Γ and Σ.

$$\frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma} \quad (\mathsf{O{\cdot}Var})$$

If Γ is a valid context in the signature Σ, then any variable x of type σ belonging to Γ also has type σ in Γ and Σ.

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau} \quad (\mathsf{O{\cdot}Abs})$$

If M has type τ in the context $\Gamma, x{:}\sigma$ and signature Σ, then the abstraction $\lambda x{:}\sigma.M$ has type $\Pi x{:}\sigma.\tau$ in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma M : \Pi x{:}\sigma.\tau \quad \Gamma \vdash_\Sigma N : \tau}{M\, N : \tau[N/x]} \quad (\mathsf{O{\cdot}App})$$

If M has type $\Pi x{:}\sigma.\tau$ in the context Γ and signature Σ, and N has type σ in Γ and Σ, then the application of N to M has type τ, in which all occurrences of x have been substituted for N, in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma M : \rho \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]} \quad (\mathsf{O{\cdot}Lock})$$

If M has type ρ in the context Γ and signature Σ, and N has type σ in Γ and Σ, then M, locked with the predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N : \sigma$ has type ρ, locked with the predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N : \sigma$, in Γ and Σ.

$$\frac{\begin{array}{l}\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma) \\ \Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]\end{array} \qquad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho} \quad (\mathsf{O{\cdot}Unlock})$$

If M has type ρ, locked with the predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N{:}\sigma$ in the context Γ and signature Σ, N has type σ in Γ and Σ, and $\mathcal{P}(\Gamma \vdash_\Sigma N{:}\sigma)$ holds, then M, unlocked with $\mathcal{P}$ on $\Gamma \vdash_\Sigma N{:}\sigma$ has type ρ in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma M : \sigma \quad \Gamma \vdash_\Sigma \sigma' : \mathsf{Type} \quad \sigma{=}_{\beta\mathcal{L}}\sigma'}{\Gamma \vdash_\Sigma M : \sigma'} \quad (\mathsf{O{\cdot}Conv})$$

If M has type σ in the context Γ and signature Σ, and σ is definitionally equal to σ', which has kind Type in Γ and Σ, then M also has type σ' in Γ and Σ.

In $\mathsf{LF}_{\mathcal{P}}$, we consider only the terms obtained after a finite number of application of the typing rules of the system. In such terms, each symbol (such as a constant, a variable, a lock, or an unlock) can appear only a finite number of times. Also, we denote by $\Gamma \vdash_\Sigma \alpha$ any typing judgement, be it $\Gamma \vdash_\Sigma T : T'$ or $\Gamma \vdash_\Sigma T$. In the two latter judgements, T will be referred to as the *subject* of that judgement.

6.2 $\beta\mathcal{L}$-reduction and Definitional Equality in $\mathsf{LF}_{\mathcal{P}}$

In $\mathsf{LF}_{\mathcal{P}}$, we have two types of reduction. The first is the standard β-reduction, while the second is a novel form of reduction, which we call $\mathcal{L}$-reduction ("lock-reduction"). $\mathcal{L}$-reduction, essentially, serves as a lock-releasing mechanism, erasing the $\mathcal{U}$-$\mathcal{L}$ pair in a term of the form $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]]$, thus effectively releasing M. Together, these two reductions combine into $\beta\mathcal{L}$-reduction, denoted by $\to_{\beta\mathcal{L}}$, and this combined reduction is the one which we take into account when considering the properties of $\mathsf{LF}_{\mathcal{P}}$. The main one-step $\beta\mathcal{L}$-reduction rules are presented in Figure 6.2. There, one can notice the new rule ($\mathcal{L}$·O·Main), which is the reduction rule illustrating the desired behavior of the lock and unlock combined - the effective release of a lock by an unlock, i.e. the unlock destructor canceling out the lock constructor. This reduction rule, together with the (O·Unlock) and (O·Lock) typing rules, provides an elegant mechanism for locking and unlocking objects. The reader can note that a similar reduction rule at the level of types is not required, as the application of the unlock destructor to a term automatically unlocks its type, as ensured by the (O·Unlock) rule.

$$(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x] \quad (\beta\cdot\mathsf{O}\cdot\mathsf{Main}) \quad \text{Standard } \beta\text{-reduction}$$

$$\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M \quad (\mathcal{L}\cdot\mathsf{O}\cdot\mathsf{Main}) \quad \text{A } \textit{lock} \text{ dissolves in the presence of an } \textit{unlock}.$$

Figure 6.2: Main one-step-$\beta\mathcal{L}$-reduction rules in $\mathsf{LF}_{\mathcal{P}}$

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\Pi x{:}\sigma.\tau \to_{\beta\mathcal{L}} \Pi x{:}\sigma'.\tau}\ (\mathsf{F}\cdot\Pi_1\cdot\beta\mathcal{L}) \qquad \frac{\tau \to_{\beta\mathcal{L}} \tau'}{\Pi x{:}\sigma.\tau \to_{\beta\mathcal{L}} \Pi x{:}\sigma.\tau'}\ (\mathsf{F}\cdot\Pi_2\cdot\beta\mathcal{L})$$

$$\frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\sigma\, N \to_{\beta\mathcal{L}} \sigma'\, N}\ (\mathsf{F}\cdot\mathsf{A}_1\cdot\beta\mathcal{L}) \qquad \frac{N \to_{\beta\mathcal{L}} N'}{\sigma\, N \to_{\beta\mathcal{L}} \sigma\, N'}\ (\mathsf{F}\cdot\mathsf{A}_2\cdot\beta\mathcal{L})$$

$$\frac{N \to_{\beta\mathcal{L}} N'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N',\sigma}[\rho]}\ (\mathsf{F}\cdot\mathcal{L}_1\cdot\beta\mathcal{L}) \qquad \frac{\sigma \to_{\beta\mathcal{L}} \sigma'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho]}\ (\mathsf{F}\cdot\mathcal{L}_2\cdot\beta\mathcal{L})$$

$$\frac{\rho \to_{\beta\mathcal{L}} \rho'}{\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho']}\ (\mathsf{F}\cdot\mathcal{L}_3\cdot\beta\mathcal{L})$$

Figure 6.3: $\beta\mathcal{L}$-closure-under-context for families of $\mathsf{LF}_{\mathcal{P}}$

The rules for one-step closure under context for families are presented in Figure 6.3, while the corresponding rules for kinds and terms are handled analogously, and are omitted. Furthermore, we will use the $\twoheadrightarrow_{\beta\mathcal{L}}$ symbol to denote the reflexive and transitive closure of $\to_{\beta\mathcal{L}}$. Finally, we introduce $\beta\mathcal{L}$-definitional equality in the standard way, as the reflexive, symmetric, and transitive closure of $\beta\mathcal{L}$-reduction on kinds, families, and objects, as illustrated in Figure 6.4.

$$\frac{T \to_{\beta\mathcal{L}} T'}{T =_{\beta\mathcal{L}} T'} \quad (\beta\mathcal{L}\cdot\mathsf{Eq}\cdot\mathsf{Main}) \qquad \frac{}{T =_{\beta\mathcal{L}} T} \quad (\beta\mathcal{L}\cdot\mathsf{Eq}\cdot\mathsf{Refl})$$

$$\frac{T =_{\beta\mathcal{L}} T'}{T' =_{\beta\mathcal{L}} T} \quad (\beta\mathcal{L}\cdot\mathsf{Eq}\cdot\mathsf{Sym}) \qquad \frac{T =_{\beta\mathcal{L}} T' \qquad T' =_{\beta\mathcal{L}} T''}{T =_{\beta\mathcal{L}} T''} \quad (\beta\mathcal{L}\cdot\mathsf{Eq}\cdot\mathsf{Trans})$$

Figure 6.4: $\beta\mathcal{L}$-definitional equality in $\mathsf{LF}_{\mathcal{P}}$

6.3 Several Comments on the $\mathsf{LF}_{\mathcal{P}}$ Typing System

In the $\mathsf{LF}_{\mathcal{P}}$ typing system, the external predicates appear only as preconditions to the application of the rule ($\mathsf{O}\cdot\mathsf{Unlock}$). In this way, we keep the predicates separate from $\beta\mathcal{L}$-reduction, in the sense that their validity is not checked at reduction-time, but rather during the construction of the unlock destructors. In this way, we do not require any conditions to be placed on the predicates when proving the confluence of $\mathsf{LF}_{\mathcal{P}}$, but only for Subject Reduction (*see* Section 7.3). An alternative approach would be to omit the unlock destructor from the object formation rules altogether, and delegate the checking of the validity of predicates to the reduction rules instead, via rules in the style of:

$$\begin{array}{lll} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \rho, & \text{if } \mathcal{P}(\Gamma \vdash_{\Sigma} N : \sigma) & (\mathsf{L}\cdot\mathsf{F}\cdot\mathsf{Main}\cdot\mathsf{Alt}), \\ \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} M, & \text{if } \mathcal{P}(\Gamma \vdash_{\Sigma} N : \sigma) & (\mathsf{L}\cdot\mathsf{O}\cdot\mathsf{Main}\cdot\mathsf{Alt}). \end{array}$$

Another interesting alternative approach would be to make the $\beta\mathcal{L}$-reduction typed, utilizing rules such as, for example:

$$\frac{\Gamma \vdash_{\Sigma} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \quad \mathcal{P}(\Gamma \vdash_{\Sigma} N : \sigma)}{\Gamma \vdash_{\Sigma} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]}$$

In this way, $\beta\mathcal{L}$-reduction would be fully incorporated into the typing system, as in [Honsell 2008b], making the proofs a little easier and a lot more uniform. However, this approach would also require us to introduce abstraction back not only into the families, but into kinds as well, which would be somewhat superfluous, as we have not found any practical usage for them in $\mathsf{LF}_{\mathcal{P}}$.

6.4 Summary

In this chapter, we have first presented in detail the judgments and the typing system of $\mathsf{LF}_{\mathcal{P}}$. Next, we have introduced and discussed the notion of $\mathcal{L}$-reduction, a new type of reduction with which we achieve the desired behaviour of a lock, in the presence of an unlock, shown how this new reduction is combined with β-reduction, and introduced corresponding rules for closure-under-context, as well as the notion of definitional equality. Finally, we have provided several comments regarding possible alternative ideas on integrating the checking of the validity of an external predicate as a part of a Logical Framework.

Chapter 7

Properties of $\mathsf{LF}_{\mathcal{P}}$

Contents

In this chapter, we present and prove the main properties of $\mathsf{LF}_{\mathcal{P}}$. Without any additional assumptions concerning predicates, we are able to prove that the type system is strongly normalizing and confluent. The former follows from strong normalization of LF (see [Harper 1993]), while the latter follows from strong normalization and local confluence, using Newman's Lemma [Newman 1942]. The proof of Subject Reduction, however, is more complicated and does require certain conditions to be placed on the predicates. These conditions are summarized in the following definition of what we term to be *well-behaved predicates*:

Definition 21 (Well-behaved predicates)**.** *A finite set of predicates* $\{\mathcal{P}_i\}_{i\in I}$ *is* well-behaved *if each* $\mathcal{P}$ *in the set satisfies the following conditions:*

Closure under signature and context weakening and permutation:

1. *If* Σ *and* Ω *are valid signatures,* $\Sigma \subseteq \Omega$*, and* $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$ *holds, then* $\mathcal{P}(\Gamma \vdash_\Omega \alpha)$ *also holds.*
2. *If* Γ *and* Δ *are valid contexts,* $\Gamma \subseteq \Delta$*, and* $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$ *holds, then* $\mathcal{P}(\Delta \vdash_\Sigma \alpha)$ *also holds.*

Closure under substitution: *If* $\mathcal{P}(\Gamma, x{:}\sigma', \Gamma' \vdash_\Sigma N : \sigma)$ *holds, and* $\Gamma \vdash_\Sigma N' : \sigma'$*, then* $\mathcal{P}(\Gamma, \Gamma'[N'/x] \vdash_\Sigma N[N'/x] : \sigma[N'/x])$ *also holds.*

Closure under reduction:

1. *If* $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ *holds, and* $N \to_{\beta\mathcal{L}} N'$ *holds, then* $\mathcal{P}(\Gamma \vdash_\Sigma N' : \sigma)$ *also holds.*
2. *If* $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ *holds, and* $\sigma \to_{\beta\mathcal{L}} \sigma'$ *holds, then* $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$ *also holds.*

7.1 Strong Normalization

In this section, we prove that $\mathsf{LF}_{\mathcal{P}}$ is strongly normalizing with respect to $\beta\mathcal{L}$-reduction. For this, we rely on the strong normalization of LF, as proven in [Harper 1993]. First, we introduce the function $^{-\mathcal{UL}} : \mathsf{LF}_{\mathcal{P}} \to \mathsf{LF}$, which maps $\mathsf{LF}_{\mathcal{P}}$ terms into LF terms by deleting the $\mathcal{L}$ and $\mathcal{U}$ symbols. The proof then proceeds by contradiction. We assume that there exists a term T with an infinite $\beta\mathcal{L}$-reduction sequence. Next, we prove that only a finite number of β-reductions can be performed within any given $\mathsf{LF}_{\mathcal{P}}$ term T. From this, we deduce that, in order for T to

have an infinite $\beta\mathcal{L}$-reduction sequence, it must have an infinite $\mathcal{L}$-sequence, which we show to be impossible, obtaining the desired contradiction.

Therefore, let us begin with the definition of the function ${}^{-\mathcal{UL}} : \mathsf{LF}_{\mathcal{P}} \to \mathsf{LF}$:

1. $\mathsf{Type}^{-\mathcal{UL}} = \mathsf{Type}$. $a^{-\mathcal{UL}} = a$. $c^{-\mathcal{UL}} = c$. $x^{-\mathcal{UL}} = x$.
2. $(\Pi x{:}\sigma.T)^{-\mathcal{UL}} = \Pi x{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}}$.
3. $(\lambda x{:}\sigma.T)^{-\mathcal{UL}} = \lambda x{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}}$.
4. $(T\,M)^{-\mathcal{UL}} = T^{-\mathcal{UL}}\,M^{-\mathcal{UL}}$.
5. $(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T])^{-\mathcal{UL}} = (\lambda x_f{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}})\,N^{-\mathcal{UL}}$.
6. $(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[T])^{-\mathcal{UL}} = (\lambda x_f{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}})\,N^{-\mathcal{UL}}$.

where x_f is a variable which *does not* have free occurrences in T. Its purpose is to preserve the N and σ, which appear in the subscript of the $\mathcal{L}$ and $\mathcal{U}$ symbols, while still being able to β-reduce to T in one step, which is in line with the main purpose of ${}^{-\mathcal{UL}}$, *i.e.* the deletion of locks and unlocks from an $\mathsf{LF}_{\mathcal{P}}$ term. We can naturally extend ${}^{-\mathcal{UL}}$ to signatures and contexts of $\mathsf{LF}_{\mathcal{P}}$, obtaining signatures and contexts of LF:

$$
\begin{aligned}
(\emptyset)^{-\mathcal{UL}} &= \emptyset \\
(\Sigma, a{:}K)^{-\mathcal{UL}} &= \Sigma^{-\mathcal{UL}}, a^{-\mathcal{UL}}{:}K^{-\mathcal{UL}} \\
(\Sigma, c{:}\sigma)^{-\mathcal{UL}} &= \Sigma^{-\mathcal{UL}}, c^{-\mathcal{UL}}{:}\sigma^{-\mathcal{UL}} \\
(\emptyset)^{-\mathcal{UL}} &= \emptyset \\
(\Gamma, x{:}\sigma)^{-\mathcal{UL}} &= \Gamma^{-\mathcal{UL}}, x^{-\mathcal{UL}}{:}\sigma^{-\mathcal{UL}}
\end{aligned}
$$

and then to judgements of $\mathsf{LF}_{\mathcal{P}}$, obtaining judgements of LF:

$$
\begin{aligned}
(\Sigma\ \mathsf{sig})^{-\mathcal{UL}} &= \Sigma^{-\mathcal{UL}}\ \mathsf{sig} \\
(\vdash_\Sigma \Gamma)^{-\mathcal{UL}} &= \vdash_{\Sigma^{-\mathcal{UL}}} \Gamma^{-\mathcal{UL}} \\
(\Gamma \vdash_\Sigma K)^{-\mathcal{UL}} &= \Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} K^{-\mathcal{UL}} \\
(\Gamma \vdash_\Sigma \sigma : K)^{-\mathcal{UL}} &= \Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \sigma^{-\mathcal{UL}} : K^{-\mathcal{UL}} \\
(\Gamma \vdash_\Sigma M : \sigma)^{-\mathcal{UL}} &= \Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}} : \sigma^{-\mathcal{UL}}
\end{aligned}
$$

With ${}^{-\mathcal{UL}}$ defined in this way, we have the following three proposition:

Proposition 1 (Connecting $\to_{\beta\mathcal{L}}$ in $\mathsf{LF}_{\mathcal{P}}$, $\twoheadrightarrow_\beta$ in LF, and ${}^{-\mathcal{UL}}$)**.**

1. *If* $K \to_{\beta\mathcal{L}} K'$ *in* $\mathsf{LF}_{\mathcal{P}}$*, then* $K^{-\mathcal{UL}} \twoheadrightarrow_\beta K'^{-\mathcal{UL}}$ *in* LF.
2. *If* $\sigma \to_{\beta\mathcal{L}} \sigma'$ *in* $\mathsf{LF}_{\mathcal{P}}$*, then* $\sigma^{-\mathcal{UL}} \twoheadrightarrow_\beta \sigma'^{-\mathcal{UL}}$ *in* LF.
3. *If* $M \to_{\beta\mathcal{L}} M'$ *in* $\mathsf{LF}_{\mathcal{P}}$*, then* $M^{-\mathcal{UL}} \twoheadrightarrow_\beta M'^{-\mathcal{UL}}$ *in* LF.

Proposition 2 (Connecting $=_{\beta\mathcal{L}}$ in $\mathsf{LF}_{\mathcal{P}}$, $=_\beta$ in LF, and ${}^{-\mathcal{UL}}$)**.**

1. *If* $K =_{\beta\mathcal{L}} K'$ *in* $\mathsf{LF}_{\mathcal{P}}$*, then* $K^{-\mathcal{UL}} =_\beta K'^{-\mathcal{UL}}$ *in* LF.
2. *If* $\sigma =_{\beta\mathcal{L}} \sigma'$ *in* $\mathsf{LF}_{\mathcal{P}}$*, then* $\sigma^{-\mathcal{UL}} =_\beta \sigma'^{-\mathcal{UL}}$ *in* LF.
3. *If* $M =_{\beta\mathcal{L}} M'$ *in* $\mathsf{LF}_{\mathcal{P}}$*, then* $M^{-\mathcal{UL}} =_\beta M'^{-\mathcal{UL}}$ *in* LF.

Proposition 3. $^{-\mathcal{UL}}$ *maps derivable judgements of* $\mathsf{LF}_{\mathcal{P}}$ *into derivable judgements of* LF.

Proof. All three of these propositions are proven simultaneously, by induction on the structure of the derivation of the reduction, the structure of the derivation of the equivalence, and the structure of the derivation of the judgement. Here, we will present the relevant cases, while the remaining ones are handled similarly.

- For Proposition 1, let $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N',\sigma}[\rho]$ from $N \to_{\beta\mathcal{L}} N'$, using the rule ($\mathsf{F}{\cdot}\mathcal{L}_1{\cdot}\beta\mathcal{L}$). From the induction hypothesis (IH), we have that $N^{-\mathcal{UL}} \twoheadrightarrow_{\beta} N'^{-\mathcal{UL}}$, while the goal we are looking for is $(\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}} \twoheadrightarrow_{\beta} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N'^{-\mathcal{UL}}$, and this follows immediately from the IH, and the rules for closure under context for β-reduction in LF.

- For Proposition 1, let $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$, using the rule ($\beta{\cdot}\mathsf{O}{\cdot}\mathsf{Main}$). Here, we have that the goal is $(\lambda x{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} \twoheadrightarrow_{\beta\mathcal{L}} M^{-\mathcal{UL}}[N^{-\mathcal{UL}}/x]$, which is, in fact, an instance of the main one-step β-reduction in LF.

- For Proposition 1, let $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M$, using the rule ($\mathcal{L}{\cdot}\mathsf{O}{\cdot}\mathsf{Main}$). We have the goal $(\lambda x_f{:}\sigma^{-\mathcal{UL}}.(\lambda y_f{:}\sigma^{-\mathcal{UL}}.M)\, N)\, N \twoheadrightarrow_{\beta} M^{-\mathcal{UL}}$, which we obtain by applying the main one-step β-reduction rule of LF, bearing in mind the nature of the choice of x_f and y_f.

- For Proposition 2, let us have that $K{=}_{\beta\mathcal{L}}K'$, from $K \to_{\beta\mathcal{L}} K'$, using the rule ($\beta\mathcal{L}{\cdot}\mathsf{Eq}{\cdot}\mathsf{Main}$). From the IH for Proposition 1, we have that $K^{-\mathcal{UL}} \twoheadrightarrow_{\beta} K'^{-\mathcal{UL}}$ in LF, from which we obtain that $K^{-\mathcal{UL}}{=}_{\beta}K'^{-\mathcal{UL}}$ in LF.

- For Proposition 3, let us have that $\Gamma \vdash_{\Sigma} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$, from $\Gamma \vdash_{\Sigma} \rho : \mathsf{Type}$ and $\Gamma \vdash_{\Sigma} N : \sigma$, using the rule ($\mathsf{F}{\cdot}\mathsf{Lock}$). From the IH, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \rho^{-\mathcal{UL}} : \mathsf{Type}$, and $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} N^{-\mathcal{UL}} : \sigma^{-\mathcal{UL}}$, while the goal is $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \mathsf{Type}$. From the Subderivation property of LF, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \sigma^{-\mathcal{UL}} : \mathsf{Type}$, and, given that we can, without loss of generality, assume that $x_f \notin \mathsf{Dom}(\Gamma^{-\mathcal{UL}})$, we obtain, using the Weakening property of LF, that $\Gamma^{-\mathcal{UL}}, x_f : \sigma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \rho^{-\mathcal{UL}} : \mathsf{Type}$, and, from there, that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}} : \Pi x_f{:}\sigma^{-\mathcal{UL}}.\mathsf{Type}$. Now, by using the application formation rule of LF, we obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \mathsf{Type}[N^{-\mathcal{UL}}/x_f]$. However, this is our claim, as, since no substitutions can occur in Type, we have that $\mathsf{Type}[N^{-\mathcal{UL}}/x_f] \equiv \mathsf{Type}$.

- For Proposition 3, let us have that $\Gamma \vdash_{\Sigma} \sigma : K'$, from $\Gamma \vdash_{\Sigma} \sigma : K$, $\Gamma \vdash_{\Sigma} K$, and $K{=}_{\beta\mathcal{L}}K'$, using the rule ($\mathsf{F}{\cdot}\mathsf{Conv}$). From the IHs, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \sigma^{-\mathcal{UL}} : K^{-\mathcal{UL}}$, $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} K'^{-\mathcal{UL}}$, and $K{=}_{\beta}K'$, from which we obtain our goal, $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \sigma^{-\mathcal{UL}} : K'^{-\mathcal{UL}}$, by using the LF conversion rule for families.

- For Proposition 3, let us have that $\Gamma \vdash_{\Sigma} \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, from $\Gamma \vdash_{\Sigma} M : \rho$ and $\Gamma \vdash_{\Sigma} N : \sigma$, using the rule ($\mathsf{O}{\cdot}\mathsf{Lock}$). From the IH, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}$ and $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} N^{-\mathcal{UL}} : \sigma^{-\mathcal{UL}}$, while the goal is $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : (\lambda y_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$. First, as earlier, we can obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \lambda x_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}} : \Pi x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}}$. Now, by using the application formation rule of LF, we obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}[N^{-\mathcal{UL}}/x_f]$. However, what we actually have is that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda x_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}$, as, by the choice of x_f, we have that $\rho^{-\mathcal{UL}}[N/x_f] \equiv \rho^{-\mathcal{UL}}$. Also, in a similar manner as before, we can obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda y_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \mathsf{Type}$. Now, as, by the choice of y_f, we have that $\rho^{-\mathcal{UL}}{=}_{\beta}(\lambda y_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$, we can use the LF conversion rule for objects to obtain our claim.

- For Proposition 3, let us have that $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho$, from $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, $\Gamma \vdash_\Sigma N : \sigma$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, using the rule ($\mathsf{O{\cdot}Unlock}$). From the IH, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}} : (\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$ and $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} N^{-\mathcal{UL}} : \sigma^{-\mathcal{UL}}$, while the goal which we would like to prove is $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} (\lambda y_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}$. Similarly to the previous item, we have that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} \rho^{-\mathcal{UL}} : \mathsf{Type}$ and that $(\lambda x_f{:}\sigma^{-\mathcal{UL}}.\rho^{-\mathcal{UL}})\, N^{-\mathcal{UL}} =_\beta \rho^{-\mathcal{UL}}$, and we can use the LF conversion rule for objects to obtain that $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}} : \rho^{-\mathcal{UL}}$. Finally, as we have that $M^{-\mathcal{UL}} =_\beta (\lambda y_f{:}\sigma^{-\mathcal{UL}}.M^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$, we obtain the desired claim by using the Subject Reduction property of LF.

□

As a consequence of Proposition 3, we have that the function $^{-\mathcal{UL}}$ maps well-typed terms of $\mathsf{LF}_{\mathcal{P}}$ into well-typed terms of LF. Next, we will denote the maximum number of β-reductions which can be executed in a given (either LF- or $\mathsf{LF}_{\mathcal{P}}$-) term T as $\max_\beta(T)$. Now, we can notice that $\mathcal{L}$-reductions cannot create entirely new β-redexes, but can only "unlock" potential β-redexes, *i.e.* expressions of the form $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{P}_{N,\sigma}[\lambda x{:}\tau.M]]\, T$, arriving at $\lambda x{:}\tau.M\, T$, which is a β-redex. Also, this resulting β-redex will be present in $(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{P}_{N,\sigma}[\lambda x{:}\tau.M]]\, T)^{-\mathcal{UL}}$. Therefore, we have that, for any $\mathsf{LF}_{\mathcal{P}}$-term T, it holds that $\max_\beta(T) \leq \max_\beta(T^{-\mathcal{UL}})$. As LF is strongly normalizing, we have that $\max_\beta(T^{-\mathcal{UL}})$ is finite, therefore forcing $\max_\beta(T)$ into being finite, leading to the following proposition:

Proposition 4. *Only finitely many β-reductions can occur within the maximal reduction sequence of any $\mathsf{LF}_{\mathcal{P}}$-term. There is no $\mathsf{LF}_{\mathcal{P}}$-term T with an infinite number of β-reductions in its maximal reduction sequence.*

Next, we notice that any $\mathsf{LF}_{\mathcal{P}}$-term has only finitely many $\mathcal{L}$-redexes before any reductions take place, and that this number can only be increased through β-reductions, and only by a finite amount per β-reduction. However, if we were to have an $\mathsf{LF}_{\mathcal{P}}$-term T which has an infinite reduction sequence, then within this sequence, there would need to be infinitely many $\mathcal{L}$-reductions, since, due to Proposition 4, the number of β-reductions in this sequence has to be finite. On the other hand, with the number of β-reductions in the sequence being finite, it would not be possible to reach infinitely many $\mathcal{L}$-reductions, and such a term T cannot exist in $\mathsf{LF}_{\mathcal{P}}$. Therefore, we have the Strong Normalization theorem:

Theorem 18 (Strong normalization of $\mathsf{LF}_{\mathcal{P}}$)**.**

1. *If $\Gamma \vdash_\Sigma K$, then K is $\beta\mathcal{L}$-strongly normalizing.*
2. *if $\Gamma \vdash_\Sigma \sigma : K$, then σ is $\beta\mathcal{L}$-strongly normalizing.*
3. *if $\Gamma \vdash_\Sigma M : \sigma$, then M is $\beta\mathcal{L}$-strongly normalizing.*

7.2 Confluence

Since $\beta\mathcal{L}$-reduction is strongly normalizing, in order to prove the confluence of the system, by Newman's Lemma, it is sufficient to show that the reduction on "raw terms" is *locally confluent*. First, we need a substitution lemma, the proof of which is routine:

Lemma 32 (Substitution lemma for local confluence)**.**

1. *If $N \to_{\beta\mathcal{L}} N'$, then $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M[N'/x]$.*
2. *If $M \to_{\beta\mathcal{L}} M'$, then $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$.*

Next, we proceed to prove local confluence:

Lemma 33 (Local confluence of $\mathsf{LF}_{\mathcal{P}}$)**.** *$\beta\mathcal{L}$-reduction is locally confluent, i.e. :*

1. *If $K \to_{\beta\mathcal{L}} K'$ and $K \to_{\beta\mathcal{L}} K''$, then there exists a K''', such that $K' \twoheadrightarrow_{\beta\mathcal{L}} K'''$ and $K'' \twoheadrightarrow_{\beta\mathcal{L}} K'''$.*

2. *If $\sigma \to_{\beta\mathcal{L}} \sigma'$ and $\sigma \to_{\beta\mathcal{L}} \sigma''$, then there exists a σ''', such that $\sigma' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$ and $\sigma'' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$.*

3. *If $M \to_{\beta\mathcal{L}} M'$ and $M \to_{\beta\mathcal{L}} M''$, then there exists a M''', such that $M' \twoheadrightarrow_{\beta\mathcal{L}} M'''$ and $M'' \twoheadrightarrow_{\beta\mathcal{L}} M'''$.*

Proof. By simultaneous induction on the two derivations $T \to_{\beta\mathcal{L}} T'$ and $T \to_{\beta\mathcal{L}} T''$. All the cases for T kind or family, as well as most of the cases for T object are proven directly using the induction hypotheses. Therefore, we will limit ourselves here to illustrating only the cases involving base reduction rules:

1. Let us have, by the base reduction rule (β·O·Main), that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$. Let us also have that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} (\lambda x{:}\sigma'.M)\, N$, from $\sigma \to_{\beta\mathcal{L}} \sigma'$, by the appropriate closure-under-context reduction rules. In this case, we will show that the required conditions are met for $M''' \equiv M[N/x]$. Indeed, by the definition of the relation $\twoheadrightarrow_{\beta\mathcal{L}}$, we have that $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M[N/x]$, and also, by the reduction rule (β·O·Main), we have that $(\lambda x{:}\sigma'.M)\, N \to_{\beta\mathcal{L}} M[N/x]$, effectively having $(\lambda x{:}\sigma'.M)\, N \twoheadrightarrow_{\beta\mathcal{L}} M[N/x]$.

2. Let us have, by the base reduction rule (β·O·Main), that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$. Let us also have that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} (\lambda x{:}\sigma.M')\, N$, from $M \to_{\beta\mathcal{L}} M'$, by the appropriate closure-under-context reduction rules. In this case, we will show that the required conditions are met for $M''' \equiv M'[N/x]$. By the reduction rule (β·O·Main), we have that $(\lambda x{:}\sigma.M')\, N \to_{\beta\mathcal{L}} M'[N/x]$, from which we obtain $(\lambda x{:}\sigma.M')\, N \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$, while we obtain that $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$ from part 2 of Lemma 32.

3. Let us have, by the base reduction rule (β·O·Main)), that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$. Let us also have that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} (\lambda x{:}\sigma.M)\, N'$, from $N \to_{\beta\mathcal{L}} N'$, by the appropriate closure-under-context reduction rule. In this case, we will show that the required conditions are met for $M''' \equiv M[N'/x]$. By the reduction rule (β·O·Main), we have that $(\lambda x{:}\sigma.M)\, N' \to_{\beta\mathcal{L}} M[N'/x]$, from which we obtain $(\lambda x{:}\sigma.M)\, N' \twoheadrightarrow_{\beta\mathcal{L}} M'[N/x]$, while we obtain that $M[N/x] \twoheadrightarrow_{\beta\mathcal{L}} M[N'/x]$ from part 1 of Lemma 32.

4. Let us have, by the base reduction rule ($\mathcal{L}$·O·Main), that $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M$, and let us also have that $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]]$, from $N \to_{\beta\mathcal{L}} N'$, by the appropriate closure-under-context reduction rule. In this case, we will show that the required conditions are met for $M''' \equiv M$. By the definition of $\twoheadrightarrow_{\beta\mathcal{L}}$, we have that $M \twoheadrightarrow_{\beta\mathcal{L}} M$, which leaves us with needing to show that $\mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \twoheadrightarrow_{\beta\mathcal{L}} M$. This we obtain by the following sequence of reductions: from $N \to_{\beta\mathcal{L}} N'$, which we have as an induction hypothesis, using the appropriate closure-under-context reduction rules, we obtain that $\mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N',\sigma}[M]]$, from which we finally obtain that $\mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N',\sigma}[M]] \to_{\beta\mathcal{L}} M$, by the reduction rule ($\mathcal{L}$·O·Main), showing that $\mathcal{U}^{\mathcal{P}}_{N',\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \twoheadrightarrow_{\beta\mathcal{L}} M$. The remaining subcases are handled very similarly.

□

Having proven local confluence, finally, from Theorem 18, Lemma 33 and Newman's Lemma, we obtain the confluence theorem for $\mathsf{LF}_{\mathcal{P}}$:

Theorem 19 (Confluence of $\mathsf{LF}_{\mathcal{P}}$). *$\beta\mathcal{L}$-reduction is confluent, meaning that:*

1. *If $K \twoheadrightarrow_{\beta\mathcal{L}} K'$ and $K \twoheadrightarrow_{\beta\mathcal{L}} K''$, then there exists a K''', such that $K' \twoheadrightarrow_{\beta\mathcal{L}} K'''$ and $K'' \twoheadrightarrow_{\beta\mathcal{L}} K'''$.*
2. *If $\sigma \twoheadrightarrow_{\beta\mathcal{L}} \sigma'$ and $\sigma \twoheadrightarrow_{\beta\mathcal{L}} \sigma''$, then there exists a σ''', such that $\sigma' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$ and $\sigma'' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$.*
3. *If $M \twoheadrightarrow_{\beta\mathcal{L}} M'$ and $M \twoheadrightarrow_{\beta\mathcal{L}} M''$, then there exists a M''', such that $M' \twoheadrightarrow_{\beta\mathcal{L}} M'''$ and $M'' \twoheadrightarrow_{\beta\mathcal{L}} M'''$.*

7.3 Subject Reduction

We begin by proving several auxiliary lemmas and propositions:

Proposition 5 (Inversion properties)**.**

1. *If $\Pi x{:}\sigma.T =_{\beta\mathcal{L}} T''$, then $T'' \equiv \Pi x{:}\sigma.'T'$, for some σ', T', such that $\sigma' =_{\beta\mathcal{L}} \sigma$, and $T' =_{\beta\mathcal{L}} T$.*
2. *If $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] =_{\beta\mathcal{L}} \theta$, then $\theta \equiv \mathcal{L}^{\mathcal{P}}_{N',\sigma'}[\rho']$, for some N', σ', and ρ', such that $N' =_{\beta\mathcal{L}} N$, $\sigma' =_{\beta\mathcal{L}} \sigma$, and $\rho' =_{\beta\mathcal{L}} \rho$.*
3. *If $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, then $\Gamma \vdash_\Sigma M : \rho$.*
4. *If $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau$, then $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau$.*

Proof. The first two items are proven directly, by inspection of the rules for $\beta\mathcal{L}$-closure-under-context for kinds and types, while the third item is proven by using the rule (F·Lock), and the second item. The fourth property follows directly from the typing and conversion rules. □

By induction on the structure of the derivation, independently of Proposition 5, we have:

Proposition 6 (Subderivation, part 1)**.**

1. *A derivation of $\vdash_\Sigma \emptyset$ has a subderivation of Σ sig.*
2. *A derivation of $\Sigma, a{:}K$ sig has subderivations of Σ sig and $\vdash_\Sigma K$.*
3. *A derivation of $\Sigma, f{:}\sigma$ sig has subderivations of Σ sig and $\vdash_\Sigma \sigma{:}\mathsf{Type}$.*
4. *A derivation of $\vdash_\Sigma \Gamma, x{:}\sigma$ has subderivations of Σ sig, $\vdash_\Sigma \Gamma$, and $\Gamma \vdash_\Sigma \sigma{:}\mathsf{Type}$.*
5. *A derivation of $\Gamma \vdash_\Sigma \alpha$ has subderivations of Σ sig and $\vdash_\Sigma \Gamma$.*
6. *Given a derivation $\mathcal{D}$ of the judgement $\Gamma \vdash_\Sigma \alpha$, and a subterm occurring in the subject of this judgement, there exists a derivation of a judgement having this subterm as a subject.*

From this point on, we will assume that the first requirement for the well-behavedness of predicates, namely the *closure under signature and context weakening and permutation*, holds. With this, we prove the following propositions by induction on the structure of the derivation:

Proposition 7 (Weakening and permutation)**.** *If predicates are closed under signature/context weakening and permutation, then:*

1. *If Σ and Ω are valid signatures, and every declaration occurring in Σ also occurs in Ω, then $\Gamma \vdash_\Sigma \alpha$ implies $\Gamma \vdash_\Omega \alpha$.*
2. *If Γ and Δ are valid contexts w.r.t. the signature Σ, and every declaration occurring in Γ also occurs in Δ, then $\Gamma \vdash_\Sigma \alpha$ implies $\Delta \vdash_\Sigma \alpha$.*

Proposition 8 (Subderivation, part 2). *If predicates are closed under signature/context weakening and permutation, then:*

1. *If* $\Gamma \vdash_\Sigma \sigma : K$, *then* $\Gamma \vdash_\Sigma K$.
2. *If* $\Gamma \vdash_\Sigma M : \sigma$, *then* $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$.

From this point on, we will assume that the second requirement for the well-behavedness of predicates, the *closure under cut*, holds as well. We prove the following propositions by induction on the structure of the derivation:

Proposition 9 (Transitivity). *If predicates are closed under signature/context weakening and permutation and under substitution, then: if* $\Gamma, x{:}\sigma, \Gamma' \vdash_\Sigma \alpha$, *and* $\Gamma \vdash_\Sigma N : \sigma$, *then* $\Gamma, \Gamma'[N/x] \vdash_\Sigma \alpha[N/x]$.

Notice that, contrary to what happens in traditional type systems, the following *closure under expansion* does not hold: $\Gamma \vdash_\Sigma M[N/x] : \tau \implies \Gamma \vdash_\Sigma (\lambda x{:}\sigma.M)N : \tau$, for $\Gamma \vdash_\Sigma N : \sigma$.

Proposition 10 (Unicity of types and kinds). *If predicates are closed under signature/context weakening and permutation and under substitution, then:*

1. *If* $\Gamma \vdash_\Sigma \sigma : K_1$ *and* $\Gamma \vdash_\Sigma \sigma : K_2$, *then* $\Gamma \vdash_\Sigma K_1 =_{\beta\mathcal{L}} K_2$.
2. *If* $\Gamma \vdash_\Sigma M : \sigma_1$ *and* $\Gamma \vdash_\Sigma M : \sigma_2$, *then* $\Gamma \vdash_\Sigma \sigma_1 =_{\beta\mathcal{L}} \sigma_2$.

Finally, for Subject Reduction, we require that the third requirements for the well-behavedness of predicates, namely the *closure under definitional equality*, also holds:

Theorem 20 (Subject reduction of $\mathsf{LF}_\mathcal{P}$). *If predicates are well-behaved, then:*

1. *If* $\Gamma \vdash_\Sigma K$, *and* $K \to_{\beta\mathcal{L}} K'$, *then* $\Gamma \vdash_\Sigma K'$.
2. *If* $\Gamma \vdash_\Sigma \sigma : K$, *and* $\sigma \to_{\beta\mathcal{L}} \sigma'$, *then* $\Gamma \vdash_\Sigma \sigma' : K$.
3. *If* $\Gamma \vdash_\Sigma M : \sigma$, *and* $M \to_{\beta\mathcal{L}} M'$, *then* $\Gamma \vdash_\Sigma M' : \sigma$.

Proof. Here, we prove Subject Reduction of a slightly extended type system. We consider the type system in which the rules (F·Lock), (O·Lock), and (O·Unlock) all have an additional premise $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$, while the rule (O·Unlock) also has another additional premise $\Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho] : \mathsf{Type}$, as shown in Figure 7.1.

$$\frac{\Gamma \vdash_\Sigma \rho : \mathsf{Type} \qquad \Gamma \vdash_\Sigma N : \sigma \qquad \Gamma \vdash_\Sigma \sigma : \mathsf{Type}}{\Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho] : \mathsf{Type}} \quad (\mathsf{F{\cdot}Lock})$$

$$\frac{\Gamma \vdash_\Sigma M : \rho \qquad \Gamma \vdash_\Sigma N : \sigma \qquad \Gamma \vdash_\Sigma \sigma : \mathsf{Type}}{\Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[M] : \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]} \quad (\mathsf{O{\cdot}Lock})$$

$$\frac{\begin{array}{lll} \Gamma \vdash_\Sigma N : \sigma & & \Gamma \vdash_\Sigma M : \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho] \\ \Gamma \vdash_\Sigma \sigma : \mathsf{Type} & \Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho] : \mathsf{Type} & \mathcal{P}(\Gamma \vdash_\Sigma N : \sigma) \end{array}}{\Gamma \vdash_\Sigma \mathcal{U}^\mathcal{P}_{N,\sigma}[M] : \rho} \quad (\mathsf{O{\cdot}Unlock})$$

Figure 7.1: An extension of $\mathsf{LF}_\mathcal{P}$ typing rules for Subject Reduction

The proof proceeds by simultaneous induction on the derivation of $\Gamma \vdash_\Sigma M$ and $M \to_{\beta\mathcal{L}} M'$. Here we will show only the case in which the base reduction rule (β·O·Main) is used, and one of the cases for which the well-behavedness of predicates is a requirement, while the other cases are handled similarly, mostly by using the induction hypotheses.

1. We have that $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M\, N : \tau[N/x]$, by the rule (O·App), from $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau$, and $\Gamma \vdash_\Sigma N : \sigma$, and that $(\lambda x{:}\sigma.M)\, N \to_{\beta\mathcal{L}} M[N/x]$ by the rule (β·O·Main). From Proposition 5, we get that $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau$, and from this and $\Gamma \vdash_\Sigma N : \sigma$, we obtain the required $\Gamma \vdash_\Sigma M[N/x] : \tau[N/x]$, by an application of Proposition 9.

2. We have that $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] : \rho$, by the rule (O·Unlock), from $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, $\Gamma \vdash_\Sigma N : \sigma$, $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, and that $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M$ by the rule ($\mathcal{L}$·O·Main). Here, we obtain the required $\Gamma \vdash_\Sigma M : \rho$ directly, using the third item of Proposition 5.

3. We have that $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho$, by the rule (O·Unlock), from $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$, $\Gamma \vdash_\Sigma N : \sigma$, $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, and that $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \to_{\beta\mathcal{L}} \mathcal{U}^{\mathcal{P}}_{N,\sigma'}[M]$, by the reduction rules for closure-under-context, from $\sigma \to_{\beta\mathcal{L}} \sigma'$. First, from the induction hypothesis we have that $\Gamma \vdash_\Sigma \sigma' : \mathsf{Type}$, and we also have, from $\sigma \to_\beta \sigma'$, that $\sigma =_{\beta\mathcal{L}} \sigma'$. From this, using $\Gamma \vdash_\Sigma N : \sigma$, and the rule (O·Conv), we obtain that $\Gamma \vdash_\Sigma N : \sigma'$. Next, since $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$ could only have been obtained by the type system rule (F·*Lock*), from $\Gamma \vdash_\Sigma \rho : \mathsf{Type}$ and $\Gamma \vdash_\Sigma N : \sigma$, and since we have $\Gamma \vdash_\Sigma N : \sigma'$, we obtain that $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho] : \mathsf{Type}$. From this, given $\sigma =_{\beta\mathcal{L}} \sigma'$, we obtain that $\mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho] \to_{\beta\mathcal{L}} \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho]$, and since we already have that $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, we can use the type system rule (O·Conv) to obtain $\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma'}[\rho]$. Finally, by the well-behavedness requirements for the predicates, we have that $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$ holds, and we can now use the type system rule (O·Unlock) to obtain the required $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma'}[M] : \rho$. Here, we can notice that there are steps in this proof (in which we obtain $\Gamma \vdash_\Sigma \sigma' : \mathsf{Type}$, and $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}$), which could not have been made had the original system not been extended for this theorem.

Now, we can prove straightforwardly that $\Gamma \vdash_\Sigma \alpha$ in the extended system *iff* $\Gamma \vdash_\Sigma \alpha$ in the original $\mathsf{LF}_{\mathcal{P}}$ system (*i.e.* that the judgements that these two systems derive are the same), by induction on the length of the derivation. With this, given that we have proven Subject Reduction of the extended system, we have that Subject Reduction also holds in the original $\mathsf{LF}_{\mathcal{P}}$ system. □

7.4 A Word on the Expressive Power of $\mathsf{LF}_{\mathcal{P}}$

Various natural questions arise as to the expressive power of $\mathsf{LF}_{\mathcal{P}}$. We outline the answers to some of them:

- $\mathsf{LF}_{\mathcal{P}}$ is decidable, if the predicates are decidable; this can be proven as usual.
- If a predicate is *definable in* LF, *i.e.* it can be encoded via the inhabitability of a suitable LF dependent type, then it is well-behaved in the sense of Definition 21.
- All well-behaved recursively enumerable predicates are LF-definable by Church's thesis. Of course, the issue is then on how "deep" the encoding is. To give a more precise answer, we would need a more accurate definition of "deep" and "shallow" encodings, which we still lack. This part of the thesis can be seen as a stepping stone towards such a theory, with the approach taken here being "shallow" by definition, and the encodings via Church's

thesis being potentially very, very deep. Consider *e.g.* the well-behaved predicate "M, N are two different closed normal forms", which can be immediately expressed in $\mathsf{LF}_{\mathcal{P}}$.

- One may wonder what is the relation between the LF encodings of, say, Modal Logics, discussed in [Avron 1998, Crary 2010], and the encodings which appear in this thesis (see Chapter 9). The former essentially correspond to the internal encoding of the predicates that are utilized in Chapter 9. In fact, one could express the mapping between the two signatures as a forgetful functor going from $\mathsf{LF}_{\mathcal{P}}$ judgements to LF judgements.

- Finally, we can say that, as far as decidable predicates, $\mathsf{LF}_{\mathcal{P}}$ is morally a *conservative extension* of LF. Of course, pragmatically, it is very different, in that it allows for the neat factoring-out of the true logical contents of derivations from the mere effective verification of other, *e.g.* syntactical or structural properties. A feature of this approach is that of making such a separation explicit.

- The main advantage of having externally verified predicates amounts to a smoother encoding (the signature is not cluttered by auxiliary notions and mechanisms needed to implement the predicate). This allows performance optimization, if the external system used to verify the predicate is an optimized tool specifically designed for the issue at hand (*e.g.* analytic tableaux methods for propositional formulæ).

7.5 Summary

In this chapter, we have shown that $\mathsf{LF}_{\mathcal{P}}$ satisfies all of the main desired meta-theoretic properties: strong normalization, confluence, subject reduction, and decidability. Strong normalization and confluence are proven without any additional assumptions on the nature of the external predicates. In order for subject reduction to hold, the predicates need to abide by certain well-behavedness conditions, while for the decidability of $\mathsf{LF}_{\mathcal{P}}$, they also need to be decidable.

Chapter 8

The Canonical $\mathsf{LF}_{\mathcal{P}}$ Framework

Contents

In this chapter, we present a *canonical* version of $\mathsf{LF}_{\mathcal{P}}$, which we will be denoting by $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, and which has been executed in the style of [Watkins 2002] and [Harper 2007]. Given that there exists another form of reduction ($\mathcal{L}$-reduction) beside β-reduction, it was necessary to extend the standard η-rule with the clause $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]] \rightarrow_{\eta} M$, corresponding to the lock type constructor. The syntax of the $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ system defines what the *normal forms* of $\mathsf{LF}_{\mathcal{P}}$ are, while the typing system captures all of the judgements that are in *η-long normal form* which are derivable in $\mathsf{LF}_{\mathcal{P}}$. The main reason for the development of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ is that it will serve as the basis for proving the adequacy of the encodings which will be presented later, in Chapter 9. As will be seen, contrary to standard LF, *not all* of the judgements derivable in $\mathsf{LF}_{\mathcal{P}}$ admit a corresponding η-long normal form. In fact, this is not the case when the predicates appearing in the $\mathsf{LF}_{\mathcal{P}}$ judgement are not satisfied in the given context. Nonetheless, although $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ is not closed under full η-expansion, it is still powerful enough for one to be able to obtain all of the relevant adequacy results.

8.1 Syntax and Type System for $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$

In $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, the families and objects have been divided into two categories: atomic and canonical, as shown in Figure 8.1.

$$
\begin{array}{rcl@{\qquad}rcll}
\alpha & \in & \mathcal{A} & \alpha & ::= & a \mid \alpha\, N & \textit{Atomic Families} \\
\sigma, \tau, \rho & \in & \mathcal{F} & \sigma & ::= & \alpha \mid \Pi x{:}\sigma.\tau \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] & \textit{Canonical Families} \\
A & \in & \mathcal{A}_o & A & ::= & c \mid x \mid A\, M \mid \mathcal{U}^{\mathcal{P}}_{N,\sigma}[A] & \textit{Atomic Objects} \\
M, N & \in & \mathcal{O} & M & ::= & A \mid \lambda x{:}\sigma.M \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] & \textit{Canonical Objects}
\end{array}
$$

Figure 8.1: Syntax of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$

The corresponding type system for $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, together with the first three judgments of $\mathsf{LF}_{\mathcal{P}}$ (see Section 6.1), proves judgements of the shape:

$$
\begin{array}{lll}
\Gamma \vdash_{\Sigma} \sigma\ \mathsf{Type} & & \sigma \text{ is a canonical family in } \Gamma \text{ and } \Sigma \\
\Gamma \vdash_{\Sigma} \alpha \Rightarrow K & & K \text{ is the kind of the atomic family } \alpha \text{ in } \Gamma \text{ and } \Sigma \\
\Gamma \vdash_{\Sigma} M \Leftarrow \sigma & & M \text{ is a canonical term of type } \sigma \text{ in } \Gamma \text{ and } \Sigma \\
\Gamma \vdash_{\Sigma} A \Rightarrow \sigma & & \sigma \text{ is the type of the atomic term } A \text{ in } \Gamma \text{ and } \Sigma
\end{array}
$$

The type system itself is presented in detail below, with the note that the we have chosen to omit the formation rules and judgements related to signatures, contexts, and kinds, since they are the same in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ as they are in $\mathsf{LF}_{\mathcal{P}}$.

Atomic Families. The atomic families contain constant types and the dependent application in types. The rule in charge of handling the application makes use of the notion of hereditary substitution, which computes the normal form resulting from the substitution of one normal form into another, and which will be presented later.

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_\Sigma a \Rightarrow K} \quad (\mathsf{A{\cdot}Const})$$

If Γ is a valid context in the signature Σ, then any family a of kind K belonging to Σ is an atomic family with kind K in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma \alpha \Rightarrow \Pi x{:}\sigma.K_1 \quad \Gamma \vdash_\Sigma M \Leftarrow \sigma \quad K_1[M/x]^K_\sigma = K}{\Gamma \vdash_\Sigma \alpha\, M \Rightarrow K} \quad (\mathsf{A{\cdot}App})$$

If the atomic family α has kind $\Pi x{:}\tau.K_1$ in the context Γ and signature Σ, and M is a canonical term of type σ in Γ and Σ, then the application of M to σ is an atomic family, and has kind K_1, in which all occurrences of x have been hereditarily substituted for M, in Γ and Σ.

Canonical Families. The remaining family constructors, for the dependent product and the locked types, as well as for the subsumption of atomic families, belong to the canonical families.

$$\frac{\Gamma \vdash_\Sigma \alpha \Rightarrow \mathsf{Type}}{\Gamma \vdash_\Sigma \alpha\ \mathsf{Type}} \quad (\mathsf{F{\cdot}Atom})$$

If α is an atomic family of kind Type in the context Γ and signature Σ, then α is also a canonical family in Γ and Σ.

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \tau\ \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.\tau\ \mathsf{Type}} \quad (\mathsf{F{\cdot}Prod})$$

If τ is a canonical family in the context $\Gamma, x{:}\sigma$ and signature Σ, then the dependent product $\Pi x{:}\sigma.\tau$ is also a canonical family in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma \rho\ \mathsf{Type} \quad \Gamma \vdash_\Sigma N \Leftarrow \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]\ \mathsf{Type}} \quad (\mathsf{F{\cdot}Lock})$$

If ρ is a canonical family in in the context Γ and signature Σ, and N is a canonical term of type σ in Γ and Σ, then the type $\mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]$, locking ρ with a predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N \Leftarrow \sigma$, is also a canonical family in Γ and Σ.

Atomic Objects. The atomic objects comprise object constants and variables, as well as the application and unlock constructors.

$$\frac{\vdash_\Sigma \Gamma \quad c{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma c \Rightarrow \sigma} \quad (\mathsf{O{\cdot}Const})$$

If Γ is a valid context in the signature Σ, then any object c of type σ in Σ is an atomic term with type σ in Γ and Σ.

$$\frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x \Rightarrow \sigma} \quad (\mathsf{O{\cdot}Var})$$

If Γ is a valid context in the signature Σ, then any variable x of type σ in Γ is an atomic term with type σ in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma A \Rightarrow \Pi x{:}\sigma.\tau_1 \quad \Gamma \vdash_\Sigma M \Leftarrow \sigma \quad \tau_1[M/x]^F_\sigma = \tau}{A\, M \Rightarrow \tau} \quad (\mathsf{O{\cdot}App})$$

If A is an atomic term with type $\Pi x{:}\sigma.\tau_1$ in the context Γ and signature Σ, and M is a canonical term of type σ in Γ and Σ, then the application of M to A has type τ_1, in which all occurrences of x have been hereditarily substituted for M, in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma A \Rightarrow \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho] \quad \Gamma \vdash_\Sigma N \Leftarrow \sigma \quad \mathcal{P}(\Gamma \vdash_\Sigma N \Leftarrow \sigma)}{\Gamma \vdash_\Sigma \mathcal{U}^\mathcal{P}_{N,\sigma}[A] \Rightarrow \rho} \quad (\mathsf{O{\cdot}Unlock})$$

If M is an atomic term of type ρ, locked with the predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N \Leftarrow \sigma$ in the context Γ and signature Σ, N is a canonical term of type σ in Γ and Σ, and $\mathcal{P}(\Gamma \vdash_\Sigma N{:}\sigma)$ holds, then M, unlocked with $\mathcal{P}$ on $\Gamma \vdash_\Sigma N{:}\sigma$ is an atomic term of type ρ in Γ and Σ.

$$\frac{}{(a)^- = a} \qquad \frac{(\alpha)^- = \rho}{(\alpha M)^- = \rho} \qquad \frac{(\sigma_2)^- = \rho_2 \quad (\sigma)^- = \rho}{(\Pi x{:}\sigma_2.\sigma)^- = \rho_2 \to \rho} \qquad \frac{(\sigma')^- = \rho'}{(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\sigma'])^- = \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho']}$$

Figure 8.2: Erasure to simple types

Canonical Objects. Canonical objects, similarly to canonical families, comprise abstraction and locking, as well as the subsumption of atomic objects.

$$\frac{\Gamma \vdash_\Sigma A \Rightarrow \alpha}{\Gamma \vdash_\Sigma A \Leftarrow \alpha} \quad (\mathsf{O{\cdot}Atom})$$

If Γ is a valid context in the signature Σ, then any atomic object c of type σ belonging to Σ is also a canonical object of type σ in Γ and Σ.

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma M \Leftarrow \tau}{\Gamma \vdash_\Sigma \lambda x{:}\sigma.M \Leftarrow \Pi x{:}\sigma.\tau} \quad (\mathsf{O{\cdot}Abs})$$

If M is a canonical term of type τ in the context $\Gamma, x{:}\sigma$ and signature Σ, then the abstraction $\lambda x{:}\sigma.M$ is a canonical term of type $\Pi x{:}\sigma.\tau$ in Γ and Σ.

$$\frac{\Gamma \vdash_\Sigma M \Leftarrow \rho \quad \Gamma \vdash_\Sigma N \Leftarrow \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \Leftarrow \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]} \quad (\mathsf{O{\cdot}Lock})$$

If M is a canonical term of type ρ in the context Γ and signature Σ, and N is a canonical term of type σ in Γ and Σ, then M, locked with the predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N \Leftarrow \sigma$ is a canonical term of type ρ, locked with the predicate $\mathcal{P}$ on $\Gamma \vdash_\Sigma N \Leftarrow \sigma$, in Γ and Σ.

In $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, the predicates are defined on judgements $\Gamma \vdash_\Sigma M \Leftarrow \sigma$, while the type system makes use, in the rules (A·App) and (O·App), of the notion of *hereditary substitution*. The general form of the hereditary substitution judgement is $T[M/x]^m_\rho = T'$, where M is the term substituted, x is the variable substituted for, T is the term substituted into, T' is the result of the substitution, ρ is the *simple type* of M, and m is the syntactic category involved in the judgement (*i.e.* kinds, atomic/canonical families, atomic/canonical objects, contexts). The simple type ρ of M is obtained via the erasure function (Figure 8.2), which maps dependent into simple types, while the rules for the hereditary substitution judgement appear in Figure 8.3 and Figure 8.4.

Substitution in Kinds

$$\frac{}{\mathsf{Type}[M_0/x_0]^K_{\rho_0} = \mathsf{Type}} \quad (\mathcal{S}{\cdot}\mathsf{K{\cdot}Type}) \qquad \frac{\sigma[M_0/x_0]^F_{\rho_0} = \sigma' \quad K[M_0/x_0]^K_{\rho_0} = K'}{(\Pi x{:}\sigma.K)[M_0/x_0]^K_{\rho_0} = \Pi x{:}\sigma'.K'} \quad (\mathcal{S}{\cdot}\mathsf{K{\cdot}Pi})$$

Substitution in Atomic Families

$$\frac{}{a[M_0/x_0]^f_{\rho_0} = a} \quad (\mathcal{S}{\cdot}\mathsf{F{\cdot}Const}) \qquad \frac{\alpha[M_0/x_0]^f_{\rho_0} = \alpha' \quad M[M_0/x_0]^O_{\rho_0} = M'}{(\alpha M)[M_0/x_0]^f_{\rho_0} = \alpha' M'} \quad (\mathcal{S}{\cdot}\mathsf{F{\cdot}App})$$

Substitution in Canonical Families

$$\frac{\alpha[M_0/x_0]^f_{\rho_0} = \alpha'}{\alpha[M_0/x_0]^F_{\rho_0} = \alpha'} \quad (\mathcal{S}{\cdot}\mathsf{F{\cdot}Atom}) \qquad \frac{\sigma_1[M_0/x_0]^F_{\rho_0} = \sigma_1' \quad \sigma_2[M_0/x_0]^F_{\rho_0} = \sigma_2'}{(\Pi x{:}\sigma_1.\sigma_2)[M_0/x_0]^F_{\rho_0} = \Pi x{:}\sigma_1'.\sigma_2'} \quad (\mathcal{S}{\cdot}\mathsf{F{\cdot}Pi})$$

$$\frac{\sigma_1[M_0/x_0]^F_{\rho_0} = \sigma_1' \quad M_1[M_0/x_0]^O_{\rho_0} = M_1' \quad \sigma_2[M_0/x_0]^F_{\rho_0} = \sigma_2'}{\mathcal{L}^{\mathcal{P}}_{M_1,\sigma_1}[\sigma_2][M_0/x_0]^F_{\rho_0} = \mathcal{L}^{\mathcal{P}}_{M_1',\sigma_1'}[\sigma_2']} \quad (\mathcal{S}{\cdot}\mathsf{F{\cdot}Lock})$$

Figure 8.3: Hereditary substitution, kinds and families

Substitution in Atomic Objects

$$\frac{}{c[M_0/x_0]^o_{\rho_0} = c}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{Const})$$

$$\frac{}{x_0[M_0/x_0]^o_{\rho_0} = M_0 : \rho_0}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{Var}\cdot\mathsf{H}) \qquad \frac{x \neq x_0}{x[M_0/x_0]^o_{\rho_0} = x}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{Var})$$

$$\frac{A_1[M_0/x_0]^o_{\rho_0} = \lambda x{:}\rho_2.M_1' : \rho_2 \to \rho \quad M_2[M_0/x_0]^O_{\rho_0} = M_2' \quad M_1'[M_2'/x]^O_{\rho_2} = M'}{(A_1 M_2)[M_0/x_0]^o_{\rho_0} = M' : \rho}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{App}\cdot\mathsf{H})$$

$$\frac{A_1[M_0/x_0]^o_{\rho_0} = A_1' \quad M_2[M_0/x_0]^O_{\rho_0} = M_2'}{(A_1 M_2)[M_0/x_0]^o_{\rho_0} = A_1' M_2'}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{App})$$

$$\frac{\sigma[M_0/x_0]^F_{\rho_0} = \sigma' \quad M[M_0/x_0]^O_{\rho_0} = M' \quad A[M_0/x_0]^o_{\rho_0} = \mathcal{L}^\mathcal{P}_{M',\sigma'}[M_1 : \mathcal{L}^\mathcal{P}_{M',\sigma'}[\rho]]}{\mathcal{U}^\mathcal{P}_{M,\sigma}[A][M_0/x_0]^o_{\rho_0} = M_1 : \rho}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{Unlock}\cdot\mathsf{H})$$

$$\frac{\sigma[M_0/x_0]^F_{\rho_0} = \sigma' \quad M[M_0/x_0]^O_{\rho_0} = M' \quad A[M_0/x_0]^O_{\rho_0} = A'}{\mathcal{U}^\mathcal{P}_{M,\sigma}[A][M_0/x_0]^o_{\rho_0} = \mathcal{U}^\mathcal{P}_{M',\sigma'}[A']}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{Unlock})$$

Substitution in Canonical Objects

$$\frac{A[M_0/x_0]^o_{\rho_0} = A'}{A[M_0/x_0]^O_{\rho_0} = A'}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{R}) \qquad \frac{A[M_0/x_0]^o_{\rho_0} = M' : \rho}{A[M_0/x_0]^O_{\rho_0} = M'}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{R}\cdot\mathsf{H})$$

$$\frac{M[M_0/x_0]^O_{\rho_0} = M'}{\lambda x{:}\sigma.M[M_0/x_0]^O_{\rho_0} = \lambda x{:}\sigma.M'}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{Abs})$$

$$\frac{\sigma_1[M_0/x_0]^F_{\rho_0} = \sigma_1' \quad M_1[M_0/x_0]^O_{\rho_0} = M_1' \quad M_2[M_0/x_0]^O_{\rho_0} = M_2'}{\mathcal{L}^\mathcal{P}_{M_1,\sigma_1}[M_2][M_0/x_0]^O_{\rho_0} = \mathcal{L}^\mathcal{P}_{M_1',\sigma_1'}[M_2']}\ (\mathcal{S}\cdot\mathsf{O}\cdot\mathsf{Lock})$$

Substitution in Contexts

$$\frac{}{\cdot[M_0/x_0]^C_{\rho_0} = \cdot}\ (\mathcal{S}\cdot\mathsf{Ctxt}\cdot\mathsf{Empty})$$

$$\frac{x_0 \neq x \quad x \notin \mathsf{Fv}(M_0) \quad \Gamma[M_0/x_0]^C_{\rho_0} = \Gamma' \quad \sigma[M_0/x_0]^F_{\rho_0} = \sigma'}{\Gamma, x{:}\sigma[M_0/x_0]^C_{\rho_0} = \Gamma', x{:}\sigma'}\ (\mathcal{S}\cdot\mathsf{Ctxt}\cdot\mathsf{Term})$$

Figure 8.4: Hereditary substitution, objects and contexts

Notice that, in the rule (O·Atom) of the type system, the syntactic restriction of the classifier to α being atomic ensures that canonical forms are indeed *η-long normal forms* for a suitable notion of η-long normal form, which extends the standard one for locked types. To illustrate, the judgement $x : \Pi z{:}a.a \vdash_\Sigma x \Leftarrow \Pi z{:}a.a$ is not derivable, as $\Pi z{:}a.a$ is not atomic, hence $\vdash_\Sigma \lambda x{:}(\Pi z{:}a.a).x \Leftarrow \Pi x{:}(\Pi z{:}a.a).\Pi z{:}a.a$ is not derivable. However, $\vdash_\Sigma \lambda x{:}(\Pi z{:}a.a).\lambda y{:}a.xy \Leftarrow \Pi x{:}(\Pi z{:}a.a).\,\Pi z{:}a.a$, where a is a family constant of kind Type, is derivable. Analogously, for locked types, the judgement $x : \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho] \vdash_\Sigma x \Leftarrow \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]$ is not derivable, since $\mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]$ is not atomic. As a consequence, $\vdash_\Sigma \lambda x{:}\mathcal{L}^\mathcal{P}_{N,\sigma}[\rho].x \Leftarrow \Pi x{:}\mathcal{L}^\mathcal{P}_{N,\sigma}[\rho].\mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]$ is not derivable. However, $x{:}\mathcal{L}^\mathcal{P}_{N,\sigma}[\rho] \vdash_\Sigma \mathcal{L}^\mathcal{P}_{N,\sigma}[\mathcal{U}^\mathcal{P}_{N,\sigma}[x]] \Leftarrow \mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]$ is derivable, if ρ is atomic and the predicate $\mathcal{P}$ holds on $\Gamma \vdash_\Sigma N \Leftarrow \sigma$. Hence $\vdash_\Sigma \lambda x{:}\mathcal{L}^\mathcal{P}_{N,\sigma}[\rho].\mathcal{L}^\mathcal{P}_{N,\sigma}[\mathcal{U}^\mathcal{P}_{N,\sigma}[x]] \Leftarrow \Pi x{:}\mathcal{L}^\mathcal{P}_{N,\sigma}[\rho].\mathcal{L}^\mathcal{P}_{N,\sigma}[\rho]$ is derivable. In Definition 24 below, we formalize the notion of η-expansion of a judgement, together with correspondence theorems between LF$_\mathcal{P}$ and LF$^\mathcal{C}_\mathcal{P}$.

8.2 Properties of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$

We start by studying basic properties of hereditary substitution and the type system. The following four lemmas and two theorems are proven directly, by either inspecting the rules for hereditary substitution and the type system for $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, and/or performing induction on the appropriate derivation or the complexity of the judgement. As the details of the proofs are purely technical, we have chosen to omit them. But, first of all, we provide a definition of well-behavedness for predicate, much like that present in $\mathsf{LF}_{\mathcal{P}}$.

Definition 22 (Well-behaved $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$-predicates). *A finite set of predicates $\{\mathcal{P}_i\}_{i\in I}$ is* well-behaved *in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ if each $\mathcal{P}_i$ is* closed under signature and context weakening, permutation, *and* hereditary substitution, *in the sense of Definition 21.*

Lemma 34 (Decidability of hereditary substitution).

1. *For any T in $\{\mathcal{K},\mathcal{A},\mathcal{F},\mathcal{O},\mathcal{C}\}$, and any M, x, and ρ, either there exists a T' such that $T[M/x]^m_\rho = T'$ or there is no such T'.*
2. *For any M, x, ρ, and A, either there exists an A' such that $A[M/x]^o_\rho = A'$, or there exist M' and ρ', such that $A[M/x]^o_\rho = M' : \rho'$, or there are no such A' or M'.*

Lemma 35 (Head substitution size). *If $A[M_0/x_0]^o_{\rho_0} = M : \rho$, then ρ is a subexpression of ρ_0.*

Lemma 36 (Uniqueness of substitution and synthesis).

1. *It is not possible that $A[M_0/x_0]^o_{\rho_0} = A'$ and $A[M_0/x_0]^o_{\rho_0} = M : \rho$.*
2. *For any T, if $T[M_0/x_0]^m_{\rho_0} = T'$, and $T[M_0/x_0]^m_{\rho_0} = T''$, then $T' = T''$.*
3. *If $\Gamma \vdash_\Sigma \alpha \Rightarrow K$, and $\Gamma \vdash_\Sigma \alpha \Rightarrow K'$, then $K = K'$.*
4. *If $\Gamma \vdash_\Sigma A \Rightarrow \sigma$, and $\Gamma \vdash_\Sigma A \Rightarrow \sigma'$, then $\sigma = \sigma'$.*

Lemma 37 (Composition of hereditary substitution). *Let us assume that $x \neq x_0$ and $x \neq M_0$. Then, we have that:*

1. *If $M_2[M_0/x_0]^O_{\rho_0} = M_2'$, $T_1[M_2/x]^m_{\rho_2} = T_1'$, and $T_1[M_0/x_0]^m_{\rho_0} = T_1''$, then there exists a T, such that $T_1'[M_0/x_0]^m_{\rho_0} = T$, and $T_1''[M_2/x]^m_{\rho_2} = M' : \rho$.*
2. *If $M_2[M_0/x_0]^O_{\rho_0} = M_2'$, $A_1[M_2/x]^o_{\rho_2} = M : \rho$, and $A_1[M_0/x_0]^o_{\rho_0} = A$, then there exists an M', such that $M[M_0/x_0]^O_{\rho_0} = M'$, and $A[M_2/x]^o_{\rho_2} = M' : \rho$.*
3. *If $M_2[M_0/x_0]^O_{\rho_0} = M_2'$, $A_1[M_2/x]^o_{\rho_2} = A$, and $A_1[M_0/x_0]^o_{\rho_0} = M : \rho$, then there exists an M', such that $A[M_0/x_0]^o_{\rho_0} = M' : \rho$, and $M[M_2/x]^O_{\rho_2} = M'$.*

Theorem 21 (Transitivity). *Let Σ sig, $\vdash_\Sigma \Gamma_L, x_0{:}\rho_0, \Gamma_R$ and $\Gamma_L \vdash_\Sigma M_0 \Leftarrow \rho_0$, and let us assume that all of the predicates are well-behaved. Then*

1. *There exists Γ_R' such that $[M_0/x_0]^C_{\rho_0} = \Gamma_R'$ and $\vdash_\Sigma \Gamma_L, \Gamma_R'$.*
2. *If $\Gamma_L, x_0{:}\rho_0, \Gamma_R \vdash_\Sigma K$ then there exists K' such that $[M_0/x_0]^K_{\rho_0} K = K'$ and $\Gamma_L, \Gamma_R' \vdash_\Sigma K'$.*
3. *If $\Gamma_L, x_0{:}\rho_0, \Gamma_R \vdash_\Sigma \sigma$ Type, then there exists σ' such that $[M_0/x_0]^F_{\rho_0}\sigma = \sigma'$ and $\Gamma_L, \Gamma_R' \vdash_\Sigma \sigma'$ Type.*
4. *If $\Gamma_L, x_0{:}\rho_0, \Gamma_R \vdash_\Sigma \sigma$ Type and $\Gamma_L, x_0{:}\rho_0, \Gamma_R \vdash_\Sigma M \Leftarrow \sigma$, then there exist σ' and M' such that $[M_0/x_0]^F_{\rho_0}\sigma = \sigma'$ and $[M_0/x_0]^O_{\rho_0} M = M'$ and $\Gamma_L, \Gamma_R' \vdash_\Sigma M' \Leftarrow \sigma'$.*

Theorem 22 (Decidability of typing)**.** *If all of the predicates in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ are decidable, then all of the judgements of the system are decidable.*

With these lemmas in theorems in place, we have all of the desired properties of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$. Now, we can focus on precisely specifying the relationships between $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ and the original $\mathsf{LF}_{\mathcal{P}}$ system.

Theorem 23 (Soundness)**.** *For any predicate $\mathcal{P}$ of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, we define a corresponding predicate in $\mathsf{LF}_{\mathcal{P}}$ as follows: $\mathcal{P}(\Gamma \vdash_\Sigma M : \sigma)$ holds if and only if $\Gamma \vdash_\Sigma M : \sigma$ is derivable in $\mathsf{LF}_{\mathcal{P}}$ and $\mathcal{P}(\Gamma \vdash_\Sigma M \Leftarrow \sigma)$ holds in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$. Then, the following claims hold:*

1. *If Σ sig is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then Σ sig is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
2. *If $\vdash_\Sigma \Gamma$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\vdash_\Sigma \Gamma$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
3. *If $\Gamma \vdash_\Sigma K$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma K$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
4. *If $\Gamma \vdash_\Sigma \alpha \Rightarrow K$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma \alpha : K$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
5. *If $\Gamma \vdash_\Sigma \sigma$ Type is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma \sigma :$ Type is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
6. *If $\Gamma \vdash_\Sigma A \Rightarrow \sigma$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma A : \sigma$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*
7. *If $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, then $\Gamma \vdash_\Sigma M : \sigma$ is derivable in $\mathsf{LF}_{\mathcal{P}}$.*

Proof. By mutual induction on the structure of the derivation. The first three items are proven directly, as the rules for signature, context, and kind formation are the same for the two systems, while for most of the rules the induction hypothesis is sufficient. In fact, only the rules of LPCP which involve hereditary substitution are interesting. Therefore, let us focus on the rule ($\mathsf{A{\cdot}App}$). Let us have $\alpha\, M \Rightarrow K[M/x]^K_\sigma$, obtained from $\Gamma \vdash_\Sigma \alpha \Rightarrow \Pi x{:}\sigma.K$, and $\Gamma \vdash_\Sigma M \Leftarrow \sigma$. By the induction hypothesis, we have that (in $\mathsf{LF}_{\mathcal{P}}$) $\Gamma \vdash_\Sigma \alpha : \Pi x{:}\sigma.K$ and $\Gamma \vdash_\Sigma M : \sigma$. Using the rule ($\mathsf{F{\cdot}App}$) of $\mathsf{LF}_{\mathcal{P}}$, we have that $\Gamma \vdash_\Sigma \alpha\, M : \tau[M/x]$, while the desired goal is $\Gamma \vdash_\Sigma \alpha\, M : \tau[M/x]^K_\sigma$. The idea here would be to apply the conversion rule ($\mathsf{F{\cdot}Conv}$), for which we need to prove that $\Gamma \vdash_\Sigma \tau[M/x]^K_\sigma$ and that $\tau[M/x] =_{\beta\mathcal{L}} \tau[M/x]^K_\sigma$. However, the former we can easily obtain from Proposition 8 and the Subject Reduction property of $\mathsf{LF}_{\mathcal{P}}$, while the latter is a direct consequence of the fact that hereditary substitution is designed to immediately calculate normal forms. The case in which the rule ($\mathsf{F{\cdot}App}$) is used is handled analogously. □

Vice versa, all $\mathsf{LF}_{\mathcal{P}}$ judgements in *η-long normal form* (η-lnf) are derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$. The definition of a judgement in η-lnf is based on the following extension of the standard η-rule to the lock constructor:

$$\lambda x{:}\sigma.Mx \to_\eta M \qquad \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]] \to_\eta M.$$

Definition 23. *An occurrence ξ of a constant or a variable in a term of an $\mathsf{LF}_{\mathcal{P}}$ judgement is* fully applied and unlocked *with respect to its type or kind $\Pi\vec{x}_1{:}\vec{\sigma}_1.\vec{\mathcal{L}}_1[\ldots\Pi\vec{x}_n{:}\vec{\sigma}_n.\vec{\mathcal{L}}_n[\alpha]\ldots]$, where $\vec{\mathcal{L}}_1,\ldots,\vec{\mathcal{L}}_n$ are vectors of locks, if ξ appears in contexts of the form $\vec{\mathcal{U}}_n[(\ldots(\vec{\mathcal{U}}_1[\xi\vec{M}_1])\ldots)\vec{M}_n]$, where $\vec{M}_1,\ldots,\vec{M}_n$, $\vec{\mathcal{U}}_1,\ldots,\vec{\mathcal{U}}_n$ have the same arities of the corresponding vectors of Π's and locks.*

Definition 24 (Judgements in η-long normal form)**.**

- *A term T in a judgement is in* η-lnf *if T is in normal form and every constant and variable occurrence in T is fully applied and unlocked w.r.t. its classifier in the judgement.*
- *A judgement is in* η-lnf *if all terms appearing in it are in η-lnf.*

Finally, we can proceed to the main theorem of this chapter, which is the correspondence theorem between derivations of $\mathsf{LF}_{\mathcal{P}}$ in η-lnf, and derivations in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$.

Theorem 24 (Correspondence)**.** *Assume that all predicates in* $\mathsf{LF}_{\mathcal{P}}$ *are well-behaved. For any predicate* $\mathcal{P}$ *in* $\mathsf{LF}_{\mathcal{P}}$*, we define a corresponding predicate in* $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ *with:* $\mathcal{P}(\Gamma \vdash_\Sigma M \Leftarrow \sigma)$ *holds if* $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ *is derivable in* $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ *and* $\mathcal{P}(\Gamma \vdash_\Sigma M : \sigma)$ *holds in* $\mathsf{LF}_{\mathcal{P}}$*. Then, we have:*

1. *If* Σ sig *is in* η*-lnf and is* $\mathsf{LF}_{\mathcal{P}}$*-derivable, then* Σ sig *is* $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$*-derivable.*
2. *If* $\vdash_\Sigma \Gamma$ *is in* η*-lnf and is* $\mathsf{LF}_{\mathcal{P}}$*-derivable, then* $\vdash_\Sigma \Gamma$ *is* $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$*-derivable.*
3. *If* $\Gamma \vdash_\Sigma K$ *is in* η*-lnf, and is* $\mathsf{LF}_{\mathcal{P}}$*-derivable, then* $\Gamma \vdash_\Sigma K$ *is* $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$*-derivable.*
4. *If* $\Gamma \vdash_\Sigma \alpha : K$ *is in* η*-lnf and is* $\mathsf{LF}_{\mathcal{P}}$*-derivable, then* $\Gamma \vdash_\Sigma \alpha \Rightarrow K$ *is* $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$*-derivable.*
5. *If* $\Gamma \vdash_\Sigma \sigma$:Type *is in* η*-lnf and is* $\mathsf{LF}_{\mathcal{P}}$*-derivable, then* $\Gamma \vdash_\Sigma \sigma$ Type *is* $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$*-derivable.*
6. *If* $\Gamma \vdash_\Sigma A : \alpha$ *is in* η*-lnf and is* $\mathsf{LF}_{\mathcal{P}}$*-derivable, then* $\Gamma \vdash_\Sigma A \Rightarrow \alpha$ *is* $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$*-derivable.*
7. *If* $\Gamma \vdash_\Sigma M : \sigma$ *is in* η*-lnf and is* $\mathsf{LF}_{\mathcal{P}}$*-derivable, then* $\Gamma \vdash_\Sigma M \Leftarrow \sigma$ *is* $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$*-derivable.*

Proof. We prove all of the items by mutual induction on the complexity of the judgement, where the complexity of a judgement is given by the sum of symbols appearing in it, provided that the complexity of the symbols Type and $\emptyset$ is 1, the complexity of a constant/variable is 2, the complexity of the symbol $\mathcal{U}$ is greater than the complexity of $\mathcal{L}$, and the complexity of the subject of the judgement is the sum of the complexities of its symbols plus the complexity of the normal form of the type of each of the subterms of the subject, derived in the given context and signature.

1. Directly, using the induction hypothesis.
2. Directly, using the induction hypothesis.
3. Directly, using the induction hypothesis.
4. If $\Gamma \vdash_\Sigma \alpha : K$ is in η-lnf, we have that $\alpha = aN_1 \dots N_n$, $K \equiv$ Type, and also that:
   (a) $a : \Pi x_1{:}\sigma_1 \dots x_n{:}\sigma_n.\mathsf{Type} \in \Sigma$, with $\Pi x_1{:}\sigma_1 \dots x_n{:}\sigma_n.\mathsf{Type}$ in η-lnf. This means that, for all $1 \leq i \leq n$, we have that σ_i is in η-lnf.
   (b) $\Gamma \vdash_\Sigma N_i : \sigma_i'$ is derivable in $\mathsf{LF}_{\mathcal{P}}$, for $1 \leq i \leq n$, where σ_i' is in η-lnf, and $\sigma_i' =_{\beta\mathcal{L}} \sigma_i[N_1/x_1, \dots, N_{i-1}/x_{i-1}]$.

   Now, from (a), we obtain that $\Gamma \vdash_\Sigma a \Rightarrow \Pi x_1{:}\sigma_1 \dots x_n{:}\sigma_n.\mathsf{Type}$ is derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$. Next, by applying the induction hypothesis to (b), we obtain that $\Gamma \vdash_\Sigma N_i \Leftarrow \sigma_i'$ is derivable in CLF for all i. From this, by repeatedly applying the rule ($A{\cdot}App$), we get that $\Gamma \vdash_\Sigma aN_1 \dots N_n \Rightarrow Type$.
5. The cases when σ is an atomic family have already been covered above, while the remaining cases are proven directly, using the induction hypothesis.
6. (a) If $\Gamma \vdash_\Sigma c : \alpha$, this could have been obtained only through the rule (O·Const) of $\mathsf{LF}_{\mathcal{P}}$, from $\vdash_\Sigma \Gamma$ and $c : \sigma \in \Gamma$. By the induction hypothesis, we get that $\vdash_\Sigma \Gamma$ in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, and that $c : \sigma \in \Gamma$, from which, using the rule (O·Const) of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, we obtain the desired $\Gamma \vdash_\Sigma c \Rightarrow \sigma$.

(b) If $\Gamma \vdash_\Sigma x : \sigma$, this could have been obtained only through the rule ($\mathsf{O{\cdot}Var}$) of $\mathsf{LF}_{\mathcal{P}}$, from $\vdash_\Sigma \Gamma$ and $x : \sigma \in \Gamma$. By the induction hypothesis, we get that $\vdash_\Sigma \Gamma$ in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, and that $x : \sigma \in \Gamma$, from which, using the rule ($\mathsf{O{\cdot}Var}$) of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, we obtain the desired $\Gamma \vdash_\Sigma x \Rightarrow \sigma$.

(c) Let $\Gamma \vdash_\Sigma AM : \alpha$ be derivable in $\mathsf{LF}_{\mathcal{P}}$, and be in η-lnf. A, as an atomic object, is then of the form:

$$\mathcal{U}^{\mathcal{P}_1}_{N_1,\sigma_1}[\ldots[\mathcal{U}^{\mathcal{P}_k}_{N_k,\sigma_k}[c|x\{M_1 \ldots M_n\}]\overrightarrow{M_k}]\overrightarrow{M_{k-1}}\ldots]\overrightarrow{M_1}.$$

Here, we have that c (or x) is fully applied and unlocked with respect to its classifier, and we also have that all N_i, σ_i, M_i, $\overrightarrow{M_i}$, as well as M and α, are in η-lnf. Also, we have that the types of M_i, $\overrightarrow{M_i}$, and M are in η-lnf, as they are recorded in the type of c (or x), which is part of the signature (or context), which is also in η-lnf. We will denote the type of M by σ, and the type of A by $\Pi z{:}\sigma.\tau$. Then, by the induction hypothesis, we have that $\Gamma \vdash_\Sigma M \Leftarrow \sigma$, and that $\Gamma \vdash_\Sigma A \Leftarrow \Pi z{:}\sigma.\tau$. From the latter, as it only could have been obtained through the rule ($\mathsf{O{\cdot}Atom}$), we have that $\Gamma \vdash_\Sigma A \Rightarrow \Pi z{:}\sigma.\tau$. Now, we have two cases to consider:

i. $\alpha \equiv a$, a constant atomic type, which is the trivial case, as a is immune to both hereditary and standard substitution. We immediately get that $\tau \equiv a$, with which we can use the rule ($\mathsf{O{\cdot}App}$) of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ to obtain the desired $\Gamma \vdash_\Sigma AM \Rightarrow \alpha$.

ii. $\alpha \equiv aM_1 \ldots M_n$, a fully applied constant atomic type of arity n. Then, we have that $\tau \equiv aM'_1 \ldots M'_n$. Due to the fact that M, and all M_i, M'_i are in normal form (as they are in η-lnf), we have that $\tau[M/z] \equiv a[M/z]M'_1[M/z]\ldots M'_n[M/z] \equiv aM_1 \ldots M_n \equiv \alpha$. However, due to the normal forms, the ordinary and hereditary substitution here coincide, yielding $\tau[M/z]^F_\sigma = \alpha$. With this, using the rule ($\mathsf{O{\cdot}App}$), we obtain the desired $\Gamma \vdash_\Sigma AM \Rightarrow \alpha$.

(d) Let us consider the case when $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[A] : \alpha'$. By inspection of the typing rules of $\mathsf{LF}_{\mathcal{P}}$, we have that the original introduction rule for our initial judgement had to have been $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[A] : \alpha$, derived from $\Gamma \vdash_\Sigma A : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\alpha]$, $\Gamma \vdash_\Sigma N : \sigma$, and $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$, where $\alpha =_{\beta\mathcal{L}} \alpha'$. By the induction hypothesis, we have $\Gamma \vdash_\Sigma N \Leftarrow \sigma$. Using the rule ($\mathsf{O{\cdot}Conv}$) of $\mathsf{LF}_{\mathcal{P}}$, we can now get $\Gamma \vdash_\Sigma A : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\alpha']$ and from this, by the induction hypothesis, we get that $\Gamma \vdash_\Sigma A \Leftarrow \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\alpha']$. Since that would only be possible through the rule ($\mathsf{O{\cdot}Atom}$), we have that $\Gamma \vdash_\Sigma A \Rightarrow \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\alpha']$ also holds. Finally, given the properties of the predicate induced in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ by the predicate $\mathcal{P}$ in $\mathsf{LF}_{\mathcal{P}}$, as stated in the formulation of the theorem, we can use the rule ($\mathsf{O{\cdot}Unlock}$) of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ to obtain the desired $\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] \Rightarrow \alpha'$.

7. (a) The cases when M is an atomic object have already been covered above.

(b) Let us consider the case when $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \theta$. By inspection of the typing rules of $\mathsf{LF}_{\mathcal{P}}$, we have that $\theta \equiv \Pi x{:}\sigma'.\tau'$, where the original introduction rule for our initial judgement had to have been $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau$, derived from $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau$, where $\sigma =_{\beta\mathcal{L}} \sigma'$, and $\tau =_{\beta\mathcal{L}} \tau'$. However, since σ, M, σ' and τ' are in η-lnf, they must also be in normal form, meaning that $\sigma \equiv \sigma'$, leaving us with $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau'$. Using the rule ($\mathsf{O{\cdot}Conv}$) of $\mathsf{LF}_{\mathcal{P}}$, we can now get $\Gamma, x{:}\sigma \vdash_\Sigma M : \tau'$, and from this, by the induction hypothesis, we get that $\Gamma, x{:}\sigma \vdash_\Sigma M \Leftarrow \tau'$, from which, using the rule ($\mathsf{O{\cdot}Abs}$) of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, we obtain the desired $\Gamma \vdash_\Sigma \lambda x{:}\sigma.M \Leftarrow \Pi x{:}\sigma.\tau'$.

(c) Let us consider the case when $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \theta$. By inspection of the typing rules of $\mathsf{LF}_{\mathcal{P}}$, we have that $\theta \equiv \mathcal{L}^{\mathcal{P}}_{N',\sigma'}[\rho']$, where the original introduction rule for our initial judgement had to have been $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, derived from $\Gamma \vdash_\Sigma M : \rho$ and

$\Gamma \vdash_\Sigma N : \sigma$, where $N =_{\beta\mathcal{L}} N'$, $\sigma =_{\beta\mathcal{L}} \sigma'$, and $\rho =_{\beta\mathcal{L}} \rho'$. However, since M, N, σ, N', σ', and ρ' are in η-lnf, they must also be in normal form, meaning that $N \equiv N'$ and $\sigma \equiv \sigma'$, leaving us with $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]'$. By the induction hypothesis, we have $\Gamma \vdash_\Sigma N \Leftarrow \sigma$. Using the rule (O·Conv) of $\mathsf{LF}_{\mathcal{P}}$, we can now get $\Gamma \vdash_\Sigma M : \rho'$ and from this, by the induction hypothesis, we get that $\Gamma \vdash_\Sigma M \Leftarrow \rho'$. Finally, we can use the rule (O·Lock) of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ to obtain the desired $\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \Leftarrow \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho']$.

□

We should notice that, by the Correspondence Theorem above, any well-behaved predicate $\mathcal{P}$ in $\mathsf{LF}_{\mathcal{P}}$ induces a well-behaved predicate in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$. Finally, and most importantly, we should notice that *not* all $\mathsf{LF}_{\mathcal{P}}$ judgements have a corresponding η-lnf. Namely, the judgement $x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \vdash_\Sigma x : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$ does not admit an η-expanded normal form when the predicate $\mathcal{P}$ does *not* hold on N, as the rule (O·Unlock) can be applied only when the predicate holds.

8.3 Summary

In this chapter, we have presented the Canonical $\mathsf{LF}_{\mathcal{P}}$ framework ($\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$), in which we make use of the notion of hereditary substitution to immediately compute the normal forms of $\mathsf{LF}_{\mathcal{P}}$, rather than having a separate reduction mechanism. Also, we have extended the notion of η-long normal forms to include the newly introduced locked and unlocked terms and types, and we have proven two theorems concerning the relationship between $\mathsf{LF}_{\mathcal{P}}$ and $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$, namely soundness and correspondence, with the former stating that all derivable judgements of $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$ are also derivable in $\mathsf{LF}_{\mathcal{P}}$, and the latter stating that all derivable judgements of $\mathsf{LF}_{\mathcal{P}}$ that are in η-long normal form are also derivable in $\mathsf{LF}^{\mathcal{C}}_{\mathcal{P}}$. This theorem will be of paramount importance for proving adequacy of the encodings we will be presenting in the next chapter.

Chapter 9

Encodings in $\mathsf{LF}_{\mathcal{P}}$

Contents

In this chapter, we illustrate how $\mathsf{LF}_{\mathcal{P}}$ can be used as a meta-language, by providing encodings for a number of crucial case studies. We place our focus on the formal systems in which derivation rules are subject to *side conditions* that are either rather difficult or impossible to encode naively, in a type theory-based LF, due to limitations of the latter or to the fact that these rules need to access the derivation context, or the structure of the derivation itself, or other structures and mechanisms which are simply not available at the object level. This is the case for sub-structural and program logics [Avron 1992, Avron 1998, Crary 2010].

We will start by providing an encoding of the well-known case of the untyped λ-calculus, with a call-by-value evaluation strategy. This allows us to illustrate yet another paradigm for dealing with free and bound variables appropriate for $\mathsf{LF}_{\mathcal{P}}$. Next, we elaborate the expressiveness of the predicates, and provide several examples of predicates which will be used in the later sections. Next, we add on top of the encoded untyped λ-calculus an extension, in order to model a sort of *design-by-contract* functional language, capturing pre- and post-conditions. Next, we discuss and provide encodings of modal logics, both in Hilbert and Natural Deduction style, and we give a sketch of how the non-commutative linear logic, introduced in [Polakow 1999], could be encoded in $\mathsf{LF}_{\mathcal{P}}$. The concluding example, in which we illustrate the encoding of a small imperative programming language logic and its Hoare logic in $\mathsf{LF}_{\mathcal{P}}$, appears in Section 9.3.

As far as comparisons with alternate encodings in LF, we refer the reader to the general comments which have been made in Section 7.4. Finally, in the formulations of adequacy theorems, we will be using the notion of *judgement in η-long normal form* (η-lnf), which has been introduced in Definition 24.

9.1 The Untyped λ-calculus

Consider the well-known untyped λ-calculus:

$$M, N, \ldots ::= x \mid M\ N \mid \lambda x.M,$$

with variables, application and abstraction. Our main goal here is to design a way of modeling free and bound variables that would both allow the formulation of well-behaved predicates over such variables, as well as preserve the benefits of the full Higher-Order-Abstract-Syntax (HOAS) approach, *i.e.* delegating α-conversion and capture-avoiding substitution to the metalanguage.

`nat : Type`	The type of natural numbers
`0 : nat`	Zero is a natural number
`S : nat -> nat`	The successor function
`free : nat -> term`	Modeling free variables
`app : term -> term -> term`	Application
`lam : (term -> term) -> term`	Abstraction

Figure 9.1: $\mathsf{LF}_{\mathcal{P}}$ signature Σ_λ for the untyped λ-calculus

In Figure 9.1, we present the $\mathsf{LF}_{\mathcal{P}}$ signature for the untyped λ-calculus. We encode natural numbers in a standard way, and we will be using the natural numbers themselves as abbreviations for repeated applications of `S` to `0`. We will be using an enumeration $\{x_i\}_{i\in\mathbb{N}\setminus\{0\}}$ of the variables in the untyped λ-calculus, we will be modeling free variables of the object language as constants of the form `(free n)`, for a suitable (encoding of a) natural number `n`, while we will be modeling bound variables using the standard HOAS approach. Next, we present the encoding function $\varepsilon_{\mathcal{X}}$, mapping the terms of the untyped λ-calculus into terms of $\mathsf{LF}_{\mathcal{P}}$:

$$\begin{aligned}
\varepsilon_{\mathcal{X}}(x_i) &= \begin{cases} x_i, \text{ if } x_i \in \mathcal{X} \\ (\texttt{free i}), \text{ if } x_i \notin \mathcal{X} \end{cases}, \\
\varepsilon_{\mathcal{X}}(MN) &= (\texttt{app}\ \varepsilon_{\mathcal{X}}(M)\,\varepsilon_{\mathcal{X}}(N)), \\
\varepsilon_{\mathcal{X}}(\lambda x_i.M) &= (\texttt{lam}\ \lambda x_i{:}\texttt{term}.\varepsilon_{\mathcal{X}\cup\{\texttt{x}_\texttt{i}\}}(M)),
\end{aligned}$$

where, in the last clause, $x_i \notin \mathcal{X}$. Therefore, the λ-term x_n (in which the variable is free) will be encoded by means of the term $\vdash_\Sigma$`(free n):term`. On the other hand, the λ-term $\lambda x_n.x_n$ (in which the variable is obviously bound) will be encoded by $\vdash_\Sigma$ (`lam` λx_n`:term.`x_n`)`. However, when we "open" the abstraction $\lambda x_n.M$, considering the body M, we will encode the body M as $\texttt{x}_\texttt{n}{:}\texttt{term} \vdash_\Sigma \varepsilon_{\{x_n\}}M$. In this case, we will refer to x_n as a *bindable* variable. These "bindable" variables must neither be confused with bound nor with free variables, and are modeled using variables of the metalanguage. By defining the encoding function in this way, we abide by the "closure under substitution" condition for external predicates, while still retaining the ability to handle "open" terms explicitly. Next, let us examine the adequacy of this encoding:

Theorem 25 (Adequacy of syntax)**.** *Let $\{x_i\}_{i\in\mathbb{N}\setminus\{0\}}$ be an enumeration of the variables in the λ-calculus. Then, the encoding function $\varepsilon_{\mathcal{X}}$ is a bijection between the λ-calculus terms with bindable variables in $\mathcal{X}$ and the terms M derivable in judgements $\Gamma \vdash_{\Sigma_\lambda} M : \texttt{term}$ in η-lnf, where $\Gamma = \{\texttt{x}_\texttt{i} : \texttt{term} \mid x_i \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a term M, with bindable variables in $\mathcal{X} = \{x_1, \ldots, x_k\}$, and $N_1, \ldots, N_k$, with bindable variables in $\mathcal{Y}$, the following holds:*

$$\varepsilon_{\mathcal{Y}}(M[N_1, \ldots, N_k/x_1, \ldots, x_k]) = \varepsilon_{\mathcal{X}}(M)[\varepsilon_{\mathcal{Y}}(N_1), \ldots, \varepsilon_{\mathcal{Y}}(N_k)/x_1, \ldots, x_k].$$

Proof. As we are not using locked types in this example, the proof of adequacy is standard. The injectivity of $\varepsilon_{\mathcal{X}}$ follows from the inspection of its definition, while the surjectivity follows by

defining the "decoding" function $\delta_{\mathcal{X}}$ on terms in η-lnf:

$$\begin{aligned}\delta_{\mathcal{X}}((\texttt{free i})) &= x_i \text{ (where } x_i \notin \mathcal{X}) \\ \delta_{\mathcal{X}}(x_i) &= x_i \text{ (where } x_i \in \mathcal{X}) \\ \delta_{\mathcal{X}}((\texttt{app M N})) &= \delta_{\mathcal{X}}(\texttt{M})\,\delta_{\mathcal{X}}(\texttt{N}) \\ \delta_{\mathcal{X}}(\texttt{lam } \lambda x_i\texttt{:term.M}) &= \lambda x.\delta_{\mathcal{X}\cup\{x_i\}}(\texttt{M})\end{aligned}$$

Given the characterization of η-lnfs, and the types of the constructors introduced in Σ_λ, it is easy to see that $\delta_{\mathcal{X}}$ is total and well-defined. It is not possible to derive an η-lnf of type `term` containing a $\mathcal{U}$-term, since no constructors in Σ_λ use $\mathcal{L}$-types. Finally, by induction on the structure of M, it is possible to check that $\delta_{\mathcal{X}}(\varepsilon_{\mathcal{X}}(M)) = M$ and that $\varepsilon_{\mathcal{X}}$ is compositional. □

9.1.1 A Closer Look at the Expressiveness of the Predicates

In this section, we will present some auxiliary functions and predicates on proof terms, the patterns of which frequently occur in the examples. The main archetype of these predicates is: "given constants or variables only occur with some modality $\mathcal{D}$ in subterms satisfying the decidable property $\mathcal{C}$", where the modalities can be, but are not limited to, any of the following phrases: "at least once", "only once","as the rightmost", "does not occur", etc. $\mathcal{C}$ can refer to the syntactic form of the subterm or to that of its type, the latter being the main reason for allowing predicates in $\mathsf{LF}_{\mathcal{P}}$ to access the context. We will require the following elements, which represent pairs of terms of the untyped λ-calculus, as presented in the previous section:

```
pair : Type                    [_, _] : term -> term -> pair
```

Further on, when we refer to free variables, we will in fact be referring to constants of the form `(free i)`, as defined in the previous section.

Counting occurrences of free variables. For one, we can count occurrences of free variables in normal forms of terms. A possible functional pseudo-code for $\mathit{count}\,(\Gamma \vdash_\Sigma [(\texttt{free i}), \texttt{M}] : \texttt{pair})$, counting occurrences of the encoding of x_i, $(i > 0)$ in the normal form of `M`, could be the following:

```
count (Γ ⊢Σ [(free i), M] : pair) ⇒
  let norm=NF(M) in
    match norm with
      |  x          => 0
      | (free j)    => if j=i then 1 else 0
      | (app N N')  => count (Γ ⊢Σ [(free i), N] : pair) + count (Γ ⊢Σ [(free i), N'] : pair)
      | (lam N)     => count (Γ ⊢Σ [(free i), (N (free 0))] : pair)
    end
```

where `(free 0)` is a "hole filler" when the recursive call crosses a `lam` binder, and `NF(M)` is a function calculating the normal form of the term `M`, possible due to the meta-properties of $\mathsf{LF}_{\mathcal{P}}$.

Listing free variables. Next, we introduce the auxiliary function $\mathit{freeVar}\,(\Gamma \vdash_\Sigma \texttt{M} : \texttt{term})$ returning as a result the list of free variables occurring in `M` in the order of their occurrence from left to right, without duplicates:

```
freeVar (Γ ⊢Σ M : term) ⇒
  let norm=NF(M) in
    match norm with
      |  x          => nil
      | (free i)    => if i=0 then nil else [(free i)]
      | (app N N')  => merge(freeVar (Γ ⊢Σ N : term), freeVar (Γ ⊢Σ N' : term))
      | (lam N)     => freeVar (Γ ⊢Σ (N (free 0)) : term)
    end
```

where *merge*(L,L') is the function merging two lists L and L' without duplicated elements. If we were to use *append* instead of *merge* in the application case in the definition of *freeVar*, we would obtain as a result the list of all the occurrences of free variables in M in their order of appearance from left to right, and we will refer to such a variant of *freeVar* as *freeVarOcc*.

Occurrence of a free variable. Whether a free variable occurs within a term or not is now easily verifiable by using the function *count*, but also directly, in the following manner:

```
freeIn(Γ ⊢Σ [(free i), M] : pair) ⇒
  let norm=NF(M) in
    match norm with
      |  x          => false
      | (free j)    => j=i
      | (app N N')  => freeIn(Γ ⊢Σ [(free i), N] : pair) ∨ freeIn(Γ ⊢Σ [(free i), N'] : pair)
      | (lam N)     => freeIn(Γ ⊢Σ [(free i), (N (free 0))] : pair)
    end
```

where $i > 0$. As the truthfulness of the predicate does neither depend on the signature nor on the context, we conclude that it is closed under signature and context weakening and permutation. As the terms of the untyped λ-calculus are encoded in such a way that they do not include any free variables of the object language or leave room for substitution, we conclude that the predicate is also closed under substitution and $\beta\mathcal{L}$-reduction. Therefore, we have that *freeIn* is a well-behaved predicate in $\mathsf{LF}_{\mathcal{P}}$. Here, we should note that we need to accommodate the possibility of M being a free variable of type term belonging to the object language, but that this does not effect the well-behavedness of the predicate.

Leftmost and rightmost free variables. As an example of the application of freeVarOcc, we define the predicates *leftmost*$(\Gamma \vdash_\Sigma [(\texttt{free i}), \texttt{M}] : \texttt{pair})$ and *rightmost*$(\Gamma \vdash_\Sigma [(\texttt{free i}), \texttt{M}] : \texttt{pair})$, which hold if (free i) is the encoding of the leftmost (rightmost, respectively) free variable occurring in M:

```
leftmost(Γ ⊢Σ [(free i), M] : pair) ⇒
  let L=freeVarOcc(Γ ⊢Σ M : term) in
    match L with
      | nil     => false
      | [l, L'] => l=(free i)
    end

rightmost(Γ ⊢Σ [(free i), M] : pair) ⇒
  let L=reverse(freeVarOcc(Γ ⊢Σ M : term)) in
    match L with
      | nil     => false
      | [l, L'] => l=(free i)
    end
```

where *reverse*(L) is the standard function reversing the order of the elements occurring in L. In the same manner as for the predicate freeIn, we can conclude that *leftmost* and *rightmost* are well-behaved predicates in $\mathsf{LF}_{\mathcal{P}}$.

Closedness of Terms. Using the function freeVar (or freeVarOcc), it is easy to inspect whether a given term M is closed or not:

```
closedTerm(Γ ⊢Σ M : term) ⇒ (freeVar(Γ ⊢Σ M : term) = nil)
```

In the same manner as for the predicate freeIn, we obtain that closedTerm is also a well-behaved predicate in $\mathsf{LF}_{\mathcal{P}}$.

9.1.2 Adding a Call-by-Value Reduction Strategy

Definition 25 (The call-by-value reduction strategy.). *The call-by-value (CBV) evaluation strategy can be specified by the following rules:*

$$\frac{}{\vdash_{CBV} M = M} \quad (\mathsf{refl}) \quad \text{Reflexivity}$$

$$\frac{\vdash_{CBV} N = M}{\vdash_{CBV} M = N} \quad (\mathsf{symm}) \quad \text{Symmetry}$$

$$\frac{\vdash_{CBV} M = N \qquad \vdash_{CBV} N = P}{\vdash_{CBV} M = P} \quad (\mathsf{trans}) \quad \text{Transitivity}$$

$$\frac{\vdash_{CBV} M = N \qquad \vdash_{CBV} M' = N'}{\vdash_{CBV} MM' = NN'} \quad (\mathsf{app}) \quad \text{Compatibility with application}$$

$$\frac{v \text{ is a value}}{\vdash_{CBV} (\lambda x.M)v = M[v/x]} \quad (\beta_{\mathsf{v}}) \quad \text{Conditioned } \beta\text{-reduction}$$

$$\frac{\vdash_{CBV} M = N}{\vdash_{CBV} \lambda x.M = \lambda x.N} \quad (\xi_{\mathsf{v}}) \quad \text{Compatibility with abstraction}$$

where values are either variables (constants in our encoding) or functions.

As the rule (β_{v}) has a side condition on its application, it will be necessary to make use of $\mathcal{L}$-types in the encoding. We proceed as follows:

Definition 26 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ_{CBV} for λ-calculus CBV reduction). *We extend the signature of Figure 9.1 with the following elements:*

```
   triple : Type
<_, _, _> : term -> (term -> term) -> (term -> term) -> triple
       eq : term -> term -> Type

     refl : ΠM:term.(eq M M)
     symm : ΠM:term.ΠN:term.(eq N M) -> (eq M N)
    trans : ΠM,N,P:term.(eq M N) -> (eq N P) -> (eq M P)
   eq_app : ΠM,N,M',N':term.(eq M N) -> (eq M' N') -> (eq (app M M') (app N N'))

    betav : ΠM:(term -> term).ΠN:term.L^Val_N[eq (app (lam M) N) (M N)]
      xiv : Πx:term.ΠM,N:(term -> term).L^ξ_⟨x,M,N⟩[(eq (M x)(N x))->(eq (lam M)(lam N))]
```

where the predicates `Val` *and* ξ *are defined as follows, and* `triple` *is the type of triples of terms with types* `term`, `term -> term` *and* `term -> term`*:*

- *The predicate* `Val` $(\Gamma \vdash_{\Sigma}$ `N : term`$)$ *holds iff either* `N` *is an abstraction or a constant (i.e. a term of the shape* `(free i)`*);*

```
Val(Γ ⊢_Σ N : term) ⇒
  let norm=NF(N) in
    match norm with
      | app M' N' => false
      | _         => true
    end
```

- *The predicate* $\xi(\Gamma \vdash_\Sigma$ `<x,M,N> : triple`$)$ *holds iff* `x` *is a constant (i.e. a term of the shape* `(free i)`*),* `M` *and* `N` *are closed and* `x` *does not occur in* `M` *and* `N`*.*

```
ξ(Γ ⊢_Σ <x,M,N> : triple) ⇒
  let nX=NF(x), nM=NF(M), nN=NF(N) in
    match nX with
      | (free i) => closedTerm(nM) ∧ ¬freeIn(Γ ⊢_Σ [(free i), nM] : pair) ∧
                    closedTerm(nN) ∧ ¬freeIn(Γ ⊢_Σ [(free i), nN] : pair)
      | _        => false
    end
```

We can easily verify that both of the predicates satisfy all of the four requirements for well-behavedness and that, thereby, their use in this encoding is allowed.

Theorem 26 (Adequacy of CBV reduction)**.** *Given an enumeration* $\{x_i\}_{i\in\mathbb{N}\setminus\{0\}}$ *of the variables in the* λ*-calculus, there is a bijection between derivations of the judgement* $\vdash_{CBV} M = N$ *on terms with no bindable variables in the CBV* λ*-calculus and proof terms* `h`*, such that* $\vdash_{\Sigma_{CBV}}$ `h` $:$ `(eq` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(N)$`)` *is in* η*-lnf.*

Proof. We define an encoding function $\varepsilon^{=}_{\emptyset}$ by induction on derivations of the form $\vdash_{CBV} M = N$ (on terms with no bindable variables) as follows:

- if ∇ is the derivation

$$\frac{}{\vdash_{CBV} M = M}$$

  then $\varepsilon^{=}_{\emptyset}(\nabla) =$ `(refl` $\varepsilon_\emptyset(M)$`)` $:$ `(eq` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(M)$`)`;

- if ∇ is the derivation with `(symm)` as last applied rule, then, by the induction hypothesis, there is a term `h` such that $\vdash_{\Sigma_{CBV}}$ `h` $:$ `(eq` $\varepsilon_\emptyset(N)$ $\varepsilon_\emptyset(M)$`)`. Hence, we have $\varepsilon^{=}_{\emptyset}(\nabla) =$ `(symm` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(N)$ `h)` $:$ `(eq` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(N)$`)`;

- if ∇ is the derivation with `(trans)` as last applied rule, then, by the induction hypothesis, we have that there exist terms `h` and `h'` such that $\vdash_{\Sigma_{CBV}}$ `h` $:$ `(eq` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(N)$`)`, and also $\vdash_{\Sigma_{CBV}}$ `h'` $:$ `(eq` $\varepsilon_\emptyset(N)$ $\varepsilon_\emptyset(P)$`)`. Hence, we will have that $\varepsilon^{=}_{\emptyset}(\nabla) =$ `(trans` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(N)$ $\varepsilon_\emptyset(P)$ `h h'`$)$ $:$ `(eq` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(P)$`)`;

- if ∇ is the derivation with `(eq_app)` as last applied rule, then, by the induction hypothesis, we have that there exist terms `h` and `h'` such that $\vdash_{\Sigma_{CBV}}$ `h` $:$ `(eq` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(N)$`)` and $\vdash_{\Sigma_{CBV}}$ `h'` $:$ `(eq` $\varepsilon_\emptyset(M')$ $\varepsilon_\emptyset(N')$`)`. Hence, we will have that $\varepsilon^{=}_{\emptyset}(\nabla) =$ `(eq_app` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(N)$ $\varepsilon_\emptyset(M')$ $\varepsilon_\emptyset(N')$ `h h'`$)$ $:$ `(eq (app` $\varepsilon_\emptyset(M)$ $\varepsilon_\emptyset(M')$`) (app` $\varepsilon_\emptyset(N)$ $\varepsilon_\emptyset(N')$`))`;

- if ∇ is the derivation

$$\frac{v \text{ is a value}}{\vdash_{CBV} (\lambda x_i.M)v = M[v/x_i]}$$

  then we will have that $\varepsilon^{=}_{\emptyset}(\nabla) = \mathcal{U}^{Val}_{\varepsilon_\emptyset(v)}$[`(betav (`$\lambda x_i$`:term.`$\varepsilon_{\{x_i\}}(M)$`)` $\varepsilon_\emptyset(v)$`)` $:$ `(eq (app (lam` λx_i`:term.`$\varepsilon_{\{x_i\}}(M)$`)` $\varepsilon_\emptyset(v)$`) ((`λx_i`:term.`$\varepsilon_{\{x_i\}}(M)$`)(`$\varepsilon_\emptyset(v)$`)))`][1];

[1] Notice the presence of the unlock operator in front of the $\mathsf{LF}_{\mathcal{P}}$ encoding: this is possible due to the fact that we know, by hypothesis (e.g., the premise of the applied derivation rule), that v is a value. Indeed, since values in the object language are either variables or abstractions and we are deriving things from the empty context, in this case v must be an abstraction $\lambda x_i.N$ (otherwise it would be a free variable and the derivation context could not be empty). Thus, it will be encoded into a term of the form `(lam` λx_i`:term.`$\varepsilon_{\{x_i\}}(N)$`)` and the predicate `Val` is defined to hold on such terms (see Definition 26), whence the predicate `Val` holds on $\vdash_{CBV} \varepsilon_\emptyset(v)$ $:$ `term`.

- if ∇ is the derivation with (ξ_v) as the last applied rule, then, by the induction hypothesis, we have that there exists a term $\mathtt{h}$, such that $\vdash_{\Sigma_{CBV}} \mathtt{h} : (\mathtt{eq}\ \varepsilon_\emptyset(M)\ \varepsilon_\emptyset(N))$[2]. Given this, we have: $\varepsilon^{=}_{\emptyset}(\nabla) = (\mathcal{U}^{\xi}_{\mathtt{T},\mathtt{triple}}[(\mathtt{xiv}\ \varepsilon_\emptyset(x_i)\ \lambda x_i{:}\mathtt{term}.\varepsilon_{\{x_i\}}(M)\ \lambda x_i{:}\mathtt{term}.\varepsilon_{\{x_i\}}(N))]\mathtt{h})$: $(\mathtt{eq}\ (\mathtt{lam}\ \lambda x_i{:}\mathtt{term}.\varepsilon_{\{x_i\}}(M))\ (\mathtt{lam}\ \lambda x_i{:}\mathtt{term}.\varepsilon_{\{x_i\}}(N)))$, with $\mathtt{T}$ being $\langle \varepsilon_\emptyset(x_i), (\lambda x_i{:}\mathtt{term}.\varepsilon_{\{x_i\}}(M)), (\lambda x_i{:}\mathtt{term}.\varepsilon_{\{x_i\}}(N))\rangle$.

The injectivity of $\varepsilon^{=}_{\emptyset}$ follows by the inspection of its definition, while the surjectivity follows by defining the "decoding" function $\delta^{=}_{\emptyset}$ by induction on the derivations of the shape $\vdash_{\Sigma_{CBV}} \mathtt{h} : (\mathtt{eq}\ \mathtt{M}\ \mathtt{N})$ in η-long normal form. Since most of the cases are purely technical, we analyze the definition concerning the main rule (β_v), as it involves an external predicate. So, if we derive from Σ_{CBV} a proof term $\mathtt{h}$ in η-long normal form such as $\mathcal{U}^{Val}_{\mathtt{N},\mathtt{term}}[\mathtt{betav}\ \mathtt{M}\ \mathtt{N}]$ whose type is $\mathtt{(eq\ (app\ (lam\ M)\ N)\ (M\ N))}$ (where $\mathtt{M} \equiv \lambda \mathtt{x_i}{:}\mathtt{term}.\mathtt{M'}$, with $\mathtt{M'}$ in η-lnf), then the predicate $\mathtt{Val}\ (\vdash_{\Sigma_{CBV}} \mathtt{N} : \mathtt{term})$ must hold, and $\mathtt{N}$ is encoding the value $\delta_\emptyset(\mathtt{N})$. Hence, the *decoding* of $\mathtt{h}$ is the following derivation:

$$\frac{\delta_\emptyset(\mathtt{N}) \text{ is a value}}{\vdash_{CBV} \delta_\emptyset(\mathtt{lam}\ (\lambda \mathtt{x_i}{:}\mathtt{term}.\mathtt{M'}))\delta_\emptyset(\mathtt{N}) = \delta_\emptyset((\lambda \mathtt{x_i}{:}\mathtt{term}.\mathtt{M'})\ \mathtt{N})},$$

and since we have that $\delta_\emptyset((\mathtt{lam}\ (\lambda \mathtt{x_i}{:}\mathtt{term}.\mathtt{M'}))) = \lambda x_i.\delta_{\{x_i\}}(\mathtt{M'})$ (see proof of Theorem 25), and that $\lambda x_i.\delta_{\{x_i\}}(\mathtt{M'})\delta_\emptyset(\mathtt{N}) = \delta_{\{x_i\}}(\mathtt{M'})[\delta_\emptyset(\mathtt{N})/x_i]$ (β-reduction in CBV λ-calculus), and also that $\delta_\emptyset((\lambda \mathtt{x_i}{:}\mathtt{term}.\mathtt{M'})\ \mathtt{N}) = \delta_\emptyset(\mathtt{M'}[\mathtt{N}/\mathtt{x_i}])$ (β-reduction in $\mathsf{LF}_\mathcal{P}$) and finally that $\delta_{\{x_i\}}(\mathtt{M'})[\delta_\emptyset(\mathtt{N})/x_i] = \delta_\emptyset(\mathtt{M'}[\mathtt{N}/\mathtt{x_i}])$ (by induction on the structure of $\mathtt{M'}$), we are done. Therefore, we can verify by induction on η-long normal forms that $\delta^{=}_{\emptyset}$ is well-defined and total. Similarly, we can prove $\delta^{=}_{\emptyset}$ to be the inverse of $\varepsilon^{=}_{\emptyset}$, making $\varepsilon^{=}_{\emptyset}$ a bijection. □

We conclude this section by illustrating the expressive power of $\mathsf{LF}_\mathcal{P}$, by encoding a restricted η-rule, which generalizes the one originally suggested in [Plotkin 1975], *i.e.* $\lambda x.xx = \lambda x.x\lambda y.xy$

$$\mathtt{eta}\ :\ \Pi \mathtt{M{:}(term\ ->\ term)}.\mathcal{L}^{\eta}_{\mathtt{M}}[\mathtt{(eq\ (lam\ M)\ (lam\ (\lambda x{:}term.(M\ (lam\ \lambda y{:}term.(app\ x\ y)))))))}]$$

where the predicate $\eta(\Gamma \vdash_\Sigma \mathtt{M\ :\ (term\ ->\ term)})$ holds if and only if the outermost abstracted variable of $\mathtt{M}$ occurs in functional position among the head variables.

9.1.3 Capturing Design-by-Contract

In this section, we extend the untyped call-by-value λ-calculus of the previous example so as to accommodate a minimal functional language supporting the *design-by-contract* paradigm (see, *e.g.*, [Meyer 1991]). More precisely, we will enrich the λ-calculus with the conditional expression *cond(C,M)*, whose intended semantics is that of checking that the constraint C applied to M (denoted by $C(M)$) holds. As we will see, such expressions can be used to validate the *entry input* and the *exit value* on a function application. The general syntax of conditions C is omitted, and a typical example is given by predicates on natural number (in)equalities, as in the Hoare Logic example in Section 9.3. Informally, C must be "something" which can be evaluated to true or false, when applied to its argument. The syntax of expressions is defined as follows:

$$M, N, \ldots ::= x \mid M\ N \mid \lambda x.M \mid cond(C, M),$$

[2]Notice that the object variable x_i occurring in M and N is represented by a constant (`(free i)`, since the encoding function takes the empty set as the set of *bindable* variables. Instead, in the next line, the encoding function will take $\{x_i\}$ as the set of bindable variables, yielding an encoding of x_i directly as the metavariable of the metalanguage of $\mathsf{LF}_\mathcal{P}$.

`nat : Type`	The type of natural numbers
`0 : nat`	Zero is a natural number
`S : nat -> nat`	The successor function
`bool : Type`	The type of boolean values
`free : nat -> term`	Modeling free variables
`app : term -> term -> term`	Application
`lam : (term -> term) -> term`	Abstraction
`cond : (term -> bool) -> term -> term`	Condition

Figure 9.2: $\mathsf{LF}_{\mathcal{P}}$ signature Σ for the design-by-contract λ-calculus

The call-by-value (CBV) evaluation strategy can be then extended by adding to the rules of Definition 25 the following one:

$$\frac{C(v) \text{ holds and } v \text{ is a value}}{\vdash_{CBV} cond(C, v) = v} \ (\mathsf{cond})$$

Here, conditional expressions are *first-class* values, *i.e.* they can be passed as arguments of a function. So far, we can encode pre- and post-conditions in our language as follows:

$$cond(Q,\ (M\ (cond(P,\ N))))$$

The key idea is that the above expression will reduce if and only if the argument N were to "pass" the pre-condition P (*i.e.* the *input contract*) and if the result of the application MN were to "pass" the post-condition Q (*i.e.* the *output contract*).

We build upon the previous case study, encoding free variables by means of constants (e.g., natural numbers) embedded into terms, while bindable and bound variables will be represented by metavariables of the metalanguage. In Figure 9.2, we provide an $\mathsf{LF}_{\mathcal{P}}$ signature for our language, extending that presented in Figure 9.1 (`bool` will represent the type of boolean values `true` and `false`, *i.e.* the result of evaluating a condition C on its argument M).

In order to make clear the role played by the types and constructors so far introduced, we fully specify the encoding function $\varepsilon_{\mathcal{X}}$, mapping terms of the source language into the corresponding terms of $\mathsf{LF}_{\mathcal{P}}$, where $\mathcal{X}$ denotes a set of bindable variables. Given an enumeration $\{x_i\}_{i\in\mathbb{N}\setminus\{0\}}$ of the variables in the source language, we have the following:

$$\begin{aligned}
\varepsilon_{\mathcal{X}}(x_i) &= \begin{cases} x_i, \text{ if } x_i \in \mathcal{X} \\ (\texttt{free i}), \text{ if } x_i \notin \mathcal{X} \end{cases} \\
\varepsilon_{\mathcal{X}}(MN) &= (\texttt{app}\ \varepsilon_{\mathcal{X}}(M)\,\varepsilon_{\mathcal{X}}(N)), \\
\varepsilon_{\mathcal{X}}(\lambda x_i.M) &= (\texttt{lam}\ \lambda x_i{:}\texttt{term}.\varepsilon_{\mathcal{X}\cup\{x_i\}}(M)), \\
\varepsilon_{\mathcal{X}}(cond(C, M)) &= (\texttt{cond}\ \varepsilon_{\mathcal{X}}(C)\ \varepsilon_{\mathcal{X}}(M)).
\end{aligned}$$

We now present the encoding of the CBV reduction of our source language encoded in $\mathsf{LF}_{\mathcal{P}}$. All the rules stated in the previous case study about the untyped λ-calculus remain unchanged. There is only the need to account for the new reduction rule involving the new constructor `cond`.

Definition 27 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ for design by contract λ-calculus CBV reduction)**.**
We extend the signature of Definition 26 by adding the following constant:

$$\texttt{condv : }\Pi\texttt{C:(term -> bool).}\Pi\texttt{M:term.}\mathcal{L}^{\texttt{Eval}}_{\texttt{(C M)},\texttt{bool}}[\texttt{(eq (cond C M) M)}]$$

where the external predicate `Eval` *is defined as follows:*

- *Eval* $(\Gamma \vdash_\Sigma$ `(C M):bool`$)$ *holds iff* `M` *is a value (i.e. an abstraction or a constant),* `C` *and* `M` *are closed and the evaluation of the condition* `C` *on the term* `M` *holds.*

The expressive power of external predicates is fully exploited in the above example. Of course, we require that the external logical conditions C (corresponding to object-level expressions `C`) allow the *Eval* predicate to satisfy the requirements of Definition 21, *i.e.* that they induce a well-behaved $\beta\mathcal{L}$-reduction. We conclude by illustrating how the expressiveness of predicates and locked types allows for a straightforward encoding of the *design-by-contract* predecessor function as follows (taking `nat` as the type of terms):

$$\texttt{(lam } \lambda\texttt{x:nat.(cond (}\lambda\texttt{z:nat.(z} \geq \texttt{0) ((cond (}\lambda\texttt{y:nat.(y > 0)) x) - 1)))}$$

9.2 Substructural Logics

Next, we turn our attention to substructural logics. In many formal systems, rules are subject to side conditions and structural constraints on the shape of assumptions or premises. Typical examples of this are the necessitation rule or the $\Box$-introduction rules in Modal logics (see, e.g., [Avron 1992, Avron 1998, Crary 2010]). For the sake of readability, we will onward often use an *infix* notation for encoding binary logic operators.

9.2.1 Modal Logics in Hilbert style.

In this section, we show how $\mathsf{LF}_{\mathcal{P}}$ allows for smooth encodings of logical systems with "rules of proof" as well as "rules of derivation". The former apply only to premises which do not depend on any assumption, such as *necessitation*, while the latter are the usual rules which apply to all premises, such as *modus ponens*. The idea is to use the appropriate locked types in rules of proof and standard types in the rules of derivation.

First, we will consider the classical Modal Logics K, KT, $K4$, $KT4$ (S_4), $KT45$ (S_5) in Hilbert style. All of these logics (K, KT, $K4$, $KT4$ (S_4), $KT45$ (S_5)) build upon the standard Hilbert axioms for classical logic ($A1 - A3$) by introducing the necessity ($\Box$) and possibility ($\Diamond$) operators, and all of them feature necessitation (the NEC rule in Figure 9.3) as a rule of proof, which can be applied only if ϕ is a theorem, *i.e.* if it does not depend on any assumptions. Therefore, our main task here would be to correctly and elegantly capture this rule of proof using the machinery of locked types.

$$\begin{array}{rcl}
\mathsf{A_1} & : & \phi \to (\psi \to \phi) \\
\mathsf{A_2} & : & (\phi \to (\psi \to \xi)) \to (\phi \to \psi) \to (\phi \to \xi) \\
\mathsf{A_3} & : & (\neg\phi \to \neg\psi) \to ((\neg\phi \to \psi) \to \phi) \\
\mathsf{K} & : & \Box(\phi \to \psi) \to (\Box\phi \to \Box\psi) \\
\top & : & \Box\phi \to \phi \\
\mathsf{4} & : & \Box\phi \to \Box\Box\phi \\
\mathsf{5} & : & \Diamond\phi \to \Box\Diamond\phi \\
\mathsf{MP} & : & \dfrac{\phi \quad \phi \to \psi}{\psi} \\
\mathsf{NEC} & : & \dfrac{\phi}{\Box\phi} \text{ if } \phi \text{ is a theorem}
\end{array}$$

C	$A_1 + A_2 + A_3 + MP$
K	$C + K + NEC$
KT	$K + T$
$K4$	$K + 4$
$KT4$	$KT + 4$
$KT45$	$KT4 + 5$

Figure 9.3: Hilbert style rules for Modal Logics

$$
\begin{array}{rl}
\mathtt{o} & : \mathtt{Type} \\
\rightarrow & : \mathtt{o} \rightarrow \mathtt{o} \rightarrow \mathtt{o} \\
\neg & : \mathtt{o} \rightarrow \mathtt{o} \\
\Box & : \mathtt{o} \rightarrow \mathtt{o} \\
\Diamond & : \mathtt{o} \rightarrow \mathtt{o} \\
\mathtt{True} & : \mathtt{o} \rightarrow \mathtt{Type} \\
\mathtt{A1} & : \Pi\phi,\psi{:}\mathtt{o}.\mathtt{True}(\phi \rightarrow (\psi \rightarrow \phi)) \\
\mathtt{A2} & : \Pi\phi,\psi,\xi{:}\mathtt{o}.\mathtt{True}(\phi \rightarrow (\psi \rightarrow \xi)) \rightarrow (\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \xi)) \\
\mathtt{A3} & : \Pi\phi,\psi{:}\mathtt{o}.\mathtt{True}((\neg\phi \rightarrow \neg\psi) \rightarrow ((\neg\phi \rightarrow \psi) \rightarrow \phi)) \\
\mathtt{K} & : \Pi\phi,\psi{:}\mathtt{o}.\mathtt{True}(\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi)) \\
\mathtt{T} & : \Pi\phi{:}\mathtt{o}.\mathtt{True}(\Box\phi \rightarrow \phi) \\
\mathtt{4} & : \Pi\phi{:}\mathtt{o}.\mathtt{True}(\Box\phi \rightarrow \Box\Box\phi) \\
\mathtt{5} & : \Pi\phi{:}\mathtt{o}.\mathtt{True}(\Diamond\phi \rightarrow \Box\Diamond\phi) \\
\mathtt{MP} & : \Pi\phi,\psi{:}\mathtt{o}.\mathtt{True}(\phi) \rightarrow \mathtt{True}(\phi \rightarrow \psi) \rightarrow \mathtt{True}(\psi) \\
\mathtt{NEC} & : \Pi\phi{:}\mathtt{o}.\Pi\mathtt{m}{:}\mathtt{True}(\phi).\mathcal{L}_{\mathtt{m}}^{\mathit{Closed}}[\mathtt{True}(\Box\phi)]
\end{array}
$$

Figure 9.4: The signature $\Sigma_{\Box}$ for Modal Logics in Hilbert style

The $\mathsf{LF}_{\mathcal{P}}$ signature corresponding to the modal logics presented in Figure 9.3 is shown in Figure 9.4. The encodings of the logical connectives and modality operators, as well as the modus ponens rule are standard. In order to capture the side-condition of the necessitation rule, we make use of the predicate $\mathit{Closed}(\Gamma \vdash_{\Sigma} \mathtt{m}{:}\mathit{True}(\phi))$, which holds iff "all of the free variables that occur in $\mathtt{m}$ are of type $\mathtt{o}$", and this is precisely what is needed to correctly encode the notion of the necessitation rule of proof, if $\mathtt{o}$ is the type of propositions. Indeed, if all of the free variables of a given proof term satisfy such a condition, it is clear, by inspection of the η-long normal forms, that there cannot be free variables of type $\mathit{True}(\ldots)$ in the proof term, *i.e.* that the encoded modal formula does not depend on any assumptions. As can be noticed, this example requires that predicates inspect the environment and be defined on *typed judgements*, as indeed is the case in $\mathsf{LF}_{\mathcal{P}}$. Furthermore, we can show without difficulties, and similarly to the previously examined predicates, that the above predicate is well-behaved, in the sense of Definition 21. Hence, we ensure a sound derivation in $\mathsf{LF}_{\mathcal{P}}$ of a proof of $\Box\phi$, by locking the type $\mathtt{True}(\Box\phi)$ with an appropriate condition in the conclusion of the encoding of the necessitation rule.

Next, we present a functional pseudo-code to illustrate the computability of the predicate Closed. The context of the typing judgement passed as argument to Closed is partitioned into two sub-contexts Γ and Γ', where the former contains all of the free variables occurring in the term $\mathtt{M}$, and the latter the remaining variables (this partitioning is re-computed with each recursive call, and is denoted by an appropriate index). We make use of two auxiliary functions which are computable (given the metaproperties of $\mathsf{LF}_{\mathcal{P}}$), namely:

- `NF`, which takes as argument a type and reduces it to its normal form;
- `TypeInf`, which takes as arguments the contexts Γ, Γ', the signature Σ and a term $\mathtt{M}$ and executes a procedure of type inference, returning as a result a type τ such that $\Gamma, \Gamma' \vdash_{\Sigma} \mathtt{M} : \tau$ holds or `undef` in case of failure.

The two base cases ensure that the propositions themselves, as well as terms that do not have any free variables are considered closed (in the sense of the predicate Closed), while the remaining cases (applications, abstractions, locked terms, and unlocked terms) recursively examine the remaining possibilities, in a standard way.

```
Closed(Γ, Γ' ⊢_Σ M : o) ⇒ true                                                    Propositions
Closed(∅, Γ' ⊢_Σ M : σ) ⇒ true                                  Terms with no free variables
Closed(Γ, Γ' ⊢_Σ MN : τ) ⇒                                                       Applications
  if (TypeInf(Γ, Γ', Σ, M)=Πx:τ1.τ2)
    then (Closed(Γ_M, Γ'_M ⊢_Σ M : Πx:τ1.τ2) ∧ Closed(Γ_N, Γ'_N ⊢_Σ N : τ1))
    else false
Closed(Γ, Γ' ⊢_Σ λx:σ.M : τ) ⇒                                                   Abstractions
  if (NF(τ)=Πx:σ'.σ'')
  then Closed(Γ_M, x:σ', Γ'_M ⊢_Σ M : σ'')
  else false
Closed(Γ, Γ' ⊢_Σ L^P_{N,σ}[M] : τ) ⇒                                                Locks
  if (NF(τ)=L^P_{N',σ'}[ρ])
  then (Closed(Γ_M, Γ'_M ⊢_Σ M : ρ) ∧ Closed(Γ_N, Γ'_N ⊢_Σ N : σ) ∧ Closed(Γ_σ, Γ'_σ ⊢_Σ σ : Type))
  else false
Closed(Γ, Γ' ⊢_Σ U^P_{N,σ}[M] : τ) ⇒                                              Unlocks
  if (TypeInf(Γ, Γ', Σ, M)=L^P_{N',σ'}[τ'])
  then (Closed(Γ_M, Γ'_M ⊢_Σ M : L^P_{N',σ'}[τ']) ∧ Closed(Γ_N, Γ'_N ⊢_Σ N : σ) ∧ Closed(Γ_σ, Γ'_σ ⊢_Σ σ : Type))
  else false
```

Figure 9.5: Pseudo-code of the predicate *Closed*

Adequacy theorems are rather straightforward to state and prove. As usual, we define an encoding function $\varepsilon_{\mathcal{X}}$ on formulas with free variables in $\mathcal{X}$ as follows, representing atomic formulas by means of $\mathsf{LF}_{\mathcal{P}}$ metavariables:

$$\begin{aligned}
\varepsilon_{\mathcal{X}}(x) &= x, \text{ where } x \in \mathcal{X},\\
\varepsilon_{\mathcal{X}}(\neg\phi) &= \neg\varepsilon_{\mathcal{X}}(\phi),\\
\varepsilon_{\mathcal{X}}(\phi \to \psi) &= \varepsilon_{\mathcal{X}}(\phi) \to \varepsilon_{\mathcal{X}}(\psi),\\
\varepsilon_{\mathcal{X}}(\Box\phi) &= \Box\varepsilon_{\mathcal{X}}(\phi),\\
\varepsilon_{\mathcal{X}}(\Diamond\phi) &= \Diamond\varepsilon_{\mathcal{X}}(\phi).
\end{aligned}$$

Then, we can prove, by structural induction on formulas, the following theorem:

Theorem 27 (Adequacy of syntax)**.** *The encoding function $\varepsilon_{\mathcal{X}}$ is a bijection between the modal logic formulas with free variables in $\mathcal{X}$ and the terms ϕ derivable in judgements $\Gamma \vdash_{\Sigma_\Box} \phi : \mathtt{o}$ in η-lnf, where $\Gamma = \{x : \mathtt{o} \mid x \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a formula ϕ, with free variables in $\mathcal{X} = \{x_1, \ldots, x_k\}$, and $\psi_1, \ldots, \psi_k$, with free variables in $\mathcal{Y}$, the following holds: $\varepsilon_{\mathcal{Y}}(\phi[\psi_1, \ldots, \psi_k/x_1, \ldots, x_k]) = \varepsilon_{\mathcal{X}}(\phi)[\varepsilon_{\mathcal{Y}}(\psi_1), \ldots, \varepsilon_{\mathcal{Y}}(\psi_k)/x_1, \ldots, x_k]$.*

If we denote by $\phi_1, \ldots, \phi_n \vdash \phi$ the derivation of the truth of a formula ϕ, depending on the assumptions $\phi_1, \ldots, \phi_n$, in Hilbert-style modal logics, the adequacy of our encoding can be stated by the following theorem:

Theorem 28 (Adequacy of the truth system)**.** *There exists a bijection between derivations $\phi_1, \ldots, \phi_k \vdash \phi$ in Hilbert-style modal logic and proof terms $\mathtt{h}$ such that $\Gamma \vdash_{\Sigma_\Box} \mathtt{h} : (\mathtt{True}\ \varepsilon_{\mathcal{X}}(\phi_1 \to \ldots \to \phi_k \to \phi))$ in η-long normal form, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is the set of propositional variables occurring in $\phi_1, \ldots, \phi_k, \phi$ and $\Gamma = \{x : \mathtt{o} \mid x \in \mathcal{X}\}$.*

9.2.2 Modal Logics in Natural Deduction Style.

In $\mathsf{LF}_{\mathcal{P}}$, one can also accommodate the classical Modal Logics S_4 and S_5 in Natural Deduction style. In particular, there are several alternative formulations for S_4 and S_5, *e.g.* as defined by Prawitz, which have rules with rather elaborate restrictions on the shape of subformulae where assumptions occur. In Figure 9.6 we present the rules allowing the specification of S_4 and S_5 (*à*

$$\frac{}{\Gamma, \phi \vdash \phi}\ (\mathsf{start})$$

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}\ (\to -\mathsf{I}) \qquad \frac{\Gamma \vdash \phi \to \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}\ (\to -\mathsf{E})$$

$$\frac{\Gamma, \neg\phi \vdash \mathit{ff}}{\Gamma \vdash \phi}\ (\mathit{ff} - \mathsf{I}) \qquad \frac{\Gamma \vdash \mathit{ff}}{\Gamma \vdash \phi}\ (\mathit{ff} - \mathsf{E})$$

$$\frac{\Delta \vdash \Box\Gamma \quad \Box\Gamma \vdash \phi}{\Delta \vdash \Box\phi}\ (\Box - \mathsf{I} \cdot \mathsf{S}_4) \qquad \frac{\Box\Gamma_0, \neg\Box\Gamma_1 \vdash \phi}{\Box\Gamma_0, \neg\Box\Gamma_1 \vdash \Box\phi}\ (\Box - \mathsf{I} \cdot \mathsf{S}_5)$$

$$\frac{\Gamma \vdash \Box\phi}{\Gamma \vdash \phi}\ (\Box - \mathsf{E}) \qquad \frac{\Gamma \vdash \Box(\phi \to \psi) \quad \Gamma \vdash \Box\phi}{\Gamma \vdash \Box\psi}\ (\to_{\Box} -\mathsf{E})$$

$$\frac{\emptyset \vdash \phi}{\emptyset \vdash \Box\phi}\ (\Box' - \mathsf{I}) \qquad \frac{\Gamma \vdash \Box\phi}{\Gamma \vdash \Box\Box\phi}\ (\Box_{\Box} - \mathsf{I}) \qquad \frac{\Gamma \vdash \Diamond\phi}{\Gamma \vdash \Box\Diamond\phi}\ (\Box_{\Diamond} - \mathsf{I})$$

NC	$\mathsf{start} + (\to -\mathsf{I}) + (\to -\mathsf{E}) + (\mathit{ff}{-}\mathsf{I}) + (\mathit{ff}{-}\mathsf{E})$
S_4	$NC + (\Box - \mathsf{I} \cdot \mathsf{S}_4) + (\Box - \mathsf{E})$
S_5	$NC + (\Box - \mathsf{I} \cdot \mathsf{S}_5) + (\Box - \mathsf{E})$
NK	$NC + (\to_{\Box} -\mathsf{E}) + (\Box' - \mathsf{I})$
NKT	$NK + (\Box - \mathsf{E})$
$NK4$	$NK + (\Box_{\Box} - \mathsf{I})$
$NKT4$	$NKT + (\Box_{\Box} - \mathsf{I})$
$NKT45$	$NKT4 + (\Box_{\Diamond} - \mathsf{I})$

Figure 9.6: Modal Logic rules in Natural Deduction style

la Prawitz), as well as the modal logics NK, NKT, $NK4$, $NKT4$, $NKT45$. So as to illustrate the flexibility of the system, the rule for $\Box$ introduction in S_4 ($(\Box\text{-I}\cdot S_4)$) allows cut-elimination.

The encodings in $\mathsf{LF}_{\mathcal{P}}$ of all of the aforementioned rules are presented in Figure 9.7. Again, the crucial role is played by three predicates, namely *Closed*, *BoxedS4* and *BoxedS5*.

```
        o : Type
       ff : o
       ¬ : o -> o
       → : o -> o -> o
       □ : o -> o
       ◊ : o -> o
    True : o -> Type
    impI : Πφ,ψ:o.(True(φ) -> True(ψ)) -> True(φ → ψ)
    impE : Πφ,ψ:o.True(φ → ψ) -> True(φ) -> True(ψ)
     ffI : Πφ:o.(True(¬φ) -> True(ff)) -> True(φ)
     ffE : Πφ:o.True(ff) -> True(φ)
  BoxIS4 : Πφ:o.Πm:True(φ).L_m^BoxedS4[True(□φ)]
  BoxIS5 : Πφ:o.Πm:True(φ).L_m^BoxedS5[True(□φ)]
    BoxE : Πφ:o.True(□φ) -> True(φ)
 impBoxE : Πφ,ψ:o.True(□(φ → ψ)) -> True(□φ) -> True(□ψ)
   BoxI' : Πφ:o. Πm:True(φ).L_m^Closed[True(□φ)]
 BoxBoxI : Πφ:o.True(□φ) -> True(□□φ)
BoxDiamondI : Πφ:o.True(◊φ) -> True(□◊φ)
```

Figure 9.7: The signature Σ_S for classic S_4 Modal Logic in $\mathsf{LF}_{\mathcal{P}}$

As in the Hilbert-style encoding, $\mathit{Closed}(\Gamma \vdash_\Sigma \mathtt{m}{:}\mathit{True}(\phi))$ holds if and only if "all of the free variables occurring in $\mathtt{m}$ have type $\mathtt{o}$". This is precisely what is needed to correctly encode the rule ($\Box'$-I), where the truth of the formula ϕ must not depend on any of the assumptions.

In the case of the modal logic S_4, the intended meaning of $\mathit{BoxedS4}\,(\Gamma \vdash_\Sigma \mathtt{m}{:}\ \mathtt{True}(\phi))$ is that all the occurrences of free variables of $\mathtt{m}$ occur in subterms whose type has the shape $\mathtt{True}(\Box\psi)$ or is $\mathtt{o}$, while in the case of the modal logic S_5, the predicate $\mathit{BoxedS5}\,(\Gamma \vdash_\Sigma \mathtt{m}{:}\ \mathtt{True}(\phi))$ holds if and only if all of the free variables of $\mathtt{m}$ have type of shape $\mathtt{True}(\Box\psi)$, $\mathtt{True}(\neg\Box\psi)$ or $\mathtt{o}$. It is easy to check that these predicates are well-behaved. Again, the "trick" to ensure a sound derivation in $\mathsf{LF}_\mathcal{P}$ of a proof of $\Box\phi$ is to lock appropriately the type $\mathtt{True}(\Box\phi)$ in the conclusion of the introduction rule $\mathtt{BoxI}$, in Figure 9.7.

As for the adequacy of our encoding, we can state Theorems 29 and 30 below. As in the previous case, we first define an encoding function $\varepsilon_\mathcal{X}$ on formulas with free variables in $\mathcal{X}$ as follows, representing atomic formulas by means of $\mathsf{LF}_\mathcal{P}$ metavariables:

$$\begin{aligned}
\varepsilon_\mathcal{X}(x) &= x, \text{ where } x \in \mathcal{X},\\
\varepsilon_\mathcal{X}(\mathit{ff}) &= \mathtt{ff},\\
\varepsilon_\mathcal{X}(\neg\phi) &= \neg\varepsilon_\mathcal{X}(\phi),\\
\varepsilon_\mathcal{X}(\phi \to \psi) &= \varepsilon_\mathcal{X}(\phi) \to \varepsilon_\mathcal{X}(\psi),\\
\varepsilon_\mathcal{X}(\Box\phi) &= \Box\varepsilon_\mathcal{X}(\phi),\\
\varepsilon_\mathcal{X}(\Diamond\phi) &= \Diamond\varepsilon_\mathcal{X}(\phi).
\end{aligned}$$

Then, we can prove, by structural induction on formulas, the following theorem:

Theorem 29 (Adequacy of syntax for modal formulas in natural deduction style)**.** *The encoding function $\varepsilon_\mathcal{X}$ is a bijection between the modal logic formulas with free variables in $\mathcal{X}$ and the terms ϕ derivable in judgements $\Gamma \vdash_\Sigma \phi : \mathtt{o}$ in η-lnf, where $\Gamma = \{x : \mathtt{o} \mid x \in \mathcal{X}\}$. Moreover, the encoding is compositional, i.e. for a formula ϕ, with free variables in $\mathcal{X} = \{x_1, \ldots, x_k\}$, and $\psi_1, \ldots, \psi_k$, with free variables in $\mathcal{Y}$, the following holds: $\varepsilon_\mathcal{Y}(\phi[\psi_1, \ldots, \psi_k/x_1, \ldots, x_k]) = \varepsilon_\mathcal{X}(\phi)[\varepsilon_\mathcal{Y}(\psi_1), \ldots, \varepsilon_\mathcal{Y}(\psi_k)/x_1, \ldots, x_k]$.*

The adequacy of the truth system of modal logics in natural deduction style can be proved by structural induction on derivations of the judgement $\Gamma \vdash \phi$:

Theorem 30 (Adequacy of modal logics in natural deduction style)**.** *Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of propositional variables occurring in formulas $\phi_1, \ldots, \phi_k, \phi$. There exists a bijection between derivations of the judgement $\{\phi_1, \ldots, \phi_k\} \vdash \phi$ in modal logics in natural deduction style, and proof terms $\mathtt{h}$ such that $\Gamma \vdash_\Sigma \mathtt{h} : (\mathtt{True}\ \varepsilon_\mathcal{X}(\phi))$ in η-lnf, where $\Gamma = \{x_1{:}\ \mathtt{o}, \ldots, x_n : \mathtt{o}$, $h_1 : (\mathtt{True}\ \varepsilon_\mathcal{X}(\phi_1)), \ldots, h_k : (\mathtt{True}\ \varepsilon_\mathcal{X}\ (\phi_k))\}$.*

9.2.3 Non-commutative Linear Logic

In this section, we outline an encoding in $\mathsf{LF}_\mathcal{P}$ of a substructural logic like the one presented in [Polakow 1999]. There, the authors presented a natural deduction system for intuitionistic non-commutative linear logic (NCLL), via a judgement of the form:

$$\Gamma; \Delta; \Omega \vdash M : A,$$

where M is a proof term, A is a formula, while Γ, Δ, and Ω are all sets of hypotheses, but with the following differences: Γ is an unrestricted intuitionistic context (allowing weakening, permutation, and contraction), Δ is a linear context (allowing only permutation), and Ω is an

ordered context. To illustrate, we present, for instance, the rules for the *ordered variables* and the introduction/elimination rules for the ordered right implication $\twoheadrightarrow$:

$$\frac{}{\Gamma;\cdot;z{:}A \vdash z{:}A}\ (\mathsf{ovar}) \qquad \frac{\Gamma;\Delta;(\Omega,z{:}A)\vdash M{:}B}{\Gamma;\Delta;\Omega\vdash \lambda^{>}z{:}A.M{:}A\twoheadrightarrow B}\ (\twoheadrightarrow \mathsf{I})$$

$$\frac{\Gamma;\Delta_1;\Omega_1\vdash M{:}A\twoheadrightarrow B \quad \Gamma;\Delta_2;\Omega_2\vdash N{:}A}{\Gamma;(\Delta_1\bowtie\Delta_2);(\Omega_1,\Omega_2)\vdash M^{>}N{:}B}\ (\twoheadrightarrow \mathsf{E})$$

where the $\bowtie$ symbol denotes the *context merge operator*.

In this system, we have that "ordered assumptions occur exactly once and in the order they were made". In order to encode the condition on the occurrence of z as the last variable in the ordered context in the introduction rule, it is sufficient to make the observation that, in an LF-based logical framework, this information is fully recorded in the proof term. The last assumption made is the rightmost variable, while the first is the leftmost. Therefore, we can, in $\mathsf{LF}_{\mathcal{P}}$, introduce suitable predicates in order to enforce such constraints, without resorting to complicated encodings.

In the following, we present an encoding of this ordered fragment of NCLL into $\mathsf{LF}_{\mathcal{P}}$. In order to give a shallow encoding, we do not represent explicitly the proof terms of the original system (see, e.g., [Polakow 1999]). The encodings of the rules $(\twoheadrightarrow \mathsf{I})$ and $(\twoheadrightarrow \mathsf{E})$ are:

```
impRightIntro : ΠA,B:o.ΠM:(True A)->(True B).L^{Rightmost}_{M,(True A)→(True B)}[(True (impRight A B))],
impRightElim : ΠA,B:o.(True (impRight A B)) -> (True A) -> (True B),
```

where `True : o -> Type` is the truth judgement on formulas (represented by type `o`) and `impRight : o -> o -> o` represents the $\twoheadrightarrow$ constructor of *right ordered implications*. Finally, the predicate ***Rightmost***$(\Gamma \vdash_{\Sigma} \mathtt{M} : \mathtt{(True\ A)\ ->\ (True\ B)})$ checks that `M` is an abstraction in normal form (*i.e.* $\mathtt{M} \equiv \lambda \mathtt{z} : \mathtt{(True\ A)}.\mathtt{M'}$, with `M'` in normal form), and that the bound variable `z` occurs only once, and as the rightmost free one in `M'`.

For what concerns the introduction/elimination rules for the left ordered implication $\rightarrowtail$, the encoding strategy is similar; indeed, the rules are the following:

$$\frac{\Gamma;\Delta;(z{:}A,\Omega)\vdash M{:}B}{\Gamma;\Delta;\Omega\vdash \lambda^{<}z{:}A.M{:}A\rightarrowtail B}\ (\rightarrowtail \mathsf{I})$$

$$\frac{\Gamma;\Delta_1;\Omega_1\vdash M{:}A\rightarrowtail B \quad \Gamma;\Delta_2;\Omega_2\vdash N{:}A}{\Gamma;(\Delta_1\bowtie\Delta_2);(\Omega_1,\Omega_2)\vdash M^{<}N{:}B}\ (\rightarrowtail \mathsf{E})$$

Therefore, we exploit the predicate ***Leftmost*** in the encoding of the introduction rule of $\rightarrowtail$:

```
impLeftIntro : ΠA,B:o.ΠM:(True A)->(True B).L^{Leftmost}_{M,(True A)→(True B)}[(True (impLeft A B))],
impLeftElim : ΠA,B:o.(True (impLeft A B)) -> (True A) -> (True B),
```

where `impLeft : o -> o -> o` represents the $\rightarrowtail$ constructor of *left ordered implications*. Finally, ***Leftmost***$(\Gamma \vdash_{\Sigma} \mathtt{M} : \mathtt{(True\ A)\ ->\ (True\ B)})$ checks that `M` is an abstraction in normal form (*i.e.* $\mathtt{M} \equiv \lambda \mathtt{z} : \mathtt{(True\ A)}.\mathtt{M'}$, with `M'` in normal form), and that the bound variable `z` occurs only once, and as the leftmost free one in `M'`.

The fragment of linear implications is, again, treated in a similar way, with a suitable predicate "ensuring" the correct introduction of the $\multimap$ constructor for linear implication. The rules

for the linear fragment of NCLL are the following:

$$\frac{}{\Gamma; y{:}A; \cdot \vdash y{:}A}\ (\mathsf{lvar}) \qquad \frac{\Gamma; (\Delta, y{:}A); \Omega \vdash M{:}B}{\Gamma; \Delta; \Omega \vdash \hat{\lambda} y{:}A.M{:}A \multimap B}\ (\multimap \mathsf{I})$$

$$\frac{\Gamma; \Delta_1; \Omega \vdash M{:}A \multimap B \quad \Gamma; \Delta_2; \cdot \vdash N{:}A}{\Gamma; (\Delta_1 \bowtie \Delta_2); \Omega \vdash M \hat{} N{:}B}\ (\multimap \mathsf{E})$$

Hence, the encodings of the rules $(\multimap \mathsf{I})$ and $(\multimap \mathsf{E})$ are as follows:

```
impLinearIntro : ΠA,B:o.ΠM:(True A)->(True B).L^Linear_{M,(True A)→(True B)}[(True (impLinear A B))],
 impLinearElim : ΠA,B:o.(True (impLinear A B)) -> (True A) -> (True B),
```

where `impLinear : o -> o -> o` represents the $\multimap$ constructor of *linear implications*. Finally, the predicate $\mathit{Linear}\,(\Gamma \vdash_\Sigma$ `M:(True A)->(True B)`$)$ verifies that `M` is an abstraction in normal form (*i.e.* `M` $\equiv \lambda$`z` : `(True A).M'`, with `M'` in normal form), and that the bound variable `z` occurs free only once in `M'`.

Finally, the encoding of the intuitionistic fragment of NCLL is straightforward, since in this part there are no restrictions about the intuitionistic variables:

$$\frac{}{(\Gamma_1, x{:}A, \Gamma_2); \cdot; \cdot \vdash x{:}A}\ (\mathsf{ivar}) \qquad \frac{(\Gamma, x{:}A); \Delta; \Omega \vdash M{:}B}{\Gamma; \Delta; \Omega \vdash \lambda x{:}A.M{:}A \to B}\ (\to \mathsf{I})$$

$$\frac{\Gamma; \Delta; \Omega \vdash M{:}A \to B \quad \Gamma; \cdot; \cdot \vdash N{:}A}{\Gamma; \Delta; \Omega \vdash MN{:}B}\ (\to \mathsf{E})$$

The encodings of the introduction/elimination rules for the intuitionistic implication are :

```
impIntro : ΠA,B:o.ΠM:(True A)->(True B). (True (imp A B)),
 impElim : ΠA,B:o.(True (imp A B)) -> (True A) -> (True B)
```

Notice that in the encodings of rules $\twoheadrightarrow_E$, $\rightarrowtail E$ and $\multimap E$ we have not enforced any conditions on the free variables occurring in the terms, in order to avoid infringing the *closure under substitution* condition for well-behavedness of predicates. Indeed, the obvious requirements surface in the following adequacy theorem:

Theorem 31 (Adequacy)**.** *Let $\mathcal{X} = \{P_1, \ldots, P_n\}$ be a set of atomic formulas (in the sense of [Polakow 1999]) occurring in formulas $A_1, \ldots, A_k, A$. Then, there exists a bijection between derivations of the judgement $(A_1, \ldots, A_{i-1}); (A_i, \ldots, A_{j-1}); (A_j, \ldots, A_k) \vdash A$ in NCLL, and proof terms* `h` *such that* $\Gamma_{\mathcal{X}}, h_1$:`(True` $\varepsilon_{\mathcal{X}}(A_1)$`)`$, \ldots,$ h_k:`(True` $\varepsilon_{\mathcal{X}}(A_k)$`)` $\vdash$ `h` : `(True` $\varepsilon_{\mathcal{X}}(A)$`)` *in η-long normal form, where the variables $h_i, \ldots, h_{j-1}$ occur in* `h` *only once, $h_j, \ldots, h_k$ occur in* `h` *only once and precisely in this order, and $\Gamma_{\mathcal{X}}$ is the context P_1 : o, $\ldots$, P_n : o, representing the object-language atomic formulas $P_1, \ldots, P_n$.*

As far as we know, this is the first example (see the discussion in, *e.g.* [Crary 2010]) of an encoding of non-commutative linear logic in an LF-like framework. Notice the peculiarity of this adequacy result, which is inevitable given the substructural nature of NCLL, but which is, nonetheless, perfectly *compositional*. The gist is the following: as far as theorems, *i.e.* proofs with no assumptions go, everything is standard; when assumptions, *i.e. truly free*, and not *bindable* variables are involved, an external requirement has to be externally checked. An alternate formulation of adequacy could be stated representing, as in the λ-calculus case, truly free variables by constants. Obviously, carrying out a deep embedding of the system, one could enforce the conditions on the variables occurring in the linear and ordered contexts by means of suitable locks at the level of the proof terms.

9.3 Imp with Hoare Logic

In this subsection, we give an encoding of an imperative language called Imp, together with its Hoare Logic. The syntax of programs in Imp is:

$$\begin{array}{rcll} p & ::= & \mathit{skip} & \text{null} \\ & & \mid x := \mathit{expr} & \text{assignment} \\ & & \mid \mathit{if}\ \mathit{cond}\ \mathit{then}\ p\ \mathit{else}\ p & \text{condition} \\ & & \mid p; p & \text{sequence} \\ & & \mid \mathit{while}\ \mathit{cond}\ \{p\} & \text{while} \end{array}$$

Other primitive notions of our object language are variables, both integer and *identifier*, and expressions. Here, identifiers denote locations. For the sake of simplicity, we assume that only integers (represented by the type `int`) as possible values for identifiers. Importantly, we follow as closely as possible the HOAS encoding, originally proposed in [Avron 1992], in order to illustrate the features and possible advantages of using $\mathsf{LF}_{\mathcal{P}}$ with respect to LF. The main difference with that approach is that here we encode *concrete* identifiers by constants of type `var`, an `int`-like type, of course different from `int` itself, so as to avoid confusion with possible values of locations. The syntax of variables and expressions is defined as follows:

Definition 28 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ for Imp)**.**

```
 int : Type                          bool : Type                          var : Type
bang : var -> int                  0,1,-1 : int                 + : int -> int -> int
  =  : int -> int -> bool                                  and, imp : bool -> bool -> bool
 not : bool -> bool                                         forall : (int -> bool) -> bool
```

Since variables of type `int` may be bound in expressions (by means of the `forall` constructor), we define explicitly the encoding function $\varepsilon^{exp}_{\mathcal{X}}$ mapping expressions with bindable variables of type `int` in $\mathcal{X}$ of the source language Imp into the corresponding terms of $\mathsf{LF}_{\mathcal{P}}$:

$$\begin{array}{rcl} \varepsilon^{exp}_{\mathcal{X}}(0) = \texttt{0},\quad \varepsilon^{exp}_{\mathcal{X}}(1) & = & \texttt{1},\quad \varepsilon^{exp}_{\mathcal{X}}(-1) = \texttt{-1} \\ \varepsilon^{exp}_{\mathcal{X}}(x) & = & \begin{cases} x \text{ if } x \in \mathcal{X} \\ (\texttt{bang x}) \text{ if } x \notin \mathcal{X} \end{cases} \\ \varepsilon^{exp}_{\mathcal{X}}(n+m) = (\texttt{+}\ \varepsilon^{exp}_{\mathcal{X}}(n)\, \varepsilon^{exp}_{\mathcal{X}}(m)), & & \varepsilon^{exp}_{\mathcal{X}}(n=m) = (\texttt{=}\ \varepsilon^{exp}_{\mathcal{X}}(n)\, \varepsilon^{exp}_{\mathcal{X}}(m)) \\ \varepsilon^{exp}_{\mathcal{X}}(\neg e) = (\texttt{not}\ \varepsilon^{exp}_{\mathcal{X}}(e)), & & \varepsilon^{exp}_{\mathcal{X}}(e \wedge e') = (\texttt{and}\ \varepsilon^{exp}_{\mathcal{X}}(e)\, \varepsilon^{exp}_{\mathcal{X}}(e')) \\ \varepsilon^{exp}_{\mathcal{X}}(e \supseteq e') = (\texttt{imp}\ \varepsilon^{exp}_{\mathcal{X}}(e)\, \varepsilon^{exp}_{\mathcal{X}}(e')), & & \varepsilon^{exp}_{\mathcal{X}}(\forall x.\phi) = (\texttt{forall}\ \lambda\texttt{x:int}.\varepsilon^{exp}_{\mathcal{X}\cup\{x\}}(\phi)) \end{array}$$

where `x` in `(bang x)` denotes the encoding of the concrete memory location (*i.e.* a constant of type `var`), representing the (free) source language identifier x; the other case represents the bindable variable x rendered as a $\mathsf{LF}_{\mathcal{P}}$ metavariable of type `int` in HOAS style. The syntax of imperative programs is defined as follows:

Definition 29 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ for Imp with command)**.**
We extend the signature of Definition 28 as follows:

```
prog   : Type
Iskip  : prog
Iseq   : prog -> prog -> prog
Iset   : var -> int -> prog
Iif    : Πe:bool.prog -> prog -> L^QF_{e,bool}[prog]
Iwhile : Πe:bool.prog -> L^QF_{e,bool}[prog]
```

where the predicate $\mathit{QF}(\Gamma \vdash_\Sigma \texttt{e:bool})$ holds iff the formula `e` is *closed* and *quantifier free*, *i.e.* it does not contain the `forall` constructor. We can look at QF as a "good formation" predicate, ruling out *bad programs with invalid boolean expressions* by means of locked terms.

The encoding function $\varepsilon^{prog}_{\mathcal{X}}$ mapping programs with free variables in $\mathcal{X}$ of the source language Imp into the corresponding terms of $\mathsf{LF}_{\mathcal{P}}$ is defined as follows:

$$\begin{aligned}
\varepsilon^{prog}_{\mathcal{X}}(skip) &= \texttt{Iskip}\\
\varepsilon^{prog}_{\mathcal{X}}(x := e) &= \texttt{Iset x } \varepsilon^{exp}_{\mathcal{X}}(e)\\
\varepsilon^{prog}_{\mathcal{X}}(p; p') &= \texttt{Iseq } \varepsilon^{prog}_{\mathcal{X}}(p)\ \varepsilon^{prog}_{\mathcal{X}}(p')\\
\varepsilon^{prog}_{\mathcal{X}}(\mathit{if}\ e\ \mathit{then}\ p\ \mathit{else}\ p') &= \mathcal{U}^{QF}_{\varepsilon^{exp}_{\mathcal{X}}(e),\texttt{bool}}[(\texttt{Iif } \varepsilon^{\texttt{exp}}_{\mathcal{X}}(\texttt{e})\ \varepsilon^{\texttt{prog}}_{\mathcal{X}}(\texttt{p})\ \varepsilon^{\texttt{prog}}_{\mathcal{X}}(\texttt{p}'))] \qquad (*)\\
\varepsilon^{prog}_{\mathcal{X}}(\mathit{while}\ e\ \{p\}) &= \mathcal{U}^{QF}_{\varepsilon^{exp}_{\mathcal{X}}(e),\texttt{bool}}[(\texttt{Iwhile } \varepsilon^{\texttt{exp}}_{\mathcal{X}}(\texttt{e})\ \varepsilon^{\texttt{prog}}_{\mathcal{X}}(\texttt{p}))] \qquad (*)
\end{aligned}$$

where (*) denotes that e is a quantifier-free formula (we are assuming to encode *legal* programs). Now, given the predicate `true: bool -> Type` such that `(true e)` holds if and only if `e` is true, we can define a judgment `hoare` as follows:

Definition 30 ($\mathsf{LF}_{\mathcal{P}}$ signature Σ for Hoare)**.**

```
args : Type
     <_ , _> : var -> (int -> bool) -> args
       hoare : bool -> prog -> bool -> Type
 hoare_Iskip : Πe:bool.(hoare e Iskip e)
  hoare_Iset : Πt:int.Πx:var.Πe:int->bool.L^{P set}_⟨x,e⟩[(hoare (e t) (Iset x t) (e (bang x))]
  hoare_Iseq : Πe,e',e'':bool.Πp,p':prog.(hoare e p e') ->
                                         (hoare e' p' e'') ->
                                         (hoare e (Iseq p p') e'')
   hoare_Iif : Πe,e',b:bool.Πp,p':prog.(hoare (b and e) p e') ->
                                       (hoare ((not b) and e) p' e') ->
                                       (hoare e U^{QF}_{b,bool}[(Iif b p p')] e')
hoare_Iwhile : Πe,b:bool.Πp:prog.(hoare (e and b) p e) ->
                                 (hoare e U^{QF}_{b,bool}[(Iwhile b p)] ((not b) and e))
 hoare_Icons : Πe,e',f,f':bool.Πp:prog.(true (imp e' e)) ->
                                       (hoare e p f) ->
                                       (true (imp f f')) ->
                                       (hoare e' p f'),
```

where $\mathit{P}^{set}(\Gamma \vdash_\Sigma \langle \texttt{x,e}\rangle\texttt{:args})$ *holds if and only if* `e` *is closed*[3]*, and the location (i.e. constant)* `x` *does not occur in* `e`*.*

Such requirements amount to formalizing that no assignment made to the location denoted by `x` affects the "meaning" or value of `e` (*non-interference* property). The intuitive idea here is that if $\texttt{e} = \varepsilon^{exp}_{\mathcal{X}}(E)$, $\texttt{p} = \varepsilon^{prog}_{\mathcal{X}}(P)$ and $\texttt{e}' = \varepsilon^{exp}_{\mathcal{X}}(E')$, then `(hoare e p e')` holds if and only if the Hoare's triple $\{E\}P\{E'\}$ holds. The advantage with respect to previous encodings (see, e.g., [Avron 1992]), is that in $\mathsf{LF}_{\mathcal{P}}$ we can delegate to the external predicates QF and P^{set} all the complicated and tedious checks concerning *non-interference* of variables and good formation clauses for guards in the conditional and looping statements. Thus, the use of lock types, which are subject to the verification of such conditions, allows us to legally derive $\Gamma \vdash_\Sigma$ `m:(hoare e p e')` only according to the Hoare semantics.

[3]Otherwise, the predicate P would not be well-behaved, in the sense of Definition 21.

9.4 Probabilistic SAT-Checkers as Oracles for LF$_\mathcal{P}$

In this section, we provide an outline of how, for instance, a probabilistic SAT-checker based on one of the logics presented in Part 1 of this thesis could be used as an external oracle in the LF$_\mathcal{P}$ framework, as part of a larger system.

For the purposes of this illustration, we will take into account the probabilistic logic $LPP_2^{\mathbb{Q}}$, presented in Chapter 3, where there exist two layers of formulas - classical and probabilistic, and the reasoning is essentially propositional. Encoding the formulas of this logic in LF$_\mathcal{P}$ is natural and follows the Coq encoding practically to the letter:

```
  pp : Type       base : pp -> forC              Pge : Q01 -> forC -> forP
forC : Type       negC : forC -> forC          negP : forP -> forP
forP : Type       impC : forC -> forC -> forC  impP : forP -> forP -> forP

 FOR : Type       Clas : forC -> FOR           Prob : forP -> FOR
```

where `Q01` is the type of rational numbers from the unit interval, which can, for instance, be defined as pairs of natural numbers such that the first is always greater than or equal to the second, and the second is necessarily greater than zero, and `pp` is the type of propositions.

One way in which we could use this logic as an oracle is by providing a predicate `Consistent`, which takes as an input a list[4] of formulas $f_1, \ldots f_n$ of type `FOR`, and returns true if and only if each of the formulas from the list is closed, and the list itself, when treated as a set, is $LPP^{\mathbb{Q}}_{2,Meas}$-satisfiable[5]. This check would be external to LF$_\mathcal{P}$, and would be accomplished by an appropriate probability SAT-checker. Since $LPP_2^{\mathbb{Q}}$ is propositional, we do not have any binders to consider, and since we require the formulas in question to be closed, the predicate `Consistent` is indeed well-behaved. This predicate could then be used as part of a larger (possibly decision-support) system encoded in LF$_\mathcal{P}$, which would make use of its own elements as propositions.

$$\mathcal{L}^{\texttt{Satisfiable}}_{<\texttt{f}_1,\ldots,\texttt{f}_n>,\texttt{list}}[M] : \mathcal{L}^{\texttt{Satisfiable}}_{<\texttt{f}_1,\ldots,\texttt{f}_n>,\texttt{list}}[\rho]$$

Here, we can again see the advantages of LF$_\mathcal{P}$ with respect to LF, as the encoding of a probabilistic SAT-checker in LF would have been a very difficult, if not an impossible task.

9.5 Several further comments on LF$_\mathcal{P}$

Arguably, one might still wonder whether develop all of the meta-theory presented in this thesis it worth it. Why should one choose to outsource parts of the formalization and verification of an object system to an external tool, when one could simply take a "monolithic" stance, encoding and checking everything within his proof assistant of choice. While this is a completely legitimate strategy, there exist several drawbacks to be addressed.

Let us give another practical application of the Poincaré Principle. Let us take ourselves to the setting of a formalization of π-calculus (see, *e.g.*, [Honsell 2001]), where we would like to prove rigorously that if a certain property $\mathcal{P}$ holds on a process P, then the same property also holds on the process $(\nu x)P$, under the condition that $x \notin fn(P)$. On paper, it is easy to conclude this, since we know that $P \equiv (\nu x)P$ under the aforementioned freshness assumption. However, proof assistants force the user to spell out in full detail even trivial proofs like $P \equiv (\nu x)P$.

[4] The encoding of a list of formulas in LF$_\mathcal{P}$, can be accomplished in the usual way, by defining the constructors `nil : list` and `cons : FOR -> list -> list`, for instance.

[5] Since we have the Strong Completeness Theorem for $LPP_2^{\mathbb{Q}}$ (Theorem 9), we can interchange at will the terms "consistent" and "satisfiable".

Therefore, if we were to formally prove the previous structural congruence statement, we would have to proceed by applying the basic axioms:

$$\begin{array}{rl} P|0 \equiv P & 0 \text{ is the identity w.r.t. parallel composition} \\ P|(\nu x)0 \equiv P & (\nu x)0 \equiv 0 \\ (\nu x)(P|0) \equiv P & (\nu x)(P|Q) \equiv P|(\nu x)Q \text{ if } x \notin fn(P) \\ (\nu x)P \equiv P & 0 \text{ is the identity w.r.t. parallel composition} \end{array}$$

Here, we illustrate only the main steps of the proof, using structural rules, transitivity and symmetry implicitly. Surely, this is rather tedious and does not require any particular skill to be carried out: it is a trivial *verification*-like checking, *e.g.* that $2 + 2 = 4$ ([Poincaré 1902, Barendregt 2002]). Of course, one will do the proof once and then save it for future re-use. However, at the level of the machinery behind the proof assistant, this implies larger proof terms, more memory consumption and, in general, a slowdown in the overall performance[6]. Thus, the opportunity of delegating such straightforward and trivial verifications to an external (and possibly optimized) tool would be very helpful during complicated formal proof developments. Furthermore, the user of the proof assistant would be free to concentrate only on the "creative" part of the entire proof and the framework itself would be free to avoid an explicit treatment of the "uninteresting" parts of the proof. Precisely this is an important aspect of the spirit behind the design of $\mathsf{LF}_{\mathcal{P}}$: allowing the user to factorize apart consolidated proof knowledge, freeing himself and the framework of recording "useless" and trivial verifications in full detail.

To illustrate this outsourcing further, in the case of the π-calculus, the reduction rule taking into account structural congruences between processes, namely

$$\frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$$

can be easily encoded in $\mathsf{LF}_{\mathcal{P}}$ using locked types in the following manner:

$$\mathcal{L}^{\mathit{Struct}}_{\langle \mathtt{P},\mathtt{P}',\mathtt{Q}',\mathtt{Q}\rangle}[(\mathtt{red\ P\ Q})]$$

where the `red` symbol serves to encode the reduction relation $\longrightarrow$, and the *external* predicate `Struct` holds if and only if $P \equiv P'$ and $Q' \equiv Q$.

Moreover, one can easily incorporate other systems separating derivation and computation within $\mathsf{LF}_{\mathcal{P}}$. For example, the rule

$$\frac{C \qquad A \to B \qquad A \equiv C}{B}$$

in *Deduction Modulo* can be represented as:

$$\supseteq_{\equiv}\!: \Pi A, B, C{:}o.\Pi x{:}\mathit{True}(A \to B).\Pi y{:}\mathit{True}(C).\mathcal{L}^{\equiv}_{\langle A,C\rangle}[\mathit{True}(B)].$$

However, the mechanism of locked/unlocked types in $\mathsf{LF}_{\mathcal{P}}$ is more general than the one provided by Deduction Modulo or the Poincaré principle. The latter can be viewed as extensions of the type Equality Rule to new definitional equalities, while $\mathsf{LF}_{\mathcal{P}}$, on the other hand, allows one to reflect into the proof objects themselves, as has been extensively shown in the examples. As a final illustration, we could address formally the issue of reflection in $\mathsf{LF}_{\mathcal{P}}$. We can already grasp the gist of this philosophy through the following principle:

$$\mathtt{Reflection\ :\ \Pi x{:}o.}\ \mathcal{L}^{\mathit{isTrue}}_{x,o}[(\mathit{True}\ x)]\,.$$

[6] As we already pointed out in the introduction, proof terms can be kept small, adopting a reflection approach (see, e.g., [Chlipala 2013]), but at the price of proving the correctness of the decision procedures internally.

9.6 Summary

In this chapter, we have presented several important case studies of encodings in $\mathsf{LF}_{\mathcal{P}}$, illustrating the potential and advantages of this framework with respect to standard LF. We have encoded the untyped λ-calculus with the call-by-value reduction strategy and shown how it can be further extended to incorporate design-by-contract. Next, we have presented encodings of various modal logics, as well as the first encoding of non-commutative linear logic in an LF-type framework. Lastly, we have shown how to encode a small imperative language featuring Hoare logic, and given an outline of how a probabilistic SAT-checker could be used as an external oracle of $\mathsf{LF}_{\mathcal{P}}$. All the while, we have been relying on locked types and external predicates to deal with the side conditions of the formal systems in question, thus obtaining an elegant separation between derivation and computation.

Part III

Conclusions

Chapter 10

Concluding Remarks and Directions for Further Work

Contents

10.1 Concluding Remarks

In this thesis, we have documented the research efforts belonging to two notable fields of mathematics - mathematical logic and type theory - two fields which are separate yet intertwined, brought together by the Curry-Howard correspondence.

Probability Logics. Following an appropriate introduction into the matter, in the first part of the thesis we have introduced one new probability logic with conditional operators - LPCP, and formally verified the key properties (soundness, strong and simple completeness, and non-compactness) of another two - $LPP_1^{\mathbb{Q}}$ and $LPP_2^{\mathbb{Q}}$.

The logic LPCP- a probability logic with conditional probability operators $CP_i, i \in \mathcal{I}$, which allows for linear combinations and comparative statements on the probabilites of formulas, has been introduced in the second chapter, along with its syntax, semantics, and a sound and strongly-complete axiomatic system, featuring an infinitary rule. The obtained formalism was shown to be fairly expressive and allows for the representation of uncertain knowledge, where uncertainty is modeled by probability formulas. It has also been shown to be decidable, with a PSPACE containment for the decision procedure, and was extended so as to represent evidence, making it the first propositional axiomatization of reasoning about evidence.

In the third and fourth chapters, we have presented probability logics $LPP_1^{\mathbb{Q}}$ and $LPP_2^{\mathbb{Q}}$, which allow for reasoning on the probability of events, together with their encodings in the proof assistant Coq. Each of these two logics has its own specificities. They both involve the extension of classical logic with modal-like probability operators $\mathcal{P}_{\geq r}$, for $r \in \mathbb{Q}_{[0,1]}$, and both are equipped with an infinitary inference rule, in order to capture strong completeness. In $LPP_2^{\mathbb{Q}}$, we have an explicit separation between classical and probabilistic formulas, resulting in a two-layered proof structure, with a probabilistic layer built atop classical logic. This prohibits the mixing of classical and probabilistic formulas and the iterations of probability operators. $LPP_1^{\mathbb{Q}}$, on the other hand, is more expressive, with no distinction between formulas, and allows iterations of probability operators. For each of these two logics, we have encoded their syntax, semantics, and axiom systems, and provided formal proofs of several important meta-theorems, notably soundness, strong and simple completeness, and non-compactness. In this way, we have formally justified the interchangeability of the syntactic and the semantic level for these logics, *i.e.* justified the use of probabilistic SAT-solvers for the checking of consistency-related questions.

The $\mathsf{LF}_{\mathcal{P}}$ Framework In the second part of the thesis, in chapters five through nine, we have presented a new Logical Framework with External Predicates, building on the well-known LF, by introducing a mechanism for locking and unlocking types and terms, allowing the use of external verification tools. We have proven that $\mathsf{LF}_{\mathcal{P}}$ satisfies the main meta-theoretic properties: strong normalization, confluence, and, with certain well-behavedness conditions enforced, subject reduction. We have also developed a corresponding canonical framework - CLFP, allowing for easy proofs of the adequacy of encodings. We have provided a number of encodings, including the simple untyped λ-calculus with a call-by-value reduction strategy, the Design-by-Contract paradigm, a small imperative language with Hoare logic, as well as Modal Logics in Hilbert and Natural Deduction style, and Non-Commutative Linear Logic, illustrating that in $\mathsf{LF}_{\mathcal{P}}$ we can encode side-conditions on the application of rules elegantly, and achieve, when necessary, a separation between verification and computation, resulting in cleaner and more readable proofs.

10.2 Further Work

Development of Probability Logics. As far as further research is concerned with respect to the logic LPCP, a similar approach could be investigated with conditional probabilities in the style of de Finetti. One alternative approach would consider dealing with first-order probabilistic formulas and has already been researched, as a continuation of the work presented in this chapter, in [Milošević 2012], where the authors have extended the formulas of LPCP to first-order formulas, provided an infinitary axiom system, and proven its soundness and strong completeness with resepct to the considered class of first-order Kripke-like models, and decidability, provided the underlying first-order theory is decidable. Also, in [Doder 2012], the authors have presented a probabilistic temporal logic which can model reasoning about evidence.

Formal Verification of Probability Logics. In this segment, we can think of two main continuations. First, the ideas used in the Coq encodings of the already verified logics can be re-used for logics which are their extensions or variations. For instance, an addition of a qualitative operator on probabilities can be handled easily. Also, once the Coq tools for handling real or non-standard analysis have reached the required level, it would be possible to extend the co-domain of the measure to real numbers or the non-standard rational unit interval. Although the code is not re-usable *per se*, the making of appropriate modifications and adjustments to the code would not require more than one week for a Coq user familiar with the proofs.

On the other hand, the next property of the already encoded logics which could be addressed formally is their decidability. Given a formal specification of the appropriate decision procedure in Coq, it would be possible to, for the first time, obtain a certified probabilistic SAT-solver. Of course, the attention here should also be directed at various optimizations of the decision process, in an effort to yield a robust and (relatively) fast code.

Prospects for $\mathsf{LF}_{\mathcal{P}}$. There exist several directions that we could consider. First, one can notice that the $\mathsf{LF}_{\mathcal{P}}$ system presented in this thesis is a purely first-order predicative type theory, corresponding to the vertex (1,0,0) of Barendregt's cube [Barendregt 1992]. One reasonable and worthwhile step further would be to try to extend it to the full impredicative higher-order Calculus of Constructions, which, given the meta-theoretic results presented in this thesis, and the manner in which they were proven, does not appear to be too difficult a task, while it would bring us a significant additional degree of expressiveness to be explored.

Next, we could investigate alternative presentations of $\mathsf{LF}_{\mathcal{P}}$, featuring, for instance, typed reductions, or removing the unlock destructors from the system altogether and delegating the checking of the validity of the predicates to $\mathcal{L}$-reduction. In this way, we would not require

the well-behavedness condition concerning closure under signature and context weakening and permutation, but we would require the predicates to be well-behaved for all meta-theoretical results. Also, in certain cases, we might even wish to keep track of the external calls made during the derivation, and one way in which this could be accomplished is to employ a variation of $\mathsf{LF}_{\mathcal{P}}$ in which $\mathcal{L}$-reductions do not fire, thus preserving the $\mathcal{U}$-$\mathcal{L}$ pairs within the term. More practical experimentations with $\mathsf{LF}_{\mathcal{P}}$ will provide more insights on these issues.

Also, the machinery of derivations with locked types is similar to δ-rules *à la* Mitschke (*cf.* [Barendregt 2013]), when we view lock rules, at the object level, as δ-rules releasing their argument once the condition has been satisfied. This connection could be investigated further. For instance, we could use the untyped object language of $\mathsf{LF}_{\mathcal{P}}$ to support the design-by-contract programming paradigm. Moreover, we believe that $\mathsf{LF}_{\mathcal{P}}$ can shed light on various concepts in need of better formal understanding, such as the topic shallow vs. deep encodings, proposition-as-types for modal operators, and combinations of different validation tools.

Next, it would be interesting to study the possibility of "embedding" the Higher-order Term Rewriting Systems, as in [Virga 1999], as well as the constraint domains of Twelf in $\mathsf{LF}_{\mathcal{P}}$ as external predicates, since the constraints imposed on the rewriting rules seem closely related to our well-behavedness properties (see Definition 21). Indeed, just as in [Virga 1999], we have also used Newman's Lemma (see Chapter 7) to prove confluence of our system, and the issue of preserving well-typedness of expressions has also been our main concern throughout the development of the meta-theory of $\mathsf{LF}_{\mathcal{P}}$.

On a different note, we believe that the $\mathsf{LF}_{\mathcal{P}}$ framework could also find its usage in modeling dynamic and reactive systems, such as bio-inspired systems, where reactions of chemical processes take place only if certain additional structural or temporal conditions hold, as well as process algebras, where no assumptions can be made about messages exchanged through the communication channels. There, it could be the case that a redex, depending on the result of a communication, can remain stuck until a "good" message arrives from a given channel, firing in that case an appropriate reduction (a common situation in many protocols, where "bad" requests are ignored and "good ones" are served). Such dynamic (run-time) behavior could hardly be captured by a rigid type discipline, where "bad" terms and hypotheses are ruled out *a priori* ([Nanevski 2008]).

Last, but by no means least, building a prototype of $\mathsf{LF}_{\mathcal{P}}$ is under consideration and is expected to be carried out in the near future. Although there are several important questions to be answered, such as whether to build the prototype from scratch or attempt to modify the kernel of one of the currently existing proof assistants, or how to design an appropriate interface between the prototype and the external oracles, we believe that this development, when completed, would present an intriguing new take on the concept of proof assistants.

Bibliography

[Abadi 1994] M. Abadi and J. Y. Halpern. *Decidability and expressiveness for first-order logics of probability.* Information and Computation, vol. 112, pages 1–36, 1994. (Cited on page 6.)

[Audebaud 2009] P .Audebaud. C. Paulin-Mohring. *Proofs of Randomized Algorithms in Coq.* Science of Computer Programming, vol. 74, no. 8, pages 568-589, 2009. (Cited on page 17.)

[Avron 1992] A. Avron, F. Honsell, I. A. Mason and R. Pollack. *Using Typed Lambda Calculus to Implement Formal Systems on a Machine.* Journal of Automated Reasoning, vol. 9, pages 309–354, 1992. (Cited on pages 97, 105, 112 and 113.)

[Avron 1998] A. Avron, F. Honsell, M. Miculan and C. Paravano. *Encoding Modal Logics in Logical Frameworks.* Studia Logica, vol. 60, no. 1, pages 161–208, 1998. (Cited on pages 66, 85, 97 and 105.)

[Barendregt 2013] H. Barendregt, W. Dekkers and R. Statman. Lambda Calculus with Types. Cambridge University Press, 2013. (Cited on pages 8 and 121.)

[Barendregt 1992] H. Barendregt. *Lambda Calculi with Types.* In Handbook of Logic in Computer Science, volume II, pages 118–310. Oxford University Press, 1992. (Cited on pages 2, 10 and 120.)

[Barendregt 2002] H. Barendregt and E. Barendsen. *Autarkic Computations in Formal Proofs.* Journal of Automated Reasoning, vol. 28, pages 321–336, 2002. (Cited on pages 66, 67, 68 and 115.)

[Barthe 2003] G. Barthe, H. Cirstea, C. Kirchner and L. Liquori. *Pure Pattern Type Systems.* In POPL'03, pages 250–261. The ACM Press, 2003. (Cited on page 68.)

[Bayes 1764] T. Bayes. *An Essay towards Solving a Problem in the Doctrine of Chances.* London, 1764. (Cited on page 3.)

[Bernoulli 1713] J. Bernoulli. *Ars Conjectandi.* Basel, 1713. (Cited on page 3.)

[Bernstein 1917] S. Bernstein. Versuch einer axiomatischen Begründung der Wahrscheinlichkeitsrechnung. *Mitteilung der Math. Gesselschaft von Charkow*, vol. 15, pages 209 – 274, 1917. (Cited on page 4.)

[Bertot 2004] Y. Bertot and P. Castéran. Interactive Theorem Proving and Program Development, Coq'Art: the Calculus of Inductive Cconstructions. Texts in Theoretical Computer Science. Springer-Verlag, 2004. (Cited on page 14.)

[Blanqui 2008] F. Blanqui, J.-P. Jouannaud and P.-Y. Strub. *From Formal Proofs to Mathematical Proofs: a Safe, Incremenntal Way for Building in First-Order Decision Procedures.* In IFIP TCS, volume 273, pages 349–365, 2008. (Cited on page 68.)

[Bohlmann 1901] G. Bohlmann. *Encyklopädie der mathematischen Wissenschaften*, vol. I, part 2, item 4b, pages 852 – 917, 1901. (Cited on page 4.)

[Boole 1847] G. Boole. The Mathematical Analysis of Logic. 1847. (Cited on page 4.)

[Boole 1854] G. Boole. An Investigation into the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probabilities. 1854. (Cited on page 4.)

[Borel 1924] E. Borel. Traite du Calcul des Probabilités. 1924. (Cited on page 4.)

[Borel 1925] E. Borel. Principes et Formules Classiques du Calcul des Probabilités. 1925. (Cited on page 4.)

[Boyer 1995] R. S. Boyer, M. Kaufmann and J. S. Moore. *The Boyer-Moore Theorem Prover and Its Interactive Enhancement*, 1995. (Cited on page 14.)

[Cardone 2005] F. Cardone, J. R. Hindley. *Lambda-Calculus and Combinators in the 20th Century* in Handbook of the History of Logic, vol. 5, pages 723-817, Elsevier, 2009. (Cited on page 7.)

[Carnap 1950] R. Carnap. Logical Foundations of Probability. University of Chicago Press. 1950. (Cited on page 4.)

[Carnap 1952] R. Carnap. The Continuum of Inductive Methods. University of Chicago Press. 1952. (Cited on page 4.)

[Carnap 1962] R. Carnap. Logical Foundations of Probability (2^{nd} edition). University of Chicago Press, 1962. (Cited on page 33.)

[Carroll 1895] L. Carroll. *What the Tortoise Said to Achilles.* Mind, vol. 4, pages 278–280, 1895. (Cited on page 67.)

[Chlipala 2013] A. Chlipala. *The Reflection Library.* Available on-line at: `http://adam.chlipala.net/cpdt/html/Reflection.html`, 2013. (Cited on pages 67 and 115.)

[Church 1932] A. Church. *A Set of Postulates for the Foundation of Logic.* Annals of Mathematics, Series 2, vol. 33, pages 346–366, 1932. (Cited on page 7.)

[Church 1933] A. Church. *A Set of Postulates for the Foundation of Logic (second paper).* Annals of Mathematics, Series 2, vol. 34, pages 839–864, 1933. (Cited on page 7.)

[Church 1936] A. Church. *An Unsolvable Problem of Elementary Number Theory.* American Journal of Mathematics, vol. 58, pages 345–363, 1936. (Cited on page 7.)

[Church and Rosser 1936] A. Church and J. B. Rosser. *Some Properties of Conversion.* Transactions of the American Mathematical Society, vol. 39, pages 472–482, 1936. (Cited on page 7.)

[Cirstea 2001] H. Cirstea, C. Kirchner and L. Liquori. *The Rho Cube.* In FOSSACS'01, volume 2030 of *LNCS*, pages 166–180, 2001. (Cited on page 68.)

[Coquand 1988] T. Coquand and G. P. Huet. *The Calculus of Constructions.* Information and Computation, vol. 76, pages 95–120, 1988. (Cited on page 14.)

[Cousineau 2007] D. Cousineau and G. Dowek. *Embedding Pure Type Systems in the Lambda-Pi-Calculus Modulo.* In Simona Ronchi Rocca, editeur, Typed Lambda Calculi and Applications, volume 4583 of *Lecture Notes in Computer Science*, pages 102–117. Springer Berlin Heidelberg, 2007. (Cited on page 68.)

[Crary 2010] K. Crary. *Higher-order Representation of Substructural Logics.* In ICFP '10, pages 131–142. ACM, 2010. (Cited on pages 66, 85, 97, 105 and 111.)

[Curry 1934] H. Curry. *Functionality in Combinatory Logic.* In Proceedings of the National Academy of Sciences, volume 20, pages 584–590. 1934. (Cited on page 12.)

[Curry 1958] H. Curry. Combinatory Logic, Volume 1. North Holland, Amsterdam, 1958. (Cited on page 12.)

[de Alfaro 1997] L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, 1997. (Cited on page 33.)

[de Bruijn 1968] N. de Bruijn. *Automath, a Language for Mathematics.* In Automation and Reasoning, volume 2, Classical papers on computational logic 1967-1970, pages 159–200. Springer Verlag, 1968. (Cited on page 67.)

[de Morgan 1847] A. de Morgan. Formal Logic. London, 1847. (Cited on page 4.)

[Doder 2010] D. Doder, B. Marinković, P. Maksimović and A. Perović. *A Logic with Conditional Probability Operators.* Publications de L'Institut Mathématique (Nouvelle Serie), vol. 87, no. 101, pages 85–96, 2010. (Cited on page 21.)

[Doder 2012] D. Doder, Z. Marković, Z. Ognjanović, A. Perović and M. Rašković. *A Probabilistic Temporal Logic that can model Reasoning about Evidence.* Annals of Mathematics and Artificial Intelligence, vol. 65, pages 278–280, 2012. (Cited on page 120.)

[Dowek 2003] G. Dowek, T. Hardin and C. Kirchner. *Theorem Proving Modulo.* Journal of Automated Reasoning, vol. 31, pages 33–72, 2003. (Cited on pages 67 and 68.)

[Fajardo 1985] S. Fajardo. *Probability Logic with Conditional Expectation. Annals of Pure and Applied Logic* vol. 28, pages 137 – 161, 1985. (Cited on page 5.)

[Fagin 1990] R. Fagin, J. Y. Halpern and N. Megiddo. *A Logic for Reasoning about Probabilities.* Information and Computation, vol. 87, no. 1/2, pages 78–128, 1990. (Cited on pages 5, 21 and 32.)

[Fagin 1994] R. Fagin and J. Halpern. *Reasoning about Knowledge and Probability.* Journal of the ACM, vol. 41, no. 2, pages 340–367, 1994. (Cited on page 6.)

[Gaifman 1986] H. Gaifman. A Theory of Higher-Order Probabilities. in: *Proceedings of the Theoretical Aspects of Reasoning about Knowledge* (eds. J.Y. Halpern), Morgan-Kaufmann, San Mateo, California, 275–292, 1986. (Cited on page 5.)

[Girard 1972] F. Girard. Interprétation fonctionnelle et élimination descoupures de l'arithmétique d'ordre supérieur. Ph.D. Thesis. Université Paris 7. 1972. (Cited on page 11.)

[Gödel 1929] K. Gödel. *Über die Vollstandigkeit des Logikkalkulus.* Ph.D. Thesis. University of Vienna. 1929. (Cited on page 4.)

[Gonthier 2004] G. Gonthier. *A Computer-Checked Proof of the Four Colour Theorem.* Available on-line at: `http://research.microsoft.com/en-us/um/people/gonthier/4colproof.pdf`, September 2004. (Cited on page 16.)

[Gonthier 2012] G. Gonthier. *Six-year Journey leads to Proof of Feit-Thompson Theorem.* Available on-line at: `http://phys.org/news/2012-10-six-year-journey-proof-feit-thompson-theorem.html`, October 2012. (Cited on page 16.)

[Good 1950] I. J. Good. Probability and the Weighing of Evidence. Charles Griffin & Co. Ltd., 1950. (Cited on page 33.)

[Gordon 2013] M. J. C. Gordon and T. F. Melham. Introduction to HOL: A Theorem Proving Environment for Higher-Order Kogic. Cambridge University Press, 2013. (Cited on page 14.)

[Hailperin 1984] T. Hailperin. Probability Logic. *Notre Dame Journal of Formal Logic*, vol. 35, no. 3:198 – 212. 1984. (Cited on page 4.)

[Hailperin 1986] T. Hailperin. Boole's Logic and Probability. North-Holland. 1986. (Cited on page 4.)

[Halpern 1990] J. Y. Halpern. *An Analysis of First-Order Logics of Probability.* Artificial Intelligence, vol. 46, pages 311 – 350, 1990. (Cited on page 6.)

[Halpern 1991] J. Y. Halpern. *Knowledge, Belief and Certainty.* Annals of Mathematics and Artificial Intelligence vol. 4, pages 301 – 322, 1991. (Cited on page 6.)

[Halpern 1993] J. Y. Halpern and M. R. Tuttle. *Knowledge, Probability, and Adversaries.* Journal of the ACM, pages 917–962, 1993. (Cited on page 33.)

[Halpern 2006] J. Y. Halpern and R. Pucella. *A Logic for Reasoning about Evidence.* Journal of Artificial Intelligence Research (JAIR), vol. 26, pages 1–34, 2006. (Cited on pages 21, 33 and 34.)

[Harper 1993] R. Harper, F. Honsell and G. Plotkin. *A Framework for Defining Logics.* Journal of the ACM, vol. 40, pages 143–184, January 1993. (Cited on pages 65, 71 and 77.)

[Harper 2007] R. Harper and D. Licata. *Mechanizing Metatheory in a Logical Framework.* J. Funct. Program., vol. 17, pages 613–673, 2007. (Cited on pages 66 and 87.)

[He 1995] J. He, A. McIver and K. Seidel. *Probabilistic Models for the Guarded Command Language.* Science of Computer Programming, vol. 28, pages 28–171, 1995. (Cited on page 33.)

[Honsell 1995] F. Honsell and M. Miculan. *A Natural Deduction Approach to Dynamic Logic.* In Stefano Berardi and Mario Coppo, editeurs, TYPES, volume 1158 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 1995. (Cited on page 40.)

[Honsell 2001] F. Honsell, M. Miculan and I. Scagnetto. *π-calculus in (Co)Inductive Type Theories.* Theoretical Computer Science, vol. 253, no. 2, pages 239–285, 2001. (Cited on page 114.)

[Honsell 2007] F. Honsell, M. Lenisa and L. Liquori. *A Framework for Defining Logical Frameworks.* Volume in Honor of G. Plotkin, ENTCS, vol. 172, pages 399–436, 2007. (Cited on page 68.)

[Honsell 2008a] F. Honsell, M. Lenisa, L. Liquori and I. Scagnetto. *A Conditional Logical Framework.* In LPAR'08, volume 5330 of *LNCS*, pages 143–157, 2008. (Cited on page 68.)

[Honsell 2008b] F. Honsell, M. Lenisa, L. Liquori and I. Scagnetto. *A Conditional Logical Framework.* In Iliano Cervesato, Helmut Veith and Andrei Voronkov, editeurs, LPAR, volume 5330 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 2008. (Cited on page 76.)

[Honsell 2013a] F. Honsell, M. Lenisa, L. Liquori, P. Maksimovic and I. Scagnetto. LF_P – *A Logical Framework with External Predicates.* In Proceedings of LFMTP 2012, pages 13–22. ACM Digital Library, 2013. (Cited on page 68.)

[Honsell 2013b] F. Honsell, M. Lenisa, L. Liquori, P. Maksimovic and I. Scagnetto. *An Open Logical Framework.* Journal of Logic and Computation, 2013. To appear. (Cited on page 68.)

[Hoover 1978] D. N. Hoover. Probability logic. *Annals of mathematical logic* 14:287 –313. 1978. (Cited on page 5.)

[Hoover 1987] D. N. Hoover. An analitic completeness theorem for logic with probability quantifiers *J. Symbolic Logic* vol. 52:802 – 816. 1987. (Cited on page 5.)

[Howard 1980] W. Howard. *The formulae-as-types notion of construction.* In To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pages 479–490. Academic Press, Boston, MA, 1980. (Cited on page 12.)

[Hurd 2002] J. Hurd. *Formal verification of probabilistic algorithms.* Ph.D. Thesis. University of Cambridge, UK, 2002. (Cited on page 17.)

[Hurd 2005] J. Hurd, A. McIver, and C. Morgan. *Probabilistic guarded commands mechanized in HOL.* In Proceedings of QAPL 2004, Electronic Notes in Theoretical Computer Science, vol. 112, pages 95-111, 2005. (Cited on page 17.)

[Ilić-Stepić 2012] A. Ilić-Stepić, Z. Ognjanović, N. Ikodinović and A. Perović. *A p-adic Probability Logic.* Mathematical Logic Quarterly, vol. 58, no. 4-5, pages 263-280, 2012. (Cited on page 6.)

[Kaufmann 2000] M. Kaufmann, P. Manolios and J. S. Moore. Computer-aided reasoning: an approach. Kluwer Academic Publishing, 2000. (Cited on page 14.)

[Keisler 1977] H. J. Keisler. Hyperfinite Model Theory. in: *Logic Colloquim 76.* (eds. R. O. Gandy, J. M. E. Hyland), 5–110. North-Holland. 1977. (Cited on page 5.)

[Keisler 1985] H. J. Keisler. Probability Quantifiers. in: *Model Theoretic Logics*, (eds. J. Barwise and S. Feferman), 509–556. Springer-Verlag, Berlin. 1985. (Cited on page 5.)

[Keisler 1986a] J. Keisler. *A Completeness Proof for Adapted Probability Logic Ann. Pure Appl. Logic* vol. 31:61 – 70. 1986. (Cited on page 5.)

[Keisler 1986b] J. Keisler. *Hyperfinite Models of Adapted Probability Logic. Ann. Pure Appl. Logic* vol. 31:71 – 86. 1986. (Cited on page 5.)

[Keynes 1921] J. M. Keynes. *Treatise on Probability.* 1921. (Cited on page 4.)

[Kleene 1936] S. C. Kleene. *λ-definability and Recursiveness.* Duke Mathematical Journal, vol. 2, pages 340–353, 1936. (Cited on page 7.)

[Kolmogorov 1933] A. Kolmogorov. Grundbegriffe der Wahrscheinlichkeitsrechnung. Springer, 1933. (Cited on page 4.)

[Leibnitz 1665] G. W. Leibnitz. De Conditionibus. 1665. (Cited on page 3.)

[Leibnitz 1669] G. W. Leibnitz. Specimen Juris. 1669. (Cited on page 3.)

[Leibnitz 1765] G. W. Leibnitz. De Nouveaux Essais. 1765. (Cited on page 3.)

[Leroy 2009a] X. Leroy. *Formal Verification of a Realistic Compiler.* Communications of the ACM, vol. 52, no. 7, pages 107–115, 2009. (Cited on page 16.)

[Leroy 2009b] X. Leroy. *A Formally Verified Compiler Back-end.* Journal of Automated Reasoning, vol. 43, no. 4, pages 363–446, 2009. (Cited on page 16.)

[Licata 2009] D. Licata and R. Harper. *A universe of Binding and Computation.* In ICFP '09, pages 123–134. ACM, 2009. (Cited on page 69.)

[Lukasiewicz 2002] T. Lukasiewicz. *Probabilistic Default Reasoning with Conditional Constraints.* Annals of Mathematics and Artificial Intelligence, vol. 34, no. 1-3, pages 35–88, 2002. (Cited on page 21.)

[Marković 2003a] Z. Marković, Z. Ognjanović, and M. Rašković. *A probabilistic extension of intuitionistic logic.* Mathematical Logic Quarterly, vol 49, pages 415 – 424, 2003. (Cited on page 6.)

[Marković 2003b] Z. Marković, Z. Ognjanović, and M. Rašković. *An intuitionistic logic with probabilistic operators.* Publications de L'Institut Mathématique (Nouvelle Serie), vol. 73, no. 87, pages 31 – 38, 2003. (Cited on page 6.)

[Marković 2004] Z. Marković, Z. Ognjanović and M. Rašković. *What is the Proper Propositional Base for Probabilistic Logic?* in L.A. Zadeh, editor, Proceedings of the Information Processing and Management of Uncertainty in Knowledge-Based Systems Conference IPMU 2004, Perugia, Italy, pages 443–450, July, 4-9, 2004. (Cited on page 6.)

[Maksimović 2008] P. Maksimović, D. Doder, B. Marinković and A. Perović. *A Logic with a Conditional Probability Operator.* In Kata Balogh, editor, Proceedings of the 13$^{\text{th}}$ ESSLLI Student Session, pages 105–114, 2008. Hamburg. (Cited on page 21.)

[Marker 2002] D. Marker. Model Theory: an Introduction. Graduate Texts in Mathematics. Springer-Verlag, New York, 2002. (Cited on page 32.)

[Meyer 1991] B. Meyer. *Design by Contract.* In Advances in Object-Oriented Software Engineering, pages 1–50. Prentice Hall , Englewood Cliffs , NJ, 1991. (Cited on page 103.)

[Milne 1996] P. Milne. $\log[P(h|eb)/P(h|b)]$ *is the One True Measure of Confirmation.* Philosophy of Science, vol. 64, no. 1, pages 21–26, 1996. (Cited on page 33.)

[Milošević 2012] M. Milošević and Z. Ognjanović. *A First-order Conditional Probability Logic.* Logic Journal of the IGPL, vol. 20, no. 1, pages 235–253, 2012. (Cited on pages 6 and 120.)

[Milošević 2013] M. Milošević and Z. Ognjanović. *A First-order Conditional Probability Logic with Iterations.* Publications de l'Institut Mathématique (Nouvelle Serie), vol. 93, no. 107, pages 19–27, 2013. (Cited on page 6.)

[Morgan 1999] C. Morgan and A. McIver. *pGCL: formal reasoning for random algorithms.* South African Computer Journal, vol. 22, pages 14-27, 1999. (Cited on page 17.)

[Nanevski 2008] A. Nanevski, F. Pfenning and B. Pientka. *Contextual Model Type Theory.* ACM Transactions on Computational Logic, vol. 9, no. 3, 2008. (Cited on page 121.)

[Newman 1942] M. H. A. Newman. *On Theories with a Combinatorial Definition of Equivalence.* Annals of Mathematics, vol. 43, no. 2, pages 223–243, 1942. (Cited on page 77.)

[Nilsson 1986] N. Nilsson. Probabilistic Logic. *Artificial Intelligence*, vol. 28, pages 71–87. 1986. (Cited on page 5.)

[Nilsson 1993] N. Nilsson. Probabilistic Logic Revisited. *Artificial Intelligence*, vol. 59, pages 39–42. 1993. (Cited on page 5.)

[Ognjanović 1996] Z. Ognjanović, M. Rašković. *A Logic with Higher-Order Probabilities.* Publications de L'Institut Mathématique (Nouvelle Serie), vol. 60, no. 74, pages 1–4, 1996. (Cited on page 6.)

[Ognjanović 1998] Z. Ognjanović. *A Logic for Temporal and Probabilistic Reasoning.* Workshop on Probabilistic Logic and Randomised Computation ESSLLI '98, Saarbruecken, Germany, 1998. (Cited on page 6.)

[Ognjanović 1999a] Z. Ognjanović. *Neke verovatnosne logike i njihove primene u računarstvu.* Ph.D. thesis. PMF Kragujevac, 1999. (Cited on page 6.)

[Ognjanović 2000] Z. Ognjanović, and M. Rašković. *Some First-Order Probability Logics.* Theoretical Computer Science, vol. 247 no. 1-2, pages 191 – 212, 2000. (Cited on page 6.)

[Ognjanović 2005] Z. Ognjanović, Z. Marković and M. Rašković. *Completeness Theorem for a Logic with Imprecise and Conditional Probabilities.* Publications de l'Institut Mathématique (Nouvelle Serie), vol. 78, no. 92, pages 35–49, 2005. (Cited on page 21.)

[Ognjanović 2006] Z. Ognjanović. Discrete Linear-time Probabilistic Logics: Completeness, Decidability and Complexity. Journal of Logic and Computation, vol. 16, no. 2, pages 257–285, 2006. (Cited on page 6.)

[Ognjanović 2008] Z. Ognjanović, A. Perović, M. Rašković. *Logics with the Qualitative Probability Operator.* Logic Journal of IGPL, vol. 16, no. 2, pages 105–120, 2008. (Cited on page 6.)

[Ognjanović 2009] Z. Ognjanović, M. Rašković and Z. Marković. Probability Logics, pages 35–111. Mathematical Institute of the Serbian Academy of Sciences and Arts, Belgrade, 2009. (Cited on pages 3 and 21.)

[Paulin-Mohring 1996] C. Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur*, Dècembre 1996. Habilitation à diriger des recherches. (Cited on page 14.)

[Park 2006] S. Park, F. Pfenning, and S. Thrun. *A Probabilistic Language based on Sampling Functions.* ACM Transactions on Programming Languages and Systems, vol. 31, no. 1, 2006. (Cited on page 17.)

[Pfenning 1993] F. Pfenning. *Intensionality, Extensionality, and Proof Irrelevance in Modal Type Theory.* In LICS'93, pages 221–230, 1993. (Cited on page 69.)

[Pfenning 2013] F. Pfenning and C. Schuermann. *Twelf User's Guide, Chapter 6: Constraint Domains.* Online at `http://www.cs.cmu.edu/~twelf/guide-1-4/twelf_6.html`, 2013. Accessed January 2013. (Cited on page 69.)

[Pientka 2008] B. Pientka and J. Dunfield. *Programming with Proofs and Explicit Contexts.* In PPDP'08, pages 163–173. ACM, 2008. (Cited on page 69.)

[Pientka 2010] B. Pientka and J. Dunfield. *Beluga: A Framework for Programming and Reasoning with Deductive Systems (System Description).* In Automated Reasoning, volume 6173 of *Lecture Notes in Computer Science*, pages 15–21. Springer Berlin / Heidelberg, 2010. (Cited on page 69.)

[Plotkin 1975] G. D. Plotkin. *Call-by-name, Call-by-value and the λ-calculus.* Theoretical Computer Science, vol. 1, no. 2, pages 125 – 159, 1975. (Cited on page 103.)

[Poincaré 1902] H. Poincaré. La Science et l'Hypothèse. Flammarion, Paris, 1902. (Cited on page 115.)

[Polakow 1999] J. Polakow and F. Pfenning. *Natural Deduction for Intuitionistic Non-commutative Linear Logic.* In TLCA'99, volume 1581 of *LNCS*, pages 644–644, 1999. (Cited on pages 66, 97, 109, 110 and 111.)

[Popper 1959] K. Popper. The Logic of Scientific Discovery. Hutchinson, 1959. (Cited on page 33.)

[Poswolsky 2009] A. Poswolsky and C. Schürmann. *System Description: Delphin - A Functional Programming Language for Deductive Systems.* Electr. Notes Theor. Comput. Sci., vol. 228, pages 113–120, 2009. (Cited on page 69.)

[Rašković 1985] M. Rašković. *Model theory for L_{AM} Logic.* Publications de L'Institut Mathématique, vol. 37, pages 17-22, 1985. (Cited on page 5.)

[Rašković 1986] M. Rašković. *Completeness Theorem for Biprobability Models.* The Journal of Symbolic Logic, vol. 51, no. 3, pages 586-590, 1986. (Cited on page 5.)

[Rašković 1988] M. Rašković. *Completeness Theorem for Singular Biprobability Models.* Proceedings of the American Mathematical Society, vol. 102, no. 2, pages 389-392, 1988. (Cited on page 5.)

[Rašković 1993] M. Rašković. *Classical Logic with some Probability Operators.* Publications de l'Institut Mathématique (Nouvelle Serie), vol. 53, no. 67, pages 1-3, 1993. (Cited on page 6.)

[Rašković 1999] M. Rašković and Z. Ognjanović. *A First-Order Probability Logic, LP_Q. Publications de L'Institut Mathématique* (Nouvelle Serie), vol. 65, no. 79, pages 1-7, 1999. (Cited on page 6.)

[Rašković 2004] M. Rašković, Z. Ognjanović and Z. Marković. *A Logic with Conditional Probabilities.* In José Júlio Alferes and João Alexandre Leite, editeurs, Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, volume 3229 of *Lecture Notes in Computer Science.* Springer, 2004. (Cited on page 21.)

[Reichenbach 1935] H. Reichenbach. Wahrscheinlichkeitslehre. 1935. (Cited on page 4.)

[Reichenbach 1949] H. Reichenbach. The Theory of Probability. University of California Press, 1949. (Cited on page 4.)

[Rodenhausen 1982] H. Rodenhausen. *The Completeness Theorem for Adapted Probability Logic.* Ph.D. thesis. Heidelberg. 1982. (Cited on page 5.)

[Russell and Whitehead 1913] B. Russell and A. N. Whitehead. Principia Mathematica. Cambridge University Press, Cambridge, England, 1913. (Cited on page 7.)

[Shoenfield 1967] J. R. Shoenfield. Mathematical Logic. Addison-Wesley Series in Logic. Reading, Mass.,Addison-Wesley Publishing Company, 1967. (Cited on page 32.)

[Shopenhauer 2008] A. Shopenhauer. The World as Will and Presentation. The Longman Library of Primary Sources in Philosophy, 2008. (Cited on page 67.)

[Stump 2009] A. Stump. *Proof Checking Technology for Satisfiability Modulo Theories.* In International Workshop on Logical Frameworks and Metalanguages: Theory and Practice (LFMTP 2008), volume 228, pages 121 – 133, 2009. (Cited on page 69.)

[The Agda development team 2013] The Agda development team. *The Agda Wiki.* Available on-line at: `http://wiki.portal.chalmers.se/agda/pmwiki.php`, 2013. (Cited on page 14.)

[The Coq development team 2013] The Coq development team. The Coq Proof Assistant Reference Manual, version 8.4. 2013. Available on-line at: `http://http://coq.inria.fr/refman/`. (Cited on page 14.)

[The Hol4 development team 2013] The Hol4 development team.A *Hol4 Kanansakis 8.* Available on-line at: `http://hol.sourceforge.net/`, 2013. (Cited on page 14.)

[The Isabelle development team 2013] The Isabelle development team. *Isabelle.* Available on-line at: `http://isabelle.in.tum.de/`, 2013. (Cited on page 14.)

[The PVS development team 2013] The PVS development team. *PVS Specification and Verification System.* Available on-line at: `http://pvs.csl.sri.com/`, 2013. (Cited on page 15.)

[The Twelf development team 2013] The Twelf development team. *The Twelf Project.* Available on-line at: `http://twelf.org/wiki/Main_Page`, 2013. (Cited on page 15.)

[Turing 1937] A. M. Turing. *Computability and λ-definability.* Journal of Symbolic Logic, vol. 2, pages 153–163, 1937. (Cited on page 8.)

[van der Hoek 1997] W. van der Hoek. *Some Considerations on the Logic PFD: A Logic Combining Modality and Probability.* Journal of Applied Non-Classical Logics, vol. 7, no. 3, 1997. (Cited on page 34.)

[Virga 1999] R. Virga. *Higher-Order Rewriting with Dependent Types.* PhD thesis, Carnegie Mellon University, Pittsburgh, USA, 1999. (Cited on pages 68 and 121.)

[Watkins 2002] K. Watkins, I. Cervesato, F. Pfenning and D. Walker. *A Concurrent Logical Framework I: Judgments and Properties.* Tech. Rep. CMU-CS-02-101, 2002. (Cited on pages 66 and 87.)