ALMA MATER STUDIORUM
UNIVERSITY OF BOLOGNA

PH.D. IN COMPUTER SCIENCE AND ENGINEERING
XXXV

Towards a Logical Foundation of Randomized Computation

Candidate
Melissa Antonelli

Supervisor
Ugo Dal Lago
Co-Supervisor
Paolo Pistone

Ph.D. Coordinator
Ilaria Bartolini

01/B1 - INFORMATICA
INF/01 - INFORMATICA

Final Exam Year 2023

Ad Asia, blue“bird” in my heart,
e a Manuela, wonder mamma,
per il loro supporto discreto e costante.

Abstract

This dissertation investigates the relations between logic and theoretical computer science in the probabilistic setting. The project was motivated by two main considerations. On the one hand, since their appearance in the 1960s-70s, probabilistic models have become more and more pervasive in several fast-growing areas of computer science and technology. On the other, the study and development of (deterministic) computational models has considerably benefitted from the mutual interchanges existing between logic and computer science. Nevertheless, there is at least one crucial aspect of the theory of computation which was only marginally touched by such fruitful interactions, namely *randomized* computation. The goal of this thesis is precisely to (start) bridg(ing) this gap, by developing logical systems corresponding to specific aspects of randomized computation and, therefore, by generalizing standard achievements to the probabilistic realm. To do so, the key ingredient of our proposal is the introduction of new, measure-sensitive quantifiers associated with quantitative interpretations.

The dissertation is tripartite. In the first part, we focus on the relation between logic and counting complexity classes. Classical propositional logic provides the first example of an $\mathbf{NP}$-complete problem, while its quantified version characterizes in the same way the full polynomial hierarchy. Yet, an analogous logical counterpart was not known for the probabilistic and counting classes as introduced by Valiant and Wagner. Here, we introduce a generalization of classical propositional logic with counting quantifiers and associated with a quantitative semantics. This system is proved able to logically characterize the full counting hierarchy, as the validity problem for counting-quantified formulae (in a specific form) captures the corresponding level in Wagner's hierarchy.

In the second part of the dissertation, we consider programming language theory. Type systems for randomized λ-calculi – also guaranteeing various forms of termination properties – were introduced in the last decades, but these are not "logically oriented" and no Curry-Howard correspondence is known for them. Following intuitions coming from counting logics, we define the first probabilistic version of the correspondence. Specifically, we present an intuitionistic counting propositional logic, which precisely corresponds to a counting-typed probabilistic event λ-calculus. Notably, this type system with counting is also proved able to capture probabilistic termination.

Finally, we consider the relationship between arithmetic and computation.

As known, several theorems from logic and recursion theory, deeply link Peano Arithmetic and deterministic computation, but, again, no similar mathematical theory is known to relate in the same way to probabilistic computation. Here, we present a quantitative extension of the language of first-order arithmetic, which allows us to formalize basic results from probability theory and to establish a probabilistic version of Gödel's arithmetization. This language is also the starting point to define a randomized bounded arithmetic and, so, to generalize standard results by Buss. Indeed, due to counting ideas, we manage to arithmetically characterize interesting probabilistic classes, as $\mathbf{BPP}$.

Acknowledgements

I am grateful to my supervisor, Ugo Dal Lago, and co-supervisor, Paolo Pistone. None of the results presented in this thesis would have been achieved without their constant guidance. They not only provided me a great topic of study and patiently supervised me, but also helped me growing as a researcher. I thank Isabel Oitavem for hosting me in Lisbon. I thank Rapahëlle Cruibillé and Willem Heijltjes for carefully reviewing this manuscript, and for their comprehensive and meliorative remarks.

I thank my colleagues and friends in Bologna – Andrea, Aurore, Davide, Francesco, Gabriele, Paolo, and Riccardo – for supporting (even in my "odd moments") and uplifting me many times. It was really fundamental for me. I am also grateful to professors who contributed to my previous education and helped me finding my own way: Giovanna Corsi, Guido Gherardi, Eugenio Orlandelli, and Jan von Plato.

Finally, yet most importantly, I would like to thank my mother and my family for their love and encouragement, even in my worst times: thank you to my mom Manuela, and to my babies Babette and Dafne; thank you to my "amate creature" Asia, Jamas, and Yanique; thank you to my grandparents – Teresa, Lodice and Mauro – and to my aunt Micaela; thank you to Valentina; thank you to Andrea (Krav) and to Simone. A special thank you goes to Giulia, whose support was so essential.

Contents

List of Figures

"Susan, enjoy the absurdity of our world. It's a lot less painful. Believe me, our world is a lot less painful than the real world."

– Tom Ford, *Nocturnal Animals*

Chapter 1

Introduction

Among the defining features of *standard* computational models there is certainly determinacy: given an algorithm and an input, the sequence of computation steps is uniquely determined. In the second half of the XX century this assumption started to be relaxed in different ways. It was in this context that randomized algorithms were first introduced, where randomized algorithms are algorithms which evolve probabilistically so that, given an input, the computation process they perform can have different outcomes and each is associated with a certain probability. This peculiar feature makes them a very efficient and powerful tool, with several applications in computer science (CS, for short) and technology.

Starting from this, the project presented in this thesis has been motivated by two main considerations. On the one hand, since their appearance in the 1950s, probabilistic computational models have become ubiquitous in several fast-growing areas of CS and, by now, related abstract models – as probabilistic Turing machines, stochastic automata or randomized λ-calculi – have been deeply studied in the literature. On the other, there exist deep and mutual interactions linking logic and theoretical computer science (TCS, for short) and, historically, the development of computational models has considerably benefitted from them. Yet, randomized computation was only marginally touched by such fruitful interchanges and, indeed, it has not found a precise logical counterpart. Such a missing connection looks even more striking nowadays, due to the increasing pervasiveness of probabilistic models in many relevant fields of information technology (IT, for short), from statistical learning [170] and cryptography [102] to approximate computing and robotics [209].

The global purpose of this study consists in laying the foundation for a uniform approach to bridge the quoted gap. To do so, our key ingredient is a family of new logics, the language of which includes non-standard quantifiers "measuring" the probability of their argument formula, and associated with inherently quantitative semantics.

1.1 On Logical Foundations of Computer Science

The existence of several and deep interactions between logic and TCS is not accidental, but rooted in the intimate correspondence connecting these disciplines. In fact, even the formal appearance of the *science of computing* was essentially motivated by foundational studies in mathematics and logic, and it was in this framework that the subject took its first steps. Later on, the back and forth between logics and CS strongly influenced the development of both and, today, numerous areas of IT – think of programming language (PL, for short) theory [198, 95, 230], verification [207] and database theory [43], computational and descriptive complexity [44, 78, 116], just to quote a few – has concretely benefitted from this dialogue. As Siekman wrote,

> [i]n many respects, logic provides computer science with both a unifying foundational framework and a tool for modeling. [190, p. 17]

Indeed, many aspects of computer *science* are intrinsically related with logic, as shown by several profound and seminal results, e.g. [215, 41, 115, 44, 144, 98]. The other side of the coin is the existence of numerous *concrete* exchanges between them. While the growing importance of IT guided and stimulated many advances in logics, logical tools have extensive applications in CS and technology – from software and hardware verification to the modelling of interactive, multi-agent and AI systems, from the study of relational databases to argumentation theory, from knowledge representation to semantic web.

The *Science of Computing* Becoming What It Is (from Logic Roots). Nowadays CS is a very broad subject, which includes numerous and different subareas, ranging from theoretical to practical ones. Besides, it has no precise birth date as a discipline [205], but, certainly, a new era in computing emerged at the end of the XIX century with the foundational crisis of mathematics. In the 1930s, early attempts to formally define the notion of algorithm led to revolutionary theoretical advances and to the first groundbreaking results in computation theory,[1] while technological innovations mostly appeared in the 1940s, with the crossing of the Newton-Maxwell gap from mechanical and electromechanical computing to fully electrical devices.[2] Of course, "abstract" and "applicative" aspects are not separable. As a paradigmatic example, think of the historical development of computing models from Turing's abstraction to concrete computing machines:

> It was van Neumann's expertise as a logician that enabled him to understand the fundamental fact that a computing machine is a logic machine. In its circuits it embodies the distilled insights of a remarkable collection of logicians, developed over the centuries. Nowadays when computer technology is advancing with such breathtaking rapidity, as we admire the truly remarkable accomplishments of the engineers, it is all too easy to overlook the logicians whose ideas made it all possible. [66, p. 31]

[1] Further details can be found in Section 12.3.1.
[2] For further details, see for example [67, 222].

Already in the 1990s, CS had so developed to gain a well-defined and unique body of basic knowledge and to become a(n increasingly more) crucial discipline for academic and industrial research. Yet, the boundaries of this science are still not precisely outlined [205].

Some Results Relating Logic and Computation. Among the many intriguing interactions existing between logic and TCS, those relating classical and intuitionistic systems – on the one hand – and computational complexity and PL theory – on the other – are full of theoretical and applicative consequences. Indeed, it is well-known that classical propositional logic (**PL**, for short) provided the first example of an interesting $\mathbf{NP}$-complete problem [44], while the Curry-Howard correspondence (CHC, for short) unveiled a fundamental correspondence between type systems for abstract functional languages and proof calculi for constructive logics [95, 198]. Another striking connection between logic and computing was provided by first-order arithmetic. Indeed, proof systems for arithmetic can be used to prove termination of certain classes of algorithms or to establish complexity bounds [100, 139], while higher-order PLs can capture the computational content of arithmetical proofs [34]. These results have then stimulated various lines of research and evolved in active sub-areas of CS: variations of standard propositional logics have been put in relation with complexity classes other than $\mathbf{P}$ and $\mathbf{NP}$ [146, 33] and with type systems other than simple types [92, 38, 224, 144], while calculi for *linear* or *bunched* logic have inspired resource-conscious type systems in which duplication and sharing are taken into account and appropriately dealt with through typing [160, 224].

1.2 Probabilistic Computation

Probabilistic computational models have been widely investigated in the last few years, and are nowadays pervasive in several areas of CS. The idea of relaxing the notion of algorithm from a *purely deterministic* to a *probabilistic* process appeared early in the history of modern computability theory. Intuitively, a randomized algorithm involves random processes – typically corresponding to "flipping a coin" – as part of its procedure. While in deterministic computation, for every input, the algorithm $\mathcal{A}$ produces (at most) one output, in randomized computation, given an input, the algorithm $\mathcal{A}_{\mathcal{R}}$ returns a set of outputs, each associated with a probability:

$$[\![\mathcal{A}]\!] : \mathbb{N} \rightharpoonup \mathbb{N} \quad \rightsquigarrow \quad [\![\mathcal{A}_{\mathcal{R}}]\!] : \mathbb{N} \to \mathcal{D}_{\mathbb{N}}.$$

In this way, these algorithms have enabled efficient solutions to several problems [153], becoming essential in disciplines like cryptography [102]. As a consequence, several probabilistic formal models were then introduced: from probabilistic Turing machines (PTM, for short) [183, 90] to Markov chains, from stochastic automata [176, 188] to probabilistic λ-calculi [180, 119]. At this point, randomized algorithms and programs are widespread, steering disciplines like robotics, verification and security coding, computer vision and NLP.

> The last decade has witnessed a tremendous growth in the area of randomized algorithms. During this period, randomized algorithms went from being a tool in computational number theory to finding widespread applications in many types of algorithms. Two benefits of randomization have spearheaded this growth: simplicity and speed. [154, p. ix]

1.2.1 On the Genealogy of Probabilistic Models

As anticipated, from the 1950s and 1960s on, probabilistic computational models started to receive attention [69], and machines including stochastic elements appeared. In the 1970s, the first formalizations of PTMs were presented [183, 90]. These new models were explicitly defined as generalizations of standard ones, but were also developed in mutual dialogue. This was the starting point of a flourishing interplay between computational complexity and randomness.[3] In the same years, randomized λ-calculi were introduced in the context of (probabilistic) PL theory, where a probabilistic program is basically one endowed with a (pseudo-)random number generator.

Early Probabilistic Machines. In 1961, Davis introduced probabilistic automata as extensions of finite deterministic ones and proved them to behave like Markov chains (and vice versa) [66, pp. 264-265]. Then, in 1963, Carlyle formally presented his stochastic sequential machine as a 4-tuple made of the input, output and state sets plus the *conditional probability function* $P(y; s' \mid s; x)$, which is the joint probability that, given input x and state s, the output is y and the new state is s' [39].[4] Carlyle presented his machines as generalizations of Moore's deterministic finite-state machine.[5] In the same year, Rabin introduced his probabilistic automata as natural extensions of deterministic machines,[6] but including a *probability* transition function, which assigns a set of probabilities to each pair $\langle s, \sigma \rangle$: when the machine is in state s and receives the input σ, it reaches any state s_i with a given probability [176]. Remarkably, Rabin's work was one of the main source of inspiration for both Santos [183] and Gill [90].

Probabilistic Turing Machines. In 1969, Santos presented his PTM as a "natural" [184, p. 165] generalization of standard TMs.[7] This work was part of

[3] In fact, many questions about the *possible* advantages of adding probabilistic elements *in terms of resources* are still open. For example, as we will see in Part III, it is difficult to give *implicit* characterizations to semantic classes like $\mathbf{BPP}$ and $\mathbf{ZPP}$.

[4] Observe that this work was quoted by Santos (and Wee), who defined machines equivalent to Carlyle's one also using similar notions (e.g. that of conditional probability function) [185].

[5] Indeed, assuming the extra condition that the probability function has value 0 and 1, Carlyle's machine corresponds to Moore's deterministic one. He also emphasized the connection with Markov chains [39, p. 168].

[6] Formally, probabilistic automata are 4-tuples $\langle S, M, s_0, F \rangle$, where S is the set of states, $M : S \times \Sigma \to [0,1]^{n+1}$ is the probability transition function, $s_0 \in S$ is the initial state, and $F \subseteq S$ is the set of finite states, over an alphabet Σ.

[7] See also [183, p. 705].

a more general study on probabilistic models, developed in a series of 1960s-1970s papers [181, 182, 183, 185, 184]. In [183], Santos' PTM is defined as a 3-tuple, made of the alphabet set U, the set of states S, and the *conditional probability function* $p : S \times U \times (U \cup \{\mathtt{R}, \mathtt{L}, \mathtt{T}\}) \times S \to [0, 1]$, with $\mathtt{R}, \mathtt{L}, \mathtt{T} \notin U$, satisfying conditions:

1. For any $s \in S, u \in U$, $\sum_{v \in (U \cup \{\mathtt{R},\mathtt{L},\mathtt{T}\})} \sum_{s' \in S} p(s, u, v, s') = 1$,
2. For any $u \in U$, if $s \neq s'$, $p(s, u, \mathtt{T}, s') = 0$.[8]

Observe that Santos' probability function *fully* enucleates non-deterministic aspects of the machine and standard TMs can be seen as special PTMs in which $p \in \{0, 1\}$.

A few years later, PTMs were also introduced by Gill [89, 90, 91] as TMs "with the ability to flip coins in order to make random decisions" [90, p. 91].[9] One of the main motivations guiding Gill's interest in these new models was their deep connection with studies on randomized algorithms and computable random functions, starting with the pioneering article [69] on the (equivalent) computational power of deterministic and probabilistic machines. Gill's PTMs can be seen as (one-way infinite tape) TMs associated with a subroutine returning either 0 or 1 with equal probability. These machines have a specific *coin-tossing state*. When entering this state, they flip an unbiased coin and branch to one of two specified states according to the given "oracle" outcome. As for Santos' machines, computation is a random process, this time driven by the random bit supplies, corresponding to "the simplest type of randomness" [91, p. 676].

In fact, there are some differences between Santos' and Gill's PTMs. As seen, the latter ones are defined allowing unbiased choice, whereas the "kind of randomness" defined by Santos is more general. A related difference is that, when entering a non-deterministic state, Gill's PTMs choose between two possible steps only, while Santos' probability functions may lead to many subsequent configurations. Gill himself presented his PTM as a special case of Santos' one, which can be generalized by relaxing some requirements. Remarkably, these original, alternative presentations have paved the way to *different* definitions, *equally* labelled as PTMs in the literature.

Probabilistic Programming and λ-Calculi. Their capability to model complex phenomena and the resulting many applications – for example, in machine learning, AI, and cognitive science – pushed forward the study of probabilistic programming languages (PPL, for short), which has significantly advanced in the last decades. Conceptually, these languages describe probabilistic models and inferences, usually incorporating probabilities as "first-class citizens". Concretely, PPLs – for instance, `Church` [103] or `IBAL` [171] or `PRISM` [186] – are languages (say, `LISP`, or `ML`, `C` or `PROLOG`) endowed with constructs for sampling from probability distributions or conditioning event probability.

[8] For further details, see Section 2.1.2.

[9] Similarly, in 1977 he wrote that a PTM is a "computer with the ability to make random decisions" [91, p. 675].

With the rapidly growing trend in research on these languages, models for higher-order functional programs have become increasingly important. In particular, probabilistic λ-calculi were introduced in the 1970s-1980s [180, 119]. Generally speaking, these are obtained by extending ordinary calculi with constructs allowing probabilistic evolution, and there are at least two main paradigms distinguished by the operators they provide [53], namely *randomized* and (more recent) *Bayesian λ-calculi*.[10] In particular, randomized calculi are obtained adding a new operator for probabilistic choice producing different outcomes in a probabilistic fashion. The simplest one is "just" a form of binary and fair choice, but is enough to model randomized algorithms. Despite their early introduction [180], many aspects of these calculi – from denotational semantics and program equivalence to type systems – started to be properly studied only recently [120, 62, 48, 56, 31], and, indeed, there is no unified view of these theories yet.

1.2.2 The Importance of Being Randomized

As seen, randomized algorithms are powerful tools with numerous applications in different fields and technology. Generally speaking, these are crucial when dealing with uncertain information or partial knowledge, namely for all systems acting in realistic contexts – for example, think of driverless cars or of computer vision modelling. In some fields, probabilistic models have become even more than optional, for instance in cryptography, where secure encryption schemas are probabilistic [102].

***Reasoning* About Uncertainty** As anticipated, the use of randomized models have spread in disciplines involving uncertain domain – that is, in all disciplines *realistically* interacting with "the world". In agent systems (whether artificial or not) reasoning is processed and decisions are made on the ground of the partial information obtained from the environment and the background knowledge.[11] Clearly, in these contexts, simplifications are needed and "probabilistic thinking" appears as a formidable tool for learning processing and decision making [170, 131]. These concrete demands led also to the first attempts to analyze probabilistic reasoning *in a formal way* and to the introduction of a few *logical* systems, starting with Nilsson's pioneering proposals in 1986:

> Because many artificial intelligence applications require the ability to reason with uncertain knowledge, it is important to seek appropriate generalizations of logic from this case [158, p. 71]

[10] Bayesian calculi describe probabilistic models, corresponding to Bayesian networks. These calculi have been introduced relatively recently [28], and adopted in concrete PPLs, e.g. `ANGLICAN` and `CHURCH`.

[11] Probabilistic models became fundamental in AI research from the 1970s-1980s on. Koller and Friedman divided the history of this disciplines in some "main phases": in the 1950s-1960s, AI focussed on problem solving (e.g. in games) and planning; in the 1970s-1980s, expert systems for realistic applications started to be developed (sometimes implying high computational complexity); in the 1990s, Bayesian networks were consolidated to develop efficient inferences and learning algorithms (together with e.g. fuzzy and non-monotonic logics); in the 2000s also probabilistic graphical models emerged [131, pp. 13ff.].

In the following years, inspired by [158, 159], some probability logics were presented and developed in the context of modal logics [20, 81].[12] Of course, these are not the only ones and there exist several alternative (logical) approaches to deal with uncertain reasoning, for example via non-monotonic and fuzzy logics or with direct numerical representations.[13]

Of Robotics and Other Demons. Applications of probabilistic algorithms and models are copious and concern a variety of different fields, from robotics to linguistics and cognitive science. For example, linguistics aims to characterize common patterns and structures of a language, analyzing (real) communication and sources. Then, extrapolation of schemes and text processing clearly include elements of approximation (sometimes errors) and partiality, to deal with which probabilistic and statistical tools become crucial. As a consequence, stochastic models are nowadays essential for linguistics and NLP [143]. Similar considerations also hold for (probabilistic) approaches to cognitive science, the purpose of which is that of modelling learning and reasoning. Indeed, observations are always sparse and data acquired is incomplete and noisy. So, even in this context, probabilistic models and PPLs become essential. Especially, the generative paradigm [104] approaches cognition as a process representing knowledge about the causal structure of the world, i.e. processes which unfold with a certain degree of randomness. Robotic mapping relies on probabilistic algorithms as well [208, 209]. Indeed, also this discipline – which is crucial to design and build autonomous robots – requires the acquisition of (spatial) models from the "external", physical environment.

1.3 Towards Logical Foundation of Randomized Computation

As seen, interchanges between logic and computation are numerous and well-studied. Yet, when switching to the randomized setting, such a deep correspondence has only been investigated rather sparsely. In probabilistic algorithms, behavioral properties like termination or equivalence have an inherently *quantitative* nature, namely any computation terminates *with a given probability* and a program might simulate a desired function *up to* some probability of error – think, for instance, to probabilistic primality tests or learning algorithms. Can such quantitative properties be studied within a logical system? In this dissertation, we focus on a few specific aspects of the interaction between quantitative logics and randomized computation, giving a positive answer to this question. The turning point of our approach consists in considering new quantitative logics able to express probability in a natural way.

[12] For further details, see Section 4.3.

[13] Pearl divides the study of uncertainty in AI into three main schools: *logicists* mostly rely on non-monotonic systems, *neo-calculists* uses numerical representations, and *neo-probabilistics* focuses on computational tools and calculi basing on probability models [170].

1.3.1 Relating Logic and Randomized Computation

We generalize a few standard results linking logic and computation to the probabilistic realm.

Complexity Theory. As it is well-known, classical propositional logic and computational complexity are connected. Checking the satisfiability of **PL**-formulae is the paradigmatic $\mathbf{NP}$-complete problem [44], while the language of classical tautologies is $\mathbf{coNP}$-complete. In the early 1970s, Meyer and Stockmeyer showed that, when switching to *quantified* propositional logic (**QPL**, for short), the full polynomial-hierarchy ($\mathbf{PH}$, for short) can be captured by a *single* logical concept and each level in it is characterized by the validity of **QPL**-formulae (in PNF), with the corresponding number of quantifier alternations [146, 147, 231, 33].[14] Nevertheless, when switching to the probabilistic framework, such a plain correspondence seems lost, as no analogous *logical* counterpart is known to relate in this way to the counting classes and hierarchy, as introduced by Valiant [217] and Wagner [225, 227]:

$$\text{Polynomial Hierarchy} : \textbf{QPL} \iff \text{Counting Hierarchy} : ?$$

In Part I, we introduce a counting propositional system, called **CPL**, which is basically a generalization of **PL** capable of expressing that a formula is true with a given probability. This logic is shown to be strongly related to counting computation and classes, being the probabilistic counterpart of **QPL**. Indeed, the counting quantifiers of **CPL** can be naturally seen as the "quantitative" versions of standard propositional ones. Then, our main result is the *purely logical* characterization of Wagner's hierarchy via complete problems defined in terms of counting-quantified formulae.

Programming Language Theory. Traditionally, the CHC relates intuitionistic **PL** and the simply-typed λ-calculus, but in the last fifty years this correspondence was shown to hold in other and more sophisticated contexts too [38, 92, 144, 165, 224]. Meanwhile, randomized λ-calculi [180, 70] and corresponding type systems were introduced, sometimes also guaranteeing desirable forms of termination properties [77, 57].[15] Yet, they are not "logically-oriented" and no (probabilistic) CHC [198] is known for them:

$$\text{simply typed } \lambda_{\to} : \text{intuitionistic } \textbf{PL} \iff \text{randomized } \lambda\text{-calculus} : ?$$

In Part II, we define an intuitionistic version of $\textbf{CPL}_0$, called $\textbf{iCPL}_0$, which is able to capture quantitative behavioral properties, together with a "counting"-typed randomized λ-calculus. Its untyped part is strongly inspired by the probabilistic event λ-calculus introduced in [57], while its types are defined mimicking counting quantifiers. Then, we provide a (static and dynamic) correspondence in the style of Curry and Howard between these two systems.

[14] Further details can be found in Section 2.1.

[15] For further details, see 6.1.3.

Arithmetic and Computation Theory. Arithmetics and the theory of deterministic computation are linked by deep results coming from logic and recursion theory – for example, arithmetization [97] or realizability [128, 139] or the *Dialectica interpretation* [98, 16]. Indeed, the language of arithmetic is able to express many interesting properties of algorithms and, due to the relation between totality (of functions) and termination (of algorithms), several issues in computation theory can be analyzed in the framework of arithmetic. Also in this case, no probabilistic theory was defined to link to probabilistic computation as Peano Arithmetic (**PA**, for short) does to deterministic one:

$$\text{deterministic computation} : \textbf{PA} \iff \text{probabilistic computation} : ?$$

In Part III, we present a quantitative extension of the language for **PA**, called **MQPA**, which allows us to formalize basic results from probability theory which are not expressible in **PA**, for example the so-called infinite monkey theorem. Then, we prove that this language is actually related to randomized computation, establishing a probabilistic version of Gödel's arithmetization [97], that is we show that each *random* function can be expressed by a formula of **MQPA**.

(Randomized) Bounded Arithmetics. One of the crucial motivations for the development of bounded arithmetics – i.e., subsystems of **PA** the induction principle of which is limited – was their connection with computational complexity [34, 35]. Indeed, not all computable functions are *feasibly* computable, and bounded theories make it possible to characterize such interesting complexity classes in terms of families of arithmetical formulae. Specifically, Buss proved that poly-time computable functions correspond to those which are Σ_1^b-definable in a given bounded theory, called $\mathbf{S}_2^1$. Although this fact is very insightful, no similar result was established in the probabilistic framework:

$$\text{deterministic classes} : \textbf{BA} \iff \text{probabilistic classes} : ?$$

Again in Part III, we introduce a *randomized* bounded theory, called $\mathbf{RS}_2^1$, to logically capture probabilistic classes, like $\mathbf{BPP}$.

1.3.2 From Evaluating to Measuring

Counting quantifiers are quantifiers of the form $\mathbf{C}^q$ and $\mathbf{D}^q$ capable of expressing probabilities within a logical language. These not only determine the *existence* of a satisfying assignment, but rather count *how many* those assignments are. In this sense, counting quantifiers can be seen as a *quantitative* generalization of standard propositional ones:

$$(\forall X)F, (\exists X)F \quad \rightsquigarrow \quad \mathbf{C}^q F, \mathbf{D}^q F.$$

Intuitively, as the **QPL**-formula $(\exists X)F$ says that there is an interpretation for X making F true, the counting-quantified formula $\mathbf{C}^{1/2}F$ expresses that F has probability greater than $\frac{1}{2}$ of being true. Dually, $\mathbf{D}^{1/2}F$ says that the argument

formula F has probability strictly smaller than $\frac{1}{2}$ of being true. Such a generalization is made possible by switching from a truth-functional to a quantitative semantics, in which formulae are no more interpreted as single truth-values but as measurable sets of models:

$$[\![F]\!]_{\mathbf{QPL}} \in \{0,1\} \quad \rightsquigarrow \quad [\![F]\!]_{\mathbf{CPL}} \subseteq 2^{\mathbb{N}}.$$

So, while (the truth of) an existentially-quantified formula of **QPL** – for instance, $(\exists X)(\exists Y)(X \wedge Y)$ – gives us information about the *existence* of a satisfying model for $X \wedge Y$, counting formulae express information about the *number* of these satisfying valuations. For instance, the (pseudo-)counting formula $\mathbf{C}^{1/4}(X \wedge Y)$ says not only that there exist a satisfying model for $X \wedge Y$, but also that *at least* one out of four possible models of the argument formula is a satisfying one.

As we shall see, this logic allows us to formally represent and study quantitative aspects of probabilistic computation in an innovative way. Yet, although it provides a natural model for stochastic events, the expressive power of **CPL** is still quite limited. So, we generalize the notion of counting quantifier and define an extended language, called **MQPA**, which is basically the language of first-order arithmetic endowed with second-order measure quantifiers and associated with a Borel semantics.

1.4 Outline of the Thesis

As explained above, our contributions concern three aspects of the interaction between *quantitative* logic and *probabilistic* computation. Consistently, the dissertation is divided into three main parts. Each one is intended to be as self-contained as possible.[16] In particular, the opening chapter is always introductorily, offering a bird's-eye view of the topic captioned in the corresponding part. It includes a brief historical overview and global motivations, together with an informal presentation of the main results we are going to introduce, without dealing with technical details.[17]

- In Part I, we introduce counting propositional logics and show them able to provide complete (logical) problems for each level of Wagner's hierarchy. In particular, in Chapters 3 and 4, we present the language of univariate $\mathbf{CPL}_0$ and multivariate **CPL** (resp.), and the associated, quantitative semantics. These logics offer a natural formalism to express stochastic events and support a suitable proof-theoretical treatment, defined in the form of sound and complete sequent calculi. Then, our main result is a logical characterization of $\mathbf{CH}$, as presented in Chapter 5.

[16] Actually, for the sake of readability, tedious and convoluted proofs are sometimes only sketched. As a disclaimer, for most of them full details can be found in [13, 8, 5, 7], [11, 12] and [9, 10, 6], respectively.

[17] Historical sections are intended to offer some background notions. Readers who are already acquainted with the corresponding standard knowledge and not interested in the historical perspective are encouraged to skip them.

- Part II is devoted to our proposal of a probabilistic CHC. In Chapter 7, we define the intuitionistic version of counting propositional logic, called **iCPL**, while, in Chapter 8, we consider its computational part, namely a slightly modified version of the probabilistic event λ-calculus [57]. The main contribution here is the definition of a *probabilistic* CHC between an intuitionisitic counting logic and a counting-type system able to express the probability of termination. This is presented in Chapter 9. Then, in Chapter 10, termination is further investigated by the introduction of an intersection type system.

- In Part III, we define a language, called **MQPA**, together with a quantitative semantics. This language extends that of **PA** via second-order measure quantifiers, which are close to counting ones. In Chapter 12, we show that **MQPA** is very expressive. Indeed, results from probability theory, which cannot be expressed in **PA**, can instead be expressed in it. Furthermore, we show that every recursive random function is represented by a formula of **MQPA**. In Chapter 13, we introduce a new randomized bounded theory and establish that the class of formulae which are Σ_1^b-representable in it is precisely that of poly-time random function. Due to this result, we also provide an *arithmetical* characterization of $\mathbf{BPP}$.

These contributions are all part of a joint work with my supervisor Ugo Dal Lago and co-supervisor Paolo Pistone. Our research about randomized bounded theory, as presented in Section 13, was developed together with Davide Davoli and Isabel Oitavem.[18]

[18] In particular, results presented in Section 13.4 were established by Davide Davoli in his Master's Thesis [68].

Part I

On Counting Logic and Wagner's Hierarchy

Chapter 2

Characterizing (Counting) Classes

In this part we focus on the relation between quantitative logics and counting complexity classes. Our overall purpose is to generalize to the probabilistic setting the insightful correspondence linking the polynomial hierarchy and **QPL**, thus providing a logical counterpart to the counting hierarchy. To do so, we consider new, inherently quantitative logics, conceived as extensions of classical **PL**. These languages are obtained by adding *counting* quantifiers, allowing to express that a formula is true *in a certain portion* of all its possible interpretations. In particular, we start by considering a univariate fragment, which is limited in expressive power but support a very natural semantics. This logic also constitutes a model for stochastic experiments (associated with dyadic distributions). The multivariate counting system is then obtained by generalizing it in the straightforward way. We conclude presenting our main result, namely the "characterization" of counting complexity classes due to our counting logics. Specifically, we prove that the complexity of the decision problem for (a special prenex form of) counting formulae perfectly matches the appropriate level of Wagner's hierarchy.

2.1 Historical Background

In 1972/73, Meyer and Stockmeyer introduced $\mathbf{PH}$, a hierarchy of complexity classes generalizing $\mathbf{NP}$ and $\mathbf{coNP}$ and defined in analogy with Kleene's arithmetical hierarchy [146, 147]. In the same years, new models of computation were developed, for example probabilistic Turing machines by Santos [183] and Gill [91], or counting and threshold machines by Valiant [217] and Simon [191]. Together with these new computational models, also probabilistic (e.g. $\mathbf{BPP}$ or $\mathbf{PP}$ [91]) and counting classes (e.g. $\sharp\mathbf{P}$ [217]) were introduced. Remarkably, enumerating and probabilistic computation are strongly linked. Indeed, in 1975, Simon proved that, within certain conditions, PTMs and thresh-

old machines yield the same complexity class [191]. In the 1980s also a hierarchy of counting classes was defined [227, 164, 212].

Although these results are well-known, we briefly recap the historical background leading to their development. In fact, terminology is sometimes imprecise or misleading. For instance, the terms *counting* and *threshold machines* are often used as interchangeable, and there exist several definitions of PTMs – as deterministic TMs accessing a random-bit source in the form of an oracle-tape [90] or as NTMs with transition functions to be chosen with (possibly equal [91]) probability [183]. Being this the root of possible misunderstanding, we aim to make some crucial notions (and differences between them) by introducing their original formulations, in the context where they first appeared.

2.1.1 The Genesis of the Polynomial Hierarchy

In *On the Computational Complexity of Algorithms* (1965), Hartmainis and Stearns laid the foundation for the study of computational complexity [109], introducing key ideas for a general approach to quantify computational resources – typically, time and space – and from the 1970s on, complexity classes started to play a central role in CS. In particular, in his seminal article *The Complexity of Theorem-Proving Procedures* (1971), Cook defined the notion of **NP**-completeness and proved SAT complete for this class [44].[1] In 1972, Karp showed that other 21 problems (including combinatorial ones) are **NP**-complete, so increasing the interest in the topic [121]. Meanwhile, in the USSR, Levin conceived similar notions and results, first presented in a series of talks, and published in [141].[2] Then, in 1973/74, Fagin also provided a logical characterization of **NP**, this constituting the groundbreaking result in descriptive complexity [78].

Shortly after, researchers started to realize the great importance of these notions and some decision problems were shown to be outside **NP**:

> There has been considerable interest recently in finding "natural" problems whose solutions require more than polynomial time. [...] Closely related to the problem of finding non polynomial time languages is the question of whether or not nondeterministic polynomial time Turing machines can recognize a larger class of languages than deterministic polynomial time machines. [146, p. 125]

Indeed, it was their search for natural problems having efficient solutions, that led Meyer and Stockmeyer to the development of a hierarchy characterizing in a precise way the inherent complexity of word problems in automata theory, logic, and arithmetic. In particular, in *The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space* (1972), they considered the complexity of the language MINIMAL, consisting of formulae not equivalent to any smaller one. Neither MINIMAL or $\overline{\texttt{MINIMAL}}$ was known to be in **NP**, but

[1] SAT denotes the language of all satisfiable CNF-formulae.

[2] Nevertheless, his results were not widely known outside from USSR.

$\overline{\texttt{MINIMAL}}$ could be checked with an oracle for testing equivalence of formulae. This suggests a hierarchy of classes *above* **NP**.

The first presentation of **PH** was in terms of polynomially-bounded quantifiers. Specifically, in [146, Sec. II], a "hierarchy of languages" is described by generalizing the Kleene's arithmetical one, so that highest levels are obtained from the preceding classes:[3]

> For languages L_1, L_2 we say $L_1 R_n L_2$ if L_1 is accepted by some nondeterministic polynomial time machine with oracle language L_2. [...]
> Definition: $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \emptyset$.
>
> $$\Sigma_{i+1}^p = \{L \mid LR_nL' \text{ for some } L' \in \Sigma_i^p\}$$
> $$\Pi_{i+1}^p = \{L \mid \neg LR_nL' \text{ for some } L' \in \Sigma_i^p\}$$
> $$\Delta_{i+1}^p = \{L \mid L \leq_p L' \text{ for some } L' \in \Sigma_i^p\}.$$
>
> [...] This hierarchy has the same inclusion structure as the Kleene arithmetical hierarchy.[4] [146, pp. 127-128]

Further details on this new hierarchy were given in *Word Problems Requiring Exponential Time* (1973), where, in Section 4, classes is defined in terms of poly-time computable predicates.

> Let $P(x_1, \ldots, x_k)$ be a predicate on words in Σ^* for some Σ. We say that P is polynomial time computable if $\{x_1 \sharp x_2 \sharp \ldots \sharp x_k \mid P(x_1, \ldots, x_k)\}$ is a set of words recognizable in deterministic polynomial time where $\sharp$ is a symbol not in Σ.
> Theorem 4.1. For $k \geq 1$ a set of words A is in Σ_k^p iff there is a deterministic polynomial time computable predicate $P(x, y_1, y_2, \ldots, y_k)$ and a polynomial p such that
>
> $$A = \{x \mid \exists y_1 \forall y_2 \exists y_3 \ldots Q_k y_k [P(x, y_1, \ldots, y_k)]\}$$
>
> where the quantifier range over $y_i \in \Sigma^*$ such that $|y_i| \leq p(|x|)$.[5] [147, p. 5]

In [146, Sec. III], Meyer and Stockmeyer even provided complete sets for each level in **PH**. Theorem 3.2 proved that for any k, the set B_k of true quantified Boolean formulae with $k-1$ alternations of quantifiers starting with $\exists$ is Σ_k^p-complete:[6] case $B_k = \texttt{SAT}$ was already considered in [44], and a similar proof is offered for $k = 2$ relying on the notion of oracle machine.[7] In [147], the

[3] The reference to Kleene's hierarchy is a constant in Meyer and Stockmeyer's works on **PH**. For example, in 1977, Stockmeyer wrote:

> Briefly, the $\mathcal{P}$-hierarchy is that subrecursive analog of the Kleene arithmetical hierarchy, in which deterministic polynomial time plays the role of recursive time. [201, p. 2]

[4] Given two languages L_1, L_2, $L_1 \leq_p L_2$ if L_1 is accepted by some deterministic poly-time machine with oracle L_2 and L_2 oracle can, in a single step of the machine, determine whether or not $y \in L_2$, y being some string written on the machine's tape [146, p. 127].

[5] As standard, $Q_k = \exists$ when k is odd and $Q_k = \forall$ when k is even.

[6] See also [147, THM. 4.1] and [200, THM. 4.1].

[7] This shows that, although the definition of **PH** is not totally stable, many key ideas were conceived, at least *in nuce*, already in 1972/73. In particular, Fortnow noticed that [146, 147] somehow predated the Backer, Gill and Solovay notions of oracle machine, as formally introduced in [21]. For further details, see [86].

definition of the set B_ω of true quantified Boolean formulae with an arbitrary number of quantifiers also appears.

These ideas were extended and made clearer in [231] and [201]. In particular, in *The Polynomial-Time Hierarchy* (1977), **PH** (called "$\mathcal{P}$-hierarchy") is defined in a formal way in terms of both polynomially-bounded alternating quantifiers and oracle (or *query*) machines. In particular, the oracle characterization below is inspired by [231]:

> The *polynomial-time hierarchy* ($\mathcal{P}$-hierarchy) is $\{\Sigma_k^p, \Pi_k^p, \Delta_k^p : k \geq 0\}$, where
>
> $$\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathcal{P};$$
>
> and for $k \geq 0$,
>
> $$\begin{aligned} \Sigma_{k+1}^p &= \mathcal{NP}(\Sigma_k^p), \\ \Pi_{k+1}^p &= \text{co-}\mathcal{NP}(\Sigma_k^p), \\ \Delta_{k+1}^p &= \mathcal{P}(\Sigma_k^p). \end{aligned}$$
>
> Also define $\mathcal{PH} = \bigcup_{k=0}^{\infty} \Sigma_k^p$. [201, pp. 5-6]

Then, [147, THM. 4.1] is re-considered as follows:

> **Theorem 3.1.** Let Θ be a finite alphabet and $A \subseteq \Theta^+$. $A \in \Sigma_k^p$ iff there is a polynomial $p(n)$, an alphabet Γ, and a $(k+1)$-ary relation $R \in \mathcal{P}$ such that for all $x \in \Theta^+$,
>
> $$x \in A \quad \text{iff} \quad (\exists y_1)(\forall y_2)(\exists y_3) \ldots (Q_k y_k)[R(x, y_1, y_2, \ldots, y_k)],$$
>
> where the quantifiers alternate (so Q_k is $\exists$ ($\forall$) if k is odd (even)), and $y_1, \ldots, y_k$ range over all words in Γ^+ of length not exceeding $p(|x|)$. Similarly, $A \in \Pi_k^p$ iff for all x,
>
> $$x \in A \quad \text{iff} \quad (\forall y_1)(\exists y_2)(\forall y_3) \ldots (Q'_k y_k)[R(x, y_1, y_2, \ldots, y_k)],$$
>
> (so Q'_k is $\forall$ ($\exists$) if k is odd (even).) [201, p. 6]

where R is an n-ary relation on words defined as P above. Here, a relevant remark is added by Stockmeyer in [201, pp. 7-8], where – following [78] – he notices that there exists a natural correspondence between **PH** and the sets of finite structures of formulae in second-order predicate logic.[8]

2.1.2 Probabilistic and Counting Models

From the 1950s on, randomized algorithms have become more and more relevant and their study started spreading [69]. Since the beginning, the development of such models has been strongly related to inquiries about their power and connected resource issues:

> Is there anything that can be done by a machine with a random element but not by a deterministic machine? [69, p. 183] [90, p. 91]

[8] Other new ways to interpret space complexity in terms of alternation were presented in [40].

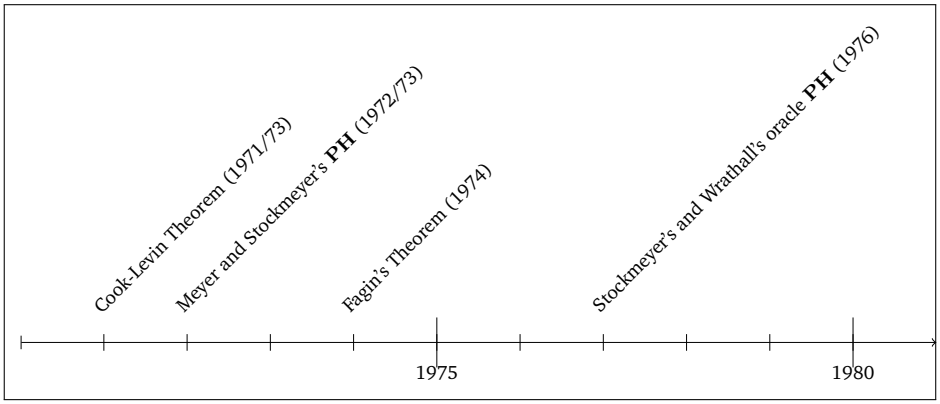


Figure 2.1: The Polynomial Hierarchy

A realistic model for real-life computers is that of machines enriched with random number generators. A corresponding precise definition was formalized in the 1970s in the form of probabilistic machines. Generally speaking, a PTM is a machine that, at each step in the computation, chooses between two transition functions – say, δ_1 and δ_2 – (usually) with equal probability $\frac{1}{2}$ and independently from previous choices.[9]

In the same decades, other non-standard models were introduced. Generally speaking, the defining feature of *threshold* and *counting machines* is that of enumerating the number of accepting paths: Valiant defined his counting machines as NTMs returning the *number* of accepting computation paths (in binary notation), while threshold machines *accept* an input when the number of accepting paths is greater than a given threshold. Other interesting models related to probabilistic computations are, for example, *alternating Turing machines* – namely, generalizations of non-deterministic TMs inspired by the notion of alternation by Kozen, Chandra, and Stockmeyer [40] – or Papadimitriou's *stochastic machines* [161, p. 293].

Probabilistic Turing Machines. PTMs were formally introduced by Santos and Gill in a series of papers in dialogue with each other. In particular, in Santos' *Probabilistic Turing Machines and Computability* (1969) and *Computability by Probabilistic Turing Machines* (1971), PTMs are considered together with random functions [183, 184]. These machines are defined as tuples $\mathscr{M} = (\mathbf{Q}, \Sigma, p)$,[10] where, as standard, $\mathbf{Q}$ and Σ are the sets of states and symbols (resp.), but $p : \mathbf{Q} \times \Sigma \times V \times \mathbf{Q} \to [0, 1]$ is a probability function with $V = \{\mathtt{R}, \mathtt{L}, \mathtt{T}\}$ and satisfying the following conditions:

i. $\sum_{v \in V} \sum_{q \in \mathbf{Q}} p(q, \sigma, v, q') = 1$ for every $q \in \mathbf{Q}$ and $\sigma \in \Sigma$

[9] In subsequent chapters, if not otherwise specified, we assume this as our standard definition.
[10] Actually, in Santos' notation the PTM is defined as follows: $Z = (U, S, p)$ [183].

ii. for every $\sigma \in \Sigma$, if $q \neq q'$, $p(q, \sigma, \mathtt{T}, q') = 0$.

This function is the *conditional probability* for "next acts" of the machine, given that it is at state q and scanning a square on which σ appears [183, p. 704] and [184, p. 12]. Then, standard TMs become a special case of PTMs in which p returns 0 and 1 only. Santos also notices that these PTMs behave like the stochastic sequential machines he defined in [181, 182] and can be associated with Markov chains, where states are represented by instantaneous expressions and a termination-state is added. Then, dynamic aspects are introduced [183, pp. 705-706]: $q_{\mathscr{M}}(\alpha, \beta)$ is the probability that the instantaneous expression of $\mathscr{M}$ is β, given that $\mathscr{M}$ starts with expression α,[11] and $t^{(n)}_{\mathscr{M}}(\alpha, \beta)$ is the probability that, after n steps, $\mathscr{M}$ terminates with instantaneous expression β, given that it starts with α.

In the same years, also Gill introduced formal machines to study the computational power of probabilistic algorithms and compared them with standard ones. His PTMs are defined as TMs with the additional ability to make random decisions – typically, corresponding to flipping a coin – so that their output is no longer uniquely determined:

> A *probabilistic Turing machine* (PTM) is a Turing machine with distinguished states called coin-tossing states. For each coin tossing state, the finite control unit specifies two possible next states. The computation of a probabilistic Turing machine is deterministic except that in coin-tossing states the machine tosses an unbiased coin to decide between the two possible next states. [91, p. 676]

Given an input, x, the computation of a PTM $\mathscr{M}$ corresponds to a stochastic process driven by the machine's "random bits supplies", and $\textsc{Prob}\{\mathscr{M}(x) = y\}$ denotes the probability that $\mathscr{M}$ returns y on input x. The output of the machine is the (at most one) number y such that $\textsc{Prob}\{\mathscr{M}(x) = y\} > \frac{1}{2}$.

Observe that Gill's definition [91, DEF. 2.1] restricts the source of randomness to a sequence of *independent* and *equiprobable* bits. By relaxing this model to allow arbitrary bias, one rather obtains machines equivalent to Santos' ones, where, as seen, transitions can be associated to *arbitrary* probability. On the other hand, despite being more limited, Gill's PTMs – in which the source of randomness is limited to independent and fair probabilities – are somehow more realistic, as implementable in concrete machines. Nevertheless, as anticipated, both these models are at the basis (usually without clear differentiation between them) of standard definitions of PTMs – as either defined as (deterministic) oracle machines with a read-only random tape or as NTM with two transition functions.

Threshold and Counting Machines. As anticipated, in this same context also threshold and counting machines were introduced. Simon defined *threshold*

[11] This function is extended to $q^n_{\mathscr{M}}(\alpha, \beta)$ to be interpreted as the probability that the instantaneous expression of $\mathscr{M}$ is β "after n steps" given that $\mathscr{M}$ starts with instantaneous expression α.

machines in Chapter 4 of his thesis *On Some Central Problems in Computational Complexity* (1975). In particular, in Section 4.4, he focusses on nondeterminism and defines classes of languages as recognized by k-threshold machines [191, pp. 90-91]. In fact, this machine is introduced to study complexity and connections with PTMs are explicitly considered in Theorem 4.4, stating that the class of probabilistic poly-time languages and that of languages recognized by poly-time threshold machines are the same [191, p. 91].

Counting machines were introduced to analyze probabilistic and counting complexity as well. In *The Complexity of Computing the Permanent* (1979), Valiant used them to define $\sharp\mathbf{P}$, the class which captures the complexity of the problem of counting the number of solution of a Boolean formula [217]. Valiant's counting machines are (poly-time) NTMs, the output of which consists in the number of their accepting paths.

> A *counting Turing machine* is a standard nondeterministic TM with an auxiliary output device that (magically) prints in binary notation on a special tape the number of accepting computations induced by the input. [217, p. 191]

Valiant himself noticed that this model is equivalent to Gill's PTM and Simon's threshold machine [217, p. 190].

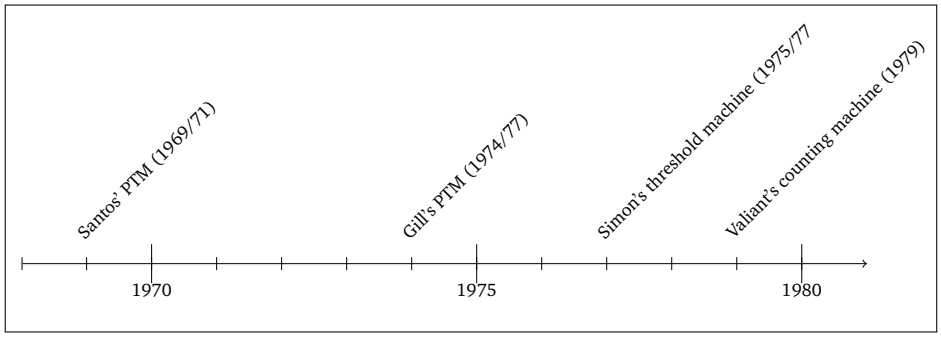


Figure 2.2: Probabilistic and Counting Machines

2.1.3 Probabilistic and Counting Classes

As anticipated, these new probabilistic and counting computational models were developed in strict connection with open problems in complexity theory:

> Since the notion of probabilistic machine was introduced by Gill, several researchers have been much interested in several questions about its computational power. [210, p. 514]

This increasing interest in complexity and the development of new models led to the discovery of many problems beyond $\mathbf{PH}$. In fact, also probabilistic and threshold machines were introduced in relation to complexity theory:

> ...there are many natural computational problems whose complexity cannot be modelized in terms of existential and universal quantifiers; on the other hand this complexity is captured by other complexity classes, more adapted to the idea of *counting*. [212, p. 213]

Probabilistic and Counting Classes. Together with PTMs, Gill introduced probabilistic classes – as $\mathbf{BPP}$ and $\mathbf{PP}$ – to capture the notion of probabilistic efficient computation [90, 91]:

> DEFINITION 5.1 (i) $\mathbf{PP}$ is the class of languages recognized by polynomial bounded PTMs. (ii) $\mathbf{BPP}$ is the class of languages recognized by polynomial bounded PTMs with bounded error probability. (iii) $\mathbf{ZPP}$ is the class of languages recognized by PTMs with polynomial bounded *average* run time and zero error probability. [91, p. 685]

In the quoted paper, Gill also considers complete problems characterizing these classes. Specifically, he takes into account MAJSAT (actually called MAJ) – defined in the standard way, i.e. as the set of propositional formulae satisfied by the majority of their interpretations – and $\sharp$SAT – defined as the set of pairs $\langle i, F\rangle$ such that the propositional formula F has more than i satisfying interpretations. He quotes Simon's result, stating that $\sharp$SAT is complete for $\mathbf{PP}$ [91, LEMMA 5.8] and extends it to MAJSAT [91, PROP. 5.10].

Meanwhile, in *On the Difference between One and Many* (1977), Simon examined whether it is more difficult to decide a problem or to count the number of its solutions and showed that the class of threshold languages corresponds to $\mathbf{PP}$ [192]. A few years later, in *The Complexity of Computing the Permanent* (1979), Valiant defined the class $\sharp\mathbf{P}$ of functions computing the number of accepting paths of a (poly-time) counting machine [217, p. 191]. Furthermore, in *Two Remarks on the Power of Computing* (1982), studying the relationship between counting classes and $\mathbf{PH}$, Papadimitriou and Zachos introduced $\oplus\mathbf{P}$ [163], the class of decision problems solvable by a poly-time NTM accepting an input when its number of accepting paths is odd. Since then, research in the area of counting complexity has been wide-spread and several definitions for such classes have been introduced.

The Counting Hierarchy. It was in this context that, in 1986, the counting hierarchy was conceived for the first time and independently defined by Wagner [225, 227, 226] and by Parberry and Schnitger [164].[12] In particular, in *The Complexity of Combinatorial Problems with Succinct Input Representation* (1986), Wagner defined $\mathbf{CH}$ as a generalization of Meyer and Stockmeyer's hierarchy, allowing to express the complexity of many natural problems in which *counting* is involved, and which are not in $\mathbf{PH}$. This was the first clear presentation of $\mathbf{CH}$ in terms of language operators. Some years later, another popular and equivalent characterization was presented by Torán [212, 214].

Notice that Wagner's operator was not the only "probabilistic" quantifier introduced in the 1980s. For example, Papadimitriou showed that the class

[12] For further details, see Chapter 5.

PPSPACE can be characterized by logical formulae, when alternating standard and *probabilistic* quantifiers, the latter expressing that more than half of the strings of a certain length satisfies the underlying predicate [161]. In 1986, Zachos and Heller defined **BPP** by means of a *random* quantifier [233], while in 1988 Zachos considered the relationship between canonical and probabilistic classes by introducing the *overwhelming* and the *majority* quantifiers [232]. Remarkably, all these operators are over (classes of) languages, rather than *stricto sensu* logical quantifiers. In this context, a relevant exception is represented by Kontinen's work [134, 135, 136], where second-order generalized quantifiers were defined in the style of descriptive complexity.

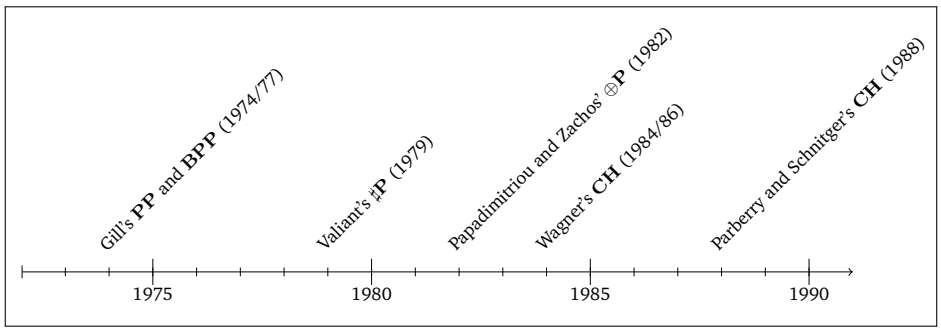


Figure 2.3: Probabilistic and Counting Classes

2.2 From Propositional to Counting Logic

From PL to QPL (and Beyond)

As seen, checking the satisfiability of a formula in classical **PL** constitutes the paradigmatic **NP**-complete problem, while determining whether any such formula is a tautology is, dually, **coNP**-complete. When switching to **QPL**, these two classes can be also uniformly captured: satisfiability of **PL** formulae corresponds to validity of existentially-quantified formulae of **QPL** in the form $(\exists X_1)\ldots(\exists X_n)F$, where F is quantifier-free, while **PL**-tautology can be seen as universally-quantified formulae $(\forall X_1)\ldots(\forall X_n)F$.

Example 2.2.1 (Formula of **QPL**)**.** For example, the formula

$$F_{\textbf{QPL}} : (\exists X_1)(\exists X_2)(\exists X_3)\big((X_1 \wedge \neg X_2) \vee (X_2 \wedge \neg X_3) \vee (X_3 \wedge \neg X_1)\big)$$

expresses that there exists *at least* one model satisfying $(X_1 \wedge \neg X_2) \vee (X_2 \wedge \neg X_3) \vee (X_3 \wedge \neg X_1)$.

Checking the validity of quantified formulae provides complete problems for the whole **PH**, so that each level in it is characterized by the number of alternations in the corresponding PNF-formula of **QPL**.

Indeed, in this case, existential and universal quantifications play the role of the *acceptance condition* in (non-deterministic) machines defining the corresponding complexity classes, that is requiring that either *(at least) one* or *all* computation paths are accepting ones. What if other kinds of quantification over computation paths replace universal and existential ones? Otherwise said, is it possible to (logically) deal with different machine models? For example, can we take into account complexity classes like $\mathbf{PP}$, which concerns problems computable by a poly-time PTM, such that the produced answer is correct for *at least half* of its accepting paths (and similarly for the non-accepting states)?

Notably, complete problems for $\mathbf{PP}$ can still be expressed in terms of **PL**-expressions, namely via the problem MAJSAT checking if a formula of **PL** is true in *at least half* of its possible interpretations. Furthermore, $\mathbf{PP}$ is related to $\mathbf{P}^{\sharp\mathtt{SAT}}$, the class of counting problems associated with the decision problems in $\mathbf{NP}$. It was starting from these classes and *in analogy with* $\mathbf{PH}$ that Wagner defined his *counting hierarchy*,

$$\mathbf{CH}_0 = \mathbf{P}$$
$$\mathbf{CH}_{n+1} = \mathbf{PP}^{\mathbf{CH}_n},$$

with $n \geq 0$. Then, a typical problem belonging to this hierarchy – in fact, one which is complete for $\mathbf{CH}_2 = \mathbf{PP}^{\mathbf{PP}}$ – is MAJMAJSAT, the problem of determining, given a formula of **PL** F containing two disjoint sets x and y of variables, whether for the majority of the valuations of variables in x, it holds that for the majority of variables in y, the resulting valuation makes F true.

From Standard to Counting Quantifiers

Wagner's hierarchy is clearly the *quantitative* counterpart of $\mathbf{PH}$, the structure of the two being very similar. So, the basic idea to generalize the insightful characterization by Meyer and Stockmeyer is to consider a *quantitative* version of **QPL**, this time able to capture "quantitative" acceptance conditions and, so, probabilistic classes. In other words, we need to consider a logic x to fill in the conceptual proportion below:

$$\frac{\textbf{QPL}}{\mathbf{PH}} = \frac{x}{\mathbf{CH}}.$$

Our core idea is to take into account more expressive quantifications and, specifically, to switch to measure-sensitive quantifiers able to *count* the number of satisfying valuations of formulae:

$$\frac{\textit{quantified} \text{ propositional logic}}{\mathbf{PH}} = \frac{\textit{measure-quantified} \text{ propositional logic}}{\mathbf{CH}}.$$

This means moving from standard quantified propositional formulae of the form,

$$(\triangleright_1 X_1)(\triangleright_2 X_2)\ldots(\triangleright_x X_n)F\{X_1,\ldots,X_n\},$$

where for any $i \in \{1, \ldots, n\}$, $\triangleright_i \in \{\exists, \forall\}$ to "counting" quantified expressions of the form

$$\mathbf{Q}_1 \mathbf{Q}_2 \ldots \mathbf{Q}_m . F,$$

where for $i \in \{1, \ldots, m\}$, $\mathbf{Q}_i$ are non-standard, measure-sensitive quantifiers.

From Qualitative to Quantitative Semantics

Yet, concretely, how could these measure quantifiers help us to define complete problems for $\mathbf{CH}$? Otherwise said, while the semantics of **QPL** is quite natural, what is the intuitive meaning of counting-quantified formulae? Clearly, this question appears as meaningless if variables one would like to quantify over only take *two* possible values. The perspective changes when allowing to simultaneously quantify over all the propositional variables which are free in the argument formula, that is when the interpretation of a formula is no longer a single truth-value got in a given valuation, but the set of *all* valuations making the formula true. So, for any valuation v,

$$[\![F]\!]_v = \text{truth-value} \quad \Longrightarrow \quad [\![F]\!]_v = \text{measurable set}$$

For example, given a formula of **PL** – say, $F = X \vee \neg Y$ – measuring the probability for it to be valid now amounts at *counting*, among the finitely many valuations of the variables of the formula, those which satisfy it – in this case three out of four.

In general, letting the semantics of a formula $[\![F]\!]$ be a (measurable) set, we can interpret the quantified formula $\mathbf{C}^q F$ as saying that F is true in *at least a fraction* $q \cdot 2^n$ of the 2^n possible assignments of its n variables. Let us consider a concrete formula similar to that of Example 2.2.1.

Example 2.2.2 (Pseudo-Formula of $\mathbf{CPL}_0$)**.** The measure-quantified formula

$$F_{\mathbf{CPL}_0} : \mathbf{C}^{1/2}\big((X_1 \wedge \neg X_2) \vee (X_2 \wedge \neg X_3) \vee (X_3 \wedge \neg X_1)\big)$$

intuitively says that $(X_1 \wedge \neg X_2) \vee (X_2 \wedge \neg X_3) \vee (X_3 \wedge \neg X_1)$ is satisfied by at least four (actually by six) of its $2^3 = 8$ models and, so, $F_{\mathbf{CPL}_0}$ is valid.

Observe also that our counting quantifier is reminiscent of the operator on classes of languages, which Wagner introduced in his seminal works on $\mathbf{CH}$ (and succinct representation) [227, 226]. Indeed, our logic precisely aims to capture *counting problems* in which one does not ask *if* a machine accepts some input, but *how many times* the machine accepts it. As seen, these problems are deeply related to the study of probabilistic classes, such as $\mathbf{PP}$, and counting quantifiers will be shown effectively able to define natural problems for them.

2.3 Outline of Part I

As anticipated, in this part of the thesis, we introduce new counting logics and investigate their connections with counting complexity classes. In particular, we proceed as follows:

- In Chapter 3, we present $\mathbf{CPL}_0$, the univariate fragment of our counting propositional logic. Although the expressive power of this language is limited, it offers a plain interpretation for formulae and its generalization to the more complex, multivariate version comes straightforwardly. This logic is also shown to admit a satisfactory proof-theoretical treatment and to be strongly related to stochastic experiments. Indeed, it provides a natural model to formalize events the measure of which is a dyadic rational.
- Then, in Chapter 4, we extend our study to multivariate **CPL**. Its language is *named*, so that relations between valuations of different groups of variables can be considered. Also in this case we define a sequent calculus and prove it sound and complete with respect to the given quantitative semantics.
- Chapter 5 is devoted to the link between our new logics and counting complexity. On the one hand, we prove that deciding the validity of $\mathbf{CPL}_0$-formulae is in $\mathbf{P}^{\sharp\mathtt{SAT}}$ (namely, the class of problems which can be solved in polynomial time when accessing a $\sharp\mathtt{SAT}$ oracle). On the other, multivariate **CPL** provides complete problems for the each level in $\mathbf{CH}$. Indeed, we establish that the deciding the validity of counting formulae (in a special form) with k nested quantifications precisely corresponds to the k-th level of Wagner's hierarchy.

Chapter 3

On Univariate Counting Propositional Logic

In the present chapter, we introduce $\mathbf{CPL}_0$, the univariate version of our counting propositional logic. Although this fragment is rather limited in expressive power, its interpretation is natural and can be extended to multivariate **CPL** plainly. Our presentation will proceed as follows. We start by summarising a few standard notions in probability theory, which are needed to define our quantitative semantics. In Section 3.2, we define the syntax and semantics of $\mathbf{CPL}_0$. Then, in Section 3.3 we introduce a rule system in the form of a labelled sequent calculus, which is proved sound and complete with respect to the given semantics. Finally, in Section 3.4, we show that $\mathbf{CPL}_0$ provides a natural model to formalize stochastic experiments the probability of which is associated with dyadic rationals.

3.1 Preliminaries

In Section 3.1.1 provide an informal introduction of basic notions in probability theory, which are needed to define the semantics of **CPL**, and make our notational convention explicit. In order for this Chapter to be self-contained, in Section 3.1.2, we present these definitions in a formal way. Readers already familiar with probability theory is encouraged to skip this latter Section.

3.1.1 A Gentle Introduction to Basic Measure Theory

Probability theory concerns mathematical foundations of stochastic phenomena. In probability theory an *outcome* or *point* ω is the result of a single execution of an experiment, the *sample space* Ω is the set of all its possible outcomes, and an *event* E is a subset of Ω. Two events, say E_1 and E_2, are *disjoint* or *mutually exclusive* when they cannot happen at the same time, that is $E_1 \cap E_2 = \emptyset$. A class

$\mathcal{F}$ of subsets of Ω is a σ-field or σ-algebra when (i.) it contains Ω, i.e. $\Omega \in \mathcal{F}$, (ii.) it is closed under complementation, i.e. if $E \in \mathcal{F}$, then $\overline{E} \in \mathcal{F}$, being $\overline{E}$ the complementation of E, and (iii.) it is closed under countable union (and intersection), i.e. if $\{E_i\}_{i\in\mathbb{N}}$, then $\bigcup_{i\in\mathbb{N}} E_i \in \mathcal{F}$. The largest σ-field on Ω is the power class 2^Ω, while the smallest one is $\{\emptyset, \Omega\}$. The *σ-field generated by* $\mathcal{F}$, $\sigma(\mathcal{F})$, is the smallest σ-algebra containing $\mathcal{F}$. A *measurable space* is a pair $(\Omega, \mathcal{F})$, where $\mathcal{F}$ is a σ-algebra over Ω.

In the 1930s, Kolmogorov introduced the notion of *probability space*, together with the axioms for probability [133]. A *probability measure* $\textsc{Prob}(\cdot)$ on a σ-field $\mathcal{F}$ associates each event $E \in \mathcal{F}$ with a number $\textsc{Prob}(E)$ so that,

i. for each $E \in \mathcal{F}$, $0 \leq \textsc{Prob}(E) \leq 1$,

ii. $\textsc{Prob}(\emptyset) = 0$ and $P(\Omega) = 1$,

iii. if $E_1, E_2, \cdots \in \mathcal{F}$ is a sequence of disjoint events, then

$$\textsc{Prob}\left(\bigcup_{k=1}^{\infty} E_k\right) = \sum_{k=1}^{\infty} \textsc{Prob}(E_k).$$

Two events are *(stochastically) independent* when the occurrence of one does not affect the probability for the other to occur. In particular, given two disjoint events, E_1 and E_2, $\textsc{Prob}(E_1 \cup E_2) = \textsc{Prob}(E_1) + \textsc{Prob}(E_2)$, while for two independent events, E_1' and E_2', $\textsc{Prob}(E_1' \cap E_2') = \textsc{Prob}(E_1') \cdot \textsc{Prob}(E_2')$.

A *probability space* $(\Omega, \mathcal{F}, \textsc{Prob})$ is a mathematical object that provides a formal model for random processes and it is made of:

- A *sample space* Ω, which is the set of all possible outcomes,
- A *σ-field* $\mathcal{F}$, which is the set of events,
- A *probability measure* PROB, which assigns to each event in $\mathcal{F}$ a probability, i.e. a number between 0 and 1 satisfying the so-called Kolmogorov axioms.

In the following we will focus on a specific probability space, namely the one related to fair coin tosses such that the set of possible outcomes of the experiment is $\{\textsc{tail}, \textsc{head}\}$. In general, if dealing with a Bernoulli experiment, we can consider the set of its possible outcomes as simply $\mathbb{B} = \{\mathbb{0}, \mathbb{1}\}$ or $2 = \{0, 1\}$.[1] The corresponding sample space is $\Omega = \mathbb{B}^{\mathbb{N}}$, to be naturally seen as the set of all infinite sequences of random bits (i.e. coin tosses) denoted as

$$\omega = \omega(1)\omega(2)\ldots,$$

with $\omega(i) \in \mathbb{B}$ for any $\omega \in \Omega$ and $i \in \mathbb{N}$. Each sequence ω can be interpreted as the result of infinitely flipping a coin.

[1] In what follows, we use all these three notations depending on the context. In particular, we use TAIL and HEAD when dealing with concrete examples concerning coin tossing. Observe however that all these sets are equivalent for our goal.

Definition 3.1.1 (Cylinder of Rank n)**.** A *cylinder of rank n* is a set of the form

$$\text{-}\mathcal{cyl}_H = \{\omega \mid \omega(1), \ldots, \omega(n) \in H\},$$

with $H \subset \mathbb{B}^n$.

Note that when $H = \{(u_1, \ldots, u_n)\}$ is a singleton ($u_1, \ldots, u_n \in \mathbb{B}$), an event $E = \{\omega \mid \omega(1), \ldots, \omega(n) = (u_1, \ldots, u_n)\}$, such that the first n repetitions of the experiment have outcomes $u_1, \ldots, u_n$ in sequence, is called a *thin cylinder*. We will be particularly interested in special thin cylinders in which the only set of H is made of one element $u_i = \mathbb{1}$.

Notation 3.1.1. Let $i \in \mathbb{N}$, we denote special thin cylinders as follows:

$$Cyl(i) = \{\omega \mid \omega(i) = \mathbb{1}\}.$$

The class of cylinders of all ranks, which is a field,[2] is denoted by $\mathscr{C}$, while $\sigma(\mathscr{C})$ indicates the σ-algebra generated by $\mathscr{C}$. It is thus possible to define a measure on it. In particular, the canonical one, consists in assigning the following probability measure $\mu_{\mathscr{C}}$ to any cylinder of rank n.

Definition 3.1.2 (Cylinder Measure)**.** Given $u \in \mathbb{B}$, let p_u denote the (non-negative and summing to 1) probability of getting u. Then, for any cylinder $\text{-}\mathcal{cyl}_H$,

$$\mu_{\mathscr{C}}(\text{-}\mathcal{cyl}_H) = \sum_H p_{u_1} \cdots p_{u_n}.$$

the sum extending over all the sequences $(u_1, \ldots, u_n) \in H$

So, in the special case of $\text{-}\mathcal{cyl}_H$ being a thin cylinder,

$$\mu_{\mathscr{C}}\big(\{\omega \mid (\omega(1), \ldots, \omega(n)) = (u_1, \ldots, u_n)\}\big) = p_{u_1} \cdots p_{u_n}.$$

This is also a mathematical model for an infinite sequence of random bits or independent tosses, each having probability $p_{\mathbb{0}_i}$ of success, and $p_{\mathbb{1}_i}$ of failure. Furthermore, when the coin is fair, for any tossing: $p_{\mathbb{0}_i} = p_{\mathbb{1}_i} = \frac{1}{2}$. In this case, since cylinders of rank n are finite sets, the following result comes out as a straightforward consequence of Definition 3.1.1.

Corollary 3.1.1. *For any cylinder of rank n, call it $\text{-}\mathcal{cyl}_H$ (such that for any $i \in \mathbb{N}$ and $p_{\mathbb{0}_i} = p_{\mathbb{1}_i} = \frac{1}{2}$), there are some $m, m' \in \mathbb{N}$ such that $\mu_{\mathscr{C}}(\text{-}\mathcal{cyl}_H) = \frac{m'}{2^m}$.*

So, going back to $\sigma(\mathscr{C})$, a well-defined probability measure can be assigned to it by simply generalizing Definition 3.1.2 in the natural way, and the probability space $\mathscr{P}_{\mathscr{C}} = (\mathbb{B}^{\mathbb{N}}, \sigma(\mathscr{C}), \mu_{\mathscr{C}})$, where $\mu_{\mathscr{C}}$ is such that, for any $i \in \mathbb{N}$, $p_{\mathbb{0}_i} = p_{\mathbb{1}_i} = \frac{1}{2}$, defines a standard model for infinite and independent tosses of a *fair* coin.[3]

[2] See [25, pp. 27-30]. Observe that Billingsley uses $\mathcal{C}_0$ to denote this class, and $\mathcal{C}$ to indicate corresponding, generated σ-algebra.

[3] For further details on the "coin toss model", see Section 3.1.2 or [25].

3.1.2 Basic Notions in Measure Theory

The following notions are standard, see for example [25, 150].

On Fields and σ-Fields. As seen, we use Ω to denote the space consisting of all possible outcomes ω of an experiment. Subsets of Ω are called events.

Definition 3.1.3 (Field)**.** A class $\mathfrak{F}$ of subsets of Ω is called a *field* if it contains Ω itself and is closed under complementation and finite unions:

i. $\Omega \in \mathfrak{F}$,

ii. if $E \in \mathfrak{F}$, then $\overline{E} \in \mathfrak{F}$,

iii. if $E_1, E_2 \in \mathfrak{F}$, then $E_1 \cup E_2 \in \mathfrak{F}$.

A field is closed under finite set-theoretic operations, while a σ-field is closed also under countable ones.

The first condition simply ensures that $\mathfrak{F}$ is nonempty. Furthermore, by De-Morgan's law (and (ii.)), condition (iii.) could be replaced by the alternative condition (iii.') saying that if $E_1, E_2 \in \mathfrak{F}$, then $E_1 \cap E_2 \in \mathfrak{F}$. In probability theory, we are particularly interested in classes that, given $\mathcal{A}$, (i.) contain $\mathcal{A}$ itself, (ii.) are σ-fields (and are as small as possible).

Definition 3.1.4 (Generated σ-Field)**.** The *σ-field generated by $\mathcal{A}$*, $\sigma(\mathcal{A})$, is the intersection of all (and only) the σ-fields containing $\mathcal{A}$:

i. $\mathcal{A} \subset \sigma(\mathcal{A})$,

ii. $\sigma(\mathcal{A})$ is a σ-field,

iii. if $\mathcal{A} \subset \mathcal{A}'$ and $\mathcal{A}'$ is a σ-field, then $\sigma(\mathcal{A}) \subset \mathcal{A}'$

Example 3.1.1 ([25])**.** Let $\mathcal{I}$ be the class of sub-intervals of $\Omega = (0, 1]$. Then, $\mathcal{B} = \sigma(\mathcal{I})$ is called the *Borel set* over the unit interval.

Basic Probability Theory. Probability functions are set functions – i.e. a real-valued functions defined on some class of subsets of Ω – satisfying specific conditions.

Definition 3.1.5 (Probability Function)**.** A *probability function* is a function $\textsc{Prob} : \mathfrak{F} \to [0, 1]$ satisfying the following conditions:

1. for any event E, $0 \leq \textsc{Prob}(E) \leq 1$,
2. $\textsc{Prob}(\Omega) = 1$,
3. for any finite or countably infinite sequence of pairwise mutually disjoint events $E_1, E_2, \ldots$ in $\mathfrak{F}$, and $\bigcup_{k=1}^{\infty} E_k \in \mathfrak{F}$, then

$$\textsc{Prob}\Big(\bigcup_{i\geq 1}^{\infty} E_i\Big) = \sum_{i\geq 1}^{\infty} \textsc{Prob}(E_i).$$

In particular, condition (3.) is called *countable additivity*.

Definition 3.1.6 (Probability Space)**.** A *probability space* is a triple $(\Omega, \mathscr{F}, \textsc{Prob})$:

I. a *sample space* Ω, which is the set of all possible outcomes,

II. a *σ-field* $\mathscr{F}$, representing the collection of all events to be considered,

III. a *probability function* $\textsc{Prob} : \mathscr{F} \to [0, 1]$.

An element of Ω is a *simple* or *elementary* event.

Observe that in a *discrete* probability space, Ω is finite or countably infinite, and $\mathscr{F}$ consists of all subsets of Ω. Furthermore, in a discrete probability space, the probability function is uniquely defined by the probabilities of simple events [150].

Since events are sets, notation comes from set theory: $E_1 \cap E_2$ denotes the occurrence of both E_1 and E_2, while $E_1 \cup E_2$ indicates the occurrence of either E_1 or E_2 or both.

Example 3.1.2. Given a fair coin, let E_1 be the event that its first flipping returns HEAD and E_2 that representing the second flipping returning HEAD. Then, $E_1 \cap E_2$ denotes the event that both coin tosses have returned HEAD, while $E_1 \cup E_2$ expresses that at least one of them has had.

$E_1 - E_2$ expresess the occurrence of E_1 but not of E_2.

Notation 3.1.2. We use $\overline{E}$ as a shorthand for $\Omega - E$.

Lemma 3.1.1. *For two events E_1 and E_2,*

$$\textsc{Prob}(E_1 \cup E_2) = \textsc{Prob}(E_1) + \textsc{Prob}(E_2) - \textsc{Prob}(E_1 \cap E_2).$$

Definition 3.1.7 (Independent Events)**.** Two events E_1, E_2 are *independent* when

$$\textsc{Prob}(E_1 \cap E_2) = \textsc{Prob}(E_1) \cdot \textsc{Prob}(E_2).$$

In general, events $E_1, \ldots, E_k$ are *mutually independent* when $\textsc{Prob}\big(\bigcap_{i\in\{1,\ldots,k\}} E_i\big) = \prod_{i\in\{1,\ldots,k\}} \textsc{Prob}(E_i)$. Observe that – as we shall see – the notion of independence is crucial in the context of randomized computation to formalize algorithm sampling with replacement [150].

Definition 3.1.8 (Random Variables)**.** A *random variable* X on a sample space Ω is a real-valued function on Ω, i.e. $X : \Omega \to \mathbb{R}$. A *discrete random variable* is a random variable that takes only a finite or countably infinite number of values.

For a discrete random variable X and a real value r, the event $X = r$ is made by all basic events of the sample space such that the random variable X assumes value r. Otherwise said, $X = r$ represents the set $\{\omega \in \Omega \mid X(\omega) = r\}$. Its probability is denoted as

$$\textsc{Prob}(X = r) = \sum_{\omega\in\Omega : X(\omega)=r} \textsc{Prob}(\omega).$$

Notice that the notion of independence is naturally generalized to this context. In this context, experiments of particular interests are those in which the probability of success is p and of failure is $1-p$. Then, we call a *Bernoulli variable* is a random variable X such that:

$$X = \begin{cases} 1 & \text{success} \\ 0 & \text{failure.} \end{cases}$$

Cylinder Probability Space We now focus on a model which allows us to make precise the idea of *tossing a (fair) coin infinitely many times*.

In [25], Billingsley considers a model to simultaneously fit random drawing of points from a segment and infinite sequences of coin tosses. He started with a finite sets of points to be regarded as the possible outcomes of an experiment, in particular letting $\Omega = 2^{\mathbb{N}}$ (or, equally, $\{\text{HEAD}, \text{TAIL}\}^{\infty}$).[4] So, $2^{\mathbb{N}}$ is an infinite Cartesian product, while $2^n = \{0,1\} \times \cdots \times \{0,1\}$ is the Cartesian product of n copies of $\{0,1\}$ and consists of the n-long sequences $(u_1, \ldots, u_n)$ of elements in $\{0,1\}$. As seen, for such a sequence, the set $\{\omega : (\omega(1), \ldots, \omega(n)) = (u_1, \ldots, u_n)\}$ represents the event that the first n repetitions of the experiment give outcomes $u_1, \ldots, u_n$ in sequence. Clearly, this notion of set of sequences is strongly related with that of *ranked cylinder* (Definition 3.1.1) and, then, defining a probability space for classes of such cylinders would also offer a formal model for coin tossing experiments.

As seen, a well-defined probability space is defined in the form of $\mathscr{P}_{\mathscr{C}} = (2^{\mathbb{N}}, \sigma(\mathscr{C}), \mu_{\mathscr{C}})$, where:

- $\mathscr{C}$ is the field set of all cylinders of any rank,
- $\sigma(\mathscr{C})$ is the σ-algebra generated by $\mathscr{C}$, that is the smallest σ-algebra containing $\mathscr{C}$ (Definition 3.1.3). (Observe that $\sigma(\mathscr{C})$ is contained in the Borel σ-algebra.),
- $\mu_{\mathscr{C}}$ denotes the standard cylinder measure over $\sigma(\mathscr{C})$, namely the unique measure on $\sigma(\mathscr{C})$ such that $\mu_{\mathscr{C}}(Cyl(i)) = \frac{1}{2}$ (Definition 3.1.2).

In order for this probability space to be well-defined it is essential that the underlying measurable space is. So, first, it is proved that $\mathscr{C}$ is a field: $2^{\mathbb{N}}$ and $\emptyset$ has the form of cylinders of rank, and $\mathscr{C}$ is closed under complementation and finite union [25, pp. 27-28]. Then, it is considered how to define the quoted, unique probability measure over $\sigma(\mathscr{C})$. In particular, letting again p_{u_i} (with $u \in \{0,1\}$ and $i \in \mathbb{N}$) denote probabilities on $\{0,1\}$, nonnegative and summing to 1, Billignsley define the set function PROB on $\mathscr{C}$ as in Definition 3.1.2 above, i.e. as

$$\text{PROB}(cyl_H) = \sum_H p_{u_1} \cdots p_{u_n}.$$

[4] Notice that we have slightly modified Billingsley's notation, according to which, for example, the infinite Cartesian product Ω is denoted as S^{∞} and ω as $z_1(\omega)$.

the sum of all the sequences $(u_1, \ldots, u_n) \in H$. This probability is finally shown to be a (unique) probability measure, as desired,[5] and defined on both $\mathscr{C}$ and (generalizable to) $\sigma(\mathscr{C})$, [25, pp.28-30].

3.2 Syntax and Semantics of $\mathbf{CPL}_0$

Standard interpretation for **PL** associates formulae to truth-values. As anticipated in Section 2.2, the core idea to develop our counting semantics consists of modifying the canonical interpretation in a quantitative sense, associating formulae with the *measurable* set of *all* valuations satisfying them. In particular, since propositional formulae may have an arbitrary number of propositional variables, valuations are taken as elements of $2^{\mathbb{N}}$. Then, any formula of $\mathbf{CPL}_0$, say F, is interpreted as the set $[\![F]\!] \subseteq 2^{\mathbb{N}}$ consisting of all maps $f \in 2^{\mathbb{N}}$ "making F true". As seen, such sets can be easily seen to belong to the standard Borel algebra over $2^{\mathbb{N}}$, $\mathscr{B}(2^{\mathbb{N}})$, thus yielding a genuinely quantitative semantics. Specifically, atomic propositions correspond to *cylinder sets* [25] of the following form:[6]

$$Cyl(i) = \{f \in 2^{\mathbb{N}} \mid f(i) = 1\}$$

where $i \in \mathbb{N}$, while molecular formulae are interpreted in a natural way by relying on the standard σ-algebra operations of complementation, finite intersection and finite union. Moreover, we let $\mu_{\mathscr{C}}$ denote the unique measure on $\sigma(\mathscr{C})$ such that $\mu_{\mathscr{C}}(Cyl(i)) = \frac{1}{2}$, see [25].

Now that formulae correspond to measurable sets, it makes sense to enrich the language of **PL** with *new* expressions able to represent conditions on the measure of such sets. By adapting Wagner's notion of counting operator [227, 226], we introduce two non-standard quantifiers, $\mathbf{C}^q$ and $\mathbf{D}^q$, with q ranging over $\mathbb{Q} \cap [0, 1]$. Then, counting-quantified formulae $\mathbf{C}^q F$ and $\mathbf{D}^q F$ intuitively express that F is satisfied in a certain portion of all its possible interpretations to be, respectively, greater or strictly smaller than the index q. For example, the formula $\mathbf{C}^{1/2} F$ expresses that F is satisfied by *at least* half of its valuations, namely F is true with probability greater than (or equal to) $\frac{1}{2}$. Similarly, the formula $\mathbf{D}^{3/4} F$ says that the probability for F to be true is strictly smaller than $\frac{3}{4}$. Semantically, this amounts to (resp.) checking that $\mu_{\mathscr{C}}([\![F]\!]) \geq \frac{1}{2}$ and $\mu_{\mathscr{C}}([\![F]\!]) < \frac{3}{4}$, where $\mu_{\mathscr{C}}$ is the standard Borel measure on $\mathscr{B}(2^{\mathbb{N}})$.

Syntax. Formally, the grammar for $\mathbf{CPL}_0$ is obtained by modifying the standard propositional language with two peculiar elements: special atoms and counting-quantified formulae.

[5] In particular since one cylinder has multiple representation, consistency problems may occur with Definition 3.1.2. For further details, see [25, pp. 28-30].

[6] This is the special thin cylinder defined in Section 3.1, but we consider 2 instead of $\mathbb{B}$ for uniformity with subsequent semantic definitions.

Definition 3.2.1 (Formulae of $\mathbf{CPL}_0$)**.** *Formulae of* $\boldsymbol{CPL}_0$ are defined by the grammar below:

$$F ::= \mathbf{i} \;\Big|\; \neg F \;\Big|\; F \wedge F \;\Big|\; F \vee F \;\Big|\; \mathbf{C}^q F \;\Big|\; \mathbf{D}^q F$$

where $i \in \mathbb{N}$ and $q \in \mathbb{Q} \cap [0,1]$.

Semantics. The formal definition of our semantics relies on the standard cylinder space $\mathscr{P}_{\mathscr{C}} = (2^{\mathbb{N}}, \sigma(\mathscr{C}), \mu_{\mathscr{C}})$, as presented in Section 3.1.

Definition 3.2.2 (Semantics of $\mathbf{CPL}_0$)**.** For each formula F of $\mathbf{CPL}_0$ its *interpretation* is the measurable set $[\![F]\!] \in \mathscr{B}(2^{\mathbb{N}})$ defined in an inductive way as follows:

$$\begin{aligned} [\![\mathbf{i}]\!] &:= Cyl(i) \\ [\![\neg G]\!] &:= 2^{\mathbb{N}} - [\![G]\!] \\ [\![G \wedge H]\!] &:= [\![G]\!] \cap [\![H]\!] \\ [\![G \vee H]\!] &:= [\![G]\!] \cup [\![H]\!] \end{aligned} \qquad\qquad [\![\mathbf{C}^q G]\!] := \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}([\![G]\!]) \geq q \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![\mathbf{D}^q G]\!] := \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}([\![G]\!]) < q \\ \emptyset & \text{otherwise.} \end{cases}$$

The examples below may help clarifying the intuitive meaning of the semantics of $\mathbf{CPL}_0$.

Example 3.2.1. Let us consider the counting formula $\mathbf{C}^{1/2}F$, where $F = G \vee H$, $G = \mathbf{0} \wedge \neg\mathbf{1}$ and $H = \neg\mathbf{0} \wedge \mathbf{1}$. The measurable sets, $[\![G]\!]$ and $[\![H]\!]$, have both measure $\frac{1}{4}$ and are disjoint. Hence, $\mu_{\mathscr{C}}\big([\![F]\!]\big) = \mu_{\mathscr{C}}\big([\![G]\!]\big) + \mu_{\mathscr{C}}\big([\![H]\!]\big) = \frac{1}{2}$. As a consequence, $[\![\mathbf{C}^{1/2}F]\!] = 2^{\mathbb{N}}$.

Example 3.2.2. Let $F = G \vee H$, where $G = (\mathbf{0} \wedge \neg\mathbf{1}) \vee \mathbf{2}$ and $H = (\neg\mathbf{0} \wedge \mathbf{1}) \vee \mathbf{2}$. Clearly, both sets $[\![G]\!]$ and $[\![H]\!]$ have measure $\frac{5}{8}$ (in fact, 5 of their 8 possible models are satisfying ones), but $[\![G]\!] \cap [\![H]\!] = Cyl(2)$. Hence, $\mu_{\mathscr{C}}\big([\![F]\!]\big) = \mu_{\mathscr{C}}\big([\![G]\!]\big) + \mu_{\mathscr{C}}\big([\![H]\!]\big) - \mu_{\mathscr{C}}(Cyl(2)) = \frac{3}{4}$.

Two formulae of $\mathbf{CPL}_0$, F and G, are said to be *logically equivalent*, noted $F \equiv G$, when $[\![F]\!] = [\![G]\!]$.

Definition 3.2.3 (Validity and Invalidity)**.** Let F be a formula of $\mathbf{CPL}_0$, F is *valid* when $[\![F]\!] = 2^{\mathbb{N}}$ and *invalid* when $[\![F]\!] = \emptyset$.

Observe that the two counting quantifiers are inter-definable, as it is easily shown semantically by Lemma 3.2.2 below. Yet they are not dual in the sense of standard modal operators: $\mathbf{C}^q F$ is *not* equivalent to $\neg\mathbf{D}^q\neg F$.

Lemma 3.2.1. *Let* F *be a formula of* $\boldsymbol{CPL}_0$*, then:*

$$[\![\mathbf{C}^0 F]\!] = 2^{\mathbb{N}} \qquad\qquad [\![\mathbf{D}^0 F]\!] = \emptyset.$$

Proof. For every F, $\mu_{\mathscr{C}}(\llbracket F \rrbracket) \geq 0$ holds. Thus,

$$\llbracket \mathbf{C}^0 F \rrbracket = \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}(\llbracket F \rrbracket) \geq 0 \\ \emptyset & \text{otherwise} \end{cases} \qquad \llbracket \mathbf{D}^0 F \rrbracket = \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}(\llbracket F \rrbracket) < 0 \\ \emptyset & \text{otherwise} \end{cases}$$
$$= 2^{\mathbb{N}} \qquad = \emptyset.$$

□

Lemma 3.2.2 (Inter-Definability)**.** *Let F be a formula of* ***CPL***$_0$ *and $q \in \mathbb{Q} \cap [0,1]$, then the following equivalences hold:*

$$\mathbf{C}^q F \equiv \neg \mathbf{D}^q F \qquad \mathbf{D}^q F \equiv \neg \mathbf{C}^q F.$$

Proof. The proof consists of simply applying Definition 3.2.2:

$$\llbracket \neg \mathbf{D}^q F \rrbracket = 2^{\mathbb{N}} - \llbracket \mathbf{D}^q F \rrbracket \qquad \llbracket \neg \mathbf{C}^q F \rrbracket = 2^{\mathbb{N}} - \llbracket \mathbf{C}^q F \rrbracket$$
$$= 2^{\mathbb{N}} - \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}(\llbracket F \rrbracket) < q \\ \emptyset & \text{otherwise} \end{cases} \qquad = 2^{\mathbb{N}} - \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}(\llbracket F \rrbracket) \geq q \\ \emptyset & \text{otherwise} \end{cases}$$
$$= \begin{cases} \emptyset & \text{if } \mu_{\mathscr{C}}(\llbracket F \rrbracket) < q \\ 2^{\mathbb{N}} & \text{otherwise} \end{cases} \qquad = \begin{cases} \emptyset & \text{if } \mu_{\mathscr{C}}(\llbracket F \rrbracket) \geq q \\ 2^{\mathbb{N}} & \text{otherwise} \end{cases}$$
$$= \llbracket \mathbf{C}^q F \rrbracket \qquad = \llbracket \mathbf{D}^q F \rrbracket.$$

□

Remarkably due to counting quantifiers it is even possible to express that a formula F is satisfied with probability *strictly greater than q* or *no smaller than q*, respectively as $\mathbf{D}^{(1-q)}\neg F$ and $\mathbf{C}^{(1-q)}\neg F$ (and combining them that a formula is true with precisely a given probability).[7]

3.3 Proof Theory of CPL$_0$

We introduce a one-sided, single-succedent sequent calculus, called $\mathbf{LK}_{\mathbf{CPL}_0}$, and prove it sound and complete with respect to the semantics of **CPL**$_0$.

The Labelled Language. The language of this rule system is constituted by *labelled* expressions of the form $\mathscr{b} \rightarrowtail F$ or $\mathscr{b} \leftarrowtail F$, where $\mathscr{b}$ is a Boolean formula and F is a formula of **CPL**$_0$. Intuitively, a labelled formula $\mathscr{b} \rightarrowtail F$ (resp., $\mathscr{b} \leftarrowtail F$) is true when the set of valuations satisfying $\mathscr{b}$ is included in (resp., includes) the interpretation of F.

Notation 3.3.1. In the following, we will use $\mathscr{b} \vDash \mathscr{c}$ for $\llbracket \mathscr{b} \rrbracket \subseteq \llbracket \mathscr{c} \rrbracket$.

Let us start by defining Boolean formulae.

[7] For further details, see Section 3.4.1.

Definition 3.3.1 (Boolean Formulae)**.** The grammar of *Boolean formulae* is as below:

$$\mathscr{b} ::= x_i \;\Big|\; \top \;\Big|\; \bot \;\Big|\; \neg\mathscr{b} \;\Big|\; \mathscr{b} \wedge \mathscr{b} \;\Big|\; \mathscr{b} \vee \mathscr{b},$$

where $i \in \mathbb{N}$. The *interpretation of a Boolean formula* $\mathscr{b}$, $[\![\mathscr{b}]\!] \in \mathscr{B}(2^{\mathbb{N}})$, is defined in an inductive way:

$$\begin{aligned} [\![x_i]\!] &:= Cyl(i) & [\![\neg\mathscr{b}]\!] &:= 2^{\mathbb{N}} - [\![\mathscr{b}]\!] \\ [\![\top]\!] &:= 2^{\mathbb{N}} & [\![\mathscr{b} \wedge \mathscr{c}]\!] &:= [\![\mathscr{b}]\!] \cap [\![\mathscr{c}]\!] \\ [\![\bot]\!] &:= \emptyset & [\![\mathscr{b} \vee \mathscr{c}]\!] &:= [\![\mathscr{b}]\!] \cup [\![\mathscr{c}]\!]. \end{aligned}$$

Given a Boolean formula $\mathscr{b}$, a *valuation* of $\mathscr{b}$ is any function $\theta : \mathsf{FV}(\mathscr{b}) \to \{\top, \bot\}$. The satisfaction relation $\theta \vDash \mathscr{b}$ is defined inductively in the obvious way.

We also express semantic properties of Boolean formulae, as $\mu([\![\mathscr{c}]\!]) \triangleright q$, where $\triangleright \in \{\geq, >, \leq, <, =\}$, $\mathscr{b}$ is a Boolean formula and $q \in \mathbb{Q} \cap [0,1]$.[8] Observe that the measure of a Boolean formula, $\mu_{\mathscr{C}}([\![\mathscr{b}]\!])$, can be related to the number $\sharp\mathtt{SAT}$ of the valuations making $\mathscr{b}$ true.

Lemma 3.3.1. *For any Boolean formula* $\mathscr{b}$ *containing exactly* n *distinct propositional variables* $x_{i_1}, \ldots, x_{i_n}$*,*

$$\mu_{\mathscr{C}}([\![\mathscr{b}]\!]) = \sharp\mathtt{SAT}(\mathscr{b}) \cdot 2^{-n}.$$

Proof. Any valuation $\theta : \{x_{i_1}, \ldots, x_{i_n}\} \mapsto 2$ is associated with a measurable set $X(\theta) \in \mathscr{B}(2^{\mathbb{N}})$ by letting

$$X(\theta) = \{f \mid (\forall j \leq n) f(j) = \theta(x_{i_j})\} = \bigcap_{i=0}^{n-1} Cyl(j)^{\theta(x_{i_j})},$$

where

$$Cyl(j)^{\theta(x_{i_j})} = \begin{cases} Cyl(j) & \text{if } \theta(x_{i_j}) = 1 \\ \overline{Cyl(j)} & \text{othrwise.} \end{cases}$$

Observe that $\mu_{\mathscr{C}}\big(X(\theta)\big) = 2^{-n}$. It is easily checked by induction on the structure of $\mathscr{b}$ that $[\![\mathscr{b}]\!] = \bigcup_{\theta \vDash \mathscr{b}} X(\theta)$. Since for all distinct θ and θ', $X(\theta) \cap X(\theta') = \emptyset$, we conclude that

$$\sharp\mathtt{SAT}(\mathscr{b}) \cdot 2^{-n} = \sum_{\theta \vDash \mathscr{b}} \mu_{\mathscr{C}}\big(X(\theta)\big) = \mu_{\mathscr{C}}\left(\bigcup_{\theta \vDash \mathscr{b}} X(\theta)\right) = \mu_{\mathscr{C}}([\![\mathscr{b}]\!]).$$

□

Then, *labelled formulae* and *external hypotheses* are introduced. Specifically, labelled formulae are defined as follows:

[8] For readability's sake, when dealing with semantic conditions in the context of sequent calculi we use μ instead of $\mu_{\mathscr{C}}$.

Definition 3.3.2 (Labelled Formulae)**.** A *labelled formula* is an expression of one of the forms $\mathscr{b} \rightarrowtail F$ and $\mathscr{b} \rightarrowtail F$, where $\mathscr{b}$ is a Boolean formula and F is a counting one. A *labelled sequent* is a sequent of the form $\vdash L$, where L is a labelled formula.

On the other hand, *external hypotheses* define a special class of formulae, which – as anticipated – express semantic properties of Boolean formulae or conditions to be checked inside $\mathscr{B}(2^{\mathbb{N}})$.

Definition 3.3.3 (External Hypothesis)**.** An *external hypothesis* is either an expression of the form $\mathscr{b} \vDash \mathscr{c}$ or of the form $\mu\big([\![\mathscr{b}]\!]\big) \triangleright q$, where $\triangleright \in \{\geq, >, \leq, <, =\}$, $\mathscr{b}, \mathscr{c}$ are Boolean formulae and $q \in \mathbb{Q} \cap [0, 1]$.

The Rule System. We now introduce a proof system for $\mathbf{CPL}_0$ in the form of a labelled sequent calculus.

Definition 3.3.4 (Sequent Calculus $\mathbf{LK}_{\mathbf{CPL}_0}$)**.** The proof system $\mathbf{LK}_{\mathbf{CPL}_0}$ is defined by the rules illustrated in Figure 3.1.

Notation 3.3.2 (μ-Rules)**.** Let us call *μ-rules* the two rules $R_{\mu}^{\rightarrowtail}$ and $R_{\mu}^{\rightarrowtail}$.

Notice that in counting rules – namely, $R_{\mathbf{C}}^{\rightarrowtail}$, $R_{\mathbf{C}}^{\rightarrowtail}$ and $R_{\mathbf{D}}^{\rightarrowtail}$, $R_{\mathbf{D}}^{\rightarrowtail}$ – the Boolean formula $\mathscr{b}$ in the conclusion is arbitrarily chosen. This is coherent with the semantics of counting-quantified formulae, which are interpreted as either $2^{\mathbb{N}}$ or $\emptyset$ (being respectively the superset and the subset of *any* set).

The use of external hypotheses – that is, of genuinely semantic conditions – as premisses of syntactic rules might seem somehow unsatisfactory. Nevertheless, such premisses do make sense from a computational viewpoint, as corresponding to the idea that, when searching for a proof of a counting formula, one might need to call an *oracle* measuring expressions of the form $\mu([\![\mathscr{b}]\!])$, in fact an oracle for $\sharp\mathtt{SAT}(\mathscr{b})$. Indeed, our calculus is designed having the counting hierarchy in mind, so accepting the idea of querying for an oracle. This is precisely "mirrored" by external hypotheses, which measure Boolean formulae. Due to these semantic conditions we can somehow divide our proof system in two parts: the counting and propositional aspects are dealt with in explicit way (by the rules), while the task of *implicitly* measuring Boolean formulae is delegated to external hypotheses. As said, this choice is guided by the overall interest of Part I, but notice that a *purely-syntactical* proof system could have been defined by substituting semantic conditions with an effective procedure to measure the probability of formulae [5].[9] Yet, as a drawback, this would lead to a more complex language and to the addition of more rules, so we preferred semantic hypotheses.

Derivations in $\mathbf{LK}_{\mathbf{CPL}_0}$ are defined in the standard way.

[9] Further studies in this direction would be of particular interest also to offer a way to provide poly-time verifiable proofs.

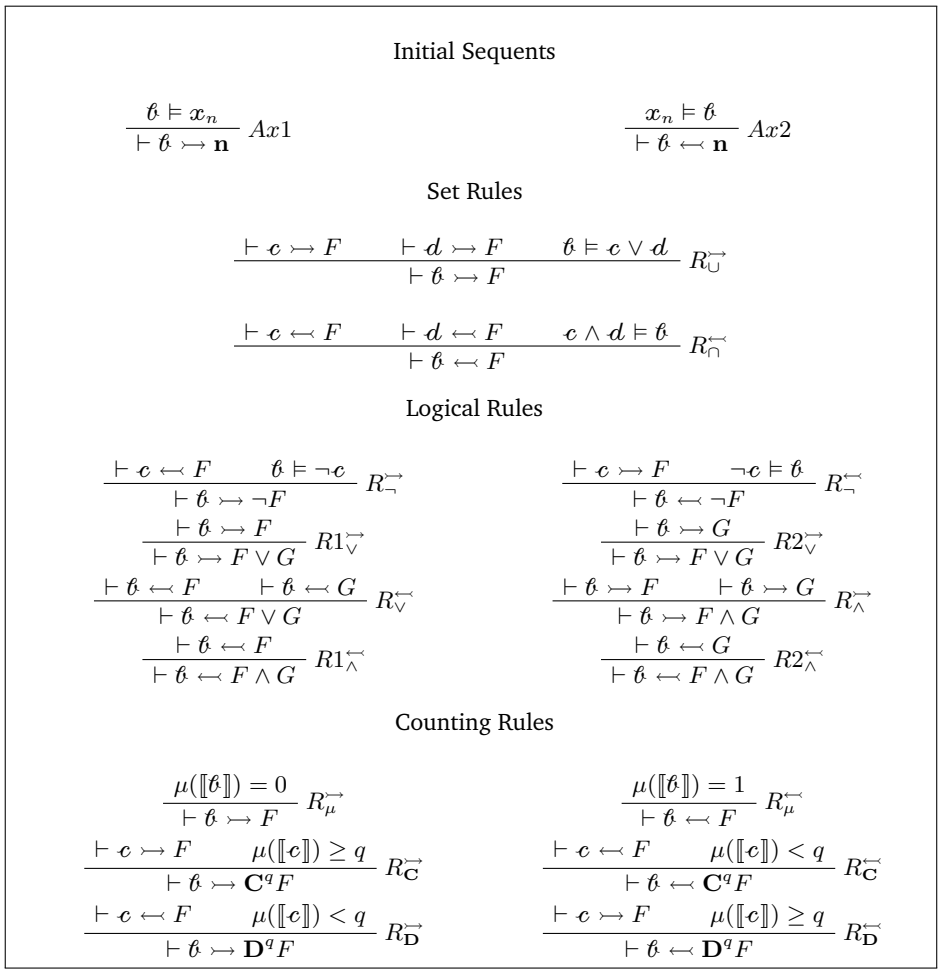


Figure 3.1: Sequent Calculus $\mathbf{LK}_{\mathbf{CPL}_0}$

Definition 3.3.5 (Derivation in $\mathbf{LK}_{\mathbf{CPL}_0}$)**.** A *derivation in* $\mathbf{LK}_{\mathbf{CPL}_0}$ is either an initial sequent, $Ax1$ and $Ax2$, or an instance of a μ-rule, $R^{\rightarrowtail}_{\mu}$ and $R^{\leftarrowtail}_{\mu}$, or is obtained by applying a rule of $\mathbf{LK}_{\mathbf{CPL}_0}$ to derivations concluding its premisses.

Let $\vdash_{\mathbf{CPL}_0} L$ indicate that $\vdash L$ is derivable by the rules in Figure 3.1. The definition of *derivation height* is canonical as well.

Definition 3.3.6 (Derivation Height)**.** The *height of a derivation in* $\mathbf{LK}_{\mathbf{CPL}_0}$ is the greatest number of successive applications of rules in it, where initial sequents and μ-rules have height 0.

In Figure 3.2 we provide an example of derivation in $\mathbf{LK}_{\mathbf{CPL}_0}$.

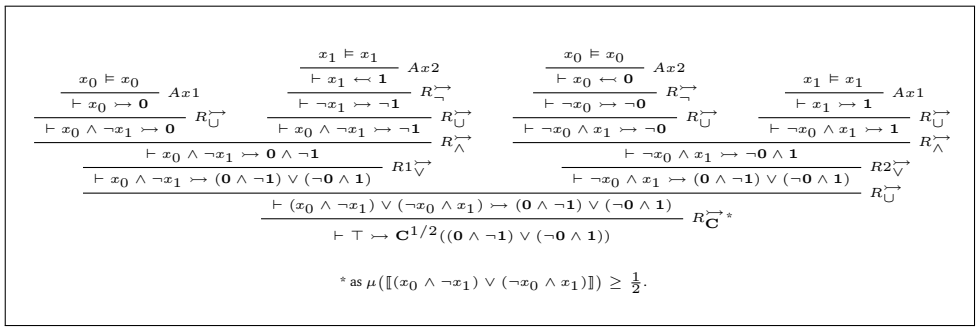


Figure 3.2: Derivation of $\vdash \top \rightarrowtail \mathbf{C}^{1/2}((\mathbf{0} \wedge \neg\mathbf{1}) \vee (\neg\mathbf{0} \wedge \mathbf{1}))$ in $\mathbf{LK}_{\mathbf{CPL}_0}$

3.3.1 Soundness and Completeness

As anticipated, $\mathbf{LK}_{\mathbf{CPL}_0}$ is sound and complete with respect to the given quantitative semantics: a labelled formula is valid when provable.

Preliminary Notions

Validity of labelled formulae and sequents is defined as follows.

Definition 3.3.7 (Validity)**.** A labelled formula $\mathscr{b} \rightarrowtail F$ (resp., $\mathscr{b} \nleftarrow F$) is *valid*, noted $\vDash \mathscr{b} \rightarrowtail F$ (resp., $\vDash \mathscr{b} \nleftarrow F$), when $[\![\mathscr{b}]\!] \subseteq [\![F]\!]$ (resp., $[\![F]\!] \subseteq [\![\mathscr{b}]\!]$). A sequent $\vdash L$ is *valid*, noted $\vDash L$, when L is a valid labelled formula.

Lemma 3.3.2. *For any formula of* $\mathbf{CPL}_0$ F*, we can construct a Boolean formula* $\mathscr{b}_F$ *such that:*

$$[\![F]\!] = [\![\mathscr{b}_F]\!].$$

Proof Sketch. The proof is trivial, except for F being of the form $F = \mathbf{C}^q G$ or $F = \mathbf{D}^q G$. In the former case, we have

$$\mathscr{b}_F = \begin{cases} \top & \text{if } \mu_{\mathscr{C}}([\![G]\!]) \geq q \\ \bot & \text{if } \mu_{\mathscr{C}}([\![G]\!]) < q. \end{cases}$$

Similarly, in the latter case, we have:

$$\mathscr{b}_F = \begin{cases} \top & \text{if } \mu_{\mathscr{C}}([\![G]\!]) < q \\ \bot & \text{if } \mu_{\mathscr{C}}([\![G]\!]) \geq q. \end{cases}$$

□

Soundness of $\mathbf{LK}_{\mathbf{CPL}_0}$

First, let us introduce the preliminary Lemma below, which is crucial to prove soundness.

Lemma 3.3.3. *For any Boolean formula $\mathscr{b}$, if $\mu_{\mathscr{C}}([\![\mathscr{b}]\!]) = 0$, then $[\![\mathscr{b}]\!] = \emptyset$.*

Proof. This is an immediate consequence of Lemma 3.3.1. The proof is by contraposition. If $[\![\mathscr{b}]\!] \neq \emptyset$, then by Lemma 3.3.1, $\mathscr{b}$ must have a satisfying model, i.e. $\mu_{\mathscr{C}}([\![\mathscr{b}]\!]) = \frac{\sharp\mathtt{SAT}(\mathscr{b})}{2^n} \geq \frac{1}{2^n}$, where n is the number of all the distinct variables occurring in $\mathscr{b}$.

□

Soundness is established by standard induction on the height of the derivation.

Proposition 3.3.1 (Soundness)**.** *If $\vdash L$ is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$, then $\vDash L$ holds.*

Proof. The proof is by induction on the height of the derivation of $\vdash L$.

Base Case. If $n = 0$, $\vdash L$ is either an initial sequent or is derived by a μ-rule. Both cases are trivial.

- $Ax1$. The derivation is of the following form:

$$\frac{\mathscr{b} \vDash x_n}{\vdash \mathscr{b} \rightarrowtail \mathbf{n}}\, Ax1$$

  Since $\mathscr{b} \vDash x_n$, then $[\![\mathscr{b}]\!] \subseteq [\![x_n]\!]$, that is $[\![\mathscr{b}]\!] \subseteq Cyl(n)$. Yet since $[\![\mathbf{n}]\!] = Cyl(n)$, also $[\![\mathscr{b}]\!] \subseteq [\![\mathbf{n}]\!]$, that is $\vDash \mathscr{b} \rightarrowtail \mathbf{n}$.

- $R^{\rightarrowtail}_{\mu}$. The derivation is of the following form:

$$\frac{\mu\big([\![\mathscr{b}]\!]\big) = 0}{\vdash \mathscr{b} \rightarrowtail F}\, R^{\rightarrowtail}_{\mu}$$

  Since $\mu\big([\![\mathscr{b}]\!]\big) = 0$, by Lemma 3.3.3, $[\![\mathscr{b}]\!] = \emptyset$. Then, for any F, $[\![\mathscr{b}]\!] \subseteq [\![F]\!]$, and we conclude $\vDash \mathscr{b} \rightarrowtail F$.

- The cases $Ax2$ and $R^{\rightarrowtail}_{\mu}$ are proved as $Ax1$ and $R^{\rightarrowtail}_{\mu}$, respectively.

Inductive case. Let us assume that soundness holds for derivations of heights up to n and show that it holds for derivations of height $n + 1$.

- $R^{\rightarrowtail}_{\cup}$. Let the last rule applied be an instance of $R^{\rightarrowtail}_{\cup}$ and the derivation be of the following form:

$$\frac{\begin{array}{ccc}\vdots & \vdots & \\ \vdash \mathscr{c} \rightarrowtail F & \vdash \mathscr{d} \rightarrowtail F & \mathscr{b} \vDash \mathscr{c} \vee \mathscr{d}\end{array}}{\vdash \mathscr{b} \rightarrowtail F}\, R^{\rightarrowtail}_{\cup}$$

By IH, $\vDash \mathscr{c} \rightarrowtail F$ and $\vDash \mathscr{d} \rightarrowtail F$, that is $[\![\mathscr{c}]\!] \subseteq [\![F]\!]$ and $[\![\mathscr{d}]\!] \subseteq [\![F]\!]$. Thus, for basic set theory, $[\![\mathscr{c}]\!] \cup [\![\mathscr{d}]\!] \subseteq [\![F]\!]$. Given the external hypothesis $\mathscr{b} \vDash \mathscr{c} \vee \mathscr{d}$ – that is, $[\![\mathscr{b}]\!] \subseteq [\![\mathscr{c}]\!] \cup [\![\mathscr{d}]\!]$ – also $[\![\mathscr{b}]\!] \subseteq [\![F]\!]$. So, we conclude $\vDash \mathscr{b} \rightarrowtail F$.

- $R^{\leftarrowtail}_{\cap}$. Assume that the last rule applied is an instance of $R^{\leftarrowtail}_{\cap}$ and that the derivation is of the following form:

$$\dfrac{\begin{array}{ccc}\vdots & & \vdots \\ \vdash \mathscr{c} \leftarrowtail F & \vdash \mathscr{d} \leftarrowtail F & \mathscr{c} \wedge \mathscr{d} \vDash \mathscr{b}\end{array}}{\vdash \mathscr{b} \leftarrowtail F}\, R^{\leftarrowtail}_{\cap}$$

By IH, $\vDash \mathscr{c} \leftarrowtail F$ and $\vDash \mathscr{d} \leftarrowtail F$, that is $[\![F]\!] \subseteq [\![\mathscr{c}]\!]$ and $[\![F]\!] \subseteq [\![\mathscr{d}]\!]$. Thus, for basic set theory, $[\![F]\!] \subseteq [\![\mathscr{c}]\!] \cap [\![\mathscr{d}]\!]$. Furthermore, for hypothesis $\mathscr{c} \wedge \mathscr{d} \vDash \mathscr{b}$, that is $[\![\mathscr{c}]\!] \cap [\![\mathscr{d}]\!] \subseteq [\![\mathscr{b}]\!]$. So, $[\![F]\!] \subseteq [\![\mathscr{b}]\!]$, and we conclude $\vDash \mathscr{b} \leftarrowtail F$.

- $R^{\rightarrowtail}_{\neg}$. Assume that the last rule applied is an instance of $R^{\rightarrowtail}_{\neg}$ and that the derivation is of the following form:

$$\dfrac{\begin{array}{cc}\vdots & \\ \vdash \mathscr{c} \leftarrowtail F & \mathscr{b} \vDash \neg\mathscr{c}\end{array}}{\vdash \mathscr{b} \rightarrowtail \neg F}\, R^{\rightarrowtail}_{\neg}$$

By IH, $\vDash \mathscr{c} \leftarrowtail F$, so $[\![F]\!] \subseteq [\![\mathscr{c}]\!]$. By basic set theory $\left(2^{\mathbb{N}} - [\![\mathscr{c}]\!]\right) \subseteq \left(2^{\mathbb{N}} - [\![F]\!]\right)$. Moreover, $\mathscr{b} \vDash \neg\mathscr{c}$, that is $[\![\mathscr{b}]\!] \subseteq (2^{\mathbb{N}} - [\![\mathscr{c}]\!])$. So $[\![\mathscr{b}]\!] \subseteq \left(2^{\mathbb{N}} - [\![F]\!]\right) = [\![\neg F]\!]$, and we conclude $\vDash \mathscr{b} \rightarrowtail \neg F$.

- $R^{\leftarrowtail}_{\neg}$. The proof is equivalent to the previous one.

- $R1^{\rightarrowtail}_{\vee}$. Assume that the last rule applied is an instance of $R1^{\rightarrowtail}_{\vee}$ and the derivation is of the following form:

$$\dfrac{\begin{array}{c}\vdots \\ \vdash \mathscr{b} \rightarrowtail F\end{array}}{\vdash \mathscr{b} \rightarrowtail F \vee G}\, R1^{\rightarrowtail}_{\vee}$$

By IH, $\vDash \mathscr{b} \rightarrowtail F$ and so $[\![\mathscr{b}]\!] \subseteq [\![F]\!]$. For basic set theory, $[\![\mathscr{b}]\!] \subseteq [\![F]\!] \cup [\![G]\!]$, that is $[\![\mathscr{b}]\!] \subseteq [\![F \vee G]\!]$, and thus $\vDash \mathscr{b} \rightarrowtail F \vee G$.

- $R2^{\rightarrowtail}_{\vee}$. Assume that the last rule applied is an instance of $R2^{\rightarrowtail}_{\vee}$ and the derivation is of the following form:

$$\dfrac{\begin{array}{c}\vdots \\ \vdash \mathscr{b} \rightarrowtail G\end{array}}{\vdash \mathscr{b} \rightarrowtail F \vee G}\, R2^{\rightarrowtail}_{\vee}$$

By IH, $\vDash \mathscr{b} \rightarrowtail G$ and so $[\![\mathscr{b}]\!] \subseteq [\![G]\!]$. For basic set theory, $[\![\mathscr{b}]\!] \subseteq [\![F]\!] \cup [\![G]\!]$ – that is, $[\![\mathscr{b}]\!] \subseteq [\![F \vee G]\!]$ – and thus $\vDash \mathscr{b} \rightarrowtail F \vee G$.

- $R^{\leftarrowtail}_{\vee}$. Assume that the last rule applied is an instance of $R^{\leftarrowtail}_{\vee}$ and the derivation is of the following form:

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \vdash \mathscr{b} \leftarrowtail F & \vdash \mathscr{b} \leftarrowtail G\end{array}}{\vdash \mathscr{b} \leftarrowtail F \vee G}\, R^{\leftarrowtail}_{\vee}$$

  By IH, $\vDash \mathscr{b} \leftarrowtail F$, and $\vDash \mathscr{b} \leftarrowtail G$, that is $[\![F]\!] \subseteq [\![\mathscr{b}]\!]$ and $[\![G]\!] \subseteq [\![\mathscr{b}]\!]$. For basic set theory, $[\![F]\!] \cup [\![G]\!] \subseteq [\![\mathscr{b}]\!]$ – that is $[\![F \vee G]\!] \subseteq [\![\mathscr{b}]\!]$ – and therefore $\vDash \mathscr{b} \leftarrowtail F \vee G$.

- $R^{\rightarrowtail}_{\wedge}, R1^{\leftarrowtail}_{\wedge}, R2^{\leftarrowtail}_{\wedge}$. These cases are equivalent to $R1^{\rightarrowtail}_{\vee}, R2^{\rightarrowtail}_{\vee}, R^{\rightarrowtail}_{\vee}$.

- $R^{\rightarrowtail}_{\mathbf{C}}$. Assume that the last rule applied is an instance of $R^{\rightarrowtail}_{\mathbf{C}}$ and the derivation is of the following form:

$$\frac{\begin{array}{cc}\vdots & \\ \vdash \mathscr{c} \rightarrowtail F & \mu([\![\mathscr{c}]\!]) \geq q\end{array}}{\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F}\, R^{\rightarrowtail}_{\mathbf{C}}$$

  By IH, $\vDash \mathscr{c} \rightarrowtail F$, that is $[\![\mathscr{c}]\!] \subseteq [\![F]\!]$. Given the external hypothesis, $\mu([\![\mathscr{c}]\!]) \geq q$, by basic measure theory, also $\mu_{\mathscr{C}}([\![F]\!]) \geq q$. Thus, $[\![\mathbf{C}^q F]\!] = 2^{\mathbb{N}}$ and for any $\mathscr{b}$, $[\![\mathscr{b}]\!] \subseteq [\![\mathbf{C}^q F]\!]$. So, we conclude $\vDash \mathscr{b} \rightarrowtail \mathbf{C}^q F$.

- $R^{\leftarrowtail}_{\mathbf{C}}$. Assume that the last rule applied is an instance of $R^{\leftarrowtail}_{\mathbf{C}}$ and the derivation is in the following form:

$$\frac{\begin{array}{cc}\vdots & \\ \vdash \mathscr{c} \leftarrowtail F & \mu([\![\mathscr{c}]\!]) < q\end{array}}{\vdash \mathscr{b} \leftarrowtail \mathbf{C}^q F}\, R^{\leftarrowtail}_{\mathbf{C}}$$

  By IH, $\vDash \mathscr{c} \leftarrowtail F$, that is $[\![F]\!] \subseteq [\![\mathscr{c}]\!]$. Since for the external hypothesis $\mu([\![\mathscr{c}]\!]) < q$, by basic measure theory, also $\mu_{\mathscr{C}}([\![F]\!]) < q$. Thus, $[\![\mathbf{C}^q F]\!] = \emptyset$ and for any $\mathscr{b}$, $[\![\mathbf{C}^q F]\!] \subseteq [\![\mathscr{b}]\!]$. So, we conclude $\vDash \mathscr{b} \leftarrowtail \mathbf{C}^q F$.

- $R^{\rightarrowtail}_{\mathbf{D}}, R^{\rightarrowtail}_{\mathbf{D}}$. These cases are proved as $R^{\leftarrowtail}_{\mathbf{C}}$ and $R^{\rightarrowtail}_{\mathbf{C}}$.

□

Completeness for $\mathbf{LK}_{\mathbf{CPL}_0}$

The proof of completeness is less straightforward. Let us start with notational conventions.

Notation 3.3.3. We use $\Psi_1, \Psi_2, \ldots$ to denote labelled sequents in the language of $\mathbf{LK}_{\mathbf{CPL}_0}$ and $S, T, \ldots$ to denote sets of such sequents.

First, we introduce a *decomposition relation*, $\rightsquigarrow$, between finite sets of sequents, allowing us to decompose the validity of a sequent into that of a finite set of less complex sequents. Specifically, $\rightsquigarrow$ is in turn defined based on the relation $\rightsquigarrow_0$ between sequents and finite sets of sequents, and letting $S \rightsquigarrow T$ whenever $S = S' \cup \{\vdash L\}$, $\vdash L \rightsquigarrow_0 T$ and $T = S' \cup T'$. Then, it is shown that a (complex) valid sequent is decomposable into a finite set of non-decomposable valid sequents, and, from the provability of the latter, one can climb back to the validity of the original sequent, using the rules of $\mathbf{LK}_{\mathbf{CPL}_0}$. Existential preservation of validity relies on the fact that $\rightsquigarrow$ is strongly normalizing. Normalization is proved in a standard way by defining a size-function $\mathsf{ms}(S)$ over finite sets of sequents and showing that, whenever $S \rightsquigarrow T$, $\mathsf{ms}(T) < \mathsf{ms}(S)$. For a sequent $\Psi = \vdash L$, let $\mathsf{cn}(\Psi)$ be the size of the counting part, say F, of L, i.e. the number of connectives in L, $\mathsf{cn}(L)(= \mathsf{cn}(F))$. Then, $\mathsf{ms}(S)$ is defined as $\sum_i^k \mathsf{cn}(\Psi_i)$, where $S = \{\Psi_1, \ldots, \Psi_k\}$. Finally, given a finite set of sequents $S = \{\Psi_1, \ldots, \Psi_k\}$, let $\vDash S$ (resp., $\vdash_{\mathbf{CPL}_0} S$) indicate that $\vDash \Psi_i$ (resp., $\vdash_{\mathbf{CPL}_0} \Psi_i$) holds for all $i = 1, \ldots, k$.

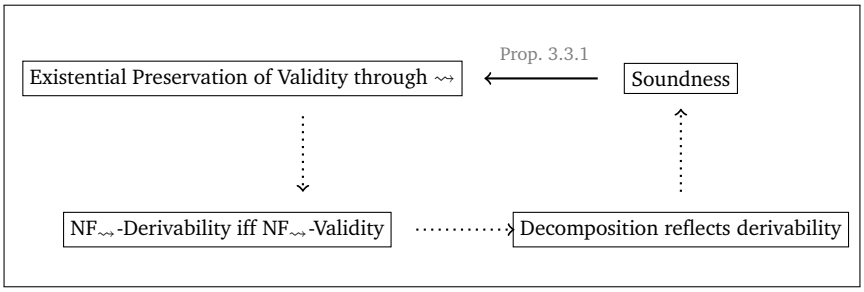


Figure 3.3: Soundness and Completeness of $\mathbf{LK}_{\mathbf{CPL}_0}$

The fundamental ingredients of the completeness proof are then expressed by the following three properties:

1. if T is $\rightsquigarrow$-normal, then $\vDash T$ if and only if $\vdash_{\mathbf{CPL}_0} T$.
2. if $\vDash S$ with S not $\rightsquigarrow$-normal, then there is a T such that $S \rightsquigarrow T$ and $\vDash T$
3. if $\vdash_{\mathbf{CPL}_0} T$ and $S \rightsquigarrow T$, then $\vdash_{\mathbf{CPL}_0} S$.

Using these, one can check that if $\vDash L$ holds, then by (2.) $\vDash S$ holds for some $\rightsquigarrow$ normal form S of $\vdash L$. Then, by (1.), $\vdash_{\mathbf{CPL}_0} S$ holds and by (3.) we conclude that $\vdash_{\mathbf{CPL}_0} L$, as summarized in Figure 3.4.

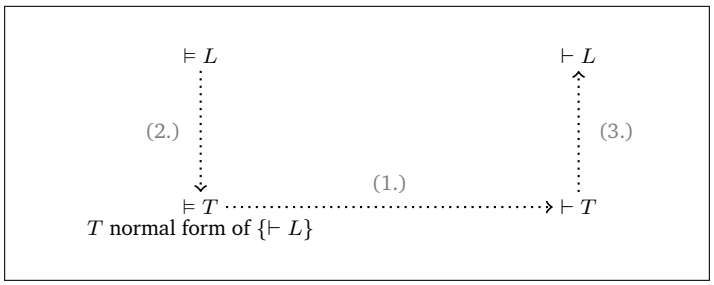


Figure 3.4: Skeleton of $\mathbf{LK}_{\mathbf{CPL}_0}$-Completeness Proof

Preliminary Notions. In order to prove completeness, some preliminary notions and lemmas are needed.

Definition 3.3.8 (Basic Formula)**.** A *basic formula* of $\mathbf{LK}_{\mathbf{CPL}_0}$ is a labelled formula, $\mathscr{b} \rightarrowtail F$ or $\mathscr{b} \leftarrowtail F$, in which the counting part, F, is atomic.

Definition 3.3.9 (Regular Sequent)**.** A *regular sequent* of $\mathbf{LK}_{\mathbf{CPL}_0}$ is a sequent of the form $\vdash L$, where L is a basic formula.

The notion of decomposition rewriting reduction is defined by the following decomposition rules.

Definition 3.3.10 (Decomposition Rewriting Reduction, $\rightsquigarrow_0$)**.** The *decomposition rewriting reduction* $\rightsquigarrow_0$, from a sequent to a set of sequents (both in the language of $\mathbf{LK}_{\mathbf{CPL}_0}$), is defined by the following decomposition rules:

$$
\begin{array}{rcl}
\text{if } \mathscr{b} \models \neg\mathscr{c},\ \vdash \mathscr{b} \rightarrowtail \neg F & \rightsquigarrow_0 & \{\vdash \mathscr{c} \leftarrowtail F\} \\
\text{if } \neg\mathscr{c} \models \mathscr{b},\ \vdash \mathscr{b} \leftarrowtail \neg F & \rightsquigarrow_0 & \{\vdash \mathscr{c} \rightarrowtail F\} \\
\text{if } \mathscr{b} \models \mathscr{c} \vee \mathscr{d},\ \vdash \mathscr{b} \rightarrowtail F \vee G & \rightsquigarrow_0 & \{\vdash \mathscr{c} \rightarrowtail F, \vdash \mathscr{d} \rightarrowtail G\} \\
\vdash \mathscr{b} \leftarrowtail F \vee G & \rightsquigarrow_0 & \{\vdash \mathscr{b} \leftarrowtail F, \vdash \mathscr{b} \leftarrowtail G\} \\
\vdash \mathscr{b} \rightarrowtail F \wedge G & \rightsquigarrow_0 & \{\vdash \mathscr{b} \rightarrowtail F, \vdash \mathscr{b} \rightarrowtail G\} \\
\text{if } \mathscr{c} \wedge \mathscr{d} \models \mathscr{b},\ \vdash \mathscr{b} \leftarrowtail F \wedge G & \rightsquigarrow_0 & \{\vdash \mathscr{c} \leftarrowtail F, \vdash \mathscr{d} \leftarrowtail G\} \\
\text{if } \mu([\![\mathscr{c}]\!]) \geq q,\ \vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F & \rightsquigarrow_0 & \{\vdash \mathscr{c} \rightarrowtail F\} \\
\text{if } \mu([\![\mathscr{c}]\!]) < q,\ \vdash \mathscr{b} \leftarrowtail \mathbf{C}^q F & \rightsquigarrow_0 & \{\vdash \mathscr{c} \leftarrowtail F\} \\
\text{if } \mu([\![\mathscr{c}]\!]) < q,\ \vdash \mathscr{b} \rightarrowtail \mathbf{D}^q F & \rightsquigarrow_0 & \{\vdash \mathscr{c} \leftarrowtail F\} \\
\text{if } \mu([\![\mathscr{c}]\!]) \geq q,\ \vdash \mathscr{b} \leftarrowtail \mathbf{D}^q F & \rightsquigarrow_0 & \{\vdash \mathscr{c} \rightarrowtail F\} \\
\text{if } \mu([\![\mathscr{b}]\!]) = 0,\ \vdash \mathscr{b} \rightarrowtail F & \rightsquigarrow_0 & \{\} \\
\text{if } \mu([\![\mathscr{b}]\!]) = 1,\ \vdash \mathscr{b} \leftarrowtail F & \rightsquigarrow_0 & \{\} \\
\text{if } \mu([\![\mathscr{b}]\!]) \neq 0,\ \vdash \mathscr{b} \rightarrowtail \mathbf{D}^0 F & \rightsquigarrow_0 & \{\vdash \bot\} \\
\text{if } \mu([\![\mathscr{b}]\!]) \neq 1,\ \vdash \mathscr{b} \leftarrowtail \mathbf{C}^0 F & \rightsquigarrow_0 & \{\vdash \bot\}.
\end{array}
$$

Preliminarily notice that – as we "measure" the size of a labelled formula considering the size of its *counting* part, the application of each of these rewriting rules decreases the size of the involved formula and then of the corresponding sequent, but not that of Boolean formulae.

Rewriting rules are so defined that each application of a decomposition reduction on an arbitrary sequent, $\vdash L$, leads to a set of sequents $\{\vdash L_1, \ldots, \vdash L_n\}$, such that for every $i \in \{1, \ldots, n\}$, the number of connectives of $\vdash L_i$ is (strictly) smaller than that of $\vdash L$. Basing on $\rightsquigarrow_0$, it is possible to define a set-decomposition reduction, $\rightsquigarrow$, from a set of sequents to a set of sequents.

Definition 3.3.11 (Set Decomposition, $\rightsquigarrow$)**.** The *set-decomposition reduction* $\rightsquigarrow$, from a set of sequents in $\mathbf{LK}_{\mathbf{CPL}_0}$ to another set of sequents is defined as follows:

$$\frac{\vdash L_i \rightsquigarrow \{\vdash L_{i_1}, \ldots, \vdash L_{i_m}\}}{\{\vdash L_1, \ldots, \vdash L_i, \ldots, \vdash L_n\} \rightsquigarrow \{\vdash L_1, \ldots, \vdash L_{i_1}, \ldots, \vdash L_{i_m}, \ldots, \vdash L_n\}}$$

Otherwise said, $\rightsquigarrow$ is the natural lifting of $\rightsquigarrow_0$ to a relation over sets. Observe, indeed, that each predicate concerning one sequent can be naturally generalized to sets of sequents by stipulating that a predicate holds for the set when it holds for every sequent in it. For clarity's sake, given a sequent $\vdash L$, we call its *corresponding set*, the set including only this sequent as its element, namely $\{\vdash L\}$.

The notion of $\rightsquigarrow$-normal form is defined as expected: a set of sequents is in $\rightsquigarrow$-normal form when no set-decomposition reduction, $\rightsquigarrow$, can be applied on it. Otherwise said, the set is in $\rightsquigarrow$-normal form if there is no sequent in it on which $\rightsquigarrow_0$ can be applied.

Definition 3.3.12 ($\rightsquigarrow$-Normal Form)**.** A sequent is a $\rightsquigarrow$*-normal form* if no decomposition rewriting reduction rule, $\rightsquigarrow_0$, can be applied on it. A set of sequents is in $\rightsquigarrow$-normal form if it cannot be reduced by any $\rightsquigarrow$ set-rewriting rule.

As anticipated, the proof of completeness relies on the given rewriting reductions, $\rightsquigarrow_0$ and $\rightsquigarrow$, in a crucial way, and is based on three auxiliary steps: (i.) validity is *existentially* preserved through $\rightsquigarrow$-decomposition, that is each valid sequent, has (at least) one valid $\rightsquigarrow$-normal form, (ii.) each $\rightsquigarrow_0$-normal form is valid when it is derivable, (iii.) derivability is "reflected by" $\rightsquigarrow_0$-decomposition, that is given a (set of) sequent(s) which is $\rightsquigarrow_0$-decomposed into another set of sequents, if the $\rightsquigarrow$-decomposed set is derivable, then it is possible to construct a derivation for the original (set of) sequent(s).

Strong Normalization of $\rightsquigarrow$. It is proved that $\rightsquigarrow$ is strongly normalizing, and so that each decomposition process terminates. A couple of auxiliary definitions are needed, first.

Definition 3.3.13 (Number of Connectives, cn)**.** The *number of connectives of a labelled formula* L, $\mathsf{cn}(L)$, is the number of connectives of its counting part –

labelled $\mathsf{cn}(F)$ as well – and is inductively defined as follows:

$$\begin{gathered}
\mathsf{cn}(\mathscr{b} \rightarrowtail \mathbf{n}) = \mathsf{cn}(\mathscr{b} \leftarrowtail \mathbf{n}) = \mathsf{cn}(\mathbf{n}) := 1 \\
\mathsf{cn}(\mathscr{b} \rightarrowtail \neg F) = \mathsf{cn}(\mathscr{b} \leftarrowtail \neg F) = \mathsf{cn}(\neg F) := 1 + \mathsf{cn}(F) \\
\mathsf{cn}(\mathscr{b} \rightarrowtail F \triangle G) = \mathsf{cn}(\mathscr{b} \leftarrowtail F \triangle G) = \mathsf{cn}(F \triangle G) := 1 + \mathsf{cn}(F) + \mathsf{cn}(G) \\
\mathsf{cn}(\mathscr{b} \rightarrowtail \bullet F) = \mathsf{cn}(\mathscr{b} \leftarrowtail \bullet F) = \mathsf{cn}(\bullet F) := 1 + \mathsf{cn}(F)
\end{gathered}$$

with $\triangle \in \{\wedge, \vee\}$ and $\bullet \in \{\mathbf{C}^q, \mathbf{D}^q\}$. The *number of connectives of the sequent* $\vdash L$, $\mathsf{cn}(\vdash L)$, is the number of connectives of its labelled formulae:

$$\mathsf{cn}(\vdash L) := \mathsf{cn}(L).$$

Definition 3.3.14 (Set Measure, ms)**.** Given a set of sequents $\{\vdash L_1, \cdots \vdash L_n\}$, its *measure*, $\mathsf{ms}(\{\vdash L_1, \ldots, \vdash L_n\})$ is defined as follows:

$$\begin{gathered}
\mathsf{ms}(\{\}) := 0 \\
\mathsf{ms}(\{\vdash L_1, \ldots, \vdash L_n\}) := \mathsf{cn}(\vdash L_1) + \cdots + \mathsf{cn}(\vdash L_n).
\end{gathered}$$

Lemma 3.3.4. *The reduction $\rightsquigarrow$ is strongly normalizing.*

Proof. That every set of sequents is $\rightsquigarrow$-strongly normalizing is proved by showing that, if $\{\vdash L_1, \ldots, \vdash L_m\} \rightsquigarrow_0 \{\vdash L'_1, \ldots, \vdash L'_m\}$, then

$$\mathsf{ms}(\{\vdash L_1, \ldots, \vdash L_m\}) > \mathsf{ms}(\{\vdash L'_1, \ldots, \vdash L'_m\}).$$

The proof is based on exhaustive inspection of all possible forms of $\rightsquigarrow$-reduction applicable to the given set, that is by dealing with all possible forms of $\rightsquigarrow_0$-reduction of one of the $\vdash L_i$, where $i \in \{1, \ldots, m\}$. For simplicity, we take (an arbitrary) $\vdash L_i$ to be the "active" sequent of $\rightsquigarrow$. We then consider all possible forms of $\rightsquigarrow_0$ on which $\rightsquigarrow$ can be based:

- $L_i = \mathscr{b} \rightarrowtail \neg F$. Assume that $\vdash L_i$ is the active sequent in the given $\rightsquigarrow$-decomposition and that $\rightsquigarrow$ is based on the $\rightsquigarrow_0$ below:

$$\vdash \mathscr{b} \rightarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \leftarrowtail F\}$$

  where $\mathscr{b} \vDash \neg\mathscr{c}$. Thus:

$$\begin{aligned}
\mathsf{ms}(\{\vdash L_1, \cdots \vdash \mathscr{c} \leftarrowtail F, \cdots \vdash L_m\}) &\overset{3.3.14}{=} \mathsf{cn}(L_1) + \cdots + \mathsf{cn}(\mathscr{c} \leftarrowtail F) + \cdots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) + \cdots + \mathsf{cn}(F) + \cdots + \mathsf{cn}(L_m) \\
&< \mathsf{cn}(L_1) + \cdots + \big(\mathsf{cn}(\neg F) + 1\big) + \cdots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) + \cdots + \mathsf{cn}(\neg F) + \cdots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) + \cdots + \mathsf{cn}(\mathscr{b} \rightarrowtail \neg F) + \ldots \mathsf{cn}(L_m) \\
&\overset{3.3.14}{=} \mathsf{ms}\big(\{\vdash L_1, \cdots \vdash \mathscr{b} \rightarrowtail \neg F, \cdots \vdash L_m\}\big).
\end{aligned}$$

- $L_i = \mathscr{b} \leftarrowtail \neg F$. Similar to the case above.

- $L_i = \mathscr{b} \rightarrowtail F \vee G$. Assume that $\vdash L_i$ is the active sequent of $\rightsquigarrow$ and that $\rightsquigarrow$ is based on the following $\rightsquigarrow_0$:

$$\vdash \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \rightarrowtail F, \vdash \mathscr{d} \rightarrowtail G\}$$

where $\mathscr{b} \vDash \mathscr{c} \vee \mathscr{d}$. Thus,

$$\begin{aligned}
\mathsf{ms}(\{\vdash L_1, \ldots \vdash \mathscr{c} \rightarrowtail F, \vdash \mathscr{d} \rightarrowtail G, \ldots \vdash L_m\}) &\overset{3.3.14}{=} \mathsf{cn}(L_1) \ldots + \mathsf{cn}(\mathscr{c} \rightarrowtail F) + \mathsf{cn}(\mathscr{d} \rightarrowtail G) \ldots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) \ldots + \mathsf{cn}(F) + \mathsf{cn}(G) \ldots + \mathsf{cn}(L_m) \\
&< \mathsf{cn}(L_1) \ldots + (\mathsf{cn}(F) + \mathsf{cn}(G) + 1) \ldots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) \ldots + \mathsf{cn}(\mathscr{b} \rightarrowtail F \vee G) \ldots + \mathsf{cn}(L_m) \\
&\overset{3.3.14}{=} \mathsf{ms}(\{\vdash L_1, \ldots \vdash \mathscr{b} \rightarrowtail F \vee G, \ldots \vdash L_m\}).
\end{aligned}$$

- $L_i = \mathscr{b} \rightarrowtail F \vee G$. Assume that $\vdash L_i$ is the active sequent of $\rightsquigarrow$ and $\rightsquigarrow$ is based on the following $\rightsquigarrow_0$:

$$\vdash \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{b} \rightarrowtail F, \vdash \mathscr{b} \rightarrowtail G\}$$

Thus,

$$\begin{aligned}
\mathsf{ms}(\{\vdash L_1, \ldots, \vdash \mathscr{b} \rightarrowtail F, \vdash \mathscr{b} \rightarrowtail G, \ldots \vdash L_m\}) &\overset{3.3.14}{=} \mathsf{cn}(L_1) \ldots + \mathsf{cn}(\mathscr{b} \rightarrowtail F) + \mathsf{cn}(\mathscr{b} \rightarrowtail G) \cdots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) \ldots + \mathsf{cn}(F) + \mathsf{cn}(G) \ldots + \mathsf{cn}(L_m) \\
&< \mathsf{cn}(L_1) \ldots + (\mathsf{cn}(F) + \mathsf{cn}(G) + 1) \ldots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) \ldots + \mathsf{cn}(\mathscr{b} \rightarrowtail F \vee G) \ldots + \mathsf{cn}(L_m) \\
&\overset{3.3.14}{=} \mathsf{ms}(\{\vdash L_1, \ldots, \vdash \mathscr{b} \rightarrowtail F \vee G, \ldots \vdash L_m\}).
\end{aligned}$$

- $L_i = \mathscr{b} \rightarrowtail F \wedge G, L_i = \mathscr{b} \rightarrowtail F \wedge G$. Similar to the two cases above.

- $L_i = \mathscr{b} \rightarrowtail \mathbf{C}^q F$. Assume that $\vdash L_i$ is the active sequent of the given $\rightsquigarrow$-decomposition and that $\rightsquigarrow$ is in its turn based on the $\rightsquigarrow_0$-decomposition below:

$$\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \rightarrowtail F\}$$

where $\mu([\![\mathscr{c}]\!]) \geq q$. Then,

$$\begin{aligned}
\mathsf{ms}(\{\vdash L_1, \ldots \vdash \mathscr{c} \rightarrowtail F, \ldots \vdash L_m\}) &\overset{3.3.14}{=} \mathsf{cn}(L_1) + \cdots + \mathsf{cn}(\mathscr{c} \rightarrowtail F) + \cdots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) + \cdots + \mathsf{cn}(F) + \cdots + \mathsf{cn}(L_m) \\
&< \mathsf{cn}(L_1) + \cdots + \big(\mathsf{cn}(F) + 1\big) + \cdots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) + \cdots + \mathsf{cn}(\mathbf{C}^q F) + \cdots + \mathsf{cn}(L_m) \\
&\overset{3.3.13}{=} \mathsf{cn}(L_1) + \cdots + \mathsf{cn}(\mathscr{b} \rightarrowtail \mathbf{C}^q F) + \cdots + \mathsf{cn}(L_m) \\
&\overset{3.3.14}{=} \mathsf{ms}\big(\{\vdash L_1, \ldots, \vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F, \ldots, \vdash L_m\}\big).
\end{aligned}$$

- $L_i = \mathscr{b} \rightarrowtail \mathbf{C}^q F, L_i = \mathscr{b} \rightarrowtail \mathbf{D}^q F, L_i = \mathscr{b} \rightarrowtail \mathbf{D}^q F$. Similar to the case above.

- $L_i = \mathscr{b} \rightarrowtail F$. Assume that $\vdash L_i$ is the active sequent in the given $\rightsquigarrow$-decomposition and that $\rightsquigarrow$ is based on the $\rightsquigarrow_0$-decomposition below:

$$\vdash \mathscr{b} \rightarrowtail F \quad \rightsquigarrow_0 \quad \{\}$$

  where $\mu([\![\mathscr{b}]\!]) = 0$. Since for Definition 3.3.13 and 3.3.14, $\mathsf{cn}(\{\}) = 0$ and $\mathsf{cn}(\vdash \mathscr{b} \rightarrowtail F) > 0$, clearly $\mathsf{cn}(\{\}) < \mathsf{cn}(\vdash \mathscr{b} \rightarrowtail F)$.

- $L_i = \mathscr{b} \rightarrowtail F, L_i = \mathscr{b} \rightarrowtail \mathbf{D}^0 F, L_i = \mathscr{b} \rightarrowtail \mathbf{C}^0 F$. Similar to the case above.

□

On $\rightsquigarrow$-Normal Sequents. It is possible to show that $\rightsquigarrow$-normal sequents are valid if and only if they are derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$. First, it is shown that sequents which are $\rightsquigarrow$-normal are regular. Then, it is proved that regular sequents are derivable if and only if they are valid. These two results together imply that $\rightsquigarrow$-normal sequents are valid when derivable.

Lemma 3.3.5. *If a (non-empty) sequent is $\rightsquigarrow$-normal, then it is regular.*

Proof. The proof is by contraposition. Given an arbitrary *non*-regular sequent, call it $\vdash L$, it is shown that it is not $\rightsquigarrow$-normal. Every possible form of labelled formula, call it L, is considered.

- $L = \mathscr{b} \rightarrowtail \neg F$. Clearly, for any $\mathscr{b}$ $[\![\mathscr{b}]\!] \subseteq 2^{\mathbb{N}}$ and $[\![\neg\bot]\!] = 2^{\mathbb{N}}$. So, $\mathscr{b} \vDash \neg\bot$ and the following decomposition is well-defined:

$$\vdash \mathscr{b} \rightarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash \bot \rightarrowtail F\}.$$

- $L = \mathscr{b} \rightarrowtail \neg F$. Clearly, for any $\mathscr{b}$ $\emptyset \subseteq [\![\mathscr{b}]\!]$, and $[\![\neg\top]\!] = \emptyset$. Then, $\neg\top \vDash \mathscr{b}$, and the following decomposition is well-defined:

$$\vdash \mathscr{b} \rightarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash \top \rightarrowtail F\}$$

- $L = \mathscr{b} \rightarrowtail F \vee G$. For any $\mathscr{b}$, $[\![\mathscr{b}]\!] \subseteq 2^{\mathbb{N}}$ and $[\![\top \vee \top]\!] = 2^{\mathbb{N}}$. So, $\mathscr{b} \vDash \top \vee \top$ and the following decomposition is well-defined:

$$\vdash \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash \top \rightarrowtail F, \vdash \top \rightarrowtail G\}.$$

- $L = \mathscr{b} \rightarrowtail F \vee G$. Then, the following decomposition is well-defined:

$$\vdash \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{b} \rightarrowtail F, \vdash \mathscr{b} \rightarrowtail G\}.$$

- $L = \mathscr{b} \rightarrowtail F \wedge G, L = \mathscr{b} \rightarrowtail F \wedge G$. Similar to the previous cases.

- $L = \mathscr{b} \rightarrowtail \mathbf{C}^q F$. For every $q \in \mathbb{Q} \cap [0,1]$, $\mu_{\mathscr{C}}(2^{\mathbb{N}}) \geq q$ and $[\![\top]\!] = 2^{\mathbb{N}}$. Then, the following decomposition is well-defined:

$$\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F \quad \rightsquigarrow_0 \quad \{\vdash \top \rightarrowtail F\}.$$

- $L = \mathscr{C} \leftarrowtail \mathbf{C}^q F$. There are two possible cases, basing on the value of $q \in \mathbb{Q} \cap [0,1]$:

  1. Let $q \neq 0$. Then, for any q, $\mu_{\mathscr{C}}(\emptyset) < q$ and, since $[\![\bot]\!] = \emptyset$, the following decomposition is well-defined:

  $$\vdash \mathscr{C} \leftarrowtail \mathbf{C}^q F \quad \rightsquigarrow_0 \quad \{\vdash \bot \leftarrowtail F\}.$$

  2. Let $q = 0$. There are two possible sub-cases:
     (a) Let $\mu([\![\mathscr{C}]\!]) \neq 1$. Then, the following decomposition is well-defined:
     $$\vdash \mathscr{C} \leftarrowtail \mathbf{C}^0 F \quad \rightsquigarrow_0 \quad \{\vdash \bot\}.$$
     (b) Let $\mu([\![\mathscr{C}]\!]) = 1$. Then, the following decomposition is well-defined:
     $$\vdash \mathscr{C} \leftarrowtail \mathbf{C}^0 F \quad \rightsquigarrow_0 \quad \{\}.$$

- $L = \mathscr{C} \rightarrowtail \mathbf{D}^q F, L = \mathscr{C} \leftarrowtail \mathbf{D}^q F$. Similar to the previous cases.

□

Lemma 3.3.6. *A regular sequent is valid if and only if it is derivable in* $\mathbf{LK}_{\mathbf{CPL}_0}$.

Proof. Let $\vdash L$ be a regular sequent, that is let L be basic.

$\Rightarrow$ Assume that $\vdash L$ is valid. Since L is a basic formula, there are two main cases – $Ax1$ and $Ax2$ – which are both trivial.

$\Leftarrow$ Assume that $\vdash L$ is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$. Then, by Proposition 3.3.1, $\vDash L$.

□

Corollary 3.3.1. *If a sequent is* $\rightsquigarrow$*-normal, then it is valid if and only if it is derivable in* $\mathbf{LK}_{\mathbf{CPL}_0}$.

Proof. By putting Lemma 3.3.5 and Lemma 3.3.6 together. □

All the given results can be generalized from single sequents to sets in a natural way, obtaining, in particular, that if a set of sequents is $\rightsquigarrow$-normal – i.e. each of its sequents is $\rightsquigarrow$-normal –, then it is valid when derivable – that is, its sequents are valid if and only if derivable.

Existential Preservation of Validity. It is now possible to prove that validity is *existentially* preserved through $\rightsquigarrow$-decomposition.

Lemma 3.3.7 (Existential Preservation of Validity)**.** *Each valid sequent has a valid* $\rightsquigarrow$*-normal form.*

Proof. Let $\vdash L$ be an arbitrary, valid sequent. It is shown that $\vdash L$ has a valid $\rightsquigarrow$-normal form. The proof is by exhaustive inspection. There are two main cases:

Case 1. Assume that $\vdash L$ is such that no $\rightsquigarrow_0$-reduction can be applied on it. Then, by Lemma 3.3.5, the sequent is either empty or basic; in other words, L is a basic formula. In both cases, the sequent is already $\rightsquigarrow$-normal and, for hypothesis, valid.

Case 2. Assume that $\vdash L$ is $\rightsquigarrow$-reducible. By Lemma 3.3.4, there is no infinite reduction sequence. Thus, it is sufficient to prove that each *reduction step* existentially preserves validity, that is for every possible $\rightsquigarrow$-reduction, based on any possible $\rightsquigarrow_0$-reduction, if $\vDash L$, then there is a $\rightsquigarrow_0$ such that $\vdash L \rightsquigarrow_0 S$ and S is valid. The proof is based on exhaustive inspection of all possible forms of $\rightsquigarrow_0$-reduction.

- $L = \mathscr{b} \rightarrowtail \neg F$. Let us consider a Boolean formula defined as $\mathscr{c} = \neg\mathscr{b}$. Thus, $[\![\neg\mathscr{c}]\!] = 2^{\mathbb{N}} - [\![\mathscr{c}]\!] = 2^{\mathbb{N}} - (2^{\mathbb{N}} - [\![\mathscr{b}]\!]) = [\![\mathscr{b}]\!]$ and so, in particular, $\mathscr{b} \vDash \neg\mathscr{c}$. Let us consider the following well-defined $\rightsquigarrow_0$-decomposition (given $\mathscr{b} \vDash \neg\mathscr{c}$):

  $$\vdash \mathscr{b} \rightarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \rightarrowtail F\}.$$

  For hypothesis $\mathscr{b} \rightarrowtail \neg F$, that is $[\![\mathscr{b}]\!] \subseteq [\![\neg F]\!]$. By basic set theory, also $[\![F]\!] \subseteq [\![\neg\mathscr{b}]\!]$. Since by construction $\mathscr{c} = \neg\mathscr{b}$, $[\![F]\!] \subseteq [\![\mathscr{c}]\!]$ holds, that is $\vDash \mathscr{c} \rightarrowtail F$ as desired. Therefore, there is a $\rightsquigarrow_0$-decomposition which preserves validity.

- $L = \mathscr{b} \rightarrowtail \neg F$. Similar to the case above.

- $L = \mathscr{b} \rightarrowtail F \vee G$. Let us consider two $\mathscr{b}_F$ and $\mathscr{b}_G$, such that $[\![\mathscr{b}_F]\!] = [\![F]\!]$ and $[\![\mathscr{b}_G]\!] = [\![G]\!]$, which exist by Lemma 3.3.2. For hypothesis $\vDash \mathscr{b} \rightarrowtail F \vee G$, and so $[\![\mathscr{b}]\!] \subseteq [\![F]\!] \cup [\![G]\!]$. Thus, by construction, also $[\![\mathscr{b}]\!] \subseteq [\![\mathscr{b}_F]\!] \cup [\![\mathscr{b}_G]\!]$, that is $\mathscr{b} \vDash \mathscr{b}_F \vee \mathscr{b}_G$. Let us consider the following reduction, which is well-defined (given $\mathscr{b} \vDash \mathscr{b}_F \vee \mathscr{b}_G$):

  $$\vdash \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{b}_F \rightarrowtail F, \vdash \mathscr{b}_G \rightarrowtail G\}.$$

  Since $[\![\mathscr{b}_F]\!] = [\![F]\!]$ and $[\![\mathscr{b}_G]\!] = [\![G]\!]$, in particular, also $[\![\mathscr{b}_F]\!] \subseteq [\![F]\!]$ and $[\![\mathscr{b}_G]\!] \subseteq [\![G]\!]$. Therefore, $\vDash \mathscr{b}_F \rightarrowtail F$ and $\vDash \mathscr{b}_G \rightarrowtail G$ and the given $\rightsquigarrow_0$-decomposition is as desired.

- $L = \mathscr{b} \rightarrowtail F \wedge G, L = \mathscr{b} \rightarrowtail F \wedge G$. Proofs are equivalent to the ones for (resp.) $\mathscr{b} \rightarrowtail F \vee G$ and $\mathscr{b} \rightarrowtail F \vee G$.

- $L = \mathscr{b} \rightarrowtail \mathbf{C}^q F$. There are two main sub-cases:

  1. Let $\mu([\![\mathscr{b}]\!]) = 0$. Then, the sequent can be decomposed by means of the following well-defined $\rightsquigarrow_0$-decomposition:

     $$\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F \quad \rightsquigarrow_0 \quad \{\}.$$

     $\{\}$ is vacuously valid. We conclude that the given decomposition is as desired.

2. Let $\mu([\![\mathscr{b}]\!]) \neq 0$. For hypothesis $\vDash \mathscr{b} \rightarrowtail \mathbf{C}^q F$, that is $[\![\mathscr{b}]\!] \subseteq [\![\mathbf{C}^q F]\!]$. Since $[\![\mathscr{b}]\!] \neq \emptyset$ (by Lemma 3.3.3), also $[\![\mathbf{C}^q F]\!] \neq \emptyset$, and so by definition $[\![\mathbf{C}^q F]\!] = 2^{\mathbb{N}}$ and, then, $\mu_{\mathscr{C}}([\![F]\!]) \geq q$. Let us consider a Boolean formula $\mathscr{b}_F$, such that $[\![\mathscr{b}_F]\!] = [\![F]\!]$, which exists due to Lemma 3.3.2. Clearly, $\mu_{\mathscr{C}}([\![\mathscr{b}_F]\!]) \geq q$ and so the following $\rightsquigarrow_0$-decomposition is well-defined:

$$\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F \quad \rightsquigarrow_0 \quad \{\mathscr{b}_F \rightarrowtail F\}.$$

For construction $[\![\mathscr{b}_F]\!] = [\![F]\!]$ so, in particular, $[\![\mathscr{b}_F]\!] \subseteq [\![F]\!]$, that is $\vDash \mathscr{b}_F \rightarrowtail F$ as desired.

- $L = \mathscr{b} \rightarrowtail \mathbf{C}^q F$. There are two main sub-cases:

  1. Let $\mu([\![\mathscr{b}]\!]) = 1$. Then, the sequent can be decomposed by means of the following well-defined $\rightsquigarrow_0$-decomposition:

  $$\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F \quad \rightsquigarrow_0 \quad \{\}.$$

  $\{\}$ is vacuously valid, so the given decomposition is as desired.

  2. Let $\mu([\![\mathscr{b}]\!]) \neq 1$. For hypothesis $\vDash \mathscr{b} \rightarrowtail \mathbf{C}^q F$, that is $[\![\mathbf{C}^q F]\!] \subseteq [\![\mathscr{b}]\!]$. Since $[\![\mathscr{b}]\!] \neq 2^{\mathbb{N}}$, $[\![\mathbf{C}^q F]\!] = \emptyset$, and by Definition 3.2.2, $\mu_{\mathscr{C}}([\![F]\!]) < q$. Let us consider a Boolean formula $\mathscr{b}_F$ such that $[\![\mathscr{b}_F]\!] = [\![F]\!]$, which can be constructed due to Lemma 3.3.2. Thus, $\mu([\![\mathscr{b}_F]\!]) < q$ and the following decomposition is well-defined:

  $$\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F \quad \rightsquigarrow_0 \quad \{\mathscr{b}_F \rightarrowtail F\}.$$

  For construction $[\![\mathscr{b}_F]\!] = [\![F]\!]$ so, in particular, $[\![F]\!] \subseteq [\![\mathscr{b}_F]\!]$, that is $\vDash \mathscr{b}_F \rightarrowtail F$ as desired.

- $L = \mathscr{b} \rightarrowtail \mathbf{D}^q F, L = \mathscr{b} \rightarrowtail \mathbf{D}^q F$. Proofs are equivalent to the ones for $\mathscr{b} \rightarrowtail \mathbf{C}^q F$ and $\mathscr{b} \rightarrowtail \mathbf{C}^q F$, respectively.

- $L = \mathscr{b} \rightarrowtail F$ and $\mu([\![\mathscr{b}]\!]) = 0$. Then, the following decomposition is well-defined:

  $$\vdash \mathscr{b} \rightarrowtail F \quad \rightsquigarrow_0 \quad \{\}.$$

  $\{\}$ is vacuously valid. We conclude that the given decomposition is as desired.

- $L = \mathscr{b} \rightarrowtail F$ and $\mu([\![\mathscr{b}]\!]) = 1$. Then, the following decomposition is well-defined:

  $$\vdash \mathscr{b} \rightarrowtail F \quad \rightsquigarrow_0 \quad \{\}.$$

  $\{\}$ is vacuously valid so the given decomposition is as desired.

There is no other possible case.[10] □

[10] Notice that if $L = \mathbf{D}^0 F$, for hypothesis $\vDash \mathscr{b} \rightarrowtail \mathbf{D}^0 F$, that is $[\![\mathscr{b}]\!] \subseteq [\![\mathbf{D}^0 F]\!]$. Clearly $[\![\mathbf{D}^0 F]\!] = \emptyset$, so $[\![\mathscr{b}]\!] = \emptyset$ and $\mu([\![\mathscr{b}]\!]) = 0$. Equally, when $L = \mathscr{b} \rightarrowtail \mathbf{C}^0 F$, $\mu([\![\mathscr{b}]\!]) = 1$.

Derivability Preservation. As anticipated, properties and proofs concerning sequents can be easily generalized to corresponding proofs about sets of sequents. Specifically, by considering Lemma 3.3.5 and Lemma 3.3.6 together, a result in the style of Corollary 3.3.1 – stating that a $\rightsquigarrow$-normal set of sequents is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ if and only if it is valid – can easily be proved.

Definition 3.3.15 (Derivable Set)**.** A set of sequents is *derivable* if and only if all of its sequents are derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$.

Proposition 3.3.2 (Decomposition Reflects Derivability)**.** *Given two sets of sequents S and T, if $S \rightsquigarrow T$ and T is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$, then S is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as well.*

Proof Sketch. Assume $S \rightsquigarrow T$. Then, there is a sequent $\vdash L \in S$ such that it is the "active" sequent on which $\rightsquigarrow$ is based, that is $\rightsquigarrow$ is based on the $\rightsquigarrow_0$-decomposition below:

$$\vdash L \quad \rightsquigarrow_0 \quad \{\vdash L_1, \ldots, \vdash L_n\}.$$

The proof is by straightforward inspection of all possible forms of $\rightsquigarrow_0$-reduction.

- $L = \mathscr{b} \rightarrowtail \neg F$ and the given $\rightsquigarrow_0$-decomposition be:

  $$\vdash \mathscr{b} \rightarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \leftarrowtail F\},$$

  where $\mathscr{b} \vDash \neg\mathscr{c}$. Since $\vdash \mathscr{c} \leftarrowtail F \in T$, for hypothesis $\vdash \mathscr{c} \leftarrowtail F$ is derivable by a derivation, call it Π. Therefore, $\vdash \mathscr{b} \rightarrowtail \neg F$ is shown derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

  $$\dfrac{\begin{array}{c}\Pi \\ \vdash \mathscr{c} \leftarrowtail F\end{array} \qquad \mathscr{b} \vDash \neg\mathscr{c}}{\vdash \mathscr{b} \rightarrowtail \neg F} R^{\rightarrowtail}_{\neg}$$

- $L = \mathscr{b} \leftarrowtail \neg F$ and the given $\rightsquigarrow_0$-decomposition be:

  $$\vdash \mathscr{b} \leftarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \rightarrowtail F\},$$

  where $\neg\mathscr{c} \vDash \mathscr{b}$. Since $\mathscr{c} \rightarrowtail F \in T$, for hypothesis $\vdash \mathscr{c} \rightarrowtail F$ is derivable by a derivation, call it Π. Therefore, $\vdash \mathscr{b} \leftarrowtail \neg F$ is shown derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

  $$\dfrac{\begin{array}{c}\Pi \\ \vdash \mathscr{c} \rightarrowtail F\end{array} \qquad \neg\mathscr{c} \vDash \mathscr{b}}{\mathscr{b} \leftarrowtail \neg F} R^{\leftarrowtail}_{\neg}$$

- $L = \mathscr{b} \rightarrowtail F \vee G$ and the given $\rightsquigarrow_0$-decomposition be:

  $$\vdash \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \rightarrowtail F, \vdash \mathscr{d} \rightarrowtail G\},$$

  where $\mathscr{b} \vDash \mathscr{c} \vee \mathscr{d}$. Since $\vdash \mathscr{c} \rightarrowtail F, \vdash \mathscr{d} \rightarrowtail G \in T$, for hypothesis they are both derivable by two derivations, call them (resp.) Π and Π'. Then, $\mathscr{b} \rightarrowtail F \vee G$ is shown derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\dfrac{\dfrac{\dfrac{\Pi}{\vdash \mathscr{c} \rightarrowtail F}}{\vdash \mathscr{c} \rightarrowtail F \vee G}\, R1^{\rightarrowtail}_{\vee} \qquad \dfrac{\dfrac{\Pi'}{\vdash \mathscr{d} \rightarrowtail G}}{\vdash \mathscr{d} \rightarrowtail F \vee G}\, R2^{\rightarrowtail}_{\vee} \qquad \mathscr{b} \vDash \mathscr{c} \vee \mathscr{d}}{\vdash \mathscr{b} \rightarrowtail F \vee G}\, R^{\rightarrowtail}_{\cup}$$

- $L = \mathscr{b} \rightarrowtail F \vee G$ and the given $\rightsquigarrow_0$-decomposition be:

$$\vdash \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{b} \rightarrowtail F, \vdash \mathscr{b} \rightarrowtail G\}$$

Since $\vdash \mathscr{b} \rightarrowtail F, \vdash \mathscr{b} \rightarrowtail G \in T$, for hypothesis they are both derivable by two derivations, call them (resp.) Π and Π'. Then, $\mathscr{b} \rightarrowtail F \vee G$ is shown derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\dfrac{\dfrac{\Pi}{\vdash \mathscr{b} \rightarrowtail F} \qquad \dfrac{\Pi'}{\vdash \mathscr{b} \rightarrowtail G}}{\vdash \mathscr{b} \rightarrowtail F \vee G}\, R^{\rightarrowtail}_{\vee}$$

- $L = \mathscr{b} \rightarrowtail F \wedge G$ and the given $\rightsquigarrow_0$-decomposition be:

$$\vdash \mathscr{b} \rightarrowtail F \wedge G \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{b} \rightarrowtail F, \vdash \mathscr{b} \rightarrowtail G\}.$$

Since $\vdash \mathscr{b} \rightarrowtail F, \vdash \mathscr{b} \rightarrowtail G \in T$, for hypothesis they are both derivable by two derivations, call them (resp.) Π and Π'. Then, $\mathscr{b} \rightarrowtail F \wedge G$ is shown derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\dfrac{\dfrac{\Pi}{\vdash \mathscr{b} \rightarrowtail F} \qquad \dfrac{\Pi'}{\vdash \mathscr{b} \rightarrowtail G}}{\vdash \mathscr{b} \rightarrowtail F \wedge G}\, R^{\rightarrowtail}_{\wedge}$$

- $L = \mathscr{b} \rightarrowtail F \wedge G$ and the given $\rightsquigarrow_0$-decomposition be:

$$\vdash \mathscr{b} \rightarrowtail F \wedge G \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \rightarrowtail F, \vdash \mathscr{d} \rightarrowtail G\},$$

where $\mathscr{c} \wedge \mathscr{d} \vDash \mathscr{b}$. Since $\vdash \mathscr{c} \rightarrowtail F, \vdash \mathscr{d} \rightarrowtail G \in T$, for hypothesis they are both derivable by two derivations, call them (resp.) Π and Π'. then, $\mathscr{b} \rightarrowtail F \wedge G$ is shown derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\dfrac{\dfrac{\dfrac{\Pi}{\vdash \mathscr{c} \rightarrowtail F}}{\vdash \mathscr{c} \rightarrowtail F \wedge G}\, R1^{\rightarrowtail}_{\wedge} \qquad \dfrac{\dfrac{\Pi'}{\vdash \mathscr{d} \rightarrowtail G}}{\vdash \mathscr{d} \rightarrowtail F \wedge G}\, R2^{\rightarrowtail}_{\wedge} \qquad \mathscr{c} \wedge \mathscr{d} \vDash \mathscr{b}}{\vdash \mathscr{b} \rightarrowtail F \wedge G}\, R^{\rightarrowtail}_{\cap}$$

- $L = \mathscr{b} \rightarrowtail \mathbf{C}^q F$ and the given $\rightsquigarrow_0$-decomposition be:

$$\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \rightarrowtail F\},$$

where $\mu([\![\mathscr{c}]\!]) \geq q$. Since $\vdash \mathscr{c} \rightarrowtail F \in T$, for hypothesis it is derivable by a derivation, call it Π. Then $\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F$ is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\frac{\begin{array}{c}\Pi\\ \vdash \mathscr{c} \rightarrowtail F\end{array} \qquad \mu([\![\mathscr{c}]\!]) \geq q}{\vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F} R^{\rightarrowtail}_{\mathbf{C}}$$

- $L = \mathscr{b} \leftarrowtail \mathbf{C}^q F$ and the given $\rightsquigarrow_0$-decomposition be:

$$\vdash \mathscr{b} \leftarrowtail \mathbf{C}^q F \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \leftarrowtail F\},$$

where $\mu([\![\mathscr{c}]\!]) < q$. Since $\vdash \mathscr{c} \leftarrowtail F \in T$, for hypothesis it is derivable by a derivation, call it Π. Then, $\vdash \mathscr{b} \leftarrowtail \mathbf{C}^q F$ is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\frac{\begin{array}{c}\Pi\\ \vdash \mathscr{c} \leftarrowtail F\end{array} \qquad \mu([\![\mathscr{c}]\!]) < q}{\vdash \mathscr{b} \leftarrowtail \mathbf{C}^q F} R^{\leftarrowtail}_{\mathbf{C}}$$

- $L = \mathscr{b} \rightarrowtail \mathbf{D}^q F$ and the given $\rightsquigarrow_0$-decomposition be:

$$\vdash \mathscr{b} \rightarrowtail \mathbf{D}^q F \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \leftarrowtail F\}$$

where $\mu([\![\mathscr{c}]\!]) < q$. Since $\vdash \mathscr{c} \leftarrowtail F \in T$, for hypothesis it is derivable by a derivation, call it Π. Then, $\vdash \mathscr{b} \rightarrowtail \mathbf{D}^q F$ is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\frac{\begin{array}{c}\Pi\\ \mathscr{c} \leftarrowtail F\end{array} \qquad \mu([\![\mathscr{c}]\!]) < q}{\vdash \mathscr{b} \rightarrowtail \mathbf{D}^q F} R^{\rightarrowtail}\mathbf{D}$$

- $L = \mathscr{b} \leftarrowtail \mathbf{D}^q F$ and the given $\rightsquigarrow_0$-decomposition be:

$$\vdash \mathscr{b} \leftarrowtail \mathbf{D}^q F \quad \rightsquigarrow_0 \quad \{\vdash \mathscr{c} \rightarrowtail F\},$$

where $\mu([\![\mathscr{c}]\!]) \geq q$. Since $\vdash \mathscr{c} \rightarrowtail F \in T$, for hypothesis it is derivable by a derivation, call it Π. Then, $\vdash \mathscr{b} \leftarrowtail \mathbf{D}^q F$ is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\frac{\begin{array}{c}\Pi\\ \vdash \mathscr{c} \rightarrowtail F\end{array} \qquad \mu([\![\mathscr{c}]\!]) \geq q}{\vdash \mathscr{b} \leftarrowtail \mathbf{D}^q F} R^{\leftarrowtail}_{\mathbf{D}}$$

- $L = \mathscr{b} \rightarrowtail F$ and the given $\rightsquigarrow_0$-decomposition be:

$$\vdash \mathscr{b} \rightarrowtail F \quad \rightsquigarrow_0 \quad \{\}$$

where $\mu([\![\mathscr{b}]\!]) = 0$. Then, $\vdash \mathscr{b} \rightarrowtail F$ is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\frac{\mu([\![\mathscr{b}]\!]) = 0}{\vdash \mathscr{b} \rightarrowtail F} R^{\rightarrowtail}_{\mu}$$

- $L = \mathscr{b} \leftarrowtail F$ and the given $\rightsquigarrow_0$-decomposition be:

$$\vdash \mathscr{b} \leftarrowtail F \quad \rightsquigarrow_0 \quad \{\}$$

where $\mu([\![\mathscr{b}]\!]) = 1$. Then, $\vdash \mathscr{b} \leftarrowtail F$ is derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$ as follows:

$$\frac{\mu([\![\mathscr{b}]\!]) = 0}{\vdash \mathscr{b} \leftarrowtail F} R^{\leftarrowtail}_{\mu}$$

□

Concluding the Proof. Putting these results together, we conclude that $\mathbf{LK}_{\mathbf{CPL}_0}$ is complete with respect to the semantics of $\mathbf{CPL}_0$.

Proposition 3.3.3 (Completeness)**.** *If* $\vDash L$ *holds, then* $\vdash L$ *is derivable in* $\mathbf{LK}_{\mathbf{CPL}_0}$.

Proof. If the sequent is valid, by Lemma 3.3.4, it has a *valid* $\rightsquigarrow$-normal form. By Corollary 3.3.1, a $\rightsquigarrow$-normal form is valid if and only if it is derivable, so the given $\rightsquigarrow$-normal form must be derivable as well. Therefore, by Proposition 3.3.2, the given (valid) sequent must be derivable in $\mathbf{LK}_{\mathbf{CPL}_0}$. □

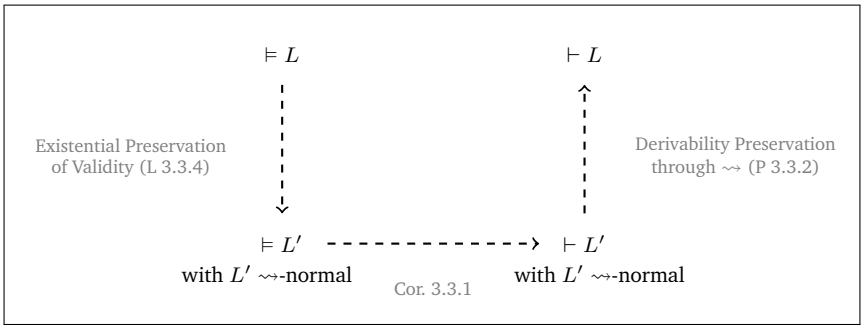


Figure 3.5: Proof Schema

Remark 3.3.1. *A remarkable consequence of the completeness theorem is that the following cut-rule turns out to be derivable in* $\mathbf{LK}_{\mathbf{CPL}_0}$*:*

$$\frac{\vdash \mathscr{c} \rightarrowtail \neg F \vee G \qquad \vdash \mathscr{d} \rightarrowtail F \qquad \mathscr{b} \vDash \mathscr{c} \wedge \mathscr{d}}{\vdash \mathscr{b} \rightarrowtail G}\, Cut$$

3.4 A Digression on the Expressive Power of $\mathbf{CPL}_0$

As said, our counting logic is strongly related to probabilistic reasoning and stochastic events. Indeed, it can be seen as offering a natural model to logically represent events the probability of which is associated with dyadic rationals.[11] We start by introducing auxiliary quantifiers, to express exact probability in a compact way (Section 3.4.1). Then, we prove that counting formulae can actually enucleate (quantitative properties of) events the probability of which is a dyadic rational (Section 3.4.2).

[11] In perspective, this result would be interesting, on the one hand, to relate (in a formal way) our logic $\mathbf{CPL}_0$ with Bernoulli experiments and, more in general, with experiments associated with dyadic distributions, on the other – given the limits of the univariate counting fragment – to extend this analysis to more expressive languages, as the ones presented in Chapter 4 and 12.

3.4.1 Expressing Exact Probability

In the language of $\mathbf{CPL}_0$, we can even express that the probability for a formula to be true is *precisely* a given one. For the sake of readability, we introduce auxiliary quantifiers, $\mathbb{C}^q$ and $\mathbb{D}^q$, intuitively meaning that the argument formula is true with probability, respectively *strictly greater* and *smaller or equal* than the index $q \in \mathbb{Q} \cap [0,1]$.

Notation 3.4.1 (Auxiliary Quantifiers)**.** The so-called *white counting quantifiers* are interpreted as follows:

$$[\![\mathbb{C}^q F]\!] := \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}([\![F]\!]) > q \\ \emptyset & \text{otherwise} \end{cases} \qquad [\![\mathbb{D}^q F]\!] := \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}([\![F]\!]) \leq q \\ \emptyset & \text{otherwise.} \end{cases}$$

Clearly, these quantifiers do not extend the expressive power of our language, as they are easily definable in terms of primitive $\mathbf{C}^q$ and $\mathbf{D}^q$.

Lemma 3.4.1. *For every formula F of $\mathbf{CPL}_0$ and $q \in \mathbb{Q} \cap [0,1]$,*

$$\mu_{\mathscr{C}}([\![F]\!]) \rhd q \quad \textit{iff} \quad \mu_{\mathscr{C}}([\![\neg F]\!]) \lhd 1 - q,$$

with $\rhd, \lhd \in \{(\geq, \leq), (\leq, \geq), (>, <), (<, >)\}$.

Proof. Let us consider the case $\leq, \geq$. Since

$$\mu_{\mathscr{C}}([\![\neg F]\!]) = \mu_{\mathscr{C}}(2^{\mathbb{N}} - [\![F]\!]) = 1 - \mu_{\mathscr{C}}([\![F]\!]),$$

trivially:

$$\begin{aligned} \mu_{\mathscr{C}}([\![F]\!]) \geq q \quad & \text{iff} \quad 1 - \mu_{\mathscr{C}}([\![F]\!]) \leq 1 - q \\ & \text{iff} \quad \mu_{\mathscr{C}}([\![\neg F]\!]) \leq 1 - q. \end{aligned}$$

All the other cases are proved in a similar way. □

Lemma 3.4.2. *For every formula F of $\mathbf{CPL}_0$ and $q \in \mathbb{Q} \cap [0,1]$,*

$$\begin{aligned} \mathbf{C}^q \neg F &\equiv \mathbb{D}^{1-q} F & \qquad \mathbf{D}^q \neg F &\equiv \mathbb{C}^{1-q} F \\ \mathbf{C}^q \neg F &\equiv \neg \mathbb{C}^{1-q} F & \qquad \mathbf{D}^q \neg F &\equiv \neg \mathbb{D}^{1-q} F. \end{aligned}$$

Proof. The proof is based on Definition 3.2.2 and Lemma 3.4.1 above:

$$\begin{aligned} [\![\mathbf{C}^q \neg F]\!] &= \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}([\![\neg F]\!]) \geq q \\ \emptyset & \text{otherwise} \end{cases} \\ &= \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}([\![F]\!]) \leq 1 - q \\ \emptyset & \text{otherwise} \end{cases} \\ &= [\![\mathbb{D}^{1-q} F]\!] \end{aligned} \qquad \begin{aligned} [\![\mathbf{D}^q \neg F]\!] &= \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}([\![\neg F]\!]) < q \\ \emptyset & \text{otherwise} \end{cases} \\ &= \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}([\![F]\!]) > 1 - q \\ \emptyset & \text{otherwise} \end{cases} \\ &= [\![\mathbb{C}^{1-q} F]\!] \end{aligned}$$

$$\begin{aligned} [\![\mathbf{C}^q \neg F]\!] &= [\![\neg \mathbf{D}^q \neg F]\!] \\ &= [\![\neg \mathbb{C}^{1-q} F]\!] \end{aligned} \qquad \begin{aligned} [\![\mathbf{D}^q \neg F]\!] &= [\![\neg \mathbf{C}^q \neg F]\!] \\ &= [\![\neg \mathbb{D}^{1-q} F]\!]. \end{aligned}$$

□

So, due to these quantifiers, we can express exact probability *in a compact way*.

Example 3.4.1. For example, we formalize that $F = \mathbf{1} \wedge \mathbf{2}$ is true with probability $\frac{1}{4}$ as follows:

$$F_{ex} : \mathbf{C}^{1/4}(\mathbf{1} \wedge \mathbf{2}) \wedge \mathbb{D}^{1/4}(\mathbf{1} \wedge \mathbf{2}).$$

We can even extend $\mathbf{LK}_{\mathbf{CPL}_0}$ with the (derivable) rules for $\mathbb{C}$ and $\mathbb{D}$ illustrated in Figure 3.6.

$$\frac{\vdash \mathscr{c} \rightarrowtail F \qquad \mu([\![c]\!]) > q}{\vdash \mathscr{b} \rightarrowtail \mathbb{C}^q F} R^{\rightarrowtail}_{\mathbb{C}} \qquad \frac{\vdash \mathscr{c} \rightarrowtail F \qquad \mu([\![c]\!]) \leq q}{\vdash \mathscr{b} \rightarrowtail \mathbb{D}^q F} R^{\rightarrowtail}_{\mathbb{D}}$$

$$\frac{\vdash \mathscr{c} \rightarrowtail F \qquad \mu([\![c]\!]) \leq q}{\vdash \mathscr{b} \rightarrowtail \mathbb{C}^q F} R^{\rightarrowtail}_{\mathbb{C}} \qquad \frac{\vdash \mathscr{c} \rightarrowtail F \qquad \mu([\![c]\!]) > q}{\vdash \mathscr{b} \rightarrowtail \mathbb{D}^q F} R^{\rightarrowtail}_{\mathbb{D}}$$

Figure 3.6: Rules for $\mathbb{C}$ and $\mathbb{D}$

3.4.2 On Formulae of $\mathbf{CPL}_0$ and Dyadic Rationals

As anticipated, it is natural to interpret atomic formulae of $\mathbf{CPL}_0$ as infinite sequences of independently and identically distributed (i.i.d. for short) random bits – i.e., more concretely, as infinite sequences of independent *fair* coin tosses. Coherently, we can see counting formulae as formalizing experiments associated with probabilities *of a very special kind*. In particular, we will show that any counting formula F is such that $\mu_{\mathscr{C}}([\![F]\!])$ is a dyadic rational.[12] Intuitively, this is the first step to "simulate" any event associated with these measures. For instance, the fact that, when tossing an unbiased coin twice, the probability that it returns HEAD both times is $\frac{1}{4}$ can be expressed in our logic by the valid formula F_{ex} of Example 3.4.1. Let us consider another very intuitive example.

Example 3.4.2. Let a *biased* coin return HEAD only 25% of the time. In this case a single toss cannot be formalized by an atomic formula of $\mathbf{CPL}_0$. Yet, as done above, it can be easily expressed using a molecular formula, namely one in the form $(\mathbf{i} \wedge \mathbf{j})$, with $i, j \in \mathbb{N}$ "fresh". Consequently, also properties concerning complex events can be "captured" via $\mathbf{CPL}_0$. For instance, that the probability for at least one of two subsequent biased tosses to return HEAD is greater than $\frac{1}{3}$ is formalized by the (valid) formula:

$$F_{bias} : \mathbf{C}^{1/3}\big((\mathbf{1} \wedge \mathbf{2}) \vee (\mathbf{3} \wedge \mathbf{4})\big).$$

More in general, basing on Definition 3.1.1 (and Corollary 3.1.1), we prove that the measure of formulae of $\mathbf{CPL}_0$ is always a dyadic rational, that is a

[12] For further details on the converse, namely, that for any dyadic rational q there is a formula $F \in \mathbf{CPL}_0$ such that $\mu_{\mathscr{C}}([\![F]\!]) = q$, see [4].

number $q \in \mathbb{Q}$ such that q can be expressed as $\frac{k}{2^m}$, where $k \in \mathbb{Z}$.[13] We start by showing that any counting formula can be interpreted as a cylinder of the proper rank. To do so, we pass through the following auxiliary lemma.

Lemma 3.4.3. *For any cylinder cyl_H of rank k_1 and $k' > k$, there is a cylinder $\text{cyl}_{H'}$ of rank k' such that*

$$\text{cyl}_H = \text{cyl}_{H'}.$$

Proof. Let us consider H' consisting of all sequences $(u_1, \ldots, u_{k'}) \in 2^{k'}$ such that $(u_1, \ldots, u_k) \in H$. Clearly, $\text{cyl}_{H'}$ is an alternative, but equivalent representation for cyl_H, of rank k'. □

Lemma 3.4.4. *For any formula of **CPL**$_0$ F, there is a cylinder of rank k, cyl_H, such that $[\![F]\!] = \text{cyl}_H$.*

Proof. The proof is by induction on the structure of F:

- $F = \mathbf{i}$ for some $i \in \mathbb{N}$. Then, $[\![\mathbf{i}]\!] = Cyl(i)$, which is a thin cylinder.
- $F = \neg G$. By IH, there is a $k \in \mathbb{N}$ and a cylinder of rank k, $\text{cyl}_{K'}$, such that $[\![G]\!] = \text{cyl}_{K'}$. Let $K = 2^k - K'$. Then,

  $$[\![\neg G]\!] = 2^{\mathbb{N}} - [\![G]\!] = 2^{\mathbb{N}} - \text{cyl}_{K'} = \text{cyl}_K$$

  is a cylinder of rank k as well.
- $F = G_1 \wedge G_2$. By IH, there exist $k_1, k_2 \in \mathbb{N}$ and cylinders of rank k_1, k_2, respectively, such that $[\![G_1]\!] = \text{cyl}_{H_1}$ and $[\![G_2]\!] = \text{cyl}_{H_2}$. Then, if $k_1 = k_2$,

  $$[\![F]\!] = [\![F_1 \wedge F_2]\!] = [\![G_1]\!] \cap [\![G_2]\!] = \text{cyl}_{H_1} \cap \text{cyl}_{H_2} = \text{cyl}_{H_1 \cap H_2},$$

  which is a cylinder of rank k_1 as well. Otherwise, assume $k_1 > k_2$ (the case $k_2 > k_1$ is equivalent). Let $\text{cyl}_{H_2'}$ be defined as in Lemma 3.4.3, so to be equivalent to cyl_{H_2} but of rank k_1 – that is H_2' consists of the sequences $(u_1, \ldots, u_{k_1}) \in 2^{k_1}$ such that the truncated sequence $(u_1, \ldots, u_{k_2})$ is in H_2. So,

  $$[\![F]\!] = [\![F_1 \wedge F_2]\!] = [\![G_1]\!] \cap [\![G_2]\!] = \text{cyl}_{H_1} \cap \text{cyl}'_{H_2} = \text{cyl}_{H_1 \cap H_2'},$$

  which is a cylinder of rank k_1.
- $F = G_1 \wedge G_2$ is similar to the case above.
- $F = \mathbf{C}^q G$. Then, by Definition 3.2.2, either $[\![F]\!] = 2^{\mathbb{N}}$ or $[\![F]\!] = \emptyset$, which are both cylinders of rank k (in particular, in the former case $k = 0$).
- $F = \mathbf{D}^q G$. Equivalent to the case above.

□

[13] Actually, in our case $k \in \mathbb{N}$.

Then, since by Corollary 3.1.1, for any k the measure of a cylinder of rank k is a dyadic rational, we conclude that for each formula of $\mathbf{CPL}_0$ its measure is dyadic as well.[14]

Lemma 3.4.5. *For any formula of $\mathbf{CPL}_0$ F, there are $n, m \in \mathbb{N}$ such that $\mu_{\mathscr{C}}([\![F]\!]) = \frac{m}{2^n}$.*

Proof. By putting Corollary 3.1.1 and Lemma 3.4.4 together. □

Remarkably, as a consequence of this Lemma, we obtain a stronger, negative result, namely that formulae of $\mathbf{CPL}_0$ cannot formalize events with probability different from $\frac{n}{2^m}$, for any $n, m \in \mathbb{N}$ (Lemma 3.4.5).[15]

[14] A "syntactic proof" of this result can also be obtained as a corollary of the procedure presented in [5].

[15] Nevertheless, we can somehow "simulate" these events *in an approximate way*. For further details, see [5, 4]. This fact also leads to natural questions of the "measure expressivity" of more general languages, as **CPL** or **MQPA**, which have been left for future study.

Chapter 4

On Multivariate Counting Propositional Logic

We introduce a more expressive counting logic, called **CPL**, in which relations between valuations of different groups of variables can be taken into account. Its language is made of *named* atomic formulae and counting quantifiers. Contextually, the corresponding quantitative semantics is subtler than the one for $\mathbf{CPL}_0$ and, in particular, the interpretation for counting-quantified formulae relies on some technical notions, presented in Section 4.1. As univariate $\mathbf{CPL}_0$, also **CPL** supports a satisfactory proof theoretical treatment. Indeed, in Section 4.2, we introduce a sound and complete calculus, which is nothing but a straightforward generalization of $\mathbf{LK}_{\mathbf{CPL}_0}$.

4.1 Syntax and Semantics of CPL

Preliminaries. Let us briefly recall useful facts about the Borel σ-algebra of the Cantor space. We will consider a countably infinite set $\mathscr{A}$ of *names* noted $a, b, c, \ldots$. For any finite subset $X \subseteq \mathscr{A}$, we let $\mathscr{B}(2^{\mathbb{N}})^X$ denote the *Borel σ-algebra* on the X-th product of the Cantor space $(2^{\mathbb{N}})^X$, that is the smallest σ-algebra containing all open sets under the product topology. For any name $a \in \mathscr{A}$ and index $i \in \mathbb{N}$, the elements of the cantor space whose value over a and i is $\mathbb{1}$ form a Borel set called a cylinder.[1] More precisely, the cylinder induced by a and i is defined as:

$$Cyl(a, i) = \{f \in (2^{\mathbb{N}})^{\mathscr{N}} \mid f(a)(i) = 1\}.$$

Importantly, there exist a *unique* measure $\mu_{\mathscr{C}}$ of $\mathscr{B}((2^{\mathbb{N}})^{\mathscr{A}})$ such that $\mu_{\mathscr{C}}(Cyl(a, i)) = \frac{1}{2}$ holds for all cylinders. For any Borel set $S \in \mathscr{B}((2^{\mathbb{N}})^{X \cup Y})$ and $f \in (2^{\mathbb{N}})^X$, the *projection of S over f* is the set $\Pi_f(S) \subseteq (2^{\mathbb{N}})^Y$ defined as:

[1] For further details, see Section 3.1 or [25].

$$\Pi_f(S) = \{g \in (2^{\mathbb{N}})^Y \mid f + g \in S\},$$

where

$$(f+g)(\alpha) = \begin{cases} f(\alpha) & \text{if } \alpha \in X \\ g(\alpha) & \text{if } \alpha \in Y. \end{cases}$$

Yet, since it is an *analytic* set [124], one can show that its Lebesgue measure is always well-defined. Moreover, the following result holds.

Theorem 4.1.1 (Theorem 14.11 and Theorem 29.26 [124])**.** *For any $S \in \mathscr{B}(2^{X \cup Y})$, with $X \cap Y = \emptyset$ and $r \in [0,1]$, $\{\omega \in (2^{\mathbb{N}})^X \mid \mu(\Pi_f(S)) \geq r\} \in \mathscr{B}(2^X)$.*

Syntax. The language of **CPL** is made of *named* propositional atoms and counting quantifiers.

Notation 4.1.1. In what follows, we use $a, b, c, ... \in \mathscr{A}$ for names and $X, Y, ... \subseteq \mathscr{A}$ for (countable) sets of names.

Counting quantifiers – indicated as $\mathbf{C}_a^q$ or $\mathbf{D}_a^q$ – now depend on the number of valuations of propositional atoms *with the corresponding name* – here, a – satisfying the argument formula F.

Definition 4.1.1 (Formulae of **CPL**)**.** *Formulae of **CPL*** are defined by the grammar below:

$$F ::= \mathbf{i}_a \mid \neg F \mid F \wedge F \mid F \vee F \mid \mathbf{C}_a^q F \mid \mathbf{D}_a^q F,$$

where $i \in \mathbb{N}$, $a \in \mathscr{A}$, and $q \in \mathbb{Q} \cap [0,1]$.

Intuitively, a named quantifier binds the occurrences of the name in the argument formula and counts models *relative* to the corresponding bounded variable.

Notation 4.1.2. Given a formula of **CPL** F, let $\mathsf{FN}(F)$ indicate the set of names occurring *free*, i.e. not bound, in F.

These names can be used to distinguish between different groups of propositional variables.

Example 4.1.1. For example, let us consider the propositional formula $F_{\mathbf{PL}} = (x_1 \vee y_1) \wedge (x_2 \vee y_2)$, which contains two groups of variables $\mathbf{x} = \{x_1, x_2\}$ and $\mathbf{y} = \{y_1, y_2\}$. In **CPL**, we can deal with different groups of variables using distinct names – for instance, a and b – and construct the counting formula

$$F_{\mathbf{CPL}} : (\mathbf{1}_a \vee \mathbf{1}_b) \wedge (\mathbf{2}_a \vee \mathbf{2}_b).$$

As we shall see in details in Chapter 5, since the intuitive meaning of $\mathbf{C}_a^q F$ is that F is true in at least q valuations of the variables with name a, counting-quantified formulae allow us to express canonical counting problems. For example,

$$\mathbf{C}_a^{1/2} \mathbf{C}_b^{1/2} F_{\mathbf{CPL}}$$

expresses the `MAJMAJSAT` problem for $F_{\mathbf{CPL}}$ (which happens to have a positive answer, in this case).

Semantics. As seen, formulae of $\mathbf{CPL}_0$ have a rather intuitive meaning. On the contrary, the semantics of **CPL** is subtler. In particular, the interpretation of a formula F now depends on the choice of a finite set of names $X \supseteq \mathsf{FN}(F)$ and is a measurable set $[\![F]\!]_X$ belonging to the Borel algebra, $\mathscr{B}\big((2^{\mathbb{N}})^X\big)$. To define it formally we need to introduce the following technical notion.

Definition 4.1.2 (f-projection)**.** Let X, Y be two disjoint, finite sets of names and $f \in (2^{\mathbb{N}})^X$. For all $\mathcal{X} \subseteq (2^{\mathbb{N}})^{X \cup Y}$, the *$f$-projection of $\mathcal{X}$* is the set:

$$\Pi_f(\mathcal{X}) := \{g \in (2^{\mathbb{N}})^Y \mid f + g \in \mathcal{X}\} \subseteq (2^{\mathbb{N}})^Y,$$

where

$$(f+g)(\alpha) := \begin{cases} f(\alpha) & \text{if } \alpha \in X \\ g(\alpha) & \text{if } \alpha \in Y. \end{cases}$$

More concretely, suppose that X and Y are two disjoint sets of names, with $\mathsf{FN}(F) \subseteq X \cup Y$. Then, if we fix a valuation $f \in (2^{\mathbb{N}})^X$ for the variables of F with names in X, the set $\Pi_f\big([\![F]\!]_{X \cup Y}\big)$ describes the set of valuations of the variables of F with names in Y, extending f.

Due to this notions we can formally define the interpretation for formulae of **CPL** as below.

Definition 4.1.3 (Semantics of **CPL**)**.** For each formula F of **CPL** and finite set of names such that $X \supseteq \mathsf{FN}(F)$, its *interpretation* $[\![F]\!]_X \subseteq (2^{\mathbb{N}})^X$ is as follows:

$$\begin{aligned}
[\![\mathbf{i}_a]\!]_X &:= \{f \mid f(a)(i) = 1\} & [\![\neg G]\!]_X &:= (2^{\mathbb{N}})^X - [\![G]\!]_X \\
[\![G \wedge H]\!]_X &:= [\![G]\!]_X \cap [\![H]\!]_X & [\![\mathbf{C}^q_a G]\!]_X &:= \{f \mid \mu_{\mathscr{C}}(\Pi_f\big([\![G]\!]_{X \cup \{a\}}\big)) \geq q\} \\
[\![G \vee H]\!]_X &:= [\![G]\!]_X \cup [\![H]\!]_X & [\![\mathbf{D}^q_a G]\!]_X &:= \{f \mid \mu_{\mathscr{C}}(\Pi_f\big([\![G]\!]_{X \cup \{a\}}\big)) < q\}.
\end{aligned}$$

That all sets $[\![F]\!]_X$ are measurable – that is $[\![F]\!]_X \in \mathscr{B}((2^{\mathbb{N}}))$ – crucially relies on some properties of f-projections, as it is proved in details in Section 4.2.1. Observe that, generally speaking, it is not always true that the projection of a measurable set is measurable. Yet, this actually holds in all the cases we are interested in.

To have a grasp of the semantics of named quantifiers, let us consider the following example:

Example 4.1.1. Let F be the formula of **CPL**:

$$F : \big(\mathbf{2}_a \wedge (\neg \mathbf{2}_b \wedge \mathbf{3}_b)\big) \vee \big(\neg \mathbf{2}_a \wedge (\mathbf{2}_b \wedge \neg \mathbf{3}_b)\big) \vee \big((\neg \mathbf{2}_a \wedge \mathbf{3}_a) \wedge \mathbf{3}_b\big).$$

The valuations $f \in (2^{\mathbb{N}})^{\{b\}}$ belonging to $[\![\mathbf{C}^{1/2}_a F]\!]_{\{b\}}$ are those which can be extended to valuations of all Boolean variables in F satisfying F in at least half of the cases. Let us list all possible cases:

1. if $f(b)(2) = f(b)(3) = 1$, then F has $\frac{1}{4}$ chances of being true, as both $\neg \mathbf{2}_a$ and $\mathbf{3}_a$ must be true,

2. if $f(b)(2) = 1$ and $f(b)(3) = 0$, then F has $\frac{1}{2}$ chances of being true, as $\neg\mathbf{2}_a$ must be true,
3. if $f(b)(2) = 0$ and $f(b)(3) = 1$, then F has $\frac{3}{4}$ chances of being true, as either $\mathbf{2}_a$ or both $\neg\mathbf{2}_a$ and $\mathbf{3}_a$ must be true,
4. if $f(b)(2) = f(b)(3) = 0$, then F has no chance of being true.

Clearly, $[\![\mathbf{C}_a^{1/2}F]\!]_{\{b\}}$ only contains the valuations which agree with cases 2. and 3. Therefore, $[\![\mathbf{C}_b^{1/2}\mathbf{C}_a^{1/2}F]\!]_{\emptyset} = 2^{\mathbb{N}}$, i.e. $\mathbf{C}_b^{1/2}\mathbf{C}_a^{1/2}F$ is valid, since half of the valuations of b has at least $\frac{1}{2}$ chances of being extended to a model of F.

Let us also consider the following slight variation of F,

$$G = \big(\mathbf{2}_a \wedge (\neg\mathbf{2}_b \wedge \mathbf{3}_b)\big) \vee (\neg\mathbf{2}_a \wedge \mathbf{2}_b \wedge \neg\mathbf{3}_b) \vee \big((\neg\mathbf{2}_a \vee \mathbf{3}_a) \wedge \mathbf{3}_b\big).$$

The cases 2. and 4. are equivalent to the ones above, but 1. and 3. are not. Indeed,

1. if $f(b)(2) = f(b)(3) = 1$, then G has $\frac{3}{4}$ chances of being true, as either $\neg\mathbf{2}_a$ or $\mathbf{3}_a$ can be true,
2. if $f(b)(2) = 1$ and $f(b)(3) = 0$, then G has $\frac{1}{2}$ chances of being true, since as before $\neg\mathbf{2}_a$ must be true,
3. if $f(b)(2) = 0$, $f(b)(3) = 1$, then G must be true, as either $\mathbf{2}_a$ or $\neg\mathbf{2}_a$ (or $\mathbf{3}_a$) is true,
4. if $f(b)(2) = f(b)(3) = 0$, then G has no chance of being true.

Coherently, logical equivalence in **CPL** is defined relatively to a set of names X, by letting $F \equiv_X G$ if and only if $\mathsf{FN}(F), \mathsf{FN}(G) \subseteq X$ and $[\![F]\!]_X = [\![G]\!]_X$.

Relating CPL and CPL$_0$.

Remark 4.1.1. *Observe that there is a strong connection between (closed) formulae of **CPL**$_0$ and (closed) formulae of **CPL** in which only one name occurs.*

This can be easily proved by defining a translation which "preserves" validity from (closed) formulae of **CPL** (with one name only) to formulae of **CPL**$_0$. Intuitively, this transformation simply deletes all occurrences of the name in the counting expression.

Definition 4.1.4. We define a translation tr_n from formulae of **CPL** with only one name occurring in it, say $a \in \mathscr{A}$, to formulae of **CPL**$_0$

- $F = \mathbf{i}_a$, for any $i \in \mathbb{N}$ and $a \in \mathscr{A}$, then $tr_n(F) = \mathbf{i}$
- $F = \neg G$, then $tr_n(\neg G) = \neg(tr_n(G))$
- $F = G_1 \triangle G_2$, then $\triangle \in \{\wedge, \vee\}$, then $tr_n(G_1 \triangle G_2) = tr_n(G_1) \triangle tr_n(G_2)$
- $F = \mathbf{C}_a^q G$, then $tr_n(\mathbf{C}_a^q G) = \mathbf{C}^q tr_n(G)$
- $F = \mathbf{D}_a^q G$, the $tr_n(\mathbf{D}_a^q G) = \mathbf{D}^q tr_n(G)$.

Then, it is proved that this translation preserves validity. Without loss of generality, given a formula of **CPL** with one name only, we can assume that it is in PNF and with no nested quantification.

Proposition 4.1.1. *Let F be a closed formula of **CPL** in PNF and with no nested quantifications, such that FN$(F) \subseteq \{a\}$ for some $a \in \mathscr{A}$. Then,*

$$[\![F]\!]_{\{a\}} = (2^{\mathbb{N}})^{\{a\}} \quad \textit{iff} \quad [\![tr_n(F)]\!] = 2^{\mathbb{N}}$$

Proof Sketch. For assumption,

$$F \equiv \mathbf{C}^q_a G,$$

where G is quantifier-free. Since a is unique, it can be proved by induction on the structure of G that $\mu_{\mathscr{C}}([\![G]\!]) = \mu_{\mathscr{C}}([\![tr_n(G)]\!])$ and, then, by semantic Definition of $\mathbf{C}^q_a$ and $\mathbf{C}^q$, we conclude that $[\![F]\!]_{\{a\}} = (2^{\mathbb{N}})^{\{a\}}$ if and only if $[\![tr_n(F)]\!] = 2^{\mathbb{N}}$, as desired. □

4.2 Proof Theory of CPL

In this Section we introduce the rule system $\mathbf{LK}_{\mathbf{CPL}}$. To do so, we start by giving an alternative characterization for formulae of **CPL** using Boolean ones, in Section 4.2.1. Then, relying on these Boolean expressions, in Section 4.2.2 we present our *labelled* calculus, generalizing $\mathbf{LK}_{\mathbf{CPL}_0}$. Finally, in Section 4.2.3 we show this rule system to be sound and complete with respect to the quantitative semantics of **CPL**.

4.2.1 Characterizing the Semantics of CPL via Boolean Formulae

The definition of $[\![\mathbf{C}^q_a F]\!]_X$ and $[\![\mathbf{D}^q_a F]\!]_X$ is not very intuitive at first glance. So, we provide an alternative characterization of such sets by means of *named* Boolean formulae and prove them measurable.

Definition 4.2.1 (Named Boolean Formulae)**.** *Named Boolean formulae* are defined by the grammar below:

$$\mathscr{b} ::= x^a_i \;\mid\; \top \;\mid\; \bot \;\mid\; \neg\mathscr{b} \;\mid\; \mathscr{b} \wedge \mathscr{b} \;\mid\; \mathscr{b} \vee \mathscr{b},$$

where $i \in \mathbb{N}$ and $a \in \mathscr{A}$.

The set of free names for named Boolean formulae is defined as for counting ones. For any named Boolean formula $\mathscr{b}$, we let $\mathsf{FN}(\mathscr{b}) \subseteq \mathscr{A}$ indicate the set of names that occur in F, and $\mathsf{FV}(\mathscr{b}) \subseteq \mathscr{A} \times \mathbb{N}$ denote the set of pairs (a, i) such that the atom x^a_i occurs in $\mathscr{b}$.

Example 4.2.1. Let us consider the named Boolean formula $\mathscr{b} = (x^0_a \wedge \neg x^3_b) \vee (\neg x^1_c \wedge x^7_a)$, such that $\mathsf{FN}(\mathscr{b}) = \{a, b, c\}$ and $\mathsf{FV}(\mathscr{b}) = \{(a, 0), (a, 7), (b, 3), (c, 1)\}$.

Definition 4.2.2 (Named Boolean Semantics)**.** Given a Boolean formula $\mathscr{b}$, with $\mathsf{FN}(\mathscr{b}) \subseteq X$, its *interpretation*, $[\![\mathscr{b}]\!]_X$, is defined as follows:

$$[\![x_i^a]\!]_X := \{f : X \to 2^{\mathbb{N}} \mid f(a)(i) = 1\} \qquad [\![\neg\mathscr{b}]\!]_X := (2^{\mathbb{N}})^X - [\![\mathscr{b}]\!]_X$$
$$[\![\top]\!]_X := (2^{\mathbb{N}})^X \qquad [\![\mathscr{b} \wedge \mathscr{c}]\!]_X := [\![\mathscr{b}]\!]_X \cap [\![\mathscr{c}]\!]_X$$
$$[\![\bot]\!]_X := \emptyset^X \qquad [\![\mathscr{b} \vee \mathscr{c}]\!]_X := [\![\mathscr{b}]\!]_X \cup [\![\mathscr{c}]\!]_X.$$

As for univariate Boolean formulae, given a named Boolean formula $\mathscr{b}$, a *valuation* of $\mathscr{b}$ is any function $\theta : \mathsf{FV}(\mathscr{b}) \to \{\top, \bot\}$. The satisfaction relation $\theta \vDash \mathscr{b}$ is defined inductively in the obvious way. Observe that, if $X \supseteq \mathsf{FN}(\mathscr{b})$, any $f \in (2^{\mathbb{N}})^X$ induces a valuation by considering only its values $f(a)(i)$, for $(a, i) \in \mathsf{FV}(\mathscr{b})$. With a slight abuse of notation, we use $f \vDash \mathscr{b}$ to mean that the unique valuation induced by f satisfies $\mathscr{b}$. In particular, the $[\![\mathscr{b}]\!]_X$ could be also defined as $[\![\mathscr{b}]\!]_X = \{f \in (2^{\mathbb{N}})^X \mid f \vDash \mathscr{b}\}$.

We also need to introduce the auxiliary notion of a-decomposition for Boolean formulae:

Definition 4.2.3 (a-decomposition)**.** Let $\mathscr{b}$ be a named Boolean formula with free names in $X \cup \{a\}$. An *a-decomposition of $\mathscr{b}$* is a Boolean formula $\mathscr{c} = \bigvee_{i=0}^{k-1} \mathscr{d}_i \wedge \mathscr{e}_i$ such that:

- $[\![\mathscr{c}]\!]_{X\cup\{a\}} = [\![\mathscr{b}]\!]_{X\cup\{a\}}$,
- $\mathsf{FN}(\mathscr{d}_i) \subseteq \{a\}$ and $\mathsf{FN}(\mathscr{e}_i) \subseteq X$,
- if $i \neq j$, then $[\![\mathscr{e}_i]\!]_X \cap [\![\mathscr{e}_j]\!]_X = \emptyset$.

Example 4.2.2. Given the Boolean formula $\mathscr{c} = (x_a^0 \wedge x_b^1) \vee (\neg x_a^1 \wedge x_b^0)$, an a-decomposition of $\mathscr{b}$ can be obtained by transforming $\mathscr{b}$ into the equivalent formula

$$\mathscr{c}' = (x_a^0 \vee \neg x_a^1 \wedge x_b^1 \wedge x_b^0) \vee (x_a^0 \wedge x_b^1 \wedge \neg x_b^0) \vee (\neg x_a^1 \wedge x_b^0 \wedge \neg x_b^1)$$

and letting $\mathscr{d}_0 = x_a^0 \vee \neg x_a^1$, $\mathscr{d}_1 = x_a^0$, $\mathscr{d}_2 = \neg x_a^1$, and $\mathscr{e}_0 = x_b^1 \wedge x_b^0$, $\mathscr{e}_1 = x_b^1 \wedge \neg x_b^0$, $\mathscr{e}_2 = x_b^0 \wedge \neg x_b^1$.

A general way to construct an a-decomposition of Boolean formulae is described by proof of the following Lemma, which is established by induction on the structure of Boolean expressions.

Lemma 4.2.1. *Any named Boolean formula $\mathscr{b}$ with $\mathsf{FN}(\mathscr{b}) \subseteq X \cup \{a\}$ (with $a \notin X$) admits an a-decomposition in X.*

Proof. We will actually prove a stronger statement saying that any named Boolean formula $\mathscr{b}$ admits an a-decomposition $\bigvee_{i=0}^{k} \mathscr{d}_i \wedge \mathscr{e}_i$, where $[\![\bigvee_{i=0}^{k} \mathscr{e}_i]\!]_X = [\![\top]\!]_X$. We argue by induction on the structure of $\mathscr{b}$:

- if $\mathscr{b} = x_i^a$ or $\mathscr{b} = \neg x_i^a$, then $k = 0$, $\mathscr{d}_0 = \mathscr{b}$ and $\mathscr{e}_0 = \top$
- if $\mathscr{b} = x_i^b$ or $\mathscr{b} = \neg x_i^b$, where $b \neq a$, then $k = 1$, $\mathscr{d}_0 = \top$, $\mathscr{d}_1 = \bot$ and $\mathscr{e}_0 = \mathscr{b}$, $\mathscr{e}_1 = \neg\mathscr{b}$.

- if $\mathcal{b} = \mathcal{b}_1 \vee \mathcal{b}_2$ then, by IH, $\mathcal{b}_1 = \bigvee_{i_1=0}^{k_1-1} \mathcal{d}_{i_1}^1 \wedge \mathcal{e}_{i_1}^1$ and $\mathcal{b}_2 = \bigvee_{i_2=0}^{k_2-1} \mathcal{d}_{i_2}^2 \wedge \mathcal{e}_{i_2}^2$. So,

$$\begin{aligned}
\mathcal{b} &\equiv \left(\bigvee_{i_1=0}^{k_1-1} \mathcal{d}_{i_1}^1 \wedge \mathcal{e}_{i_1}^1\right) \vee \left(\bigvee_{i_2=0}^{k_2-1} \mathcal{d}_{i_2}^2 \wedge \mathcal{e}_{i_2}^2\right) \\
&\equiv \left(\bigvee_{i_1=0}^{k_1-1} \mathcal{d}_{i_1}^1 \wedge \mathcal{e}_{i_1}^1 \wedge \top\right) \vee \left(\bigvee_{i_2=0}^{k_2-1} \mathcal{d}_{i_2}^2 \wedge \mathcal{e}_{i_2}^2 \wedge \top\right) \\
&\equiv \left(\bigvee_{i_1=0}^{k_1-1} \mathcal{d}_{i_1}^1 \wedge \mathcal{e}_{i_1}^1 \wedge \bigvee_{i_2=0}^{k_2-1} \mathcal{e}_{i_2}^2\right) \vee \left(\bigvee_{i_2=0}^{k_2-1} \mathcal{d}_{i_2}^2 \wedge \mathcal{e}_{i_2}^2 \wedge \bigvee_{i_1=0}^{k_1-1} \mathcal{e}_{i_1}^1\right) \\
&\equiv \left(\bigvee_{i_1=0,i_2=0}^{k_1-1,k_2-1} (\mathcal{d}_{i_1}^1 \wedge \mathcal{e}_{i_1}^1 \wedge \mathcal{e}_{i_2}^2)\right) \vee \left(\bigvee_{i_2=0,i_1=0}^{k_2-1,k_1-1} (\mathcal{d}_{i_2}^2 \wedge \mathcal{e}_{i_2}^2 \wedge \mathcal{e}_{i_1}^1)\right) \\
&\equiv \bigvee_{i_1=0,i_2=0}^{k_1-1,k_2-1} (\mathcal{d}_{i_1}^1 \vee \mathcal{d}_{i_2}^2) \wedge (\mathcal{e}_{i_1}^1 \wedge \mathcal{e}_{i_2}^2).
\end{aligned}$$

  Let $k = k_1 \cdot k_2$. We can identify any $l \leq k-1$ with a pair (i_1, i_2), where $i_1 < k_1$ and $i_2 < k_2$. Let $\mathcal{d}_{i_1,i_2} = \mathcal{d}_{i_1}^1 \vee \mathcal{d}_{i_2}^2$ and $\mathcal{e}_{i_1,i_2} = \mathcal{e}_{i_1}^1 \vee \mathcal{e}_{i_2}^2$. Then,

$$\mathcal{b} \equiv \bigvee_{i_1=0,i_2=0}^{k_1-1,k_2-1} \mathcal{d}_{i_1,i_2} \wedge \mathcal{e}_{i_1,i_2}.$$

  Observe that for $(i_1, i_2) \neq (i_1', i_2')$, $\mathcal{e}_{i_1,i_2} \wedge \mathcal{e}_{i_1',i_2'} \equiv \bot$. Moreover, $\bigvee_{i_1,i_2} \mathcal{e}_{i_1,i_2} \equiv \bigvee_{i_1,i_2} \mathcal{e}_{i_1}^1 \vee \mathcal{e}_{i_2}^2 \equiv \bigvee_1 \mathcal{e}_{i_1}^1 \vee \bigvee_{i_2} \mathcal{e}_{i_2}^2 \equiv \top \vee \top \equiv \top$.

- $\mathcal{b} = \mathcal{b}_1 \wedge \mathcal{b}_2$, then, by IH, $\mathcal{b}_1 \equiv \bigvee_{i_1=0}^{k_1-1} \mathcal{d}_{i_1}^1 \wedge \mathcal{e}_{i_1}^1$ and $\mathcal{b}_2 \equiv \bigvee_{i_2=0}^{k_2-1} \mathcal{d}_{i_2}^2 \wedge \mathcal{e}_{i_2}^2$. So,

$$\begin{aligned}
\mathcal{b} &\equiv \left(\bigvee_{i_1=0}^{k_1-1} \mathcal{d}_{i_1}^1 \wedge \mathcal{e}_{i_1}^1\right) \wedge \left(\bigvee_{i_2=0}^{k_2-1} \mathcal{d}_{i_2}^2 \wedge \mathcal{e}_{i_1}\right) \\
&\equiv \bigvee_{i_1=0,i_2=0}^{k_1-1,k_2-1} \mathcal{d}_{i_1}^1 \wedge \mathcal{e}_{i_1}^1 \wedge \mathcal{d}_{i_2}^2 \wedge \mathcal{e}_{i_2}^2 \\
&\equiv \bigvee_{i_1=0,i_2=0}^{k_1-1,k_2-1} (\mathcal{d}_{i_1}^1 \wedge \mathcal{d}_{i_2}^2) \wedge (\mathcal{e}_{i_1}^1 \wedge \mathcal{e}_{i_2}^2).
\end{aligned}$$

  As in the case above, let $k = k_1 \cdot k_2$. We can identify any $l \leq k-1$ with a pair (i_1, i_2), $i_1 < k_1$ and $i_2 < k_2$. Let $\mathcal{d}_{i_1,i_2} = \mathcal{d}_{i_1}^1 \wedge \mathcal{d}_{i_2}^2$ and $\mathcal{e}_{i_1,i_2} = \mathcal{e}_{i_1}^1 \wedge \mathcal{e}_{i_2}^2$ We have than that

$$\mathcal{b} \equiv \bigvee_{i_1=0,i_2=0}^{k_1-1,k_2-1} \mathcal{e}_{i_1,i_2} \wedge \mathcal{e}_{i_1,i_2}.$$

As in the previous case we have that for $(i_1, i_2) \neq (i'_1, i'_2)$, $\mathscr{e}_{i_1,i_2} \wedge \mathscr{e}_{i'_1,i'_2} \equiv \bot$ and $\bigvee_{i_1,i_2} \mathscr{e}_{i_1,i_2} \equiv \bigvee_{i_1,i_2} \mathscr{e}^1_{i_1} \wedge \mathscr{e}^2_{i_2} \equiv \bigvee_{i_1} \mathscr{e}^1_{i_1} \wedge \bigvee_{i_2} \mathscr{e}^2_{i_2} \equiv \top \wedge \top \equiv \top$.

□

It is worth observing that, while an a-decomposition $\mathscr{c}$ of $\mathscr{b}$ always exists, finding it may be complex in computational terms, since $\mathscr{c}$ can be of exponential length with respect of $\mathscr{b}$. Yet, a-decompositions can be used to show that the interpretation of a quantified formula is a finite union of measurable sets.

Lemma 4.2.2 (Fundamental Lemma)**.** *Let $\mathscr{b}$ be a named Boolean formula with $\mathsf{FN}(\mathscr{b}) \subseteq X \cup \{a\}$ and $\mathscr{c} = \bigvee_{i=0}^{k-1} \mathscr{d}_i \wedge \mathscr{e}_i$ be an a-decomposition of $\mathscr{b}$. Then, for all $q \in \mathbb{Q} \cap [0,1]$,*

$$\{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![\mathscr{b}]\!]_{X\cup\{a\}})) \geq q\} = \bigcup_i \{[\![\mathscr{e}_i]\!]_X \mid \mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q\}$$

$$\{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![\mathscr{b}]\!]_{X\cup\{a\}})) < q\} = \bigcup_i \{[\![\mathscr{e}_i]\!]_X \mid \mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) < q\}.$$

Proof. We prove the first equality, the second one being established in a similar way. First, notice that if $q = 0$, then both sets are equal to $(2^{\mathbb{N}})^X$. So, let us suppose $q > 0$.

- $\subseteq$ Suppose $\mu_{\mathscr{C}}(\Pi_f([\![\mathscr{b}]\!]_{X\cup\{a\}})) \geq q$. Then, since $\Pi_f([\![\mathscr{b}]\!]_{X\cup\{a\}})$ is non-empty, from $\mathscr{b} \equiv \bigvee_i^k \mathscr{d}_i \wedge \mathscr{e}_i$, by Definition 4.2.3, we deduce that there exists an $i \leq k$ such that $f \in [\![\mathscr{e}_i]\!]_X$ and for each $g \in [\![\mathscr{d}_i]\!]_{\{a\}}$, $f + g \in [\![\mathscr{d}_i \wedge \mathscr{e}_i]\!]_{X\cup\{a\}}$. This implies then that $[\![\mathscr{d}_i]\!]_{\{a\}} \subseteq \{g \in (2^{\mathbb{N}})^{\{a\}} \mid f + g \in [\![\mathscr{d}_i \wedge \mathscr{e}_i]\!]_{X\cup\{a\}}\} \overset{D\,4.1.2}{=} \Pi_f([\![\mathscr{b}]\!]_{X\cup\{a\}})$. Moreover, since the sets $[\![\mathscr{e}_i]\!]_X$ are pairwise disjoint, for all $j \neq i$, $f \notin [\![\mathscr{e}_j]\!]_X$, which implies that $\Pi_f([\![\mathscr{b}]\!]_{X\cup\{a\}}) \subseteq [\![\mathscr{d}_i]\!]_{\{a\}}$. Hence, $\Pi_f([\![\mathscr{b}]\!]_{X\cup\{a\}}) = [\![\mathscr{d}_i]\!]_{\{a\}}$, which implies $\mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q$.
- $\supseteq$ If $f \in [\![\mathscr{e}_i]\!]_X$, where $\mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q$, then, since $\mathscr{d}_i \wedge \mathscr{e}_i \vDash^{X\cup\{a\}} \mathscr{b}$, $\mu_{\mathscr{C}}(\Pi_f([\![\mathscr{b}]\!]_{X\cup\{a\}})) \geq \mu_{\mathscr{C}}(\Pi_f([\![\mathscr{d}_i \wedge \mathscr{e}_i]\!]_{X\cup\{a\}})) \overset{D\,4.1.2}{=} \mu_{\mathscr{C}}(\{g \in (2^{\mathbb{N}})^{\{a\}} \mid f + g \in [\![\mathscr{d}_i \wedge \mathscr{e}_i]\!]_{X\cup\{a\}}\}) = \mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q$, that is $\mu_{\mathscr{C}}(\Pi_f([\![\mathscr{b}]\!]_{X\cup\{a\}} \geq q$ as desired.

□

The importance of this Lemma 4.2.2 consists of its being used to associate any formula of **CPL** F (where $\mathsf{FN}(F) \subseteq X$), with a Boolean formula $\mathscr{b}_F$, such that $[\![F]\!]_X = [\![\mathscr{b}_F]\!]_X$. Then, the crucial cases concerning $[\![\mathbf{C}^q_a F]\!]_X$ and $[\![\mathbf{D}^q_a F]\!]_X$ are handled using the fact that these sets are finite unions of measurable sets of the form $[\![\mathscr{e}_i]\!]_X$, where $\bigvee_{i=0}^{k-1} \mathscr{e}_i \wedge \mathscr{d}_i$ is an a-decomposition of $\mathscr{b}_F$. As an immediate consequence of Lemma 4.2.2, it follows that the Borel sets which are interpretations of counting formulae are precisely those whcih are the interpretations of Boolean formulae with names. In other words, the formula of **CPL** do not increase the set of "definable" Borel sets.

Corollary 4.2.1. *For any formula F (with $\mathsf{FN}(F) \subseteq X$), there is a Boolean formula $\mathscr{b}_F$, such that $[\![F]\!]_X = [\![\mathscr{b}_F]\!]_X$.*

Proof. The proof is by induction on the structure of F. The crucial steps concerning $[\![\mathbf{C}_a^q F]\!]_X$ and $[\![\mathbf{D}_a^q F]\!]_X$ are handled using the fact that, thanks to Lemma 4.2.2, these sets are finite unions of sets of the form $[\![e_i]\!]_X$, where $\bigvee_{i=0}^{k-1} d_i \wedge e_i$ is an a-decomposition of $\mathscr{b}_F$.

□

Observe again that, given a counting formula F, finding a quantifier-free $\mathscr{b}$ such that $[\![F]\!]_X = [\![\mathscr{b}]\!]_X$ can be complex, computationally, as the construction of $\mathscr{b}$ crucially depends on the construction of a-decompositions for any sub-formula of F of the form $\mathbf{C}_a^q G$ and $\mathbf{D}_a^q G$.

Remark 4.2.1. *To show that the sets $[\![F]\!]_X$ are Borel we exploited Theorem 4.1.1, which is a non-trivial measure-theoretic result, to account for the case of quantifiers. Yet, using the result above one can get rid of this Lemma and prove that the sets $[\![F]\!]_X$ are Borel using, in the inductive steps related to quantifiers, the fact that $[\![\mathbf{C}^q F]\!]_X$ and $[\![\mathbf{D}^q F]\!]_X$ are finite unions of sets which must all be Borel by the inductive hypothesis.*

As for **CPL**, the measure of named Boolean formulae can be related with $\sharp\mathtt{SAT}$. In particular, given a named Boolean formula $\mathscr{b}$, let

$$\sharp\mathtt{SAT}(\mathscr{b}) = \sharp\{\theta : \mathsf{FN}(\mathscr{b}) \to \{0,1\} \mid \theta \vDash \mathscr{b}\}.$$

This function can be used to compute the measure of the finitary Borel set $[\![\mathscr{b}]\!]_X$. The proof is very similar to the corresponding one for $\mathbf{CPL}_0$, as provided in Chapter 3.

Lemma 4.2.3. *For any named Boolean formula $\mathscr{b}$ and $X \supseteq \mathsf{FN}(\mathscr{b})$,*

$$\mu_{\mathscr{C}}([\![\mathscr{b}]\!]_X) = \sharp\mathtt{SAT}(\mathscr{b}) \cdot 2^{-\sharp \mathsf{FV}(\mathscr{b})}.$$

Proof. For any valuation θ of $\mathscr{b}$, let $X(\theta) \subseteq (2^{\mathbb{N}})^X$ be the following measurable set:

$$X(\theta) = \{f \mid \forall (a,i) \in \mathsf{FN}(\mathscr{b}), f(a)(i) = \theta(a,i)\} = \bigcap_{i=0}^{n-1} Cyl(a,i)^{\theta(a,i)},$$

where

$$Cyl(a,i)^{\theta(a,i)} = \begin{cases} Cyl(a,i) & \text{if } \theta(a,i) = 1 \\ \overline{Cyl(a,i)} & \text{if } \theta(a,i) = 0. \end{cases}$$

One can check by induction that for any Boolean formula $\mathscr{b}$, $[\![\mathscr{b}]\!]_X = \bigcup_{\theta \vDash \mathscr{b}} X(\theta)$. Then, since for all distinct θ', θ'', $X(\theta') \cap X(\theta'') = \emptyset$, letting $N = \sharp\mathsf{FV}(\mathscr{b})$, we have $\sharp\mathtt{SAT}(\mathscr{b}) \cdot 2^{-N} = \sum_{\theta \vDash \mathscr{b}} X(\theta) \cdot 2^{-N} = \sum_{\theta \vDash \mathscr{b}} \mu_{\mathscr{C}}(X(\theta)) = \mu_{\mathscr{C}}\Big(\bigcup_{\theta \vDash \mathscr{b}} X(\theta)\Big) = \mu_{\mathscr{C}}([\![\mathscr{b}]\!]_X)$. □

As a of Lemma 4.2.3, $\mu_{\mathscr{C}}\big([\![\mathscr{b}]\!]_X\big)$ is always a rational number and it is independent from the choice of $X \supseteq \mathsf{FN}(\mathscr{b})$.

Furthermore, due to Lemma 4.2.2 we can prove that all sets $[\![F]\!]_X$ are measurable.[2]

Corollary 4.2.2. *For any formula of* ***CPL*** F *and* $X \supseteq \mathsf{FN}(F)$, $[\![F]\!]_X \in \mathscr{B}\big((2^{\mathbb{N}})^X\big)$.

Proof. The proof is by induction on the structure of formulae of **CPL**.

- $F = \mathbf{i}_a$, for some $i \in \mathbb{N}$ and $a \in \mathscr{A}$. Then, by Definition 4.1.3, $[\![\mathbf{i}_a]\!]_X = \{f \mid f(a)(i) = 1\} = Cyl(a,i)$ (with $X \supseteq \{a\}$), which is measurable and, in particular, – as seen in Section 4.1 – $\mu_{\mathscr{C}}(Cyl(a,i)) = \frac{1}{2}$ (for any $i \in \mathbb{N}$ and $a \in \mathscr{A}$).
- $F = \neg G$. Then, by Definition 4.1.3, $[\![\neg G]\!]_X = (2^{\mathbb{N}})^X - [\![G]\!]_X$ (with $X \supseteq \mathsf{FN}(G)$). For IH, $[\![G]\!]_X \in \mathscr{B}\big((2^{\mathbb{N}})^X\big)$ so, clearly, by the axioms of the (generated) σ-algebra, $[\![\neg G]\!]_X = 2^{\mathbb{N}} - [\![G]\!] \in \mathscr{B}\big((2^{\mathbb{N}})^X\big)$ as well.
- $F = G \wedge H$. Then, by Definition 4.1.3, $[\![F]\!]_X = [\![G]\!]_X \cap [\![H]\!]_X$ (with $X \supseteq \mathsf{FN}(F)$). For IH $[\![G]\!]_X, [\![H]\!]_X \in \mathscr{B}\big((2^{\mathbb{N}})^X\big)$, so clearly, by the axioms of the (generated) σ-algebra $[\![F]\!]_X = [\![G]\!]_X \cap [\![H]\!]_X \in \mathscr{B}\big((2^{\mathbb{N}})^X\big)$.
- $F = G \vee H$. Analogous to the case above.
- $F = \mathbf{C}^q_a G$. Then, by Definition 4.1.3, $[\![\mathbf{C}^q_a F]\!]_X = \{f \mid \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q\}$ (with $X \supseteq F$), that is, for the Lemma 4.2.2, $[\![\mathbf{C}^q_a F]\!] = \bigcup_i \{[\![\mathscr{e}_i]\!]_X \mid \mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q\}$, where $\mathscr{c} = \bigvee_{i=0}^{k-1} \mathscr{d}_i \wedge \mathscr{e}_i$ is an a-decomposition of $\mathscr{b}_G$. Furthermore, by IH, $[\![G]\!] (= [\![\mathscr{b}_G]\!]) \in \mathscr{B}\big((2^{\mathbb{N}})^X\big)$, so, by the axioms of the σ-algebra, also $\bigcup_i \{[\![\mathscr{d}_i]\!]_X \mid \mu_{\mathscr{C}}([\![\mathscr{e}_i]\!]_{\{a\}}) \geq q\} \in \mathscr{B}\big((2^{\mathbb{N}})^X\big)$.
- $F = \mathbf{D}^q_a G$. Similar to the case above.

□

4.2.2 The Sequent Calculus $\mathbf{LK}_{\mathbf{CPL}}$

We now introduce a sound and complete labelled calculus for **CPL**, called $\mathbf{LK}_{\mathbf{CPL}}$. Using the Fundamental Lemma 4.2.2, we could define this proof system in analogy with the one introduced for $\mathbf{CPL}_0$.

The Labelled Language. As for $\mathbf{LK}_{\mathbf{CPL}_0}$, the language of $\mathbf{LK}_{\mathbf{CPL}}$ is labelled.

Definition 4.2.4 (Named External Hypothesis)**.** A *named external hypothesis* is an expression of one of the following forms:

- $\vDash \{a\} \in X$
- $\mathscr{b} \vDash^X \mathscr{c}$
- $\mu\big([\![\mathscr{b}]\!]_X\big) = 0$ and $\mu\big([\![\mathscr{b}]\!]_X\big) = 1$

[2] Observe that this could also be proved as an immediate consequence of Corollary 4.2.1.

- $\mathscr{b} \models^X \bigvee_i \{\mathscr{e}_i \mid \mu(\llbracket \mathscr{d}_i \rrbracket_Y) \triangleright q\}$ and $\bigvee_i \{\mathscr{e}_i \mid \mu(\llbracket \mathscr{d}_i \rrbracket_Y) \triangleright q\} \models^X \mathscr{b}$,

where $i \in \mathbb{N}$, $X, Y \subseteq \mathscr{A}$, $\mathscr{b}, \mathscr{d}_i, \mathscr{e}_i$ are named Boolean formulae, $q \in \mathbb{Q} \cap [0,1]$ and $\triangleright \in \{\geq, \leq, >, <, =\}$.

Definition 4.2.5 (Named Sequent of $\mathbf{LK_{CPL}}$)**.** A *named sequent of* $\mathbf{LK_{CPL}}$ is an expression of the form $\vdash^X L$, where L is a labelled formula – i.e., it is either $\mathscr{b} \rightarrowtail F$ or $\mathscr{b} \leftarrowtail F$ – with $\mathscr{b}$ being a named Boolean formula and $\mathsf{FN}(\mathscr{b}) \cup \mathsf{FN}(F) \subseteq X$.

The Rules System. $\mathbf{LK_{CPL}}$ is a one-sided, single-succedent and labelled sequent calculus, the rules of which are obtained as straightforward extensions of $\mathbf{LK_{CPL_0}}$. The only ones which are substantially different with respect to the corresponding univariate ones are initial sequents and counting rules.

Definition 4.2.6 (Sequent Calculus $\mathbf{LK_{CPL}}$)**.** The proof system $\mathbf{LK_{CPL}}$ is defined by the rules illustrated in Figure 4.1.

Remark 4.2.2. *Rules $R^{\rightarrowtail}_{\mathbf{C}}$ and $R^{\leftarrowtail}_{\mathbf{D}}$ look significantly simpler than the corresponding rules $R^{\leftarrowtail}_{\mathbf{C}}$ and $R^{\rightarrowtail}_{\mathbf{D}}$, which refer to the a-decomposition of $\mathscr{c}$. Actually, it can easily shown that the following variants $R^{\rightarrowtail *}_{\mathbf{C}}$ and $R^{\leftarrowtail *}_{\mathbf{D}}$ – which are similar to the corresponding univariate ones – are admissible in* $\mathbf{LK_{CPL}}$*:*

$$\frac{\vdash^{X \cup \{a\}} \mathscr{c} \rightarrowtail F \qquad \mathscr{b} \models \bigvee_i \{\mathscr{e}_i \mid \mu(\llbracket \mathscr{d}_i \rrbracket_{\{a\}}) \geq q\}}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{C}^q_a F} R^{\rightarrowtail *}_{\mathbf{C}}$$

$$\frac{\vdash^{X \cup \{a\}} \mathscr{c} \rightarrowtail F \qquad \neg\mathscr{b} \models \bigvee_i \{\mathscr{e}_i \mid \mu(\llbracket \mathscr{d}_i \rrbracket_{\{a\}}) \geq q\}}{\vdash^X \mathscr{b} \leftarrowtail \mathbf{D}^q_a F} R^{\leftarrowtail *}_{\mathbf{D}}$$

where $\bigvee_i \mathscr{e}_i \wedge \mathscr{d}_i$ is an a-decomposition of $\mathscr{c}$.
Let us consider for example $R^{\rightarrowtail *}_{\mathbf{C}}$. Assume that $\bigvee_{i=0}^{k-1} \mathscr{e}_i \wedge \mathscr{d}_i$, is an a-decomposition of $\mathscr{c}$, with $\mathsf{FN}(\mathscr{e}_i) \subseteq \{a\}$ and $\mathsf{FN}(\mathscr{e}_i) \subseteq X$. Then, for all $j \in \{0, \ldots k-1\}$ *such that $\mu_{\mathscr{C}}(\llbracket \mathscr{d}_j \rrbracket_{\{a\}}) \geq q$ we apply $R^{\rightarrowtail}_{\mathbf{C}}$. Then, we conclude due to $R\cup$:*

$$\dfrac{\dfrac{\vdash \mathscr{d}_{j_1} \wedge \mathscr{e}_{j_1} \rightarrowtail F \qquad \mu(\llbracket \mathscr{d}_{j_1} \rrbracket) \geq q}{\vdash^X \mathscr{e}_{j_1} \rightarrowtail \mathbf{C}^q_a F} \quad \ldots \quad \dfrac{\vdash \mathscr{d}_{j_n} \wedge \mathscr{e}_j \rightarrowtail F \qquad \mu(\llbracket \mathscr{d}_{j_n} \rrbracket) \geq q}{\vdash^X \mathscr{e}_{j_n} \rightarrowtail \mathbf{C}^q_a F} \qquad \mathscr{b} \models^X \bigvee_j \mathscr{e}_j}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{C}^q_a F} R\cup$$

Clearly, $\bigvee_j^{\{j_1, \ldots, j_n\}} \mathscr{e}_j$ corresponds to $\bigvee_i \{\mathscr{e}_i \mid \mu(\llbracket \mathscr{e}_i \rrbracket_{\{a\}}) \geq q\}$ as desired.

4.2.3 Soundness and Completeness

The proofs of soundness and completeness for $\mathbf{LK_{CPL}}$ are structurally equivalent to the ones for $\mathbf{LK_{CPL_0}}$. In this Section we briefly illustrate the skeleton of the proof for **CPL**, trying to emphasize the discrepancies between the two.

Initial Sequents

$$\frac{\vDash \{a\} \in X \qquad \mathscr{b} \vDash^X x_i^a}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{i}_a}\ Ax1 \qquad \frac{\vDash \{a\} \in X \qquad x_i^a \vDash^X \mathscr{b}}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{i}_a}\ Ax2$$

Union Rule

$$\frac{\vdash^X \mathscr{c} \rightarrowtail F \qquad \vdash^X \mathscr{d} \rightarrowtail F \qquad \mathscr{b} \vDash^X \mathscr{c} \vee \mathscr{d}}{\vdash^X \mathscr{b} \rightarrowtail F}\ R\cup$$

Intersection Rule

$$\frac{\vdash^X \mathscr{c} \rightarrowtail F \qquad \vdash^X \mathscr{d} \rightarrowtail F \qquad \mathscr{c} \wedge \mathscr{d} \vDash^X \mathscr{b}}{\vdash^X \mathscr{b} \rightarrowtail F}\ R\cap$$

Logical Rules

$$\frac{\vdash^X \mathscr{c} \rightarrowtail F \qquad \mathscr{b} \vDash^X \neg\mathscr{c}}{\vdash^X \mathscr{b} \rightarrowtail \neg F}\ R^{\rightarrowtail}_{\neg} \qquad \frac{\vdash^X \mathscr{c} \rightarrowtail F \qquad \neg\mathscr{c} \vDash^X \mathscr{b}}{\vdash^X \mathscr{b} \rightarrowtail \neg F}\ R^{\rightarrowtail}_{\neg}$$

$$\frac{\vdash^X \mathscr{b} \rightarrowtail F}{\vdash^X \mathscr{b} \rightarrowtail F \vee G}\ R1^{\rightarrowtail}_{\vee} \qquad \frac{\vdash^X \mathscr{b} \rightarrowtail G}{\vdash^X \mathscr{b} \rightarrowtail F \vee G}\ R2^{\rightarrowtail}_{\vee}$$

$$\frac{\vdash^X \mathscr{b} \rightarrowtail F \qquad \vdash^X \mathscr{b} \rightarrowtail G}{\vdash^X \mathscr{b} \rightarrowtail F \wedge G}\ R^{\rightarrowtail}_{\wedge} \qquad \frac{\vdash^X \mathscr{b} \rightarrowtail F \qquad \vdash^X \mathscr{b} \rightarrowtail G}{\vdash^X \mathscr{b} \rightarrowtail F \wedge G}\ R^{\rightarrowtail}_{\wedge}$$

$$\frac{\vdash^X \mathscr{b} \rightarrowtail F}{\vdash^X \mathscr{b} \rightarrowtail F \wedge G}\ R1^{\rightarrowtail}_{\wedge} \qquad \frac{\vdash^X \mathscr{b} \rightarrowtail G}{\vdash^X \mathscr{b} \rightarrowtail F \wedge G}\ R2^{\rightarrowtail}_{\wedge}$$

Counting Rules

$$\frac{\mu\big([\![\mathscr{b}]\!]_X\big) = 0}{\vdash^X \mathscr{b} \rightarrowtail F}\ R^{\rightarrowtail}_{\mu} \qquad \frac{\mu\big([\![\mathscr{b}]\!]_X\big) = 1}{\vdash^X \mathscr{b} \rightarrowtail F}\ R^{\rightarrowtail}_{\mu}$$

$$\frac{\vdash^{X\cup\{a\}} \mathscr{b} \wedge \mathscr{c} \rightarrowtail F \qquad \mu([\![\mathscr{c}]\!]) \geq q}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{C}_a^q F}\ R^{\rightarrowtail}_{\mathbf{C}} \qquad \frac{\vdash^{X\cup\{a\}} \neg\mathscr{b} \wedge \mathscr{c} \rightarrowtail F \qquad \mu([\![\mathscr{c}]\!]) \geq q}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{D}_a^q F}\ R^{\rightarrowtail}_{\mathbf{D}}$$

where $\mathsf{FN}(\mathscr{b}) \subseteq X, \mathsf{FN}(\mathscr{c}) \subseteq \{a\}$ and $\mathsf{FN}(\mathscr{b}) \cap \mathsf{FN}(\mathscr{c}) = \emptyset$

where $\mathsf{FN}(\mathscr{b}) \subseteq X, \mathsf{FN}(\mathscr{c}) \subseteq \{a\}$ and $\mathsf{FN}(\mathscr{b}) \cap \mathsf{FN}(\mathscr{c}) = \emptyset$

$$\frac{\vdash^{X\cup\{a\}} \mathscr{c} \rightarrowtail F \qquad \bigvee_i \{e_i \mid \mu\big([\![d_i]\!]_{\{a\}}\big) \geq q\} \vDash^X \mathscr{b}}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{C}_a^q F}\ R^{\rightarrowtail}_{\mathbf{C}}$$

where $\bigvee_i e_i \wedge d_i$ is an a-decomposition of $\mathscr{c}$.

$$\frac{\vdash^{X\cup\{a\}} \mathscr{c} \rightarrowtail F \qquad \mathscr{b} \vDash^X \bigvee_i \{e_i \mid \mu\big([\![d_i]\!]_{\{a\}}\big) < q\}}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{D}_a^q F}\ R^{\rightarrowtail}_{\mathbf{D}}$$

where $\bigvee_i e_i \wedge d_i$ is an a-decomposition of $\mathscr{c}$.

Figure 4.1: Sequent Calculus $\mathbf{LK}_{\mathbf{CPL}}$

Preliminaries. First of all, in the multivariate setting, the notion of validity for labelled formulae and sequents is defined in relation to a set of names X.

Definition 4.2.7 (X-Validity)**.** Given a Boolean formula $\mathscr{b}$, and a **CPL**-formula F, with $\mathsf{FN}(\mathscr{b}) \cup \mathsf{FN}(F) \subseteq X$, the labelled formula $\mathscr{b} \rightarrowtail F$ (resp., $\mathscr{b} \leftarrowtail F$) is *$X$-valid* when $[\![\mathscr{b}]\!]_X \subseteq [\![F]\!]_X$ (resp., $[\![F]\!]_X \subseteq [\![\mathscr{b}]\!]_X$).

Similarly, an external hypothesis containing n named Boolean variables $\mathscr{b}_{i_1}, \ldots, \mathscr{b}_{i_n}$, with $i_1, \ldots, i_n \in \mathbb{N}$, is valid when for every X such that $\bigcup_{i_j=1}^{n} \mathsf{FN}(\mathscr{b}_{i_j}) \subseteq X$, for $j \in \{1, \ldots, n\}$, the hypothesis is valid. So, for example, $\mathscr{b} \vDash^X x_i^a$ (resp., $x_i^a \vDash^X \mathscr{b}$) is valid if and only if for every $X \supseteq \mathsf{FN}(\mathscr{b}) \cup \mathsf{FN}(x_i^a)$, $[\![\mathscr{b}]\!]_X \subseteq [\![x_i^a]\!]_X$ (resp., $[\![x_a^i]\!]_X \subseteq [\![\mathscr{b}]\!]_X$).

Notation 4.2.1. Let us write $F \equiv_X G$ when $[\![F]\!]_X = [\![G]\!]_X$.

Soundness. In order to establish soundness we need to first introduce the auxiliary Corollary 4.2.3 below.

Corollary 4.2.3. *For any Boolean formula $\mathscr{b}$ and $X \supseteq \mathscr{b}$, if $\mu_{\mathscr{C}}([\![\mathscr{b}]\!]_X) = 0$, then $[\![\mathscr{b}]\!]_X = (\emptyset)^X$.*

Proof. Since $[\![\mathscr{b}]\!]_X \neq (\emptyset)^X$, $\mathscr{b}$ must have at least one satisfying model, defined by the set satisfying a *finite* set of conditions. Since $[\![\mathscr{b}]\!]_X$ includes all models models of $\mathscr{b}$, $\mu([\![\mathscr{b}]\!]_X) > 0$. □

As for $\mathbf{LK}_{\mathbf{CPL}_0}$, soundness is proved by standard induction.

Proposition 4.2.1 (Soundness of $\mathbf{LK}_{\mathbf{CPL}}$)**.** *If a sequent is derivable in $\mathbf{LK}_{\mathbf{CPL}}$, then is valid.*

Proof. The proof is by induction on the height n, of the derivation for $\vdash^X L$.

Base case. The sequent is either an initial sequent or is derived by a μ-rule. Let us consider the cases $Ax1$ and $R_{\mu}^{\rightarrowtail}$. The other ones are proved in an analogous way.

- $Ax1$. Let the derivation be of the following form:

$$\frac{\vDash \{a\} \in X \qquad \mathscr{b} \vDash^X x_i^a}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{i}_a} Ax1$$

  As $\mathscr{b} \vDash^X x_i^a$, $\mathsf{FN}(\mathscr{b}) \cup \mathsf{FN}(x_i^a) \subseteq X$, so $[\![\mathscr{b}]\!]_X \subseteq [\![x_i^a]\!]_X$, that is $[\![\mathscr{b}]\!]_X \subseteq \{f \in (2^{\mathbb{N}})^X \mid f(a)(i) = 1\}$. By Definition 4.1.3, $[\![\mathbf{i}_a]\!]_X = \{f \in (2^{\mathbb{N}})^X \mid f(a)(i) = 1\}$, so $[\![\mathscr{b}]\!]_X \subseteq [\![\mathbf{i}_a]\!]_X$ and $\mathscr{b} \rightarrowtail \mathbf{i}_a$ is X-valid. We conclude that $\vdash^X \mathscr{b} \rightarrowtail \mathbf{i}_a$ is valid as well.

- $R_{\mu}^{\rightarrowtail}$. Let the derivation be of the following form:

$$\frac{\mu\big([\![\mathscr{b}]\!]_X\big) = 0}{\vdash^X \mathscr{b} \rightarrowtail F} R_{\mu}^{\rightarrowtail}$$

Since $\mu([\![\mathcal{b}]\!]_X) = 0$, by Corollary 4.2.3, $[\![\mathcal{b}]\!]_X = (\emptyset)^X$. Then, trivially, for every F, $[\![\mathcal{b}]\!]_X \subseteq [\![F]\!]_X$, that is $\mathcal{b} \rightarrowtail F$ is X-valid. We conclude that $\vdash^X \mathcal{b} \rightarrowtail F$ is valid.

- The proofs for $Ax2$ and $R^{\leftarrowtail}_{\mu}$ are similar.

Inductive Case. Let us assume soundness to hold for derivations of height up to n and show it holds for derivations of height $n+1$. The proof is equivalent to that for $\mathbf{LK}_{\mathbf{CPL}_0}$, so let us just consider a few examples only. All the other cases are similar.

- $R\cup$. The derivation is of the following form:

$$\frac{\begin{array}{ccc}\vdots & & \vdots \\ \vdash^X \mathcal{c} \rightarrowtail F & \vdash^X \mathcal{d} \rightarrowtail F & \mathcal{b} \vDash^X \mathcal{c} \vee \mathcal{d}\end{array}}{\vdash^X \mathcal{b} \rightarrowtail F} R\cup$$

The premisses are valid (IH), so $\vDash^X \mathcal{c} \rightarrowtail F$ and $\vDash^X \mathcal{d} \rightarrowtail F$, that is $[\![\mathcal{c}]\!]_X \subseteq [\![F]\!]_X$ and $[\![\mathcal{d}]\!]_X \subseteq [\![F]\!]_X$. Then, for basic set theory, also $[\![\mathcal{c} \cup \mathcal{d}]\!]_X \subseteq [\![F]\!]_X$. Furthermore, $\mathcal{b} \vDash^X \mathcal{c} \vee \mathcal{d}$, then $[\![\mathcal{b}]\!]_X \subseteq [\![\mathcal{c}]\!]_X \cup [\![\mathcal{d}]\!]_X$. So, for the transitivity of the subset relation, $[\![\mathcal{b}]\!]_X \subseteq [\![F]\!]_X$. We conclude that $\vdash^X \mathcal{b} \rightarrowtail F$ is valid as desired.

- $R\cap$. The derivation is of the following form:

$$\frac{\begin{array}{ccc}\vdots & & \vdots \\ \vdash^X \mathcal{c} \leftarrowtail F & \vdash^X \mathcal{d} \leftarrowtail F & \mathcal{c} \wedge \mathcal{d} \vDash^X \mathcal{b}\end{array}}{\vdash^X \mathcal{b} \leftarrowtail F} R\cap$$

The premisses are valid (IH), so $\vDash^X \mathcal{c} \leftarrowtail F$ and $\vDash^X \mathcal{d} \leftarrowtail F$, that is $[\![F]\!]_X \subseteq [\![\mathcal{c}]\!]_X$ and $[\![F]\!]_X \subseteq [\![\mathcal{d}]\!]_X$. Then, for basic set theory, $[\![F]\!] \subseteq [\![\mathcal{c}]\!]_X \cup [\![\mathcal{d}]\!]_X$. Furthermore as $\mathcal{c} \wedge \mathcal{d} \vDash^X \mathcal{b}$, also $[\![\mathcal{c}]\!]_X \cap [\![\mathcal{d}]\!]_X \subseteq [\![\mathcal{b}]\!]_X$. So, for the transitivity of subset relaiton, $[\![F]\!]_X \subseteq [\![\mathcal{b}]\!]_X$. We conclude that $\vdash^X \mathcal{b} \leftarrowtail F$ is valid, as desired.

- $R^{\rightarrowtail}_{\neg}$. The derivation is of the following form:

$$\frac{\begin{array}{c}\vdots \\ \vdash^X \mathcal{c} \leftarrowtail F \qquad \mathcal{b} \vDash^X \neg\mathcal{c}\end{array}}{\vdash^X \mathcal{b} \rightarrowtail \neg F} R^{\rightarrowtail}_{\neg}$$

The premiss is valid (IH), so $\vDash^X \mathcal{c} \leftarrowtail F$, that is $[\![F]\!]_X \subseteq [\![\mathcal{c}]\!]_X$. Then, for basic set theory, $(2^{\mathbb{N}})^X - [\![\mathcal{c}]\!]_X \subseteq (2^{\mathbb{N}})^X - [\![F]\!]_X$, i.e. by Definition 4.1.3 and 4.2.1, $[\![\neg\mathcal{c}]\!]_X \subseteq [\![\neg F]\!]_X$. Furthermore as $\mathcal{b} \vDash^X \neg\mathcal{c}$, also $[\![\mathcal{b}]\!]_X \subseteq [\![\neg\mathcal{c}]\!]_X$. So, for the transitivity of subset relation, $[\![\mathcal{b}]\!]_X \subseteq [\![\neg F]\!]_X$. We conclude that $\vdash_X \mathcal{b} \rightarrowtail \neg F$ is valid, as desired.

- $R\neg^{\rightarrowtail}$. The proof is similar to the one above.
- $R1^{\rightarrowtail}_{\vee}$. The derivation is of the following form:

$$\dfrac{\begin{array}{c}\vdots\\ \vdash^X \mathscr{b} \rightarrowtail F\end{array}}{\vdash^X \mathscr{b} \rightarrowtail F \vee G}\, R1^{\rightarrowtail}_{\vee}$$

  The premiss is valid (for IH), so $\vDash^X \mathscr{b} \rightarrowtail F$, that is $[\![\mathscr{b}]\!]_X \subseteq [\![F]\!]_X$. Then, for basic set theory, $[\![\mathscr{b}]\!]_X \subseteq [\![F]\!]_X \cup [\![G]\!]_X$, i.e. by Definition 4.1.3 $[\![\mathscr{b}]\!]_X \subseteq [\![F \vee G]\!]_X$. We conclude that $\vdash_X \mathscr{b} \rightarrowtail G \vee H$ is valid, as desired.
- $R2^{\rightarrowtail}_{\vee}$. The proof is equivalent to the one above.
- $R^{\leftarrowtail}_{\vee}$. The derivation is of the following form:

$$\dfrac{\begin{array}{cc}\vdots & \vdots\\ \vdash^X \mathscr{b} \leftarrowtail F & \vdash^X \mathscr{b} \leftarrowtail G\end{array}}{\vdash^X \mathscr{b} \leftarrowtail F \vee G}\, R^{\leftarrowtail}_{\vee}$$

  The premisses are valid (for IH), so $\vDash^X \mathscr{b} \leftarrowtail F$ and $\vDash^X \mathscr{b} \leftarrowtail G$, that is $[\![F]\!]_X \subseteq [\![\mathscr{b}]\!]_X$ and $[\![G]\!]_X \subseteq [\![\mathscr{b}]\!]_X$. Then, also $[\![F]\!]_X \cup [\![G]\!]_X \subseteq [\![\mathscr{b}]\!]_X$, namely $[\![F \vee G]\!]_X \subseteq [\![\mathscr{b}]\!]_X$. We conclude that $\vdash^X \mathscr{b} \leftarrowtail F \vee G$ is valid.
- The proofs for $R^{\rightarrowtail}_{\wedge}$, $R1^{\leftarrowtail}_{\wedge}$, $R2^{\leftarrowtail}_{\wedge}$ are proved in a similar way.
- $R^{\rightarrowtail}_{\mathbf{C}}$. Let $\mathsf{FN}(\mathscr{b}) \subseteq X, \mathsf{FN}(\mathscr{c}) \subseteq \{a\}$ and $\mathsf{FN}(\mathscr{b}) \cap \mathsf{FN}(\mathscr{c}) = \emptyset$. Then, the derivation is of the following form:

$$\dfrac{\begin{array}{c}\vdots\\ \vdash^{X\cup\{a\}} \mathscr{b} \wedge \mathscr{c} \rightarrowtail F \qquad \mu([\![\mathscr{c}]\!]) \geq q\end{array}}{\vdash^X \mathscr{b} \rightarrowtail \mathbf{C}^q_a F}\, R^{\rightarrowtail}_{\mathbf{C}}$$

  For IH, $\mathscr{b} \wedge \mathscr{c} \rightarrowtail F$ is $X \cup \{a\}$-valid, that is $[\![\mathscr{b} \wedge \mathscr{c}]\!]_{X\cup\{a\}} \subseteq [\![F]\!]_{X\cup\{a\}}$, i.e. $[\![\mathscr{b}]\!]_X \cap [\![\mathscr{c}]\!]_{\{a\}} \subseteq [\![F]\!]_{X\cup\{a\}}$. Furthermore, since $\mu([\![\mathscr{c}]\!]) \geq q$, also $\mu_{\mathscr{C}}\big([\![\mathscr{c}]\!]_{\{a\}}\big) \geq q$. By Definition 4.1.2, $\Pi_f([\![\mathscr{b}\wedge\mathscr{c}]\!]_{X\cup\{a\}}) = \{g \in (2^{\mathbb{N}})^{\{a\}} \mid f+g \in [\![\mathscr{b}\wedge\mathscr{c}]\!]_{X\cup\{a\}}\}$, and, given $\mu_{\mathscr{C}}\big([\![\mathscr{c}]\!]_{\{a\}}\big) \geq q$, $\mathscr{b} \subseteq \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![\mathscr{b}\wedge\mathscr{c}]\!]_{X\cup\{a\}})) \geq q\}$. Moreover, since $[\![\mathscr{b} \wedge \mathscr{c}]\!]_{X\cup\{a\}} \subseteq [\![F]\!]_X$ (and $\mathsf{FN}(\mathscr{b}) \subseteq X$, $\mathsf{FN}(\mathscr{c}) \subseteq \{a\}$, $\mathsf{FN}(\mathscr{b}) \cap \mathsf{FN}(\mathscr{c}) = \emptyset$), we conclude $\mathscr{b} \subseteq \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X\cup\{a\}})) \geq q\} \overset{D\ 4.1.3}{=} \mathbf{C}^q_a F$. Therefore $\vdash^X \mathscr{b} \rightarrowtail \mathbf{C}^q_a F$ is valid.
- $R^{\leftarrowtail}_{\mathbf{C}}$. Let $\bigvee_i \mathscr{e}_i \wedge \mathscr{d}_i$ be an a-decomposition of $\mathscr{c}$. Then, the derivation is of the following form:

$$\dfrac{\begin{array}{c}\vdots\\ \vdash^{X\cup\{a\}} \mathscr{c} \leftarrowtail F \qquad \bigvee_i\{\mathscr{e}_i \mid \mu([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q\} \vDash^X \mathscr{b}\end{array}}{\vdash^X \mathscr{b} \leftarrowtail \mathbf{C}^q_a F}\, R^{\leftarrowtail}_{\mathbf{C}}$$

For IH, $\mathscr{c} \leftharpoondown F$ is $X \cup \{a\}$-valid, that is $[\![F]\!]_{X\cup\{a\}} \subseteq [\![\mathscr{c}]\!]_{X\cup\{a\}}$. Furthermore, $\bigvee_i\{\mathscr{e}_i \mid \mu([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q\} \vDash^X \mathscr{b}$, that is $\bigcup_i\{[\![\mathscr{e}_i]\!]_X \mid \mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_X) \geq q\} \subseteq [\![\mathscr{b}]\!]_X$. By Lemma 4.2.2 (and being $\bigvee_i \mathscr{e}_i \wedge \mathscr{d}_i$ an a-decomposition of $\mathscr{c}$), $\{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![\mathscr{c}]\!]_{X\cup\{a\}})) \geq q\} \subseteq [\![\mathscr{b}]\!]_X$ and, since (for IH) $[\![F]\!]_{X\cup\{a\}} \subseteq [\![\mathscr{c}]\!]_{X\cup\{a\}}$, also $\{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X\cup\{a\}})) \geq q\} \subseteq [\![\mathscr{b}]\!]_X$. We conclude $[\![\mathbf{C}_a^q F]\!]_X \subseteq [\![\mathscr{b}]\!]_X$, that is $\mathscr{b} \leftharpoondown \mathbf{C}_a^q F$ is X-valid and, thus, $\vdash^X \mathscr{b} \leftharpoondown \mathbf{C}_a^q F$ is valid as well.

- $R^{\rightharpoonup}_{\mathbf{D}}$. Let $\bigvee_i \mathscr{e}_i \wedge \mathscr{d}_i$ be an a-decomposition of $\mathscr{c}$. Then, the derivation is of the following form:

$$\dfrac{\begin{array}{c}\vdots\\ \vdash^{X\cup\{a\}} \mathscr{c} \leftharpoondown F \qquad \mathscr{b} \vDash^X \bigvee_i\{\mathscr{e}_i \mid \mu([\![\mathscr{d}_i]\!]_{\{a\}}) < q\}\end{array}}{\vdash^X \mathscr{b} \rightharpoonup \mathbf{D}_a^q F} R^{\rightharpoonup}_{\mathbf{D}}$$

  For IH, $\mathscr{c} \leftharpoondown F$ is $X \cup \{a\}$-valid. Furthermore, $\bigvee_i\{\mathscr{e}_i \mid \mu([\![\mathscr{d}_i]\!]_{\{a\}}) < q\}$, that is $[\![\mathscr{b}]\!]_X \subseteq \bigcup_i\{[\![\mathscr{e}_i]\!]_X \mid \mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) < q\}$. By Lemma 4.2.2, $\bigcup_i\{[\![\mathscr{e}]\!]_X \mid \mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) < q\} = \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![\mathscr{c}]\!]_{X\cup\{a\}})) < q\}$. Then $[\![\mathscr{b}]\!]_X \subseteq \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![\mathscr{c}]\!]_{X\cup\{a\}})) < q\}$. Since (for IH) $[\![F]\!]_{X\cup\{a\}} \subseteq [\![\mathscr{c}]\!]_{X\cup\{a\}}$, also $[\![\mathscr{b}]\!]_X \subseteq \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X\cup\{a\}}) < q\}$. We conclude then that $[\![\mathscr{b}]\!]_X \subseteq [\![\mathbf{D}_a^q F]\!]_X$, that is $\vdash^X \mathscr{b} \rightharpoonup \mathbf{D}_a^q F$ is valid.

- $R^{\leftharpoondown}_{\mathbf{D}}$. Let $\mathsf{FN}(\mathscr{b}) \subseteq X, \mathsf{FN}(\mathscr{c}) \subseteq \{a\}$, and $\mathsf{FN}(\mathscr{b}) \cap \mathsf{FN}(\mathscr{c}) = \emptyset$. Then, the derivation is of the following form:

$$\dfrac{\begin{array}{c}\vdots\\ \vdash^{X\cup\{a\}} \neg\mathscr{b} \wedge \mathscr{c} \rightharpoonup F \qquad \mu([\![\mathscr{c}]\!]) \geq q\end{array}}{\vdash^X \mathscr{b} \leftharpoondown \mathbf{D}_a^q F} R^{\leftharpoondown}_{\mathbf{D}}$$

  For IH, $\neg\mathscr{b} \wedge \mathscr{c} \rightharpoonup F$ is $X \cup \{a\}$-valid, that is $[\![\neg\mathscr{b} \wedge \mathscr{c}]\!]_{X\cup\{a\}} \cap [\![\mathscr{c}]\!]_{X\cup\{a\}} \subseteq [\![F]\!]_{X\cup\{a\}}$. Furthermore, since $\mu(\mathscr{c}) \geq q$, $\mu_{\mathscr{C}}([\![\mathscr{c}]\!]_{\{a\}}) \geq q$. Then, $\Pi_f([\![\neg\mathscr{b} \wedge \mathscr{c}]\!]_{X\cup\{a\}}) = \{g \in (2^{\mathbb{N}})^{\{a\}} \mid f + g \in [\![\neg\mathscr{b} \wedge \mathscr{c}]\!]_X\}$ and since, for assumption, $\mu(\mathscr{c}) \geq q$ and so $\mu_{\mathscr{C}}([\![\mathscr{c}]\!]_{\{a\}}) \geq q$, $\neg\mathscr{b} \subseteq \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![\neg\mathscr{b} \wedge \mathscr{c}]\!]_{X\cup\{a\}})) \geq q\}$. Given $[\![\neg\mathscr{b} \wedge \mathscr{c}]\!]_{X\cup\{a\}} \subseteq F$, we conclude $\neg\mathscr{b} \subseteq \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X\cup\{a\}})) \geq q\}$, that is $\mathscr{b} \supseteq \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X\cup\{a\}} < q\} \overset{4.1.3}{=} \mathbf{D}_a^q F$. Thus, $\mathscr{b} \leftharpoondown \mathbf{D}_a^q F$ is X-valid.

□

Completeness. The proof of completeness for $\mathbf{LK}_{\mathbf{CPL}}$ is similar to the one for $\mathbf{LK}_{\mathbf{CPL}_0}$ as well. We start by defining the decomposition relation $\rightsquigarrow$ between sets of sequents, which is in its turn based on $\rightsquigarrow_0$.

Definition 4.2.8 (Decomposition Rewriting Reduction, $\rightsquigarrow_0$)**.** The *decomposition rewriting reduction*, $\rightsquigarrow_0$, from a sequent to a set of sequents (both in the language of $\mathbf{LK_{CPL}}$), is defined by the following decomposition rewriting rules:

$$
\begin{array}{rcl}
\text{if } \mathscr{b} \vDash^X \neg \mathscr{c}, \vdash^X \mathscr{b} \rightarrowtail \neg F & \rightsquigarrow_0 & \{\vdash^X \mathscr{c} \leftarrowtail F\} \\
\text{if } \neg \mathscr{c} \vDash^X \mathscr{b}, \vdash^X \mathscr{b} \leftarrowtail \neg F & \rightsquigarrow_0 & \{\vdash^X \mathscr{c} \rightarrowtail F\} \\
\text{if } \mathscr{b} \vDash^X \mathscr{c} \vee \mathscr{d}, \vdash^X \mathscr{b} \rightarrowtail F \vee G & \rightsquigarrow_0 & \{\vdash^X \mathscr{c} \rightarrowtail F, \vdash^X \mathscr{d} \rightarrowtail G\} \\
\vdash^X \mathscr{b} \leftarrowtail F \vee G & \rightsquigarrow_0 & \{\vdash^X \mathscr{b} \leftarrowtail F, \vdash^X \mathscr{b} \leftarrowtail G\} \\
\vdash^X \mathscr{b} \rightarrowtail F \wedge G & \rightsquigarrow_0 & \{\vdash^X \mathscr{b} \rightarrowtail F, \vdash^X \mathscr{b} \rightarrowtail G\} \\
\text{if } \mathscr{c} \wedge \mathscr{d} \vDash^X \mathscr{b}, \vdash^X \mathscr{b} \leftarrowtail F \wedge G & \rightsquigarrow_0 & \{\vdash^X \mathscr{c} \leftarrowtail F, \vdash^X \mathscr{d} \leftarrowtail G\} \\
\text{if } \mathscr{c} = \bigvee_i e_i \wedge \mathscr{d}_i \text{ is an } a\text{-decomposition of } \mathscr{b} & & \\
\text{and } \mathscr{b} \vDash^X \bigvee_i \{e_i \mid \mu([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q\}, \vdash^X \mathscr{b} \rightarrowtail \mathbf{C}^q_a F & \rightsquigarrow_0 & \{\vdash^{X \cup \{a\}} \mathscr{c} \rightarrowtail F\} \\
\text{if } \mathscr{c} = \bigvee_i e_i \wedge \mathscr{d}_i \text{ is an } a\text{-decomposition of } \mathscr{b} & & \\
\text{and } \bigvee_i \{e_i \mid \mu([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q\} \vDash^X \mathscr{b}, \vdash^X \mathscr{b} \leftarrowtail \mathbf{C}^q_a F & \rightsquigarrow_0 & \{\vdash^{X \cup \{a\}} \mathscr{c} \leftarrowtail F\} \\
\text{if } \mathscr{c} = \bigvee_i e_i \wedge \mathscr{d}_i \text{ is an } a\text{-decomposition of } \mathscr{b} & & \\
\text{and } \mathscr{b} \vDash \bigvee_i \{e_i \mid \mu([\![\mathscr{d}_i]\!]_{\{a\}}) < q\} \vdash^X \mathscr{b} \rightarrowtail \mathbf{D}^q_a F & \rightsquigarrow_0 & \{\vdash^{X \cup \{a\}} \mathscr{c} \leftarrowtail F\} \\
\text{if } \mathscr{c} = \bigvee_i e_i \wedge \mathscr{d}_i \text{ is an } a\text{-decomposition of } \mathscr{b} & & \\
\text{and } \neg \mathscr{b} \vDash^X \bigvee_i \{e_i \mid \mu([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q\}, \vdash^X \mathscr{b} \leftarrowtail \mathbf{D}^q_a F & \rightsquigarrow_0 & \{\vdash^{X \cup \{a\}} \mathscr{c} \rightarrowtail F\} \\
\text{if } \mu([\![\mathscr{b}]\!]_X) = 0, \vdash^X \mathscr{b} \rightarrowtail F & \rightsquigarrow_0 & \{\} \\
\text{if } \mu([\![\mathscr{b}]\!]_X) = 1, \vdash^X \mathscr{b} \leftarrowtail F & \rightsquigarrow_0 & \{\} \\
\text{if } \mu([\![\mathscr{b}]\!]) \neq 0, \vdash^X \mathscr{b} \rightarrowtail \mathbf{D}^0 F & \rightsquigarrow_0 & \{\vdash \bot\} \\
\text{if } \mu([\![\mathscr{b}]\!]) \neq 1, \vdash^X \mathscr{b} \leftarrowtail \mathbf{C}^0 F & \rightsquigarrow_0 & \{\vdash \bot\}.
\end{array}
$$

It is basing on $\rightsquigarrow_0$ that we define set-decomposition reduction $\rightsquigarrow$, from a set of sequents to a set of sequents.

Definition 4.2.9 (Set Decomposition, $\rightsquigarrow$)**.** The *set-decomposition reduction* $\rightsquigarrow$, from a set of sequents to another set of sequents is defined as follows:

$$
\frac{\vdash^X L_i \rightsquigarrow_0 \{\vdash^{X'_1} L_{i_q}, \ldots, \vdash^{X'_m} L_{i_m}\}}{\{\vdash^{X_1} L_1, \ldots, \vdash^{X_i} L_i, \ldots, \vdash^{X_n} L_n\} \rightsquigarrow \{\vdash^{X_1} L_1, \ldots, \vdash^{X'_1} L_{i_1}, \ldots, \vdash^{X'_m} L_{i_m}, \ldots, \vdash^{X_n} L_n\}}
$$

As for $\mathbf{LK_{CPL}}_0$, $\rightsquigarrow$ is a natural lifting of $\rightsquigarrow_0$ to a relation between sets of sequents. Again, predicates about sequents can be generalized to ones on sets, and the definitions of corresponding sets, $\rightsquigarrow_0$-normal form, and normalizations are equivalent to the ones given in Section 3.3. The notions of basic and regular sequents are also close to those for $\mathbf{LK_{CPL}}_0$.

Definition 4.2.10 (Regular Sequent)**.** A *basic formula* is a named labelled formula – i.e. either $\mathcal{b} \rightarrowtail F$ or $\mathcal{b} \leftarrowtail F$ – where F is atomic. A *regular sequent of* $\mathbf{LK}_{\mathsf{CPL}}$ is a sequent of the form $\vdash^X L$, such that L is a (named) basic formula.

All the following lemmas and proofs are analogous to those for $\mathbf{LK}_{\mathsf{CPL}_0}$.

Lemma 4.2.4. *A regular sequent is valid if and only if derivable in* $\mathbf{LK}_{\mathsf{CPL}}$.

Proof. Let $\vdash^X L$ be an arbitrary, *regular* sequent, that is let L be basic.

$\Rightarrow$ Assume that L is an X-valid formula. There are two possible cases.

Let $L = \mathcal{b} \rightarrowtail \mathbf{i}_a$ for some $i \in \mathbb{N}$. Then, $[\![\mathcal{b}]\!]_X \subseteq [\![\mathbf{i}_a]\!]_X$, that is $[\![\mathcal{b}]\!]_X \subseteq \{f \in (2^{\mathbb{N}})^X \mid f(a)(i) = 1\} = [\![x_i^a]\!]_X$, with $\mathsf{FN}(\mathbf{i}_a) \subseteq X$. Thus, $\vdash^X \mathcal{b} \rightarrowtail \mathbf{i}_a$ can be derived by means of $Ax1$ as follows:

$$\frac{a \in X \qquad \mathcal{b} \vDash^X x_i^a}{\vdash^X \mathcal{b} \rightarrowtail \mathbf{i}_a}\, Ax1$$

Let $L = \mathcal{b} \leftarrowtail \mathbf{i}_a$ for some $i \in \mathbb{N}$. Then, $[\![\mathbf{i}_a]\!]_X \subseteq [\![\mathcal{b}]\!]_X$, which is $[\![x_i^a]\!]_X = \{f \in (2^{\mathbb{N}})^X \mid f(a)(i) = 1\} \subseteq [\![\mathcal{b}]\!]_X$, with $\mathsf{FN}(\mathbf{i}_a) \subseteq X$. Thus, $\vdash^X \mathcal{b} \leftarrowtail \mathbf{i}_a$ is derivable by means of $Ax2$ as follows:

$$\frac{a \in X \qquad x_i^a \vDash^X \mathcal{b}}{\vdash^X \mathcal{b} \leftarrowtail \mathbf{i}_a}\, Ax2$$

$\Leftarrow$ By Proposition 4.2.1.

□

Lemma 4.2.5. *If a (non-empty) sequent is* $\rightsquigarrow_0$*-normal, then is regular.*

Proof. The proof is by contraposition. Assume that $\vdash^X L$ is an arbitrary, non-regular sequent, that is let L be *non*-basic. We prove that the sequent is not $\rightsquigarrow$-normal by inspecting all possible cases. Since they are all similar, we actually deal with a few cases only.

- $L = \mathcal{b} \rightarrowtail \neg F$. For every $\mathcal{b}$, $[\![\mathcal{b}]\!]_X \subseteq 2^{\mathbb{N}}$ and, since $[\![\neg\bot]\!]_X = 2^{\mathbb{N}}$, $\mathcal{b} \vDash^X \neg\bot$. Thus, the following $\rightsquigarrow_0$-decomposition is well-defined:

$$\vdash^X \mathcal{b} \rightarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash^X \bot \leftarrowtail F\}.$$

- $L = \mathcal{b} \rightarrowtail F \vee G$. For every $\mathcal{b}$, $\mathcal{b} \vDash^X \top \vee \top$, so the following $\rightsquigarrow_0$-decomposition is well-defined:

$$\vdash^X \mathcal{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash^X \top \rightarrowtail F, \vdash^X \top \rightarrowtail G\}.$$

- $L = \mathcal{b} \leftarrowtail F \vee G$. Then, the following $\rightsquigarrow_0$-decomposition is well-defined:

$$\vdash^X \mathcal{b} \leftarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash^X \mathcal{b} \leftarrowtail F, \vdash^X \mathcal{b} \leftarrowtail G\}.$$

- $L = \mathscr{b} \rightarrowtail \mathbf{C}_a^q F$. Let $\top = \bigvee \top \wedge \top$ be an a-decomposition of $\top$. Then, for every $\mathscr{b}$ and $q \in \mathbb{Q} \cap [0,1]$, $\mathscr{b} \vDash^X \bigvee\{\top \mid \mu([\![\top]\!]) \geq q\}$. The following $\rightsquigarrow_0$-decomposition is well-defined:

$$\vdash^X \mathscr{b} \rightarrowtail \mathbf{C}_a^q F \quad \rightsquigarrow_0 \quad \{\vdash^{X \cup \{a\}} \top \rightarrowtail F\}.$$

□

Corollary 4.2.1. *If a sequent is $\rightsquigarrow_0$-normal, then it is valid when derivable in* $\mathbf{LK}_{\mathsf{CPL}}$.

Proof. It is a straightforward consequence of Lemma 4.2.4 and Lemma 4.2.5. □

Lemma 4.2.6. *Reduction $\rightsquigarrow$ is strongly normalizing.*

Proof Sketch. The proof is based on the notion of sequent measure. It is shown that if $\{\vdash^{X_1} L_1, \cdots \vdash^{X_m} L_m\} \rightsquigarrow \{\vdash^{X'_1} L'_1, \ldots, \vdash^{X'_m} L'_m\}$, then

$$\mathsf{ms}\big(\{\vdash^{X_1} L_1, \ldots, \vdash^{X_m} L_m\}\big) > \mathsf{ms}\big(\{\vdash^{X'_1} L'_1, \ldots, \vdash^{X'_m} L'_m\}\big).$$

This property is established by exhaustive analysis of all possible forms of $\rightsquigarrow_0$-reduction applicable to the set, that is by dealing with all possible forms of $\rightsquigarrow_0$-reduction of $\vdash^{X_i} L_i$, with $i \in \{1, \ldots, m\}$, and where $\vdash^{X_i} L_i$ is the active sequent on which the $\rightsquigarrow_0$-reduction is based. These cases are proved in a similar way with respect to the corresponding ones in Section 3.3, so only a few examples are considered:

- $L_i = \mathscr{b} \leftarrowtail \neg F$. Assume that $\vdash^X \mathscr{b} \leftarrowtail \neg F$ is the sequent on which the considered $\rightsquigarrow_0$-reduction of $\{\vdash^{X_1} L_1, \ldots, \vdash^{X_m} L_m\}$ is based and, specifically, that the sequent is $\rightsquigarrow_0$-reduced as follows:

$$\vdash^X \mathscr{b} \leftarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash^X \mathscr{c} \rightarrowtail F\}$$

  for some $\mathscr{c}$ such that $\neg\mathscr{c} \vDash_X \mathscr{b}$. By Definition 3.3.13, $\mathsf{cn}(\mathscr{b} \leftarrowtail \neg F) = \mathsf{cn}(\mathscr{c} \rightarrowtail F) + 1$. Since the considered $\rightsquigarrow_0$-step reduces the active sequent $\vdash^X \mathscr{b} \leftarrowtail \neg F$ only, for Definition 3.3.14, we conclude

$$\mathsf{ms}\big(\{\vdash^{X_1} L_1 \ldots \vdash^X \mathscr{b} \leftarrowtail \neg F \ldots \vdash^{X_m} L_m\}\big) > \mathsf{ms}\big(\{\vdash^{X_1} L_1 \ldots \vdash^X \mathscr{c} \rightarrowtail F \ldots \vdash^{X_m} L_m\}\big).$$

- $L_i = \mathscr{b} \rightarrowtail F \vee G$. Assume that $\vdash^X \mathscr{b} \rightarrowtail F \vee G$ is the sequent on which the considered $\rightsquigarrow_0$-reduction of $\{\vdash^{X_1} L_1, \ldots, \vdash^{X_m} L_m\}$ is based and, specifically, it is $\rightsquigarrow_0$-reduced as follows:

$$\vdash^X \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash^X \mathscr{c} \rightarrowtail F, \vdash^X \mathscr{d} \rightarrowtail F\}$$

  for some $\mathscr{c}$ and $\mathscr{d}$ such that $\mathscr{b} \vDash^X \mathscr{c} \vee \mathscr{d}$. Thus, by Definition 3.3.13 and 3.3.14 we conclude:

$$\begin{aligned}
\mathsf{ms}(\{\vdash^{X_1} L_1 \ldots \vdash^X \mathscr{c} \rightarrowtail F, \vdash^X \mathscr{d} \rightarrowtail G \ldots \vdash^{X_m} L_m\}) &= \mathsf{cn}(L_1) \ldots \mathsf{cn}(\mathscr{c} \rightarrowtail F) + \mathsf{cn}(\mathscr{d} \rightarrowtail G) \ldots \mathsf{cn}(L_m) \\
&= \mathsf{cn}(L_1) \ldots + \mathsf{cn}(F) + \mathsf{cn}(G) \ldots + \mathsf{cn}(L_m) \\
&< \mathsf{cn}(L_1) \ldots + \mathsf{cn}(F) + \mathsf{cn}(G) + 1 \ldots + \mathsf{cn}(L_m) \\
&= \mathsf{cn}(L_1) \ldots + \mathsf{cn}(\mathscr{b} \rightarrowtail F \vee G) \ldots + \mathsf{cn}(L_m) \\
&= \mathsf{ms}(\{\vdash^{X_1} L_1 \ldots \vdash^X \mathscr{b} \rightarrowtail F \vee G \ldots \vdash^{X_m} L_m\}).
\end{aligned}$$

- $L_i = \mathscr{b} \leftarrowtail \mathbf{C}_a^q F$. Assume that $\vdash^X \mathscr{b} \leftarrowtail \mathbf{C}_a^q F$ is the sequent on which the considered $\rightsquigarrow_0$-reduction of $\{\vdash^{X_1} L_1, \cdots \vdash^{X_m} L_m\}$ is based and, specifically, that the sequent is $\rightsquigarrow_0$-reduced as follows:

$$\vdash^X \mathscr{b} \leftarrowtail \mathbf{C}_a^q F \quad \rightsquigarrow_0 \quad \{\vdash^{X \cup \{a\}} \mathscr{c} \leftarrowtail F\}$$

for some $\mathscr{c}$ such that $\mathscr{c} = \bigvee_i \mathscr{e}_i \wedge \mathscr{d}_i$ is an a-decomposition of $\mathscr{c}$ and $\bigvee_i \{\mathscr{e}_i \mid \mu([\![\mathscr{d}_i]\!]_{\{a\}} \geq q\} \vDash^X \mathscr{b}$. By Definition 3.3.13, $\mathsf{cn}(\mathscr{b} \leftarrowtail \mathbf{C}_a^q F) = \mathsf{cn}(\mathscr{d} \leftarrowtail F) + 1$. Thus, since the considered $\rightsquigarrow_0$-step reduces the active sequent $\vdash \mathscr{b} \leftarrowtail \mathbf{C}_a^q F$ only, for Definition 3.3.14,

$$\mathsf{ms}\big(\{\vdash^{X_1} L_1 \ldots \vdash^X \mathscr{b} \leftarrowtail \neg F \ldots \vdash^{X_m} L_m\}\big) > \mathsf{ms}\big(\{\vdash^{X_1} L_1 \ldots \vdash^X \mathscr{c} \rightarrowtail F \ldots \vdash^{X_m} L_m\}\big).$$

□

Lemma 4.2.7 (Existential Preservation of Validity for $\mathbf{LK}_{\mathsf{CPL}}$)**.** *For each sequent of* $\mathbf{LK}_{\mathsf{CPL}}$*, if it is valid, then it has a valid* $\rightsquigarrow$*-normal form.*

Proof Sketch. The proof is by exhaustive inspection of all possible cases. Given an arbitrary sequent $\Psi = \vdash^X L$, there are two main, possible cases:

Case 1. Let Ψ be such that no $\rightsquigarrow_0$-reduction can be applied on it. So, the sequent is $\rightsquigarrow_0$-normal, that is either empty or regular (Lemma 4.2.5). Therefore, it is already a $\rightsquigarrow$-normal form and, for hypothesis, it is valid.

Case 2. Let Ψ be $\rightsquigarrow_0$-reducible. By Lemma 4.2.6, $\rightsquigarrow$ is strongly normalizing and no infinite reduction sequence is possible. Consequently, it is enough to prove that each reduction step existentially preserves validity, that is for each possible $\rightsquigarrow$-reduction, based on a $\rightsquigarrow_0$-reduction, if Ψ is valid, there is a set of sequents $\{\Psi_1, \ldots, \Psi_m\}$ such that $\Psi \rightsquigarrow_0 \{\Psi_1, \ldots, \Psi_m\}$ and $\{\Psi_1, \ldots, \Psi_m\}$ is valid. The proof consists of taking into account all possible forms of $\rightsquigarrow_0$ on which the reduction step can be based. These proofs are analogous to the corresponding univariate ones We consider a few examples only:

- $L = \mathscr{b} \rightarrowtail \neg F$. We show that there is a well-defined $\rightsquigarrow_0$-reduction of Ψ such that the reduced set is valid. For hypothesis $\vdash \mathscr{b} \rightarrowtail \neg F$ is valid, that is $\mathscr{b} \rightarrowtail \neg F$ is X-valid. Let us consider a Boolean expression $\mathscr{c} = \neg \mathscr{b}$. Then,

$$[\![\neg \mathscr{c}]\!]_X = (2^{\mathbb{N}})^X - [\![\mathscr{c}]\!]_X = (2^{\mathbb{N}})^X - \big((2^{\mathbb{N}})^X - [\![\mathscr{b}]\!]_X\big) = [\![\mathscr{b}]\!]_X$$

and in particular $[\![\mathscr{b}]\!]_X \subseteq [\![\neg \mathscr{c}]\!]_X$, that is $\mathscr{b} \vDash^X \neg \mathscr{c}$. Consequently, the following $\rightsquigarrow_0$-reduction is well-defined,

$$\vdash^X \mathscr{b} \rightarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash^X \mathscr{c} \leftarrowtail F\}.$$

- $L = \mathscr{b} \leftarrowtail \neg F$. We show that there is a well-defined $\rightsquigarrow_0$-reduction of Ψ such that the reduced set is valid. For hypothesis $\vdash^X \mathscr{b} \leftarrowtail \neg F$ is valid, that is $\mathscr{b} \leftarrowtail \neg F$ is X-valid. Let us consider a Boolean expression $\mathscr{c} = \neg \mathscr{b}$. Then,

$$[\![\neg \mathscr{c}]\!]_X = (2^{\mathbb{N}})^X - [\![\mathscr{c}]\!]_X = (2^{\mathbb{N}})^X - \big((2^{\mathbb{N}})^X - [\![\mathscr{b}]\!]_X\big) = [\![\mathscr{b}]\!]_X$$

and, in particular, $[\![\neg \mathscr{c}]\!]_X \subseteq [\![\mathscr{b}]\!]_X$, that is $\neg \mathscr{c} \models^X \mathscr{b}$. Consequently, the following $\rightsquigarrow_0$-reduction is well-defined,

$$\vdash^X \mathscr{b} \hookleftarrow \neg F \quad \rightsquigarrow_0 \quad \{\vdash^X \mathscr{c} \rightarrowtail F\}.$$

Since $\mathscr{b} \hookleftarrow \neg F$ is X-valid, $[\![\neg F]\!]_X \subseteq [\![\mathscr{b}]\!]_X$. Thus, for basic set theory, $[\![\neg \mathscr{b}]\!]_X \subseteq [\![F]\!]_X$. But, by construction, $\mathscr{c} = \neg \mathscr{b}$, so $[\![\mathscr{c}]\!]_X \subseteq [\![F]\!]_X$, that is $\mathscr{c} \rightarrowtail F$ is X-valid. We conclude that $\vdash^X \mathscr{c} \rightarrowtail F$ is valid.

- $L = \mathscr{b} \rightarrowtail F \vee G$. Let us consider two $\mathscr{b}_F$ and $\mathscr{b}_G$, such that $[\![\mathscr{b}_F]\!]_X = [\![F]\!]_X$ and $[\![\mathscr{b}_G]\!]_X = [\![G]\!]_X$. For hypothesis $\mathscr{b} \rightarrowtail F \vee G$ is X-valid, that is $[\![\mathscr{b}]\!]_X \subseteq [\![F]\!]_X \cup [\![G]\!]_X$. Thus, by construction, also $[\![\mathscr{b}]\!]_X \subseteq [\![\mathscr{b}_F]\!]_X \cup [\![\mathscr{b}_G]\!]_X$, that is $\mathscr{b} \models^X \mathscr{b}_F \vee \mathscr{b}_G$. Let us consider the following reduction, which is well defined, given $\mathscr{b} \models^X \mathscr{b}_F \vee \mathscr{b}_G$:

$$\vdash^X \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash^X \mathscr{b}_F \rightarrowtail F, \vdash^X \mathscr{b}_G \rightarrowtail G\}.$$

Since $[\![\mathscr{b}_F]\!]_X = [\![F]\!]_X$ and $[\![\mathscr{b}_G]\!]_X = [\![G]\!]_X$, in particular, also $[\![\mathscr{b}_F]\!]_X \subseteq [\![F]\!]_X$ and $[\![\mathscr{b}_G]\!]_X \subseteq [\![G]\!]_X$. Therefore, $\models^X \mathscr{b}_F \rightarrowtail F$ and $\models^X \mathscr{b}_G \rightarrowtail G$ and the given $\rightsquigarrow_0$-decomposition is as desired.

- $L = \mathscr{b} \hookleftarrow F \vee G, L = \mathscr{b} \rightarrowtail F \wedge G, L = \mathscr{b} \hookleftarrow F \wedge G$. Proofs are similar to the one above.
- $L = \mathscr{b} \rightarrowtail \mathbf{C}_a^q F$. There are two main sub-cases:

  1. Let $\mu_{\mathscr{C}}([\![\mathscr{b}]\!]) = 0$. Then, the sequent can be decomposed by means of the following well-defined $\rightsquigarrow_0$-decomposition:

     $$\vdash^X \mathscr{b} \rightarrowtail \mathbf{C}_a^q F \quad \rightsquigarrow_0 \quad \{\}.$$

     $\{\}$ is vacuously valid. We conclude that the given decomposition is as desired.
  2. Let $\mu_{\mathscr{C}}([\![\mathscr{b}]\!]) \neq 0$. For hypothesis $\models^X \mathscr{b} \rightarrowtail \mathbf{C}_a^q F$, that is $[\![\mathscr{b}]\!]_X \subseteq [\![\mathbf{C}_a^q F]\!]_X = \{f \mid \mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X \cup \{a\}})) \geq q\}$. Let us consider $\mathscr{b}_F$ such that $[\![\mathscr{b}_F]\!]_X = [\![F]\!]_X$. By Lemma 4.2.1, it admits an a-decomposition $\mathscr{c} = \bigvee_i \mathscr{e}_i \wedge \mathscr{d}_i$. By the Lemma 4.2.2, $\{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_f([\![\mathscr{b}_F]\!]_{X \cup \{a\}})) \geq q\} = \bigcup_i \{[\![\mathscr{e}_i]\!]_X \mid \mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q\}$, so $[\![\mathscr{b}]\!]_X \subseteq \bigcup_i \{[\![\mathscr{e}_i]\!]_X \mid \mu_{\mathscr{C}}([\![\mathscr{d}_i]\!]_{\{a\}}) \geq q\}$. Then the following $\rightsquigarrow_0$-decomposition is well-defined:

     $$\vdash^X \mathscr{b} \rightarrowtail \mathbf{C}_a^q F \quad \rightsquigarrow_0 \quad \{\vdash^{X \cup \{a\}} \mathscr{e} \rightarrowtail F\}$$

     For construction $[\![\mathscr{e}_F]\!]_{X \cup \{a\}} = [\![\mathscr{c}]\!]_{X \cup \{a\}}$ and so $\models^{X \cup \{a\}} \mathscr{c} \rightarrowtail F$.
- $L = \mathscr{b} \hookleftarrow \mathbf{C}_a^q F, L = \mathscr{b} \rightarrowtail \mathbf{D}_a^q F, L = \mathscr{b} \hookleftarrow \mathbf{D}_a^q F$. Proofs are similar to the one above.

□

Finally, it is proved that derivability in $\mathbf{LK}_{\mathbf{CPL}}$ is preserved in the following sense.

Lemma 4.2.8 (Derivability Reflection for $\mathbf{LK}_{\mathsf{CPL}}$)**.** *Given two sets of sequents S and T, if $S \rightsquigarrow T$ and T is derivable in $\mathbf{LK}_{\mathsf{CPL}}$, then S is derivable in $\mathbf{LK}_{\mathsf{CPL}}$ as well.*

Proof. For hypothesis $S \rightsquigarrow T$, that is for some $\Psi \in S$, there is a $\rightsquigarrow_0$-decomposition

$$\Psi \quad \rightsquigarrow_0 \quad \{\Psi_1, \ldots, \Psi_m\}$$

on which the considered $\rightsquigarrow_0$ is based, so that $\Psi_1, \ldots, \Psi_m \in S$. The number of possible $\rightsquigarrow_0$ is finite, so the proof is obtained by considering all forms of reduction applicable to the set. Let us consider just a few cases.

- $L = \mathscr{b} \rightarrowtail \neg F$. Assume $\Psi \equiv \vdash^X \mathscr{b} \rightarrowtail \neg F$ and that the considered $\rightsquigarrow$ is based on the $\rightsquigarrow_0$-reduction below:

  $$\vdash^X \mathscr{b} \rightarrowtail \neg F \quad \rightsquigarrow_0 \quad \{\vdash^X \mathscr{c} \rightarrowtail F\}$$

  for some $\mathscr{c}$ such that $\mathscr{b} \vDash^X \neg\mathscr{c}$. For hypothesis T is derivable, so each of its sequents is. Thus, $\vdash^X \mathscr{c} \rightarrowtail F$ ($\in T$) is derivable. Given that $\mathscr{b} \vDash^X \neg\mathscr{c}$, it is possible to derive $\vdash^X \mathscr{b} \rightarrowtail \neg F$ by applying $R^{\rightarrowtail}_{\neg}$ as follows:

  $$\dfrac{\begin{array}{c}\vdots\\ \vdash^X \mathscr{c} \rightarrowtail F\end{array} \qquad \mathscr{b} \vDash^X \neg\mathscr{c}}{\vdash^X \mathscr{b} \rightarrowtail \neg F}\, R^{\rightarrowtail}_{\neg}$$

- $L = \mathscr{b} \rightarrowtail F \vee G$. Assume that $\Psi \equiv \vdash^X \mathscr{b} \rightarrowtail F \vee G$ and that the considered $\rightsquigarrow$-reduction is based on the $\rightsquigarrow_0$-reduction below:

  $$\vdash^X \mathscr{b} \rightarrowtail F \vee G \quad \rightsquigarrow_0 \quad \{\vdash^X \mathscr{c} \rightarrowtail F, \vdash^X \mathscr{d} \rightarrowtail G\}$$

  where $\mathscr{c}$ and $\mathscr{d}$ are two Boolean formulae such that $\mathscr{b} \vDash^X \mathscr{c} \vee \mathscr{d}$. For hypothesis T is derivable, so each of its sequents is. Therefore, both $\vdash^X \mathscr{c} \rightarrowtail F$ ($\in T$) and $\vdash^X \mathscr{d} \rightarrowtail F$ ($\in T$) are derivable and, since $\mathscr{b} \vDash^X \mathscr{c} \vee \mathscr{d}$, we conclude that also $\vdash^X \mathscr{b} \rightarrowtail F \vee G$ is derivable as follows:

  $$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash^X \mathscr{c} \rightarrowtail F\end{array}}{\vdash^X \mathscr{c} \rightarrowtail F \vee G}\, R1^{\rightarrowtail}_{\vee} \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash^X \mathscr{d} \rightarrowtail G\end{array}}{\vdash^X \mathscr{d} \rightarrowtail F \vee G}\, R2^{\rightarrowtail}_{\vee} \qquad \mathscr{b} \vDash^X \mathscr{c} \vee \mathscr{d}}{\vdash^X \mathscr{b} \rightarrowtail F \vee G}\, R^{\rightarrowtail}_{\cup}$$

- $L = \mathscr{b} \leftarrowtail \mathbf{C}^q_a F$. Assume $\Psi \equiv \vdash^X \mathscr{b} \leftarrowtail \mathbf{C}^q_a F$ and that the considered $\rightsquigarrow$-reduction is based on the $\rightsquigarrow_0$-reduction below:

  $$\vdash^X \mathscr{b} \leftarrowtail \mathbf{C}^q_a F \quad \rightsquigarrow_0 \quad \{\vdash^{X\cup\{a\}} \mathscr{c} \leftarrowtail F\}$$

  where $\mathscr{c} = \bigvee e_i \wedge d_i$ is an a-decomposition of $\mathscr{c}$ and $\bigvee_i \{e_i \mid \mu(\llbracket d_i \rrbracket_{\{a\}}) \geq q\} \vDash \mathscr{b}$. For hypothesis T is derivable, so also $\vdash^{X\cup\{a\}} \mathscr{c} \leftarrowtail F$ ($\in T$) is and $\vdash^X \mathscr{c} \leftarrowtail \mathbf{C}^q_a F$ can be shown derivable as follows:

$$\dfrac{\begin{array}{c}\vdots\\ \vdash^{X\cup\{a\}} \mathscr{c} \leftharpoondown F \end{array} \qquad \bigvee_i \{e_i \mid \mu([\![d_i]\!]_{\{a\}}) \geq q\} \vDash^X \mathscr{b}}{\vdash^X \mathscr{b} \leftharpoondown \mathbf{C}_a^q F} R^{\leftharpoondown}_{\mathbf{C}}$$

□

Proposition 4.2.2 (Completeness)**.** *If a sequent is valid, then it is derivable in* $\mathbf{LK}_{\mathsf{CPL}}$.

Proof. By combining the given Lemma 4.2.7, Corollary 4.2.1 and Lemma 4.2.8. □

4.3 Related Works

Several authors – at least from the late XIX century on – have tried to define logics to model probability. In the last decades, studies dealing with uncertain reasoning have also spread in CS, but – as Fagin, Halpern, and Megiddo noticed [81, p. 79] – these not always attempt at constructing a *logic* to reason *explicitly* about probability. Generally speaking, when moving towards this direction, the modal approach seems prevailing in the literature. Yet, remarkably, most proposals in this area are not focussed on computational aspects and, indeed, our notion of counting quantifier is mostly inspired by Wagner's operator over classes of languages [227].

One of the first works on probability logic in the context of possible-world semantics was by Nilsson [158, 159].[3] Although he did not present a formal semantics, his ideas are at the basis of many subsequent formalizations, for instance those by Bacchus and Halpern. Indeed, first probability logical *systems* were defined in the 1990s. These can be divided into two main classes: logics for *statistical probability*, which treat probability as a distribution over the structure domain, and logics expressing probability as *degrees of belief*, which models probability via sets of possible worlds. The two approaches were developed in the same years, by the same authors and can be "combined".[4] In this context, particularly relevant are Bacchus' [20, 17, 18, 19] and Fagin, Halpern and Megiddo's [81] proposals.[5] More recently, similar probability logics have been studied by Finger and others in connection with non-standard systems – for example, Lukasiewicz's logic [85].

[3] Given a sentence, say F, two sets of possible worlds are considered: one where $F = 1$, the other where $F = 0$. Given L sentences, there will be 2^L sets of worlds and the probability of each sentence is defined as the sum of probabilities of the sets of possible worlds in which it is true.

[4] Observe that Bacchus also related his logics with standard epistemic systems, like **KD45** [19].

[5] In particular, Bacchus defined *probability terms* by means of a modal operator prob which computes the probability of certain events, and *probability formulae*, which are equalities between probability terms and numbers, such as $\mathsf{prob}(\alpha) = \frac{1}{2}$. Observe that this is not too different from the intuitive meaning we associate to $\mathbf{CPL}_0$-formula $\mathbf{C}^{1/2}$ (see also the measuring procedure presented in [5]). Remarkably, Bacchus' prob yields terms, whereas $\mathbf{C}^q$ yields formulae. A similar notion was that of *weight term*, as defined in [81, 79, 80, 106, 107].

Another class of probabilistic modal logics have been designed to model Markov chains and similar structures [108, 137, 140]. Some of these logics are probabilistic extensions of **CTL**, the standard logic for model-checking. Differently from **CPL**, in these systems modal operators have a dynamic meaning, as they describe transitions in MDP. A notable example is *Riesz modal logic* [87], which admits a sound and complete proof system. Other sequent calculi expressing probabilities are developed in [26, 27] and probabilistic proof systems are also studied in [114].[6] Complete axiomatizations have been provided for both the probability logics quoted above [19, 81]. On the other hand, our calculi are mostly inspired by (non-probabilistic) labelled systems – e.g. $\mathbf{G3K}^*$ and $\mathbf{G3P}^*$ – as presented in [156, 96].

[6] We thank Raphaëlle Crubillé for pointing us to these works.

Chapter 5

On Counting Logics and Wagner's Hierarchy

In this Chapter, we investigate the profound relation existing between our counting logics and (probabilistic) complexity classes. First, in Section 5.1, we briefly recap salient aspects of Wagner's hierarchy directly referring to the original source. In Section 5.2, we consider univariate $\mathbf{CPL}_0$ and show that deciding the validity of its formulae is complete for the class $\mathbf{P}^{\sharp\mathtt{SAT}}$ of problems which can be solved in polynomial time when accessing an oracle for $\sharp\mathtt{SAT}$. Finally, in Section 5.3, we establish our main result, by relating the decision problem for formulae of **CPL** (in a special prenex form) with the corresponding level in $\mathbf{CH}$.

5.1 The Counting Hierarchy

As for $\mathbf{PH}$, there exist (at least) two main, *equivalent* presentations of the counting hierarchy: Wagner's original one [227, 226], in terms of alternating quantifiers, and Torán's oracle characterization [212, 214]. The latter one is very similar to the corresponding polynomial version, but its building blocks are obtained by replacing $\mathbf{NP}$ with $\mathbf{PP}$, where $\mathbf{PP}$ can either be defined as the class of languages recognized by a poly-time *probabilistic Turing machine* with an error probability smaller than $\frac{1}{2}$ [90, 91] or as the counting classes of languages recognized by a poly-time *threshold machine*, accepting an input when the majority of its computation paths are accepting ones [217, 191].[1]

Definition 5.1.1 (Counting Hierarchy, Oracle Characterization [212])**.** Let $k \geq$

[1] As seen in Section 2.1, probabilistic and threshold models are strongly related, as proved for example in [191, 193].

0,

$$\mathbf{CH}_0 = \mathbf{P}$$
$$\mathbf{CH}_{k+1} = \mathbf{PP}^{\mathbf{CH}_k}.$$

So, for example, $\mathbf{CH}_1 = \mathbf{PP}$ and $\mathbf{CH}_2 = \mathbf{PP}^{\mathbf{PP}}$.

Remarkably, in [227], Wagner not only introduced **CH**, but also defined canonical complete problems for each level in it.

Historical Background. The increasing interest in complexity theory and the development of new computational models, led to the discovery of problems beyond **PH**. It was in this context that in the 1980s the counting hierarchy was conceived for the first time both by Wagner [226] and by Parberry and Schnitger [164]. As said, Wagner defined it in terms of a special counting operator but, remarkably, this was not the only "probabilistic" (class) quantifier that appeared in those decades to characterize complexity classes. For example, Papdimitriou introduced a *probabilistic quantifier* to capture **PSPACE** [161], Zachos and Heller defined *random quantifiers* to characterize **BPP** [233], and (again) Zachos in [232] considered the *overwhelming* and *majority* quantifiers. Furthermore, the counting classes were shown not only interesting in themselves, but also for their relations with standard classes, as shown for instance by Toda's Theorem [210, 211].

Wagner's Characterization In 1984/86, starting from the investigation of languages for succinct representation of combinatorial problems, Wagner introduced **CH** to classify natural problems in which counting is involved [225, 226, 227]. His characterization relies on the notion of counting operator over classes of languages, which was inspired by that of threshold machine: $\mathsf{C}_{f(x)}^{p(n)} y P(y)$ expresses that there are at least $f(x)$ strings y of length $p(n)$ satisfying P. In particular, a clear presentation was offered in *The Complexity of Combinatorial Problems with Succinct Input Representation* (1986), where, in Section 3, the so-called $\mathscr{CPH}$ was introduced as an extension of **PH** obtained by adding C to standard existential and universal quantifiers.

> The counting quantifier C is defined as follows. For every formula $H(x, y)$ with the free variables x and y (which can be n-tuples),
>
> $$\mathsf{C}_y^k H(x, y) \quad \leftrightarrow \quad \text{card}\{y : H(x, y) \text{ is true}\} \geq k.$$
>
> The polynomially bounded version of the existential, universal, and counting quantifiers give rise to the operators $\bigvee, \bigwedge$ and C, resp., which are defined as follows. Let $\mathcal{C}$ be a class of languages.
>
> $A \in \bigvee \mathcal{K}$ iff there exist a $B \in \mathcal{K}$ and a polynomial p such that
>
> $$x \in A \leftrightarrow \bigvee_{y, |y| \leq p(|x|)} (x, y) \in B.$$

$A \in \bigwedge \mathcal{K}$ iff there exist a $B \in \mathcal{K}$ and a polynomial p such that

$$x \in A \leftrightarrow \bigwedge_{y,|y|\leq p(|x|)} (x,y) \in B.$$

$A \in \mathsf{C}\mathcal{K}$ iff there exist a $B \in \mathcal{K}$, a polynomial-time computable function f and a polynomial p such that

$$x \in A \leftrightarrow \mathsf{C}^{f(x)}_{y,|y|\leq p(|x|)}(x,y) \in B.$$

[227, p. 335]

Then, **CH** was defined as the smallest family of classes of languages including **P** and closed under existential, universal and counting operators [227, p. 335]. For each level in the hierarchy Wagner showed how to construct a complete set for it.

For $k \geq 1, Q_1, ..., Q_{k-1} \in \{\bigwedge, \bigvee, \mathsf{C}\}$ and $Q_k \in \{\bigvee, \mathsf{C}\}$ we define:

$$(F(\tilde{x_1}, ..., \tilde{x_k}), m_1, ..., m_k) \in Q_1, ..., Q_k B_{be} \leftrightarrow F \text{ is a Boolean expression in conjunctive normal form and } m_1, ..., m_k \in \mathbb{N} \text{ such that } \mathsf{Q}^{m_1}_{\tilde{\alpha_1}} ... \mathsf{Q}^{m_k}_{\tilde{\alpha_k}} F(\tilde{\alpha_1}, ..., \tilde{\alpha_k}) = 1,$$

$$(F(\tilde{x_1}...\tilde{x_k}), m_1...m_k) \in Q_1...Q_{k-1} \bigwedge B_{be} \leftrightarrow (F(\tilde{x_1}...\tilde{x_k}), m_1...m_k) \notin \overline{Q_1}...\overline{Q_{k-1}} \bigvee B_{be}$$

[...] Let $\leq^{log}_m$ denote the logarithic-space m-reducibility.

Theorem 7. For every $k \geq 1$ and every $Q_1, ..., Q_k \in \{\bigvee, \bigwedge, \mathsf{C}\}$,

$$Q_1...Q_k B_{be} \text{ is } \leq^{log}_m\text{-complete in } Q_1...Q_k\mathcal{P}.$$

[227, p. 338]

Torán's Characterization. Some years later, an alternative, but equivalent oracle characterization for **CH** was introduced by Torán [212, 213]. First, Torán presented a slightly modified notion of C, the so-called *exact counting quantifier*.

The polynomial counting quantifier C is defined in the following way; for a function $f : \Sigma^* \to \mathbb{N}$, $f \in \mathbf{FP}$, a polynomial p, and a two argument predicate P,

$$\mathsf{C}^p_{f(x)} y : P(x,y) \quad \Leftrightarrow \quad ||\{y : |y| \leq p(|x|) \text{ and } P(x,y)\}|| \geq f(x).$$

If $\mathcal{K}$ is a language class, for any set A, $A \in \mathsf{C}\mathcal{K}$ if there is a function $f \in \mathbf{FP}$, such that for every x, $f(x) > 0$, a polynomial p and a language $B \in \mathcal{K}$ such that for any $x \in \Sigma^*$,

$$x \in A \quad \Leftrightarrow \quad \mathsf{C}^p_{f(x)} y : \langle x, y\rangle \in B.$$

We alternate now the polynomial counting quantifier C with the existential and universal quantifiers in order to define the counting hierarchy. [...] For a function $f : \mathbb{N} \to \mathbb{N}$, $f \in \mathbf{FP}$, a polynomial p and two argument predicate P,

$$\mathsf{C}^p_{f(x)} y : P(x,y) \quad \Leftrightarrow \quad ||\{y : |y| \leq p(|x|) \text{ and } P(x,y)\}|| = f(x).$$

[212, p. 215]

Then, in [212, Sec. 4], the oracle characterization was presented. This definition is close to the oracle characterization of $\mathbf{PH}$, but PTMs are used instead of NTMs.

Theorem 15. For any class $\mathcal{K}$ in $\mathbf{CH}$,

i. $\mathbf{PP}^{\mathcal{K}} = \mathsf{C}\mathcal{K}$
ii. $\mathbf{NP}^{\mathcal{K}} = \exists\mathcal{K}$. [212, p. 219]

Torán also proved this characterization equivalent to Wagner's one.[2] These new approaches to counting classes are clarified and summarised a few years later, for example in Allender and Wagner's *Counting Hierarchies: Polynomial Time and Constant Depth Circuits* (1993). In this paper also combinations with other probabilistic operators – e.g. those by [163] – were considered [2, pp. 469-470].

5.2 On $\mathbf{CPL}_0$ and $\mathbf{P}^{\sharp\mathtt{SAT}}$

We have seen that quantified formulae of $\mathbf{CPL}_0$ can be proved valid by invoking an *oracle*, which provides a suitable measurement $\mu_{\mathscr{C}}(\llbracket \mathscr{b} \rrbracket)$ for any Boolean formula $\mathscr{b}$. This corresponds to actually *counting* the number of valuations satisfying the corresponding formula.[3] We now make this intuition more precise by showing that the validity of a $\mathbf{CPL}_0$-formula can be decided by a *poly-time* algorithm accessing an oracle for the problem $\sharp\mathtt{SAT}$, i.e. an oracle counting the models of a Boolean formula. To do so, we start by introducing the notion of closed formula.

Definition 5.2.1 (Closed Formula)**.** A formula of $\mathbf{CPL}_0$, say F, is said to be *closed* if it is either of the form $\mathbf{Q}^q F$ with $\mathbf{Q} \in \{\mathbf{C}, \mathbf{D}\}$, or a negation, conjunction or disjunction of closed formulae.

It can be easily checked by induction on the structure of closed formulae that for any closed F, either $\llbracket F \rrbracket = 2^{\mathbb{N}}$ or $\llbracket F \rrbracket = \emptyset$. Then, we define by mutual recursion, two poly-time algorithms $\mathtt{Bool}$ and $\mathtt{Var}$ as illustrated in Figure 5.1. For each counting formula F, $\mathtt{Bool}(F)$ computes a Boolean fromula $\mathscr{b}_F$, such that $\llbracket F \rrbracket = \llbracket \mathscr{b}_F \rrbracket$ and, for any closed formula F,

$$\mathtt{Var}(F) = \begin{cases} \top & \text{if } \llbracket F \rrbracket = 2^{\mathbb{N}} \\ \bot & \text{if } \llbracket F \rrbracket = \emptyset. \end{cases}$$

Notice that the algorithm $\mathtt{Var}$ is defined relying on a $\sharp\mathtt{SAT}$ oracle. So both $\mathtt{Bool}$ and $\mathtt{Var}$ belong to $\mathbf{P}^{\sharp\mathtt{SAT}}$, where – as said – $\mathbf{P}^{\sharp\mathtt{SAT}}$ is the class made of problems which can be decided in polynomial time having access to an oracle for $\sharp\mathtt{SAT}$ [14].

Proposition 5.2.1. *Validity of $\mathbf{CPL}_0$-formulae is in $\mathbf{P}^{\sharp\mathtt{SAT}}$.*

[2] Furthermore, in [212, Lemma 3, p. 215], Torán proved $\mathsf{C}\mathbf{P} = \mathbf{PP}$. So, basically, $\mathbf{CH}_0 = \mathsf{C}\mathbf{P} = \mathbf{PP}$ and $\mathbf{CH}_k = \mathbf{PP}^{\mathbf{CH}_{k-1}}$.

[3] See also Lemma 3.3.1.

$$
\begin{aligned}
\texttt{Bool}(n) &= x_n \\
\texttt{Bool}(\neg F) &= \neg\texttt{Bool}(F) \\
\texttt{Bool}(F \wedge G) &= \texttt{Bool}(F) \wedge \texttt{Bool}(G) \\
\texttt{Bool}(F \vee G) &= \texttt{Bool}(F) \vee \texttt{Bool}(G) \\
\texttt{Bool}(\mathbf{C}^q F) &= \texttt{Var}(\mathbf{C}^q F) \\
\texttt{Bool}(\mathbf{D}^q F) &= \texttt{Var}(\mathbf{D}^q F)
\end{aligned}
\qquad
\begin{aligned}
\texttt{Var}(\neg F) &= \texttt{not Var}(F) \\
\texttt{Var}(F \wedge G) &= \texttt{Var}(F)\ \texttt{and}\ \texttt{Var}(F) \\
\texttt{Var}(F \vee G) &= \texttt{Var}(F)\ \texttt{or}\ \texttt{Var}(G) \\
\texttt{Var}(\mathbf{C}^q F) &= \texttt{let}\ \mathscr{b} = \texttt{Bool}(F)\ \texttt{in} \\
&\quad \texttt{let}\ n = \sharp\texttt{Var}(\mathscr{b})\ \texttt{in} \\
&\quad \frac{\sharp\texttt{SAT}(\mathscr{b})}{2^{\mathbb{N}}} \geq q \\
\texttt{Var}(\mathbf{D}^q F) &= \texttt{let}\ \mathscr{b} = \texttt{Bool}(F)\ \texttt{in} \\
&\quad \texttt{let}\ n = \sharp\texttt{Var}(\mathscr{b})\ \texttt{in} \\
&\quad \frac{\sharp\texttt{SAT}(\mathscr{b})}{2^{\mathbb{N}}} < q
\end{aligned}
$$

where $\sharp\texttt{SAT}(\mathscr{b})$ denote the number of propositional variables in the Boolean formula $\mathscr{b}$.

Figure 5.1: $\texttt{Bool}(\cdot)$ and $\texttt{Var}(\cdot)$

5.3 On CPL and Wagner's Hierarchy

Clearly, counting problems are not restricted to those in $\mathbf{P}^{\sharp\texttt{SAT}}$. In this section we provide a *logical* characterization of Wagner's hierarchy, showing that the validity of multivariate counting formulae yields a new family of complete problems for the corresponding level of $\mathbf{CH}$.

5.3.1 Towards a Logical Characterization of the Hierarchy

The introduction of *named* atoms and *named* quantifiers makes it possible to relate valuations of *different* groups of variables. For instance, the intuitive meaning of the quantified formula $\mathbf{C}_a^q F$ is that "F is true in at least $q \cdot n$ of the n valuations of variables labelled with name a". As seen, in this way, a problem in MAJMAJSAT can be captured by formulae in the form $\mathbf{C}_a^q \mathbf{C}_b^p F$, as shown by the example of Chapter 4 restated below.

Example 5.3.1. Given the formula of **PL** $F = (X_1 \vee Y_1) \wedge (X_2 \vee Y_2)$, containing two disjoint sets of variables, $\mathtt{X} = \{X_1, X_2\}$ and $\mathtt{Y} = \{Y_1, Y_2\}$, we can express its MAJMAJSAT problem [24, 219, 220], by the **CPL**-formula below,

$$\mathbf{C}_a^{1/2} \mathbf{C}_b^{1/2} \big((\mathbf{1}_a \vee \mathbf{1}_b) \wedge (\mathbf{2}_a \vee \mathbf{2}_b) \big).$$

It is precisely by making this intuition formal that we provide a characterization of the full $\mathbf{CH}$ via **CPL**. In particular, we show that: (i.) any counting formula is equivalent to one in the special prenex normal form $\mathbf{C}_{a_1}^{q_1} \ldots \mathbf{C}_{a_k}^{q_k} F$, where F is quantifier-free and (ii.) prenex formulae with k nested quantifiers characterize the level k of Wagner's $\mathbf{CH}$.

Specifically, our logical characterization is basically obtained by *internalizing* Wagner's constructions inside a proper logical system, namely **CPL**. Our starting point is Wagner's Theorem 7, which provides a complete problem for each level in $\mathbf{CH}$ [227]. We notice that there is a mismatch between the form of Wagner's problems and that of our counting formulae. Indeed, on the one hand, counting quantifiers can appear deep-inside formulae, rather than just at top-level and, on the other, the quantifier $\mathbf{D}$ has no counterpart in Wagner's constructions. So, in order to obtain the desired characterization we pass through a *specific* prenex form, called *positive prenex normal form* (PPNF, for short). A formula is in PPNF if it is both in PNF and $\mathbf{D}$-free. In Section 5.3.2 and Section 5.3.3, we prove that any formula of **CPL** can be converted into this special PPNF. Finally, in Section 5.3.4, we show that, as desired, this form perfectly matches that of Wagner's problems and, so, **CPL** can be seen as the *counting* or *quantitative* counterpart of **QPL**. Observe that in all the following lemmas and proofs, formulae of **CPL** are considered modulo α-conversion.

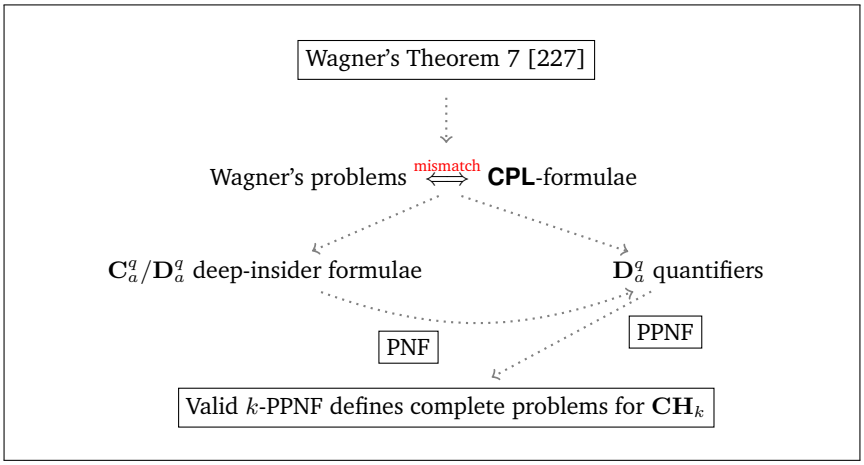


Figure 5.2: Proof Schema

5.3.2 Prenex Normal Form

First, we show that any formula of **CPL** can be converted into prenex normal form.[4]

Definition 5.3.1 (Prenex Normal Form)**.** A formula of **CPL** is a(n *n-ary) prenex normal form* (PNF, for short) if it can be written as:

$$\mathbf{Q}_1 \dots \mathbf{Q}_n F,$$

[4]I thank Raphaëlle Crubillé for suggesting me to add several details and proofs to this Section.

where $\mathbf{Q}_i \in \{\mathbf{C}^q_a, \mathbf{D}^q_a\}$, for any $a \in \mathscr{A}$, $q \in \mathbb{Q} \cap [0,1]$ and $i \in \{1, \ldots, n\}$, and F is quantifier-free. The formula F is said to be the *matrix* of the PNF.

To convert a formula of **CPL** into an equivalent one in PNF, some intermediate lemmas are needed. Preliminarily notice that as for **QPL**, conversion into CNF of (the matrix of) counting formulae can have high complexity.

Converting Conjunction and Disjunction. Without loss of generality, we assume $a \notin \mathsf{FN}(F)$. First, let us show that counting quantifiers occurring inside conjunction and disjunction can be extruded from them. Preliminarily notice that the following results hold.

Lemma 5.3.1. *For any X such that $\mathsf{FN}(F) \subseteq X \cup \{a\}$ and $a \notin X$,*

$$[\![\mathbf{C}^0_a F]\!]_X = (2^{\mathbb{N}})^X \qquad\qquad [\![\mathbf{D}^0_a F]\!]_X = \emptyset^X.$$

Proof. Let us consider $[\![\mathbf{C}^0_a F]\!]_X = (2^{\mathbb{N}})^X$. Since by Definition 4.1.2, for every $\Pi_f([\![F]\!]_{X\cup\{a\}}) \subseteq (2^{\mathbb{N}})^{\{a\}}$, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) \geq 0$ holds. So, trivially,

$$\begin{aligned}[\![\mathbf{C}^0_a F]\!]_X &= \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) \geq 0\} \\ &= (2^{\mathbb{N}})^X.\end{aligned}$$

Let us consider $[\![\mathbf{D}^0_a F]\!]_X = \emptyset^X$. Again by Definition 4.1.2, for no $\Pi_f([\![F]\!]_{X\cup\{a\}})$, $\mu\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) < 0$ holds. So, trivially,

$$\begin{aligned}[\![\mathbf{D}^0_a F]\!]_X &= \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) < 0\} \\ &= \emptyset^X.\end{aligned}$$

□

Corollary 5.3.1. *For any X such that $\mathsf{FN}(F) \cup \mathsf{FN}(G) \subseteq X \cup \{a\}$ and $a \notin X$:*

$$\begin{aligned} F \wedge \mathbf{C}^0_a G &\equiv_X F & \qquad F \vee \mathbf{C}^0_a G &\equiv_X (2^{\mathbb{N}})^X \\ F \wedge \mathbf{D}^0_a G &\equiv_X \emptyset^X & \qquad F \vee \mathbf{D}^0_a G &\equiv_X F. \end{aligned}$$

Proof. Let us consider $F \wedge \mathbf{C}^0_a G \equiv_X F$, that is:

$$[\![F \wedge \mathbf{C}^0_a G]\!]_X = [\![F]\!]_X.$$

By Definition 4.1.3,

$$\begin{aligned}[\![F \wedge \mathbf{C}^0_a G]\!]_X &= [\![F]\!]_X \cap [\![\mathbf{C}^0_a G]\!]_X \\ &\overset{L.5.3.1}{=} [\![F]\!]_X \cap (2^{\mathbb{N}})^X \\ &= [\![F]\!]_X.\end{aligned}$$

Let us now consider $F \vee \mathbf{C}^0_a G \equiv_X (2^{\mathbb{N}})^X$, that is:

$$[\![F \vee \mathbf{C}^0_a G]\!]_X = (2^{\mathbb{N}})^X.$$

Again, by Definition 4.1.3,

$$\begin{aligned}\llbracket F \vee \mathbf{C}_a^0 G \rrbracket_X &= \llbracket F \rrbracket_X \cup \llbracket \mathbf{C}_a^0 G \rrbracket_X \\ &\overset{L.5.3.1}{=} \llbracket F \rrbracket_X \cup (2^{\mathbb{N}})^X \\ &= (2^{\mathbb{N}})^X.\end{aligned}$$

Let us consider $F \wedge \mathbf{D}_a^0 G \equiv_X \emptyset^X$, that is:

$$\llbracket F \wedge \mathbf{D}_a^0 G \rrbracket_X = \emptyset^X.$$

By Definition 4.1.3,

$$\begin{aligned}\llbracket F \wedge \mathbf{D}_a^0 G \rrbracket_X &= \llbracket F \rrbracket_X \cap \llbracket \mathbf{D}_a^0 G \rrbracket_X \\ &\overset{L\ 5.3.1}{=} \llbracket F \rrbracket_X \cap \emptyset^X \\ &= \emptyset^X.\end{aligned}$$

Let us consider $F \vee \mathbf{D}_a^0 G \equiv_X F$, that is:

$$\llbracket F \vee \mathbf{D}_a^0 G \rrbracket_X = \llbracket F \rrbracket_X.$$

By Definition 4.1.3:

$$\begin{aligned}\llbracket F \vee \mathbf{D}_a^0 G \rrbracket_X &= \llbracket F \rrbracket_X \cup \llbracket \mathbf{D}_a^0 G \rrbracket_X \\ &\overset{L\ 5.3.1}{=} \llbracket F \rrbracket_X \cup \emptyset^X \\ &= \llbracket F \rrbracket_X.\end{aligned}$$

□

Let X, Y be two disjoint sets of names. For any $\mathcal{X} \subseteq (2^{\mathbb{N}})^X$, the set $X^{\Uparrow Y} \subseteq (2^{\mathbb{N}})^{X \cup Y}$ is defiend by

$$\mathcal{X}^{\Uparrow Y} = \{f \in (2^{\mathbb{N}})^{X \cup Y} \mid f_X \in \mathcal{X}\}$$

where $f_X \in (2^{\mathbb{N}})^X$ denotes the function obtained from f by restricting its domain to X. We can formally define the *extension of $\mathcal{X}$ to Y* as follows.

Definition 5.3.2 (Extension of $\mathcal{X}$ of Y)**.** Given $\mathcal{X} \subseteq (2^{\mathbb{N}})^X$ and Y such that $X \cap Y = \emptyset$, we use $\mathcal{X}^{\Uparrow Y} \subseteq (2^{\mathbb{N}})^{X \cup Y}$ to denote the *extension of $\mathcal{X}$ to Y*. Since we assume $X \cap Y = \emptyset$, given $\mathcal{X} \subseteq (2^{\mathbb{N}})^X$, the *extension of $\mathcal{X}$ to Y* is

$$\mathcal{X}^{\Uparrow Y} := \{f + g \in (2^{\mathbb{N}})^{X \cup Y} \mid f \in \mathcal{X} \ \&\ g \in (2^{\mathbb{N}})^Y\},$$

where

$$(f+g)(a) = \begin{cases} f(a) & \text{if } a \in X \\ g(a) & \text{if } a \in Y. \end{cases}$$

Then, even the following auxiliary lemmas are proved:

Lemma 5.3.2. *Given $\mathcal{Z} \subseteq (2^{\mathbb{N}})^{X \cup Y}$ and $f \in \mathcal{X} \subseteq (2^{\mathbb{N}})^X$, then*

$$\Pi_f(\mathcal{Z}) = \Pi_f\big(\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y}\big).$$

Proof. The proof is by cases:

- $\subseteq$ If $g \in \Pi_f(\mathcal{Z})$, as by Definition 4.1.2, $\Pi_f(\mathcal{Z}) = \{g' \in (2^{\mathbb{N}})^Y \mid f + g' \in \mathcal{Z}\} \subseteq (2^{\mathbb{N}})^Y$, $f + g \in \mathcal{Z}$. Since $f \in \mathcal{X}$, by Definition of $\mathcal{X}^{\Uparrow Y} (= \{f + h \in (2^{\mathbb{N}})^{X \cup Y} \mid f \in \mathcal{X}\})$, also $f + g \in \mathcal{X}^{\Uparrow Y}$. As a consequence, $f + g \in \mathcal{Z} \cap \mathcal{X}^{\Uparrow Y}$ and, because again by Definition 4.1.2, $\Pi_f(\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y}) = \{g \in (2^{\mathbb{N}})^X \mid f + g \in \mathcal{Z} \cap \mathcal{X}^{\Uparrow}\}$, we conclude that $g \in \Pi_f\big(\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y}\big)$.
- $\supseteq$ Trivial since the projection operator is monotone.

□

Lemma 5.3.3. *Let $f \in (2^{\mathbb{N}})^X$, $\mathcal{Z} \subseteq (2^{\mathbb{N}})^{X \cup Y}$ and $\mathcal{X} \subseteq (2^{\mathbb{N}})^X$ with $X \cap Y = \emptyset$, if $\mu_{\mathscr{C}}\big(\Pi_f(\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y})\big) > 0$, then $f \in \mathcal{X}$.*

Proof. The proof is by contraposition. Assume $f \notin \mathcal{X}$. Then, there are two possible cases:

- If $\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y} = \emptyset$, then

$$\begin{aligned}\mu_{\mathscr{C}}\big(\Pi_f(\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y})\big) &= \mu_{\mathscr{C}}(\Pi_f(\emptyset)) \\ &= \mu_{\mathscr{C}}(\emptyset) \\ &= 0.\end{aligned}$$

- Otherwise, by Definition 4.1.2, $\Pi_f(\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y}) = \{g \in (2^{\mathbb{N}})^Y \mid f + g \in (\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y})\}$. Furthermore, by Definition of Y-extension of $\mathcal{X}$, $\mathcal{X}^{\Uparrow Y} = \{h + h' \in (2^{\mathbb{N}})^X \cup Y \mid h \in \mathcal{X}\}$ and, since ($X \cap Y = \emptyset$ and) by assumption $f \notin \mathcal{X}$, $\{g \in (2^{\mathbb{N}})^Y \mid f + g \in \mathcal{X}^{\Uparrow Y}\} = \emptyset$. Then, trivially also $\{g \in (2^{\mathbb{N}})^Y \mid f + g \in \mathcal{Z} \cap \mathcal{X}^{\Uparrow Y}\} = \emptyset$. Therefore,

$$\begin{aligned}\mu_{\mathscr{C}}\big(\Pi_f(\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y})\big) &= \mu_{\mathscr{C}}\big(\{g \in (2^{\mathbb{N}})^Y \mid f + g \in (\mathcal{Z} \cap \mathcal{X}^{\Uparrow Y})\}\big) \\ &\overset{f \notin \mathcal{X}}{=} \mu_{\mathscr{C}}(\emptyset) \\ &= 0.\end{aligned}$$

□

It is now possible to prove that, for $a \notin X$, $\mathsf{FN}(F) \subseteq X$ and $f \in (2^{\mathbb{N}})^X$, if $f \in [\![F]\!]_X$, then $\Pi_f\big([\![F]\!]_{X \cup \{a\}}\big) = (2^{\mathbb{N}})^{\{a\}}$. Intuitively, since $a \notin \mathsf{FN}(F)$, $[\![F]\!]_{X \cup \{a\}}$ is a set of $f \in (2^{\mathbb{N}})^{X \cup \{a\}}$, the characterization of which *does not* rely on $\{a\}$. So, for each $\alpha \in Y$, either $f(\alpha) \in [\![F]\!]_X$, and $f + g \in [\![F]\!]_{X \cup \{a\}}$, or $f(\alpha) \notin [\![F]\!]_X$, and $f + g \notin [\![F]\!]_{X \cup \{a\}}$. Therefore, if $f \in [\![F]\!]_X$, $\Pi_f\big([\![F]\!]_{X \cup \{a\}}\big) = (2^{\mathbb{N}})^{\{a\}}$.

The following auxiliary Proposition is proved by simply applying Definitions 4.1.2 and 5.3.2.

Proposition 5.3.1. *Let X, Y be finite disjoint sets, $\mathcal{Z} \subseteq (2^{\mathbb{N}})^{X \cup Y}$, $f \in (2^{\mathbb{N}})^X$ and $g \in (2^{\mathbb{N}})^Y$, then*

$$\Pi_{f+g}(\mathcal{Z}^{\Uparrow Y}) = \Pi_f(\mathcal{Z}).$$

Lemma 5.3.4. *For any **CPL**-formula F, $f \in (2^{\mathbb{N}})^X, g \in (2^{\mathbb{N}})^{\{a\}}, a \notin X$, and $\mathsf{FN}(F) \subseteq X$:*

$$f \in [\![F]\!]_X \quad \textit{iff} \quad f + g \in [\![F]\!]_{X \cup \{a\}}$$

Proof. The proof is by induction on the structure of F:

- $F = \mathbf{i}_b$. Then,
  - $\Rightarrow$ Assume $f \in [\![\mathbf{i}_b]\!]_X$, with $\{b\} \subseteq X$. By Definition 4.1.3, $[\![\mathbf{i}_b]\!]_X = \{h \in (2^{\mathbb{N}})^X |\ h(b)(i) = 1\}$, with $a \notin X$, and for assumption $f \in [\![\mathbf{i}_b]\!]_X$, i.e. $f(b)(i) = 1$. Again by Definition 4.1.3, $[\![\mathbf{i}_b]\!]_{X \cup \{a\}} = \{h' \in (2^{\mathbb{N}})^{X \cup \{a\}} \mid h'(b)(i) = 1\}$, i.e. $\{f' + g' \in (2^{\mathbb{N}})^{X \cup \{a\}} \mid f' \in (2^{\mathbb{N}})^X \ \&\ g' \in (2^{\mathbb{N}})^{\{a\}} \ \&\ f'(b)(i) = 1\} = \{f' + g' \in (2^{\mathbb{N}})^{X \cup \{a\}} \mid f' \in [\![\mathbf{i}_b]\!]_X \ \&\ g' \in 2^{\mathbb{N}}\}$. So, we conclude $(f + g) \in [\![\mathbf{i}_b]\!]_{X \cup \{a\}}$, as desired.
  - $\Leftarrow$ The proof is by contraposition. Assume $f \notin [\![\mathbf{i}_b]\!]_X$, with $\{b\} \subseteq X$. As seen, by Definition 4.1.3, $f(b)(i) = 0$ and since (for $a \notin X$) $[\![\mathbf{i}_b]\!]_{X \cup \{a\}} = \{f' + g' \in (2^{\mathbb{N}})^{X \cup \{a\}} \mid f' \in (2^{\mathbb{N}})^X \ \&\ g' \in (2^{\mathbb{N}})^{\{a\}} \ \&\ f'(b)(i) = 1\}$, we conclude $f + g \notin [\![\mathbf{i}_b]\!]_{X \cup \{a\}}$.
- $F = \neg G$. Then,
  - $\Rightarrow$ For assumption $f \in [\![\neg G]\!]_X$, with $\mathsf{FN}(\neg G) \subseteq X$, i.e. by Definition 4.1.3, $f \in (2^{\mathbb{N}})^X - [\![G]\!]_X$ and, so $f \notin [\![G]\!]_X$. Then, by IH, $f + g \notin [\![G]\!]_{X \cup \{a\}}$, with $a \notin X$, and so, again by Definition 4.1.3, $f + g \in [\![\neg G]\!]_{X \cup \{a\}}$ as desired.
  - $\Leftarrow$ The proof is by contraposition. Assume $f \notin [\![\neg G]\!]_X$, with $\mathsf{FN}(\neg G) \subseteq X$. Then, by Definition 4.1.3, $f \in [\![G]\!]_X$ and, so, by IH, $f + g \in [\![G]\!]_{X \cup \{a\}}$, with $a \notin X$. So also $f + g \notin (2^{\mathbb{N}})^{X \cup \{a\}} - [\![G]\!]_{X \cup \{a\}}$. By Definition 4.1.3, we conclude $f + g \notin [\![\neg G]\!]_{X \cup \{a\}}$.
- $F = G_1 \wedge G_2$. Then,
  - $\Rightarrow$ For assumption $f \in [\![G_1 \wedge G_2]\!]_X$, with $\mathsf{FN}(G_1 \wedge G_2) \subseteq X$, i.e. by Definition 4.1.3, $f \in [\![G_1]\!]_X \cap [\![G_2]\!]_X$. Then, $f \in [\![G_1]\!]_X$ and $f \in [\![G_2]\!]_X$. By IH, $f + g \in [\![G_1]\!]_{X \cup \{a\}}$ and $f + g \in [\![G_2]\!]_{X \cup \{a\}}$, with $a \notin X$, so, by basic set theory, also $f + g \in [\![G_1]\!]_{X \cup \{a\}} \cap [\![G_2]\!]_{X \cup \{a\}}$. Again by Definition 4.1.3, we conclude $f + g \in [\![G_1 \wedge G_2]\!]_{X \cup \{a\}}$, as desired.
  - $\Leftarrow$ Assume $f + g \in [\![G_1 \wedge G_2]\!]_{X \cup \{a\}}$, with $\mathsf{FN}(G_1 \wedge G_2) \subseteq X$ and $a \notin X$. Then, by Definition 4.1.3, $f + g \in [\![G_1]\!]_{X \cup \{a\}} \cap [\![G_2]\!]_{X \cup \{a\}}$, and so for basic set theory $f + g \in [\![G_1]\!]_{X \cup \{a\}}$ and $f + g \in [\![G_2]\!]_{X \cup \{a\}}$. By IH, also $f \in [\![G_1]\!]_X$ and $f \in [\![G_2]\!]_X$. Then, $f \in [\![G_1]\!]_X \cap [\![G_2]\!]_X$ and, by Definition 4.1.3, we conclude $f \in [\![G_1 \wedge G_2]\!]_X$.

- $F = G_1 \vee G_2$. Then,

  $\Rightarrow$ The proof is by contraposition. Assume $f+g \notin [\![G_1 \vee G_2]\!]_{X\cup\{a\}}$, with $\mathsf{FN}(G_1 \vee G_2) \subseteq X$ and $a \notin X$. Then, $f+g \in (2^{\mathbb{N}})^{X\cup\{a\}} - [\![G_1 \vee G_2]\!]_{X\cup\{a\}}$, that is, $f+g \in \big((2^{\mathbb{N}})^{X\cup\{a\}} - [\![\neg G_1]\!]_{X\cup\{a\}}\big) \cap \big((2^{\mathbb{N}})^{X\cup\{a\}} - [\![G_2]\!]_{X\cup\{a\}}\big)$. So, $f+g \in (2^{\mathbb{N}})^{X\cup\{a\}} - [\![G_1]\!]_{X\cup\{a\}}$ and $f+g \in (2^{\mathbb{N}})^{X\cup\{a\}} - [\![G_2]\!]_{X\cup\{a\}}$, which are equivalent to $f+g \notin [\![G_1]\!]_{X\cup\{a\}}$ and $f+g \notin [\![G_2]\!]_{X\cup\{a\}}$. By IH, $f \notin [\![G_1]\!]_X$ and $f \notin [\![G_2]\!]_X$, i.e. $f \in (2^{\mathbb{N}})^X - [\![G_1]\!]_X$ and $f \in (2^{\mathbb{N}})^X - [\![G_2]\!]_X$. So, $f \in ((2^{\mathbb{N}})^X - [\![G_1]\!]_X) \cap ((2^{\mathbb{N}})^X - [\![G_2]\!]_X)$, and, for basic set theory, also $f \in [\![G_1]\!]_X \cup [\![G_2]\!]$. By Definition 4.1.3, $f \in (2^{\mathbb{N}})^X - [\![G_1 \vee G_2]\!]_X$, so we conclude $f \notin [\![G_1 \vee G_2]\!]$, as desired.

  $\Leftarrow$ The proof is by contraposition. Assume $f \notin [\![G_1 \vee G_2]\!]_X$, with $\mathsf{FN}(G_1 \vee G_2) \subseteq X$. Then, $f \in (2^{\mathbb{N}})^X - ([\![G_1 \vee G_2]\!]_X)$ and, by Definition 4.1.3, $f \in \big((2^{\mathbb{N}})^X - [\![G_1]\!]_X\big) \cap \big((2^{\mathbb{N}})^X - [\![G_2]\!]_X\big)$, i.e. $f \in \big((2^{\mathbb{N}})^X - [\![G_1]\!]_X\big)$ and $f \in \big((2^{\mathbb{N}})^X - [\![G_2]\!]_X\big)$, implying $f \notin [\![G_1]\!]_X$ and $f \notin [\![G_2]\!]$. By IH, $f+g \notin [\![G_1]\!]_{X\cup\{a\}}$ and $f+g \notin [\![G_2]\!]_{X\cup\{a\}}$, with $a \notin X$, i.e. $f+g \in (2^{\mathbb{N}})^{X\cup\{a\}} - [\![G_1]\!]_{X\cup\{a\}}$ and $f+g \in (2^{\mathbb{N}})^{X\cup\{a\}} - [\![G_2]\!]_{X\cup\{a\}}$. So, $f+g \in \big((2^{\mathbb{N}})^{X\cup\{a\}} - [\![G_1]\!]_{X\cup\{a\}}\big) \cap \big((2^{\mathbb{N}})^{X\cup\{a\}} - [\![G_2]\!]_{X\cup\{a\}}\big) = (2^{\mathbb{N}})^{X\cup\{a\}} - ([\![G_1]\!]_{X\cup\{a\}} \cup [\![G_2]\!]_{X\cup\{a\}})$. Then, $f+g \notin [\![G_1]\!]_{X\cup\{a\}} \cup [\![G_2]\!]_{X\cup\{a\}}$ and, by Definition 4.1.3, we conclude $f+g \notin [\![G_1 \vee G_2]\!]_{X\cup\{a\}}$.

- $F = \mathbf{C}_b^q G$. Without loss of generality assume $a \neq b$. Then,

  $\Rightarrow$ Assume $f \in [\![\mathbf{C}_b^q G]\!]_X$, for $\mathsf{FN}(\mathbf{C}_b^q G) \subseteq X$. By Definition 4.1.3, $[\![\mathbf{C}_b^q G]\!]_X = \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_h([\![G]\!]_{X\cup\{b\}})) \geq q\}$ and, by IH, $f \in [\![G]\!]_{X\cup\{b\}}$ when $f+g \in [\![G]\!]_{X\cup\{b\}\cup\{a\}}$. Then, since for hypothesis $f \in (2^{\mathbb{N}})^X$, $g \in (2^{\mathbb{N}})^{\{a\}}$ and $a \notin \mathsf{FN}(\mathbf{C}_b^q G)$, $\Pi_{f+g}([\![G]\!]_{X\cup\{a\}\cup\{b\}}) = \Pi_{f+g}([\![G]\!]^{\Uparrow\{a\}}_{X\cup\{b\}}) = \Pi_f([\![G]\!]_{X\cup\{b\}})$, by Proposition 5.3.1 and Definition 4.1.2. So, we conclude that also $\mu_{\mathscr{C}}(\Pi_{f+g}\big([\![G]\!]_{X\cup\{a\}\cup\{b\}}\big)) \geq q$ and, then, also $f+g \in [\![\mathbf{C}_b^q G]\!]_{X\cup\{a\}}$.

  $\Leftarrow$ The proof is by contraposition. Assume $f \notin [\![\mathbf{C}_b^q G]\!]_X$. Then, by Definition 4.1.3, $[\![\mathbf{C}_b^q G]\!]_X = \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_h([\![G]\!]_{X\cup\{b\}})) < q\}$. Again, by IH, $f \in [\![G]\!]_{X\cup\{b\}}$ when $f+g \in [\![G]\!]_{X\cup\{b\}\cup\{a\}}$. Also in this case, by Proposition 5.3.1 and Definition 4.1.2, $\Pi_{f+g}([\![G]\!]_{X\cup\{a\}\cup\{b\}}) = \Pi_f([\![G]\!]_{X\cup\{b\}})$. Thus, we conclude that $\mu_{\mathscr{C}}(\Pi_{f+g}([\![G]\!]_{X\cup\{a\}\cup\{b\}})) \geq q$ and, so, $f+g \in [\![\mathbf{C}_b^q G]\!]_{X\cup\{a\}}$ as desired.

- $F = \mathbf{D}_a^q G$. The proof is similar to the equivalent above.

□

Lemma 5.3.5. *Let $a \notin X$, $\mathsf{FN}(F) \subseteq X$, and $f \in (2^{\mathbb{N}})^X$:*

i. if $f \in [\![F]\!]_X$, then $\Pi_f\big([\![F]\!]_{X\cup\{a\}}\big) = (2^{\mathbb{N}})^{\{a\}}$

ii. if $f \notin [\![F]\!]_X$*, then* $\Pi_f\big([\![F]\!]_{X\cup\{a\}}\big) = \emptyset^{\{a\}}$

or equivalently, $[\![F]\!]_{X\cup\{a\}} = [\![F]\!]_X^{\Uparrow\{a\}}$.

Proof. As clear, there are two cases to be taken into account:

i. Let $f \in [\![F]\!]_X$. Then,

$$\begin{aligned}\Pi_f([\![F]\!]_{X\cup\{a\}}) &\overset{D}{=}^{4.1.2} \{g \in (2^{\mathbb{N}})^{\{a\}} \mid f + g \in [\![F]\!]_{X\cup\{a\}}\}\\ &\overset{L}{=}^{5.3.4} \{g \in (2^{\mathbb{N}})^{\{a\}} \mid f \in [\![F]\!]_X\}\\ &= (2^{\mathbb{N}})^{\{a\}}.\end{aligned}$$

ii. Let $f \notin [\![F]\!]_X$. Then,

$$\begin{aligned}\Pi_f([\![F]\!]_{X\cup\{a\}}) &\overset{D}{=}^{4.1.2} \{g \in (2^{\mathbb{N}})^{\{a\}} \mid f + g \in [\![F]\!]_{X\cup\{a\}}\}\\ &\overset{L}{=}^{5.3.4} \{g \in (2^{\mathbb{N}})^{\{a\}} \mid f \in [\![F]\!]_X\}\\ &= \emptyset^{\{a\}}.\end{aligned}$$

□

Lemma 5.3.6. *Let* $a \notin X, \mathsf{FN}(F) \subseteq X, f \in (2^{\mathbb{N}})^X$*, and* $q > 0$*. Then:*[5]

$$\begin{aligned}\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) \geq q \textit{ iff } f \in [\![F]\!]_X\\
\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q \textit{ iff } f \in [\![F]\!]_X \wedge \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q\\
\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q \textit{ iff } f \in [\![F]\!]_X \vee \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q\\
\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q \textit{ iff } f \notin [\![F]\!]_X \wedge \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q\\
\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q \textit{ iff } f \notin [\![F]\!]_X \vee \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q.\end{aligned}$$

Proof. Let us consider each case.

Case I. Let us consider

$$\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) \geq q \quad \text{iff} \quad f \in [\![F]\!]_X.$$

$\Rightarrow$ The proof is by contraposition. Assume $f \notin [\![F]\!]_X$. Since $a \notin X$ and $f \in (2^{\mathbb{N}})^X$ for hypothesis, we apply Lemma 5.3.5.ii, obtaining $\Pi_f([\![F]\!]_{X\cup\{a\}}) = \emptyset^{\{a\}}$. Then, $\mu_{\mathscr{C}}(\Pi_f([\![G]\!]_{X\cup\{a\}})) = \mu_{\mathscr{C}}(\emptyset^{\{a\}}) = 0$, that is $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) < q$.

$\Leftarrow$ Assume $f \in [\![F]\!]_X$. Since $a \notin X$ and $f \in (2^{\mathbb{N}})^X$ for hypothesis, we apply Lemma 5.3.5.i, obtaining $\Pi_f([\![F]\!]_{X\cup\{a\}}) = (2^{\mathbb{N}})^{\{a\}}$. Then, $\mu_{\mathscr{C}}\big(\Pi_f\big([\![F]\!]_{X\cup\{a\}}\big)\big) = \mu_{\mathscr{C}}\big((2^{\mathbb{N}})^{\{a\}}\big) = 1$, that is $\mu_{\mathscr{C}}\big(\Pi_f\big([\![F]\!]_{X\cup\{a\}}\big)\big) \geq q$.

[5]Notice that, given cases II and III, cases IV and V become somehow pleonastic, but for the sake of clarity (in particular, in connection with the proof of Lemma 5.3.8 above) we present them explicitly.

Case II. Let us consider

$$\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\geq q \text{ iff } f\in[\![F]\!]_X\wedge\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\geq q.$$

Preliminarily notice that:

- If $f\in[\![F]\!]_X$, then, since $a\notin X$ and $f\in(2^{\mathbb{N}})^X$, by Lemma 5.3.5.i, $\Pi_f\big([\![F]\!]_{X\cup\{a\}}\big)=(2^{\mathbb{N}})^{\{a\}}$. So:

$$\begin{aligned}\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)&=\mu_{\mathscr{C}}\big((2^{\mathbb{N}})^{\{a\}}\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\\&=\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big).\end{aligned}$$

- If $f\notin[\![F]\!]_X$, then, since $a\notin X$ and $f\in(2^{\mathbb{N}})^X$, by Lemma 5.3.5.ii, $\Pi_f([\![F]\!]_{X\cup\{a\}})=\emptyset^X$. So:

$$\begin{aligned}\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)&=\mu_{\mathscr{C}}\big((\emptyset)^{\{a\}}\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\\&=\mu_{\mathscr{C}}(\emptyset^{\{a\}})\\&=0.\end{aligned}$$

  Then, we conclude the proof as follows:

  $\Leftarrow$ For hypothesis $f\in[\![F]\!]_X$ and $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\geq q$. Then, for the first clause above together with the first hypothesis, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)=\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)$ and, clearly, for the second hypothesis – namely, $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\geq q$ – $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\geq q$, as desired.

  $\Rightarrow$ The proof is by contraposition. If $f\notin[\![F]\!]_X$, then, for the second clause above, $\mu\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)=0$. So, trivially, for any $q\in\mathbb{Q}_{[0,1]}$, we conclude $\mu_{\mathscr{C}}(\Pi_f([\![F]\!]_X)\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)<q$. Otherwise, $f\in[\![F]\!]_X$ and $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)<q$. Observe that for the first clause above, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)=\mu_{\mathscr{C}}(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)<q$. Then, clearly, we conclude $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\cap\Pi_f([\![G]\!]_{X\cup\{a\}})\big)<q$.

Case III. Let us consider

$$\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\cup\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\geq q \text{ iff } f\in[\![F]\!]_X\vee\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\geq q.$$

Preliminarily, notice that:

- If $f\in[\![F]\!]_X$, then, since $a\notin X$ and $f\in(2^{\mathbb{N}})^X$, by Lemma 5.3.5.i, $\Pi_f([\![F]\!]_{X\cup\{a\}})=(2^{\mathbb{N}})^{\{a\}}$ So:

$$\begin{aligned}\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\cup\Pi_f([\![G]\!]_{X\cup\{a\}})\big)&=\mu_{\mathscr{C}}\big((2^{\mathbb{N}})^{\{a\}}\cup\Pi_f([\![G]\!]_{X\cup\{a\}})\big)\\&=\mu_{\mathscr{C}}\big((2^{\mathbb{N}})^{\{a\}}\big)\\&=1.\end{aligned}$$

- If $f \notin [\![F]\!]_X$, then, since $a \notin X$ and $f \in (2^{\mathbb{N}})^X$, by Lemma 5.3.5.ii, $\Pi_f\big([\![F]\!]_{X\cup\{a\}}\big) = \emptyset^X$. So:

$$\begin{aligned}\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) &= \mu_{\mathscr{C}}\big(\emptyset^{\{a\}} \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big)\\ &= \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big).\end{aligned}$$

Then, we conclude the proof as follows:

$\Leftarrow$ If $f \in [\![F]\!]_X$, then, for the first clause above, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) = 1$. It is then clear that, for any $q \in \mathbb{Q}_{[0,1]}$, $\mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X\cup\{a\}} \cup \Pi_f([\![G]\!]_{X\cup\{a\}})) \geq q$.

If $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$, then – since clearly $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)$ – also $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$.

$\Rightarrow$ The proof is by contraposition. Assume $f \notin [\![F]\!]$ and $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$. Furthermore, for the second clause above $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) = \mu_{\mathscr{C}}(\Pi_f([\![G]\!]_{X\cup\{a\}}))$. So, we conclude $\mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})) < q$.

Case IV. Let us consider

$$\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q \text{ iff } f \notin [\![F]\!]_X \wedge \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q.$$

Notice that preliminary observations of Case III above will be useful also to treat Case IV. Then, we conclude the proof as follows:

$\Leftarrow$ Assume $f \notin [\![F]\!]_X$ and $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}}\big) < q$. Observe that for the second clause above, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) = \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)$. Then, since for hypothesis $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$ so also $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$.

$\Rightarrow$ The proof is by contraposition. If $f \in [\![F]\!]_X$. then for the first clause above $\mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) = 1$. Then, clearly, for every $q \in \mathbb{Q}_{[0,1]}$ such that $q > 0$ (for hypothesis), $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$.

If $f \notin [\![F]\!]$ and $\mu(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$. Observe that, for the second clause above, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) = \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!])_{X\cup\{a\}}\big)$. But, for hypothesis, $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$ holds. So, we conclude $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$.

Case V. Let us consider

$$\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q \text{ iff } f \notin [\![F]\!]_X \vee \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q.$$

Again notice that preliminary observations of Case II will be useful also to treat Case V. Then, we conclude the proof as follows:

$\Leftarrow$ If $f \notin [\![F]\!]_X$, then, for the second clause in Case II, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) = 0$. So, trivially, for any $q \in \mathbb{Q}_{[0,1]}$ such that $q > 0$ (for hypothesis), we conclude $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_X) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$.

Otherwise, $f \in [\![F]\!]_X$ and $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$. Observe that for the first clause in Case II, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) = \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$. Then, we conclude $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$.

$\Rightarrow$ The proof is by contraposition. Assume $f \in [\![F]\!]_X$ and $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$. Then, for the first clause in Case II together with the first hypothesis, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}} \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) = \mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big)$. So, clearly, for the second hypothesis, we conclude $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$.

□

Lemma 5.3.7. *For any **CPL**-formulae F and G such that $\mathsf{FN}(F), \mathsf{FN}(G) \subseteq X$ and $f \in (2^{\mathbb{N}})^X$*

$$\Pi_f([\![F]\!]_X) \cap \Pi_f([\![G]\!]_X) = \Pi_f([\![F \wedge G]\!]_X) \tag{1}$$
$$\Pi_f([\![F]\!]_X) \cup \Pi_f([\![G]\!]_X) = \Pi_f([\![F \vee G]\!]_X) \tag{2}$$

Proof. (1) By Definition 4.1.2, $\Pi_f([\![F]\!]_X) = \{g \in (2^{\mathbb{N}})^X \mid f + g \in [\![F]\!]_X\}$ and $\Pi_f([\![G]\!]) = \{g' \in (2^{\mathbb{N}})^X \mid f + g \in [\![G]\!]_X\}$. Then, $\Pi_f([\![F]\!]_X) \cap \Pi_f([\![G]\!]_X) = \{g \in (2^{\mathbb{N}})^X \mid f + g \in [\![F]\!]_X\} \cap \{g' \in (2^{\mathbb{N}})^X \mid f + g' \in [\![G]\!]_X\} = \{h \in (2^{\mathbb{N}})^X \mid f + h \in [\![F]\!]_X \cap [\![G]\!]_X\} \overset{D\ 4.1.3}{=} \{h \in (2^{\mathbb{N}})^X \mid f + h \in [\![F \wedge G]\!]_X\}$. So, again by Definition 4.1.2, we conclude $\Pi_f([\![F]\!]_X) \cap \Pi_f([\![G]\!]_X) = \Pi_f([\![F \wedge G]\!]_X)$.

(2) Clearly, again by Definition 4.1.2, $\Pi_f([\![F]\!]_X) \cup \Pi_f([\![G]\!]_X) = \{g \in (2^{\mathbb{N}})^X \mid f + g \in [\![F]\!]_X\} \cup \{g' \in (2^{\mathbb{N}})^X \mid f + g' \in [\![G]\!]_X\} = \{h \in (2^{\mathbb{N}})^X \mid f + h \in [\![F]\!]_X \cup [\![G]\!]_X\} \overset{D\ 4.1.3}{=} \{h \in (2^{\mathbb{N}})^X \mid f + h \in [\![F \vee G]\!]_X\}$. So, again by Definition 4.1.2, we conclude $\Pi_f([\![F]\!]_X) \cup \Pi_f([\![G]\!]_X) = \Pi_f([\![F \vee G]\!]_X)$. □

We can now prove the desired lemma, stating that counting quantifiers can be extruded from conjunctions and disjunctions.

Lemma 5.3.8. *Let $a \notin \mathsf{FN}(F)$ and $q > 0$. Then, for every X such that $\mathsf{FN}(F) \cup \mathsf{FN}(G) \subseteq X \cup \{a\}$, and $a \notin X$, the following holds:*

$$F \wedge \mathbf{C}^q_a G \equiv_X \mathbf{C}^q_a(F \wedge G) \qquad F \vee \mathbf{C}^q_a G \equiv_X \mathbf{C}^q_a(F \vee G)$$
$$F \wedge \mathbf{D}^q_a G \equiv_X \mathbf{D}^q_a(\neg F \vee G) \qquad F \vee \mathbf{D}^q_a G \equiv_X \mathbf{D}^q_a(\neg F \wedge G).$$

Proof. Let us deal with each case separately from the others.

Case I. Let us consider

$$[\![F \wedge \mathbf{C}^q_a G]\!]_X = [\![\mathbf{C}^q_a(F \wedge G)]\!]_X$$

$\subseteq$ We will prove that for any $f \in (2^{\mathbb{N}})^X$, if $f \in [\![F \wedge \mathbf{C}_a^q G]\!]_X$, then $f \in [\![\mathbf{C}_a^q(F \wedge G)]\!]_X$. Assume $f \in [\![F \wedge \mathbf{C}_a^q G]\!]_X \overset{D\ 4.1.3}{=} [\![F]\!]_X \cap [\![\mathbf{C}_a^q G]\!]_X$. So, by basic measure theory $f \in [\![F]\!]_X$ and $f \in [\![\mathbf{C}_a^q G]\!]_X \overset{D\ 4.1.3}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_h([\![G]\!]_{X\cup\{a\}})) \geq q\}$, i.e. $\mu_{\mathscr{C}}(\Pi_f([\![G]\!]_{X\cup\{a\}})) \geq q$. Then, by Lemma 5.3.6.2$\Leftarrow$, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$, that is, by Definition 4.1.3, $f \in \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![F]\!]_{X\cup\{a\}}) \cap \Pi_h([\![G]\!]_X)\big) \geq q\} \overset{L\ 5.3.7.1}{=} \{h \in (2^{\mathbb{N}})^{X\cup\{a\}} \mid \mu_{\mathscr{C}}(\Pi_h[\![F \wedge G]\!]_{X\cup\{a\}}) \geq q\}$. So, again by Definition 4.1.3, we conclude $f \in [\![\mathbf{C}_a^q(F \wedge G)]\!]_X$.

$\supseteq$ We will prove that, for any $f \in (2^{\mathbb{N}})^X$, if $f \in [\![\mathbf{C}_a^q(F \wedge G)]\!]_X$, then $f \in [\![F \wedge \mathbf{C}_a^q G]\!]_X$. Assume $f \in [\![\mathbf{C}_a^q(F \wedge G)]\!]_X = \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![F \wedge G]\!]_{X\cup\{a\}})\big) \geq q\} \overset{L\ 5.3.7.1}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![F]\!]_{X\cup\{a\}}) \cap \Pi_h([\![G]\!]_{X\cup\{a\}}) \geq q\}$, i.e. $\mu_{\mathscr{C}}(\Pi_f([\![F]\!])_{X\cup\{a\}} \cap \Pi_f([\![G]\!]_{X\cup\{a\}})) \geq q$. So, by Lemma 5.3.6.2$\Rightarrow$, $f \in [\![F]\!]_X$ and $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$. Then, also $f \in \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_h([\![G]\!]_{X\cup\{a\}})) \geq q\} \overset{D\ 4.1.3}{=} [\![\mathbf{C}_a^q G]\!]_X$. So, since $f \in [\![F]\!]_X$ and $f \in [\![\mathbf{C}_a^q G]\!]_X$, also $f \in [\![F]\!]_X \cap [\![\mathbf{C}_a^q G]\!]_X$ and, by Definition 4.1.3, we conclude $f \in [\![F \wedge \mathbf{C}_a^q G]\!]_X$.

Case II. Let us consider

$$[\![F \vee \mathbf{C}_a^q G]\!]_X = [\![\mathbf{C}_a^q(F \vee G)]\!]_X$$

$\subseteq$ We will prove that, for any $f \in (2^{\mathbb{N}})^X$, if $f \in [\![F \vee \mathbf{C}_a^q G]\!]_X$, then $f \in [\![\mathbf{C}_a^q(F \vee G)]\!]_X$. Assume $f \in [\![F \vee \mathbf{C}_a^q G]\!]_X \overset{D\ 4.1.3}{=} [\![F]\!]_X \cup [\![\mathbf{C}_a^q G]\!]_X$. Then, for basic set theory, $f \in [\![F]\!]_X$ or $f \in [\![\mathbf{C}_a^q G]\!]_X \overset{D\ 4.1.3}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![G]\!]_{X\cup\{a\}})\big) \geq q\}$, i.e. $\mu_{\mathscr{C}}(\Pi_f([\![G]\!]_{X\cup\{a\}})) \geq q$. Then, by Lemma 5.3.6.3$\Leftarrow$, $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$, that is, by Definition 4.1.2, $f \in \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_h([\![F]\!]_{X\cup\{a\}}) \cup \Pi_h([\![G]\!]_{X\cup\{a\}})) \geq q\} \overset{L\ 5.3.7.2}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![F \vee G]\!]_{X\cup\{a\}})\big) \geq q\}$. So, again by Definition 4.1.3, we conclude $f \in [\![\mathbf{C}_a^q(F \vee G)]\!]_X$.

$\supseteq$ We will prove that, for any $f \in (2^{\mathbb{N}})^X$, if $f \in [\![\mathbf{C}_a^q(F \vee G)]\!]_X$, then $f \in [\![F \vee \mathbf{C}_a^q G]\!]_X$. Assume $f \in [\![\mathbf{C}_a^q(F \vee G)]\!] = \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_h([\![F \vee G]\!]_{X\cup\{a\}})) \geq q \overset{L\ 5.3.7.2}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![F]\!]_{X\cup\{a\}}) \cup \Pi_h([\![G]\!]_{X\cup\{a\}}) \geq q\}$, that is $\mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}}) \cup \Pi_f\big([\![G]\!]_{X\cup\{a\}}\big)\big) \geq q$. So, by Lemma 5.3.6.3$\Rightarrow$ $f \in [\![F]\!]_X$ or $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) \geq q$. Then, also $f \in \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![G]\!]_{X\cup\{a\}})\big) \geq q\} \overset{D\ 4.1.3}{=} [\![\mathbf{C}_a^q G]\!]_X$. Then, since $f \in [\![F]\!]_X$ or $f \in [\![\mathbf{C}_a^q G]\!]_X$, also $f \in [\![F]\!]_X \cup [\![\mathbf{C}_a^q G]\!]_X$ and, by Definition 4.1.3, we conclude $f \in [\![F \vee \mathbf{C}_a^q G]\!]_X$.

Case III. Let us consider

$$[\![F \wedge \mathbf{D}_a^q G]\!]_X = [\![\mathbf{D}_a^q(\neg F \vee G)]\!]_X$$

$\subseteq$ We will prove that, for ny $f \in (2^{\mathbb{N}})^X$, if $f \in [\![F \wedge \mathbf{D}_a^q G]\!]_X$, then $f \in [\![\mathbf{D}_a^q(\neg F \vee G)]\!]_X$. Assume $f \in [\![F \wedge \mathbf{D}_a^q G]\!]_X \overset{D\ 4.1.3}{=} [\![F]\!]_X \cap [\![\mathbf{D}_a^q G]\!]_X$. So, for basic set theory, $f \in [\![F]\!]_X$ and $f \in [\![\mathbf{D}_a^q G]\!]_X \overset{D\ 4.1.3}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_h([\![G]\!]_{X\cup\{a\}})) < q\}$, i.e. $\mu_{\mathscr{C}}(\Pi_f([\![G]\!]_{X\cup\{a\}})) < q$. Furthermore, since $f \in [\![F]\!]_X$, $f \notin (2^{\mathbb{N}})^X - [\![F]\!]_X \overset{D\ 4.1.3}{=} f \notin [\![\neg F]\!]_X$. Then, by Lemma 5.3.6.4$\Leftarrow$, $\mu_{\mathscr{C}}\big(\Pi_f([\![\neg F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$, that is by Lemma 5.3.7.2, $\mu_{\mathscr{C}}\big(\Pi_f([\![\neg F \vee G]\!]_{X\cup\{a\}})\big) < q$. So, by Definition 4.1.2, $f \in \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_f([\![\neg F \vee G]\!]_{X\cup\{a\}})\big) < q$. Therefore, again by Definition 4.1.3, $f \in [\![\mathbf{D}_a^q(\neg F \vee G)]\!]_X$.

$\supseteq$ We will prove that, for any $f \in (2^{\mathbb{N}})^X$, if $f \in [\![\mathbf{D}_a^q(\neg F \vee G)]\!]_X$, then $f \in [\![F \wedge \mathbf{D}_a^q G]\!]_X$. Assume $f \in [\![\mathbf{D}_a^q(\neg F \vee G)]\!]_X \overset{D\ 4.1.3}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}(\Pi_h([\![\neg F \vee G]\!]_{X\cup\{a\}})) < q\} \overset{L\ 5.3.7.2}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![\neg F]\!]_{X\cup\{a\}}) \cup \Pi_h([\![G]\!]_{X\cup\{a\}})\big) < q\}$, that is $\mu_{\mathscr{C}}\big(\Pi_f([\![\neg F]\!]_{X\cup\{a\}}) \cup \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$. Then, by Lemma 5.3.6.4$\Rightarrow$, $f \notin [\![\neg F]\!]_X$ and $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$. In particular, since $f \notin [\![\neg F]\!]_X$, $f \in (2^{\mathbb{N}})^X - [\![\neg F]\!]_X$ and by Definition 4.1.3, $f \in (2^{\mathbb{N}})^X - ((2^{\mathbb{N}})^X - [\![F]\!]_X)$, that is $f \in [\![F]\!]_X$. Furthermore, since $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$, $f \in \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![G]\!]_{X\cup\{a\}})\big) < q\} \overset{D\ 4.1.3}{=} [\![\mathbf{D}_a^q G]\!]_X$ Thus, $f \in [\![F]\!]_X$ and $f \in [\![\mathbf{D}_a^q G]\!]_X$. We conclude $f \in [\![F]\!]_X \cap [\![\mathbf{D}_a^q G]\!]_X = [\![F \vee \mathbf{D}_a^q G]\!]_X$, as desired.

Case IV. Let us consider

$$[\![F \vee \mathbf{D}_a^q G]\!]_X = [\![\mathbf{D}_a^q(\neg F \wedge G)]\!]_X$$

$\subseteq$ We will prove that for any $f \in (2^{\mathbb{N}})^X$, if $f \in [\![F \vee \mathbf{D}_a^q G]\!]_X$, then $f \in [\![\mathbf{D}_a^q(\neg F \wedge G)]\!]_X$. Assume $f \in [\![F \vee \mathbf{D}_a^q G]\!]_X = [\![F]\!]_X \cup [\![\mathbf{D}_a^q G]\!]_X$. So, for basic measure theory, $f \in [\![F]\!]_X$ or $f \in [\![\mathbf{D}_a^q G]\!]_X \overset{D\ 4.1.3}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) < q\}$, i.e. $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$. Furthermore, since $f \in [\![F]\!]_X$, $f \notin (2^{\mathbb{N}})^X - [\![F]\!]_X \overset{D\ 4.1.3}{=} f \notin [\![\neg F]\!]_X$. Then, by Lemma 5.3.6.5$\Leftarrow$, $\mu_{\mathscr{C}}(\Pi_f([\![\neg F]\!]_{X\cup\{a\}} \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$, that is, by Lemma 5.3.7.2 $\mu_{\mathscr{C}}\big(\Pi_f([\![\neg F \wedge G]\!]_{X\cup\{a\}})\big) < q$. Thus, by Definition 4.1.2, $f \in \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_f([\![\neg F \wedge G]\!]_{X\cup\{a\}})\big) < q$ and, again by Definition 4.1.3, we conclude $f \in [\![\mathbf{D}_a^q(\neg F \wedge G)]\!]_X$.

$\supseteq$ We will prove that for any $f \in (2^{\mathbb{N}})^X$, if $f \in [\![\mathbf{D}_a^q(\neg F \wedge G)]\!]_X$, then $f \in [\![F \vee \mathbf{D}_a^q G]\!]_X$. Assume $f \in [\![\mathbf{D}_a^q(\neg F \wedge G)]\!]_X \overset{D\ 4.1.3}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![\neg F \wedge G]\!]_{X\cup\{a\}})\big) < q\} \overset{L\ 5.3.7.1}{=} \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![\neg F]\!]_{X\cup\{a\}}) \cap \Pi_h([\![G]\!]_{X\cup\{a\}})\big) < q\}$, that is $\mu_{\mathscr{C}}\big(\Pi_f([\![\neg F]\!]_{X\cup\{a\}}) \cap \Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$. Then, by Lemma 5.3.6.5$\Rightarrow$, $f \notin [\![\neg F]\!]_X$ or $\mu_{\mathscr{C}}\big(\Pi_f([\![G]\!]_{X\cup\{a\}})\big) < q$. In particular, if $f \notin [\![\neg F]\!]_X$, by Definition 4.1.3, $f \in [\![F]\!]_X$. Furthermore, if $\mu_{\mathscr{C}}(\Pi_f([\![F]\!]_{X\cup\{a\}})) < q$, then $f \in \{h \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_h([\![G]\!]_{X\cup\{a\}})\big) < q\}$

$\overset{D\ 4.1.3}{=} [\![\mathbf{D}_a^q G]\!]_X$. Therefore, $f \in [\![F]\!]_X$ or $f \in [\![\mathbf{D}_a^q G]\!]_X$. So, for basic measure theory and Definition 4.1.3, we conclude $f \in [\![F]\!]_X \cup [\![\mathbf{D}_a^q G]\!]_X = [\![F \vee \mathbf{D}_a^q G]\!]_X$, as desired.

□

Observe that the proof of Lemma 5.3.8 relies on the possibility of *renaming* variables and, indeed, a corresponding Lemma does *not* hold for $\mathbf{CPL}_0$.

Inter-Definability of Quantifiers. Let us now consider negation. Inter-definability of univariate counting quantifiers, $\mathbf{C}^q$ and $\mathbf{D}^q$, can be generalized to the multivariate case. This allows us to get rid of negations which lie between an occurrence of a counting quantifier and the formula's root.

Lemma 5.3.9. *For every $q \in \mathbb{Q} \cap [0,1]$, name $a \in \mathscr{A}$, and X such that $\mathsf{FN}(F) \subseteq X \cup \{a\}$ and $a \notin X$,*

$$\neg \mathbf{D}_a^q F \equiv_X \mathbf{C}_a^q F \qquad\qquad \neg \mathbf{C}_a^q F \equiv_X \mathbf{D}_a^q F.$$

Proof. Let us consider the first one only. (The second is proved in a similar way.) There are two possible cases to be taken into account:

- If $q = 0$,

$$\begin{aligned} [\![\neg \mathbf{D}_a^0 F]\!] &= (2^{\mathbb{N}})^X - [\![\mathbf{D}_a^0 F]\!]_X \\ &\overset{L\ 5.3.1}{=} (2^{\mathbb{N}})^X - \emptyset^X \\ &= (2^{\mathbb{N}})^X \\ &\overset{L\ 5.3.1}{=} [\![\mathbf{C}_a^0 F]\!]. \end{aligned}$$

- If $q > 0$,

$$\begin{aligned} [\![\neg \mathbf{D}_a^q F]\!]_X &= (2^{\mathbb{N}})^X - [\![\mathbf{D}_a^q F]\!]_X \\ &= (2^{\mathbb{N}})^X - \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X \cup \{a\}})\big) < q\} \\ &= \{f \in (2^{\mathbb{N}})^X \mid \mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X \cup \{a\}})\big) \geq q\} \\ &= [\![\mathbf{C}_a^q F]\!]_X. \end{aligned}$$

□

Therefore, using Lemma 5.3.8 and 5.3.9, we conclude that – as desired – every formula of **CPL** can be converted into PNF.

Proposition 5.3.2. *For any formula of **CPL**, F, there is a formula in PNF G, such that for any X with $\mathsf{FN}(F) \cup \mathsf{FN}(G) \subseteq X$,*

$$F \equiv_X G.$$

Moreover, G can be computed in polynomial time from F.

5.3.3 Positive Prenex Normal Form

Reducing formulae to PNF is close to what we need, but there is one last step to be made, namely getting rid of the quantifier $\mathbf{D}$, which does not have any counterpart in Wagner's construction. In other words, we need to reduce formulae of **CPL** to prenex normal forms *of a special kind*:

Definition 5.3.3 (Positive Prenex Normal Form)**.** A formula of **CPL** is said to be a *positive prenex normal form* (PPNF, for short) when it is both in PNF and $\mathbf{D}$-free.

The gist to convert counting formulae into (equivalent) PPNF, consists of two main steps: (i.) converting each instance of $\mathbf{D}$ into one of $\mathbf{C}$, using Lemma 5.3.9, and (ii.) applying the Lemma below which states that $\mathbf{C}$ enjoys a specific, weak form of self-duality, to push the negation inside the matrix. In order to prove (ii.), we need to introduce some auxiliary definition and to establish some results, in particular the so-called Epsilon Lemma.

Epsilon Lemma. For any $k \in \mathbb{N}$, let $[0,1]_k$ indicate the set of rationals of the form $q = \sum_{i=0}^{k} b_i \cdot 2^{-i}$, where $b_i \in \{0,1\}$. Notice that, for all $p \leq 2^k$, $\frac{p}{2^k} \in [0,1]_k$.

Lemma 5.3.10. *For each $p \leq 2^k$, $\frac{p}{2^k} \in [0,1]_k$.*

Proof. Let $b_0 \dots b_k = (p)_2$ the base 2 of p, with possibly all 0s at the end so that the length is precisely k, then $p = \sum_{i=0}^{k} b_i \cdot 2^i$:

$$\begin{aligned}\frac{p}{2^k} &= \frac{\sum_{i=0}^{k} b_i \cdot 2^i}{2^k} \\ &= \sum_{i=0}^{k} b_i \cdot 2^{-k+i} \\ &= \sum_{i=0}^{k} b_{k-i} \cdot 2^{-i}.\end{aligned}$$

□

Lemma 5.3.11. *For every formula of **CPL** F, there is a named Boolean formula $\mathscr{b}_F$, such that for every X, with $\mathsf{FN}(F) \cup \mathsf{FN}(G) \subseteq X$:*

$$[\![F]\!]_X = [\![\mathscr{b}_F]\!]_X.$$

Proof Sketch. The proof is by simple induction on the structure of F, where the quantified cases rely on Lemma 4.2.2. □

Lemma 5.3.12. *For each Boolean formula $\mathscr{b}$ with $\mathsf{FN}(\mathscr{b}) \subseteq \{a\}$,*

$$\mu\big([\![\mathscr{b}]\!]_{\{a\}}\big) \in [0,1]_k,$$

where k is the maximum natural number such that x_k^a occurs in $\mathscr{b}$.

Proof. By Lemma 3.3.1 and 5.3.10:

$$\mu\big([\![\mathscr{b}]\!]_{\{a\}}\big) = \sharp\mathscr{b} = \frac{\sharp\{x_0^a, \dots x_k^a\} \to \{0,1\} \mid m \vdash \mathscr{b}\}}{2^k} \in [0,1]_k.$$

□

Lemma 5.3.13. *For all $S \in \mathscr{B}\big((2^{\mathbb{N}})^{X\cup\{a\}}\big)$ and $f : X \to 2^{\mathbb{N}}$,*

$$\overline{\Pi_f(S)} = \Pi_f(\overline{S}).$$

Proof. Indeed,

$$\begin{aligned}\overline{\Pi_f(S)} &= \overline{\{g : \{a\} \to 2^{\mathbb{N}} \mid f + g \in S\}} \\ &= \{g : \{a\} \to 2^{\mathbb{N}} \mid f + g \notin S\} \\ &= \{g : \{a\} \to 2^{\mathbb{N}} \mid f + g \in \overline{S}\} \\ &= \Pi_f(\overline{S}).\end{aligned}$$

□

Lemma 5.3.14. *For all $S \in \mathscr{B}\big((2^{\mathbb{N}})^{X}\big)$ and $r \in [0,1]$,*

$$\mu_{\mathscr{C}}(S) \leq r \quad \textit{iff} \quad \mu_{\mathscr{C}}(\overline{S}) \geq 1 - r.$$

Proof. The claim follows from:

$$1 - \mu_{\mathscr{C}}\big((2^{\mathbb{N}})^{X}\big) = \mu_{\mathscr{C}}(S \cup \overline{S}) = \mu_{\mathscr{C}}(S) + \mu_{\mathscr{C}}(\overline{S}).$$

□

We now have all ingredients to prove the crucial Epsilon Lemma.

Lemma 5.3.15 (Epsilon Lemma)**.** *For every formula of **CPL** F, and $q \in \mathbb{Q}\cap[0,1]$, there is a $p \in \mathbb{Q}\cap[0,1]$, such that for every X with $\mathsf{FN}(F) \subseteq X$ and $a \notin X$,*

$$\neg\mathbf{C}_a^q F \equiv_X \mathbf{C}_a^p \neg F.$$

Moreover, p can be computed from q in polynomial time.

Proof. Let $\mathscr{b}_F$ be a Boolean formula satisfying $[\![F]\!]_{X\cup\{a\}} = [\![\mathscr{b}_F]\!]_{X\cup\{a\}}$, which exists by Lemma 5.3.11. Let $\mathscr{b}_F$ be a-decomposable as $\bigvee_i^n \mathscr{d}_i \wedge \mathscr{e}_i$ and let k be the maximum such that x_k^a occurs in $\mathscr{b}_F$. By Lemma 5.3.12, for all $i = 0, \dots, n$, $\mu_{\mathscr{C}}([\![\mathscr{d}]\!]_{\{a\}}) \in [0,1]_k$. This implies in particular that for all $f : X \to 2^{\mathbb{N}}$, $\mu\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) \in [0,1]_k$, since $\Pi_f\big([\![F]\!]_{X\cup\{a\}}\big)$ coincides with the unique $[\![\mathscr{d}_i]\!]_{\{a\}}$ such that $f \in [\![\mathscr{e}_i]\!]_X$, by Lemma 4.2.2. Now, if $q \notin [0,1]_k$, let $\epsilon = 0$ and if $q \in [0,1]_k$, then:

$$\epsilon = \begin{cases} -2^{-(k+1)} & \text{if } q = 1 \\ 2^{-(k+1)} & \text{if } q \neq 1. \end{cases}$$

So, in any case, $q + \epsilon \notin [0,1]_k$ and we conclude:

$$\begin{aligned}
[\![\neg \mathbf{C}_a^q F]\!]_X &= \{f : X \to 2^{\mathbb{N}} \mid \mu_{\mathscr{C}}\big(\Pi_f([\![F]\!]_{X\cup\{a\}})\big) \leq q + \epsilon\} \\
&\overset{L\ 5.3.14}{=} \{f : X \to 2^{\mathbb{N}} \mid \mu_{\mathscr{C}}\big(\overline{\Pi_f([\![F]\!]_{X\cup\{a\}})}\big) \geq 1 - (q + \epsilon)\} \\
&\overset{L\ 5.3.12}{=} \{f : X \to 2^{\mathbb{N}} \mid \mu_{\mathscr{C}}(\Pi_f([\![\neg F]\!]_{X\cup\{a\}})) \geq 1 - (q + \epsilon)\} \\
&= [\![\mathbf{C}_a^{1-(q+\epsilon)} \neg F]\!]_X.
\end{aligned}$$

□

Actually, the value of p is very close to $1 - q$, the difference between the two being easily computable from the formula F.

Concluding the Proof. We now conclude that any counting formula can be converted into the desired PPNF.

Proposition 5.3.3. *For every formula of **CPL** F, there is a PPNF G, such that for every X, with $\mathsf{FN}(F) \cup \mathsf{FN}(G) \subseteq X$,*

$$F \equiv_X G.$$

Moreover, G can be computed from F in polynomial time.

Proof. Due to Lemma 5.3.15. □

5.3.4 CPL and the Counting Hierarchy

As seen, in 1986, Wagner not only introduced his counting operator and hierarchy, but also defined complete problems for *each level* of $\mathbf{CH}$. Below, we present a slightly weaker version of Wagner's Theorem 7 [226, pp. 338-339], which perfectly fits our needs. Assume that $\mathcal{L}$ is a subset of S^n, where S is a set, $1 \leq m \leq n$, and $b \in \mathbb{N}$. We define $\mathbf{C}_m^q \mathcal{L}$ as the following subset of S^{n-m}:

$$\{(a_n, \ldots, a_{m+1}) \mid \sharp(\{(a_m, \ldots, a_1) \mid (a_n, \ldots, a_1) \in \mathcal{L}\}) \geq q\}.$$

Let $\top$ and $\bot$ indicate the usual formulae of **PL**. For any natural number $n \in \mathbb{N}$, let $\mathcal{TF}^n$ be the subset of $\mathbf{PL}^{n+1}$ containing all tuples in the form $(F, t_1, \ldots, t_n)$, where F is a propositional formula in CNF with at most n free variables and $t_1, \ldots, t_n \in \{\top, \bot\}$ render F true. Finally, for every $k \in \mathbb{N}$, we denote as $\mathcal{W}^k$ the language consisting of all (binary encodings of) tuples of the form $(F, m_1, \ldots, m_k, b_1, \ldots, b_k)$ such that $F \in \mathbf{C}_{m_1}^{b_q} \ldots \mathbf{C}_{m_k}^{b_k} \mathcal{TF}^{\sum m_i}$.

Theorem 5.3.1 (Theorem 7 [227])**.** *For every k, the language $\mathcal{W}^k$ is complete for $\mathbf{CH}_k$.*

Observe that elements of $\mathcal{W}^k$ can be seen as alternative representations for **CPL**-formulae in PPNF once any m_i is replaced by $\min\{1, \frac{m_i}{2^{b_i}}\}$. Consequently,

Corollary 5.3.2. *The closed and valid k-ary PPNFs, the matrix of which is in CNF, define a complete set for $\mathbf{CH}_k$.*

Proof Sketch. Putting Proposition 5.3.3 and Theorem 5.3.1 together. □

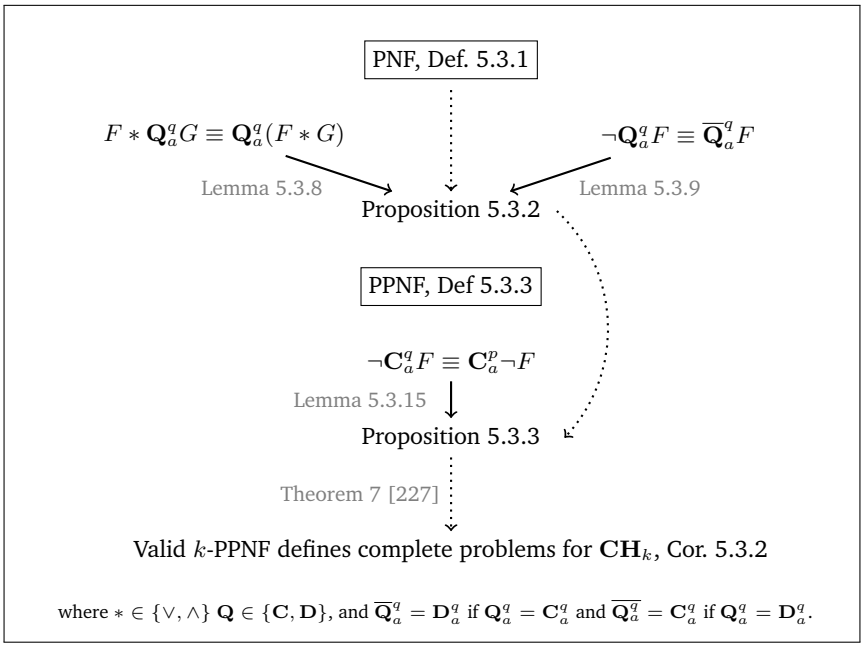


Figure 5.3: The Proof of Corollary 5.3.2

Part II

Curry and Howard Meet Borel

Chapter 6

Towards a Probabilistic Correspondence (and Beyond)

We introduce the intuitionistic version of **CPL**, called **iCPL**$_0$, associated with a quantitative Kripke-style semantics and equipped with a sound and complete natural deduction system. We then show that it corresponds, in the sense of Curry and Howard, to an expressive type system for a specific randomized λ-calculus, namely a variant of the probabilistic event λ-calculus [57]. Proofs for formulae of **iCPL**$_0$ (resp., types) do not guarantee that validity (resp., termination) holds, but also *reveal* the underlying probability. Furthermore, by endowing our counting-type systems with intersection operators, we obtain a system *precisely* capturing the probabilistic behavior of λ-terms.

6.1 Background

Early in its history, the notion of algorithm was extended in a quantitative sense, obtaining efficient solutions to several computational problems. In these cases, elementary computation steps are not *purely* deterministic anymore, but enable probabilistic outcomes. From the viewpoint of PL theory, a probabilistic program is an algorithm which is able to make random choices and, thus, the execution of which can be described by a stochastic model. Concretely, a classical program defines a deterministic input-output relation: given an input, it terminates or not. On the contrary, a probabilistic program generates a probabilistic distribution over the set of possible outcomes: given an input, it terminates *with a given probability* and – even when terminating with probability 1 – it can possibly run for infinitely many steps.

On the other hand, among the many ways in which mathematical logic influenced PL theory, the CHC is certainly one of the most intriguing and signif-

icant. Traditionally, the correspondence identified by Curry [51] and formalized by Howard [115] relates intuitioinstic **PL** and the simply-typed λ-calculus. Though, it holds in other contexts too and, indeed, in the last fifty years more sophisticated type systems have been put in relation with logical formalism: from polymorphism [92, 95] to various forms of session typing [38, 224], from control operators [165] to dependent types [144, 198]. Yet, these generalizations do not involve probabilistic effects. In the following chapters, we will bridge this gap by introducing a probabilistic intuitionistic logic and a counting-typed randomized λ-calculus, thus providing a probabilistic version of the CHC:

$$\frac{\text{intuitionistic } \textbf{PL}}{\text{simply typed } \lambda\text{-calculus}} = \frac{\text{intuitionistic version of } \textbf{CPL}}{\text{counting-typed probabilistic } \lambda\text{-calculus}}$$

6.1.1 On the Versatility of the λ-Calculus

In the 1930s, Church and Curry introduced two new, embryonal models of computation: combinatory logics and the λ-calculus. In particular, the λ-calculus – the core notion of which is β-reduction – has turned out to be extremely versatile. Indeed, in subsequent years, a collection of formal systems (with different grammars) have been defined around the heart of these models [112].

Historical Overview. Combinators were first presented in a public talk in 1920 by the mathematician Schönfinkel [187] and, in 1927, combinators and combinatory logics were (independently) conceived by Curry as well. A few years later, the λ-calculus was formally defined by Church and Curry, in relation to studies on *effective computability*, which – according to Church's conjecture – can be formalized in terms of λ-definability. This calculus was further studied and developed together with Rosser and Kleene. In 1936, Kleene showed that λ-definability is equivalent to Gödel-Herbrand recursiveness [127]. When in 1936/37 TMs were defined [215, 216], also Turing's notion of computability was proved equivalent to λ-definability. Then, in the 1940s, a new area of research was initiated by Church's introduction of a typed version of the λ-calculus, as a means to avoid paradoxes.

Nevertheless, until about the 1960s, combinatory logics and λ-calculi were studied by small communities only. This picture substantially changed with the development of functional PLs. In the late 1950s, McCarthy introduced the functional style of programming by designing `LISP`, the first higher-order language. `LISP` was explicitly inspired by the λ-calculus (and is, indeed, based on a form of "λ-notation").[1] Thus, the properties of functional languages reflect those of abstract λ-calculi; for example typed languages are safer, as errors are *statically* checked, while untyped ones are more flexible. From then on, λ-calculi have been recognized as fruitful tools in the design, implementation and theory of PLs, and have soon attracted fast-growing interest.

[1] The λ-calculus has had a profound impact on the development of functional programming languages, both typed and untyped, e.g. `GEDANKEN`, `Scheme` and `ML`, and the purely functional `Haskell` and `Miranda`.

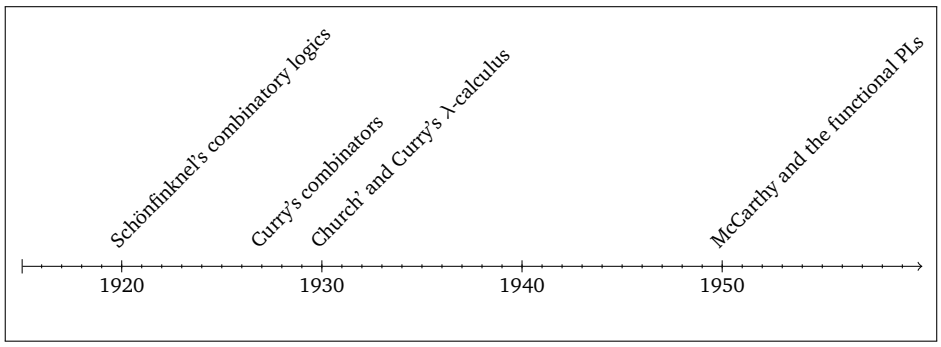


Figure 6.1: Combinatory Logics and the λ-Calculus

A Versatile Computational Model. From the mathematical viewpoint, the λ-calculus is a theory of "functions as rules", defining how to determine values from arguments. A λ-term, say $\lambda x.\mathtt{t}$, can be seen as the description of a nameless function that, given x, produces t. As Kleene showed, this calculus represents a model for the intuitive notion of computable function, in the same way as Turing's machines or Herbrand, Gödel and Kleene's partial recursive functions do. In the λ-calculus, the notion of computation is expressed in a clear way by β-reduction, obtaining a model which is dynamic (as TMs), but, at the same time, abstracts from implementation details (as recursive functions).

For a computer scientist, the λ-calculus is an abstract model for functional programs and, as such, has had a profound impact on the development of programming language theory. In functional languages, functions are treated as "first-class citizen" – indeed, λ-terms represent both programs and data – and computation consists in the evaluation of expressions in a given environment, that is to a *process of reduction*. This way the λ-calculus can be seen as a paradigmatic (higher-order) PL.[2] Moreover, due to the absence of side-effects and the related referential transparency, properties – like correctness – are easier to study for functional languages than for the corresponding imperative ones. The λ-calculus being a *pure* model, it is crucial to analyze properties of programs in the abstract setting and to design languages, think, for example, of how advanced type theory can support program verification and computer-assisted reasoning.

Reduction Strategies and Termination. One of the aspects characterizing functional languages is the reduction strategy they are associated with. Indeed, when an expression is made of more than one redex, one can perform rewriting steps in different orders and the order of evaluation is precisely prescribed by

[2]Notably, also the terminology for constructs in functional programs is the same as that of the λ-calculus, e.g. *redex* for *red*ucible *ex*pression of the form ((`fnx` $\Rightarrow$ `body`)`arg`) or *value* for an expression which cannot be rewritten.

the *reduction strategy*, that is a policy guiding the choice for a β-redex to be reduced. One of the most common evaluation strategies is *call by value* (CbV, for short), consisting in never reducing a redex when its argument is not a value. Another common strategy is *call by name* (CbN, for short), according to which arguments are not evaluated before the function is called. So in CbN, the redex can be reduced even when its argument is not a value.[3]

Due to the absence of side-effects, the result of a computation, *when defined*, is not determined by the reduction strategy – that is, different strategies never lead to different results. Otherwise said the result returned by a (terminating) computation does not depend on the language in which the program is written. From the abstract perspective, this corresponds to the fact that in the λ-calculus reduction is *confluent*, meaning that every term has a *unique* normal form. Let us consider a concrete example.

Example 6.1.1. An interpreter based on the CbV reduction strategy – for instance that of `Scheme` – would evaluate the term $(\texttt{return}(*1(+2\ 2)))$ as follows:

$$(\texttt{return}(*\ 1\ (+2\ 2))) \to_{\mathsf{CbV}} (\texttt{return}(*\ 1\ 4)) \to_{\mathsf{CbV}} (\texttt{return}(4)) \to_{\mathsf{CbV}} 4.$$

On the other hand, interpreters supporting CbN evaluation – for example that of (pseudo) `Algol60` – would proceed as follows:

$$\texttt{return}(1 * (2 + 2)) \to_{\mathsf{CbN}} \texttt{return}(2 + 2) \to_{\mathsf{CbN}} \texttt{return}(+4) \to_{\mathsf{CbN}} +4.$$

Clearly, these two reduction sequences are different, but lead to the same value.

Actually, although different strategies cannot reduce to different results (or normal forms), they do not necessarily produce the same output, since – even if a normal form exists – certain strategies might still not terminate. This fact can be concisely illustrated through a simple example.

Example 6.1.2. As is standard, let $\Omega = (\lambda x.xx)(\lambda x.xx)$ and u be a (closed) normal form. Then the evaluation of the term $\mathsf{t} = (\lambda x.\mathsf{u})\Omega$ either produces a normal form or diverges, depending on the strategy it is associated with:

$$\mathsf{u}\ {}_{\mathsf{CbN}} \leftarrow \underline{(\lambda x.\mathsf{u})\Omega} \to_{\mathsf{CbV}} (\lambda x.\mathsf{u})\Omega \to_{\mathsf{CbV}} \cdots$$

Clearly, CbN substitution immediately leads to the normal form u, while, following CbV, which prescribes that the argument is evaluated before applying substitution, evaluation does not terminate.

Notably, the evaluation (*alias* reduction) of the same program (*alias* λ-term) may lead to a normal form based on a specific strategy – in Example 6.1.2, CbN – and diverge under another – in Example 6.1.2, CbV.

[3] As said, each programming language is associated with a default reduction strategy, for example `LISP` and `Scheme` use the CbV strategy, while `Miranda`'s and `Haskell`'s calculation is based on CbN evaluation.

6.1.2 The Curry-Howard Correspondence

Although it is linked to previous intuitions, the CHC has been progressively developed from the 1950s on, offering deep insights on the connection between logical systems and PL theory.

Historical Overview. As said, type systems are widely used in programming theory as an efficient tool to prevent errors: a language based on (static) typing only diverges as a result of the explicit use of diverging constructs. In the 1940s, Church introduced the simply-typed λ-calculus ($\lambda_{\rightarrow}$, for short) as a means to avoid paradoxes resulting from self-application. It was then proved consistent by Rosser. On the other hand, its expressive power is limited if compared to that of the untyped calculus and, from a computational viewpoint, algorithms representable in $\lambda_{\rightarrow}$ are restricted to terminating ones.

Together with the development of $\lambda_{\rightarrow}$, also its relation with Gentzen's deductive systems started to come to light. Insightful results in this direction are known as the *formulae-as-types* or *Curry-Howard correspondence*. These umbrella terms actually denote several aspects of the analogy between the two "systems". On the one hand, there is a *static* correspondence, first discovered by Curry (and Feys), between intuitionistic formulae and proofs, on one side, and types and (typed) terms of $\lambda_{\rightarrow}$, on the other [51, 52, 50]. On the other, starting from a few observations by Curry himself and Tait [203], an analogy between *dynamic* aspects of computation in $\lambda_{\rightarrow}$ – namely, β-reduction – and normalization for the implicational fragment of intuitionistic natural deduction ($\mathbf{NI}_{\rightarrow}$, for short) was revealed and made precise by Howard [115]. Since then, the CHC has been generalized to more expressive logics and type systems [92, 144, 165, 224].

The Core of the Correspondence. The core of the CHC lies in a *computational* understanding of intuitionistic logic as defined by the BHK interpretation. Intuitionistically, formulae are defined by the set of their proofs. For instance, an implicational formula $F \rightarrow G$, corresponds to a function from F to G, and its proof is a procedure that transforms a proof of F into a proof of G. In the same way, a type $\sigma \Rightarrow \tau$ denotes a function from σ to τ. So, if interpreting $\Rightarrow$ as (intuitionistic) implication, we can see open types as propositional formulae. On the other hand, λ-terms can be interpreted as (operational) representations of programs In $\lambda_{\rightarrow}$, types describe the *functional behavior* of terms. Otherwise said, the type of a term (or program) represents its specification, namely what the program abstractly does. Therefore, under CHC, we interpret terms as corresponding to (constructive) logical deductions: $\mathtt{t} : \sigma_F$ expresses that $\mathtt{t}$ is a proof of F, where F is another, logical notation for σ_F.

Furthermore, type-assignment rules can be seen as annotated versions of $\mathbf{NI}_{\rightarrow}$ (Figure 6.2),[4] while rules for $\mathbf{NI}_{\rightarrow}$ are special ways of constructing func-

[4]Notice that, in [115], the deduction system is presented as a so-called "sequent calculus". Actually, Howard used a hybrid system containing introduction and elimination rules (as natural deduc-

tions: terms in **I**-rules corresponds to constructors and in **E**-rules to selector. Then, a proof of F is linearly codified by a term of type σ_F, and if $\Gamma \vdash \mathtt{t} : \sigma_F$ in $\lambda_\to$, then $rg(\Gamma) \vdash F$ in $\mathbf{NI}_\to$ and vice versa. The correspondence lifts to the dynamics. So, proof normalization, that is detour ($\to$ **I**/$\to$ **E**) conversion in $\mathbf{NI}_\to$

$$\dfrac{\dfrac{\begin{array}{c}[F]^1\\ \vdots\\ G\end{array}}{F \to G}\to \mathbf{I},1 \qquad \begin{array}{c}\vdots\\ F\end{array}}{G}\to \mathbf{E} \quad \rightsquigarrow \quad \begin{array}{c}\vdots\\ F\\ \vdots\\ G\end{array}$$

corresponds to β-reduction

$$\dfrac{\dfrac{x : \sigma_F \vdash \mathtt{t}_G : \sigma_G}{\vdash \lambda x.\mathtt{t}_G : \sigma_F \Rightarrow \sigma_G}(\lambda) \qquad \mathtt{t}_F : \sigma_F \vdash \mathtt{t}_F : \sigma_F}{\mathtt{t}_F : \sigma_F \vdash \lambda x.\mathtt{t}_G(\mathtt{t}_F) : \sigma_G}(@) \quad \to_\beta \quad x : \sigma_F \vdash \mathtt{t}_G[\mathtt{t}_F/x] : \sigma_G$$

Then, the theory of computation (expressed in the language of functions) and proof theory turn out to be two sided of the same coin: (inhabited) types correspond to (provable) propositional formulae, (typed) terms to proofs, **I**-rules translate to constructors, **E**-rules to destructors, redex to detours, reduction to normalization, normal forms to normal proofs, the inhabitation problem to the provability problem, subject reduction to removing detours, the Church-Rosser theorem to the fact that the order of detour conversion is irrelevant, ...

$$\dfrac{\dfrac{\begin{array}{c}[F]^1\\ \vdots\\ G\end{array}}{F \to G}\to \mathbf{I},1 \qquad \begin{array}{c}\vdots\\ F\end{array}}{G}\to \mathbf{E} \qquad \dfrac{\Gamma \vdash \mathtt{t} : \sigma_F \Rightarrow \sigma_G \qquad \Gamma \vdash \mathtt{u} : \sigma_F}{\Gamma \vdash \mathtt{tu} : \sigma_G}$$

$$\dfrac{\begin{array}{c}\vdots\\ F\end{array}}{x : \sigma_F, \Gamma \vdash x : \sigma_F} \qquad \dfrac{\begin{array}{c}[F]^1\\ \vdots\\ G\end{array}}{F \to G}\to \mathbf{I},1 \qquad \dfrac{\Gamma, x : \sigma_F \vdash \mathtt{t} : \sigma_G}{\Gamma \vdash \lambda x.\mathtt{t} : \sigma_F \Rightarrow \sigma_G}$$

Figure 6.2: Comparing $\mathbf{NI}_\to$ and $\lambda_\to$

tion), together with structural rules. From the algorithmic perspective, sequent calculus does not support straightforward CHC. So, subsequent authors have usually interpreted proofs by Howard as in $\mathbf{NI}_\to$.

6.1.3 On Probabilistic λ-Calculi

The pervasive role of stochastic models in a variety of domains – from machine learning to NLP and verification – has prompted a vast body of research also on probabilistic PLs. Generally speaking, these languages support discrete probability distributions providing an operator which models sampling.

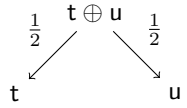
Probabilistic λ-Calculi. Studies on the functional style of probabilistic programming, pioneered by Saheb-Djaromi [180], have attracted increasing interest as allowing higher-order computation and offering a noteworthy level of abstraction. Early works [132, 167, 168, 177], have evolved in a growing body of software developments and theoretical research. To model higher-order probabilistic computation, it is natural to enrich basic λ-calculi by one (or more) probabilistic construct(s). There exist at least two main paradigms of probabilistic λ-calculi, distinguished by the additional operator they provide: *Bayesian* and *randomized* ones.[5] In what follows, we focus on the latter one only.

Randomized λ-Calculi. The usual way to define randomized calculi is to equip untyped λ-calculi with a probabilistic choice operator, as done, for example in [70, 71, 74, 63].

Definition 6.1.1 (Minimal Probabilisitic λ-Terms)**.** *Probabilistic λ-terms* are defined by the following (minimal) grammar:

$$\mathsf{t} := x \;\Big|\; \lambda x.\mathsf{t} \;\Big|\; \mathsf{tt} \;\Big|\; \mathsf{t} \oplus \mathsf{t}.$$

Actually, various choice operators can be considered, the simplest one being a form of binary and fair probabilistic choice. One can form terms, say $\mathsf{t} \oplus \mathsf{u}$, which evolve like t or u depending on the outcome of a probabilistic process, typically corresponding to coin-tossing. The outcome is thus a probabilistic event and different coin flips are taken as independent events:



Probabilistic Confluence and Termination. One of the most relevant properties of ordinary λ-calculi is *confluence*, making its form of non-determinism

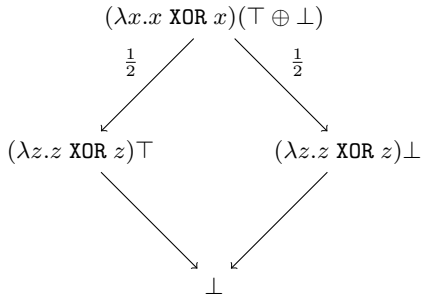
[5] Generally speaking, Bayesian λ-calculi endow the class of terms with two new constructs, one modelling *sampling* (similar to the probabilistic choice operator of randomized λ-calculi) and the other *conditioning* the underlying distribution based on external evidence. This paradigm has been adopted in concrete PLs like `Anglican` and `Church`. For further details, see [53]. Another relevant (orthogonal) distinction is that between languages that handle *discrete* probability distributions and languages that handles *continuous* probability distributions. The approach we pursue basing on counting is limited to discrete distributions. We thank Raphaëlle Crubillé for pointing this distinction to us.

benign since the reduction strategy does not influence the final result of the computation. Nevertheless not all strategies are guaranteed to lead to a normal form. This nice picture is not there anymore when adding the operator $\oplus$, as the resulting probabilistic λ-calculus is non-confluent.

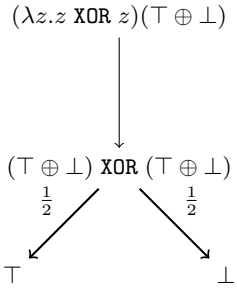
Example 6.1.3 (Non-Confluence [77])**.** Let us consider terms $\mathtt{t} = (\lambda x.x\ \mathtt{XOR}\ x)$ and $\mathtt{u} = (\top \oplus \bot)$, where XOR denotes for exclusive OR (and $\top = \lambda x.y.x$ and $\bot = \lambda xy.y$). Then, when dealing with.

$$\mathtt{tu} = (\lambda z.z\ \mathtt{XOR}\ z)(\top \oplus \bot),$$

- if first reducing $\mathtt{u}$, either $(\lambda z.z\ \mathtt{XOR}\ z)\top$ or $(\lambda z.z\ \mathtt{XOR}\ z)\bot$ are obtained, with equal probability $\frac{1}{2}$. Then, $\mathtt{tu}$ evaluates to $\bot$ with probability 1.



- if reducing the outermost redex first, $\mathtt{tu}$ reduces to $(\top \oplus \bot)\mathtt{XOR}(\top \oplus \bot)$, and, then, the term evaluates either to $\top$ with probability $\frac{1}{2}$ and to $\bot$ again with probability $\frac{1}{2}$:

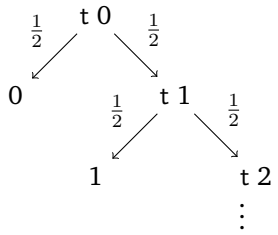


The problem with confluence is handled in the literature in several ways, often consisting in fixing the reduction strategy.[6] Another key property of functional

[6]Indeed, the semantics of a probabilistic λ-calculus usually depends on the choice of a reduction strategy and terms only have meaning in the context of that strategy. When distinguishing between *calculus*, as defined by reduction rules (independently from reduction strategies), and *programming languages* (associated with a reduction strategy) [173], it emerges that probabilistic computation works well only in the latter case (that is, when the execution of a program can be associated with CbV [63, 77] or CbN [74, 77]).

programs is termination. As said, in presence of probabilistic choice, termination becomes a probabilistic event and, as such, happens *with a certain probability*. Even if the termination probability is 1 (almost sure termination), the degree of certitude is typically not reached in any finite number of steps, but appears *as a limit* [180, 145, 29]. For instance, in the informal Example 6.1.4 below, the probability that the evaluation of the term terminates is $\sum_{i=0}^{\infty} \frac{1}{2^{n+1}}$, namely 1. As such $n\mathtt{t}$ is said to be an almost surely terminating term.[7]

Example 6.1.4 (Probabilistic Termination [53])**.** Let $\mathtt{t}\ n$ deterministically reduce to $n \oplus (\mathtt{t}\ (n+1))$, where $\oplus$ is *fair* probabilistic choice operator and $n \in \mathbb{N}$. At each reduction step, the term $\mathtt{t}\ n$ either reduces to n with probability $\frac{1}{2}$, or proceeds as $n\ \mathtt{t}$, again with probability $\frac{1}{2}$. In particular, the computational behavior of $\mathtt{t}\ 0$ can be graphically represented as the infinite binary tree below:



6.2 To a Probabilistic Correspondence (and Beyond)

No logical counterpart, in the sense of Curry and Howard, was established for the class of PLs and type systems involving *probabilistic effects*, and these type-theoretic accounts have recently been put forward in various ways, e.g. type systems based on sized types [56], intersection types [31], or type systems in the style of so-called amortized analysis [228]. In all the aforementioned cases, a type system is built by modifying well-known type systems for deterministic languages *without* being guided by logic, and instead incepting inherently quantitative concepts directly from probability theory. Our perspective is rather different: we search for a logical system to possibly play the role of **PL** in suggesting meaningful and expressive type systems for a λ-calculus endowed with probabilistic choice effects.

A tempting answer is to start from modal logic, which is known to correspond in the Curry-Howard sense to staged computation and to algebraic effects [49, 65, 234]. But – with a few notable exceptions (for example [87]) – there remains one aspect of randomized computation that modal logic fails to capture, namely the *probability* of certain notable events, typically termina-

[7] Observe that, as noticed by Dal Lago, this tree do not correspond to the reduction tree of an ordinary λ-term. Here, branching is modelling a form of nondeterminism coming from a somehow external choice of the next redex to fire (and do not rely on the presence of a choice operator occurring inside the given term).

tion.[8] For example, in many of the probabilistic type systems mentioned above, a term t receives a type that captures the fact that t has a certain probability q (perhaps strictly smaller than 1) of reducing to a value. This probability is an essential part of what we want to observe about the dynamic behavior of t and, as such, has to be captured by its type, at least if one wants the type system to be expressive.

6.2.1 Randomized Programs and Counting Quantifiers

As seen, a probabilistic functional program is a functional program with the additional ability of sampling from some distributions, or to proceed by performing some form of discrete probabilistic choice.[9] Consequently, program evaluation becomes an essentially stochastic process, while programs satisfy a given specification *up to* a certain probability.

Example 6.2.1. Let us consider the λ-term

$$\mathsf{t}_{\mathsf{half}} = \lambda x.\lambda y.x \oplus y$$

where $\oplus$ is a binary infix operator for fair probabilistic choice. When applied to two arguments t and u assumed to be normal form, the evaluation on $\mathsf{t}_{\mathsf{half}}$ results in either t, with probability $\frac{1}{2}$, or in u, again with probability $\frac{1}{2}$.

When trying to consider $\mathsf{t}_{\mathsf{half}}$ as a proof of a standard propositional formula, it becomes clear that **PL** is not rich enough to capture the behavior above. Indeed, given that $\mathsf{t}_{\mathsf{half}}$ is a function with two arguments, it is natural to see it as a proof of an implication $F \to G \to H$, that is, following the BHK interpretation, as a function turning a proof of F and a proof of G into a proof of H. What is H then? Is it F or G? Actually, it could be both *with some degree of uncertainty*, but **PL** is not able to express all this. It is at this point counting quantifiers can come to the rescue. Indeed, while intutionistic **PL** cannot represent the behavior of H, this can be expressed by the counting formulae $\mathbf{C}^{1/2}F$ – saying that F has probability at least probability $\frac{1}{2}$ to hold – and by $\mathbf{C}^{1/2}G$ – equally saying that G has probability at least $\frac{1}{2}$ to hold.

Extending this intuition, we introduce the intuitionistic counting logic **iCPL**. Its language extends that of intuitionistic **PL** with Boolean variables and the counting quantifier $\mathbf{C}^q$, to be interpreted as in Chapter 3. Intuitively, if a proof of a formula F can be seen as a deterministic program satisfying the specification F, a proof of $\mathbf{C}^q F$ corresponds to a *probabilistic* program that satisfies the specification F *with probability* q. This logic (actually the corresponding computational fragment), together with its natural deduction system, provides the logical part of our probabilistic CHC. Our main result consists in showing that intuitionistic, counting proofs correspond to functional probabilistic programs, and that normalization in this logic describes probabilistic evaluation.

[8] Notably, several other properties – such as reachability and safety – can be reduced to termination. For further details, see [130].

[9] We are not concerned with sampling from continuous distributions, nor with capturing any form of conditioning.

6.2.2 Making CbN and CbV Evaluation Coexist

There are two main evaluation strategies: CbN might duplicate choices before evaluating them, while CbV evaluates choices before possibly duplicating their outcomes. Then, the probability of termination of a program might differ depending on the chosen strategy.

Example 6.2.2. Consider the following term:

$$2 = \lambda yx.y(yx),$$

corresponding to the second Church numeral and its application to $\mathsf{I} \oplus \Omega$,

$$2(\mathsf{I} \oplus \Omega),$$

where $\mathsf{I} = \lambda x.x$ and $\Omega = (\lambda x.xx)\lambda x.xx$ is diverging.

- Under CbN the redex $2(\mathsf{I}\oplus\Omega)$ first reduces $\lambda x.(\mathsf{I}\oplus\Omega)((\mathsf{I}\oplus\Omega)x)$, and then to different terms, each with probability $\frac{1}{4}$. In particular, since only $\lambda x.\mathsf{I}(\mathsf{I}x)$ converges, the global probability of convergence is $\frac{1}{4}$.
- Under CbV, we first evaluate $\mathsf{I} \oplus \Omega$ and then passes the result to 2, hence returning either the converging term $\mathsf{I}(\mathsf{I}x)$ or the diverging one, each with probability $\frac{1}{2}$.

When interpreted as a logical proof, the Church numeral would prove different counting formulae depending on the reduction strategy we are considering. Given that $\mathsf{I} \oplus \Omega$ proves $\mathbf{C}^{\frac{1}{2}}(F \to F)$ in the CbN case, 2 proves $\mathbf{C}^{1/2}(F \to F) \to F \to \mathbf{C}^{1/4}F$. Indeed, only in one case out of four it yields a proof of F. On the other hand, in the CbV case, 2 proves the formula $\mathbf{C}^{1/2}(F \to F) \to F \to \mathbf{C}^{1/2}F$, as it yields a proof of F in one case out of two.

Going back to the computational model, we have seen that in the literature on randomized λ-calculi, the apparent incompatibility of CbN and CbV is usually resolved by considering calculi associated with one or the other strategy, but the observation above suggests that, if functional programs are typed using counting quantifiers, it should become possible to make the two evaluation strategies coexist, by assigning them different types. As a starting point, we consider a few recent approaches already offering ways to make CbN and CbV mutually compatible [54, 57, 73, 77] and, in particular, in the *probabilistic event λ-calculus*, in which the choice operator $\oplus$ is decomposed into two different syntactic objects, yielding a confluent calculus [57]. This calculus constitutes an ideal candidate for our CHC, especially when dealing with the dynamic aspects and, indeed, it would be our (untyped) basis to define the computational side of the probabilistic correspondence.

6.2.3 Capturing Probability of Normalization Through Types

As said, a fundamental quantitative property we would like to observe using types is the *probability of termination*, but as known reduction in standard type

systems for randomized λ-calculus is purely deterministic. So, what notions of probabilistic termination should we actually observe? First, we notice that rather than evaluating programs by implementing probabilistic choices, reduction in the probabilistic event λ-calculus has the effect of progressively generating the *full tree* of outcomes of (sequences of) probabilistic choices, giving rise to a *distribution* of values. Therefore, given a term $\mathtt{t}$, rather than asking whether *some* or *all* reductions of $\mathtt{t}$ terminate, it makes sense to ask *what is the probability* for a normal form to be found by giving all probabilistic outcomes of $\mathtt{t}$. Following these intuitions, in Chapter 10, we show that when the type $\mathbf{C}^q\sigma$ is assigned to a program $\mathtt{t}$, the value $q \in \mathbb{Q} \cap (0, 1]$ provides a lower bound for the actual probability of finding a (head) normal form in the development of $\mathtt{t}$. Then, we extend this type system with an intersection operator, which allows us to obtain also as upper bound for probability, and thus fully characterize the distribution of values associated with a term.

6.3 Outline of Part II

This part of the thesis is devoted to the presentation of our probabilistic CHC. We first present its main ingredients: on the logical side we introduce an *intuitionistic* version of **CPL**, while the computational part is defined by (a slight variation of) the probabilistic event λ-calculus by [57] together with an expressive type systems based on the notion of counting quantifiers. Finally, we establish their correspondence showing that, in this context, proofs (resp., types) do not guarantee that validity (resp., termination) holds, but rather *reveal* the underlying probability.

Concretely, the presentation is structured as follows:

- In Chapter 7 we introduce an intuitionistic version of **CPL** together with a Kripke-style semantics based on the Borel σ-algebra of the Cantor space. We also identify its "computational fragment" and design its sound and complete natural deduction system.
- In Chapter 8 we present a variant of the probabilistic event λ-calculus [57], which is a vehicle calculus capable of expressing both CbV and CbN. In our case, the standard underlying probability space is replaced by the Cantor space.
- In Chapter 9 we define counting-type system(s) decorating derivations with terms of our probabilistic λ-calculus. We show that natural deduction derivations translate into typing derivations in it, with normalization precisely corresponding to reduction.
- Finally, in Chapter 10, we introduce an intersection type system derived from the one defined in Chapter 9, and prove it able to capture the normalization probability of terms in the given randomized λ-calculus. We conclude by comparing our results and systems with the ones offered in the literature.

Chapter 7

The Logical Side: iCPL

In Section 7.1, we introduce an intuitionistic counting logic, called **iCPL**, and the corresponding (sound and complete) natural deduction system. It is self-contained, but in the perspective of CHC, this logic is both *too much* and *not enough*. So, in Section 7.2, we slightly modify it to obtain the minimal, computational fragment, called **iCPL**$_0$, which perfectly meets our needs. Also in this case, we provide a natural deduction rule system together with its normalization proof. As we shall see in Chapter 9, this calculus is strongly connected with probabilistic type systems for CbV evaluation. Before concluding, in Section 7.3, we introduce an alternative proof system for **iCPL**$_0$ able to capture CbN evaluation.

7.1 Intuitionistic Counting Propositional Logic

We introduce a counting propositional logic, called **iCPL**, which extends the standard intuitionistic system with Boolean variables and counting quantifiers. Intuitively, this logic combines *constructive* reasoning, corresponding under CHC to functional programming, with *semantic* reasoning (in the form of Boolean formulae and their models), corresponding to discrete probabilistic reasoning. Then, formulae of **iCPL** are somehow *hybrid*, comprising both a countable set of intuitionistic propositional variables, $\mathfrak{P} = \{P_1, P_2, \dots\}$, and named Boolean variables of the form x_a^i where $i \in \mathbb{N}$ and a is a name.

Before introducing the semantics of **iCPL** in a formal way, let us consider two simple examples to clarify the intuitive meaning of intuitionistic, counting formulae.

Example 7.1.1. Let F be the following formula of **iCPL**:

$$F : P_1 \to P_2 \to (x_a^0 \wedge P_1) \vee (\neg x_a^0 \wedge P_2).$$

Intuitively, proving F amounts to showing that, whenever both P_1 and P_2 hold, then either x_a^0 and P_1 or $\neg x_a^0$ and P_2 do. Let us now suppose to test F against

some "environment" ω in $(2^{\mathbb{N}})^{\{a\}}$: given assumptions P_1 and P_2, we conclude P_1 and $\boldsymbol{x}_a^0$ when $\omega(0) = 1$ – i.e. if ω satisfies $\boldsymbol{x}_a^0$ –, but conclude P_2 and $\neg\boldsymbol{x}_a^0$ when $\omega(0) = 0$ – i.e. if ω does not satisfy $\boldsymbol{x}_a^0$. Otherwise said, given the "environment" ω, a proof of F would be something of the form:

$$\lambda xy.x \oplus_a y.$$

where $\oplus_a$ is a (fair) probabilistic operator, depending on a. So, assuming that ω is uniformly sampled, what are the chances that our strategy will actually yield a proof of F? As seen, there are two possible cases, and in both of them we get a proof of one of the disjuncts $\boldsymbol{x}_a^0 \wedge P_1$ and $\neg\boldsymbol{x}_a^0 \wedge P_2$, thus a proof of F. We can conclude that a proof of F is obtained with probability 1 – that is we conclude $\mathbf{C}_a^1 F$ –, which no more depends on any "environment". Observe that this proof looks precisely as the closed term:

$$\nu a.\lambda xy.x \oplus_a y.$$

Example 7.1.2. Let us consider another formula of **iCPL**:

$$G := P_1 \to P_2 \to \boldsymbol{x}_a^0 \wedge P_1,$$

and let ω be an "environment". For any ω such that $\omega(a)(0) = 1$, we can build a proof of G under ω by concluding $P_1 \wedge \boldsymbol{x}_a^0$. For ω such that $\omega(a)(0) = 0$, we are instead not able to build a proof of G. Then, when we build the finial proof by probabilistically summing over the possible cases for $\omega_a(0)$ (as in Example 7.1.1 above), not all the environments ω would give a correct proof. Yet, when the proof they provide is not correct, we obtain a proof of $\mathbf{C}^q G$ (instead of a proof of G) and $1 - q$ corresponds to the probability of the set of all environments ω with incorrect proof.

7.1.1 Syntax and Semantics of iCPL

Syntax. As anticipated, the grammar of **iCPL** is obtained by endowing that of intuitionistic **PL** with named expressions and C-quantified formulae.

Definition 7.1.1 (Formulae of **iCPL**)**.** *Formulae of* ***iCPL*** are defined by the grammar below:

$$F ::= \top \;\Big|\; \bot \;\Big|\; \boldsymbol{x}_a^i \;\Big|\; P \;\Big|\; F \wedge F \;\Big|\; F \vee F \;\Big|\; F \to F \;\Big|\; \mathbf{C}_a^q F,$$

with $a \in \mathscr{A}$, $P \in \mathfrak{P}$ and $q \in \mathbb{Q} \cap [0, 1]$.

Semantics. A natural semantics for formulae of **iCPL** is given in terms of Kripke-like structures.

Definition 7.1.2 (**iCPL**-Structure)**.** An ***iCPL****-structure* is a triple $\mathcal{M} = (W, \preceq, \mathcal{I})$, where W is a countable set, $\preceq$ is a preorder on W, and $\mathcal{I} : \mathfrak{P} \to W^{\uparrow}$ is an interpretation of propositional atoms to the set $W^{\uparrow}$ of upper-closed subsets of W.

The interpretation of formulae in **iCPL**-structures is defined combining a set W of worlds – for the interpretation of intuitionistic propositional variables – with the choice of an element from the Cantor space – for the interpretation of Boolean variables.

Definition 7.1.3 (Semantics for **iCPL**)**.** For any **iCPL**-structure $\mathcal{M} = (W, \preceq, \mathcal{I})$ and finite set X, we define the relation $w, \omega \Vdash^X_{\mathcal{M}} F$, where $w \in W, \omega \in (2^{\mathbb{N}})^X$ and $\mathsf{FN}(F) \subseteq X$, by induction:

$$
\begin{array}{rll}
w, \omega \nVdash^X_{\mathcal{M}} \bot & & \\
w, \omega \Vdash^X_{\mathcal{M}} \top & & \\
w, \omega \Vdash^X_{\mathcal{M}} x^i_a & \text{iff} & \omega(a)(i) = 1 \\
w, \omega \Vdash^X_{\mathcal{M}} P & \text{iff} & w \in \mathcal{I}(P) \\
w, \omega \Vdash^X_{\mathcal{M}} F \wedge G & \text{iff} & w, \omega \Vdash^X_{\mathcal{M}} F \text{ and } w, \omega \Vdash^X_{\mathcal{M}} G \\
w, \omega \Vdash^X_{\mathcal{M}} F \vee G & \text{iff} & w, \omega \Vdash^X_{\mathcal{M}} F \text{ or } w, \omega \Vdash^X_{\mathcal{M}} G \\
w, \omega \Vdash^X_{\mathcal{M}} F \to G & \text{iff} & \text{for all } w \preceq w', w', \omega \Vdash^X_{\mathcal{M}} F \text{ implies } w', \omega \Vdash^X_{\mathcal{M}} G \\
w, \omega \Vdash^X_{\mathcal{M}} \mathbf{C}^q_a F & \text{iff} & \mu\Big(\Big\{\omega' \in 2^{\mathbb{N}} \mid w, \omega +_a \omega' \Vdash^{X \cup \{a\}}_{\mathcal{M}} F\Big\}\Big) \geq q,
\end{array}
$$

where $\omega +_a \omega' \in (2^{\mathbb{N}})^{X \cup \{a\}}$ is given by $(\omega +_a \omega')(b)(n) = \omega(b)(n)$ for any $b \in X$ and $(\omega +_a \omega')(a)(n) = \omega'(n)$.

Using the properties of the Borel σ-algebra $\mathcal{B}_X$, it is shown that for any $w \in W$ and formula F, the set $\{\omega \in (2^{\mathbb{N}})^X \mid w, \omega \Vdash^X_{\mathcal{M}} F\}$ is a Borel set and, thus, is measurable. The notions of validity of a formula in a model and of validity are standard.

Notation 7.1.1. We write $\Gamma \Vdash^X_{\mathcal{M}} F$, when for all $w \in W$ and $\omega \in (2^{\mathbb{N}})^X$, whenever $w, \omega \Vdash^X_{\mathcal{M}} \Gamma$ holds, also $w, \omega \Vdash^X_{\mathcal{M}} F$ holds. Moreover, $\Gamma \vDash F$ holds when for any **iCPL**-structure $\mathcal{M}$ and $X \supseteq \mathsf{FN}(F)$, $\Gamma \Vdash^X_{\mathcal{M}} F$ holds.

Notation 7.1.2. We use $\neg F$ as a shorthand for $F \to \bot$.

7.1.2 Proof Theory of iCPL

We define a sound and complete natural deduction system, called $\mathbf{ND}_{\mathbf{iCPL}}$.[1] In particular, we start from usual rules for intuitionistic logic, and add the excluded middle (**EM**, for short) for Boolean variables, namely $x^i_a \vee \neg x^i_a$, as well as suitable rules and axioms for counting quantifiers.

Definition 7.1.4. Rules for $\mathbf{ND}_{\mathbf{iCPL}}$ are illustrated in Figure 7.1, and satisfies the following provisos:

i. for readability's sake, we use $\vdash$ instead of $\vdash^X$ for sequents in the form $\Gamma \vdash F$, assuming $\mathsf{FN}(\Gamma), \mathsf{FN}(F) \subseteq X$.

[1] For the proof of soundness and completeness of $\mathbf{ND}_{\mathbf{iCPL}}$ with respect to **iCPL** semantics, see [12, Section A.4] or [11].

ii. in the rule **CI** it is assumed that $\mathsf{FN}(\mathscr{b}) \subseteq \{a\}$,

together with all instances of the two axiom schemas below:

$$\mathbf{C}_a^q(F \vee G) \to F \vee (\mathbf{C}_a^q G) \qquad (a \notin \mathsf{FN}(F)) \qquad (\mathbf{C}\vee)$$

$$\neg\mathbf{C}_a^q \mathscr{b} \qquad (\mathsf{FN}(\mathscr{b}) \subseteq \{a\}, \mu_{\mathscr{C}}(\mathscr{b}) < q). \qquad (\mathbf{C}\bot)$$

As for **CPL**$_0$ and **CPL**, some rules include semantic premisses, which intuitively ask for an oracle to count the models of the Boolean formulae.

Classical Identity

$$\frac{}{\Gamma \vdash x_a^i \vee \neg x_a^i}\ \mathbf{C_{id}}$$

Intuitionistic Identity

$$\frac{}{\Gamma, F \vdash F}\ \mathbf{I_{id}}$$

Logical Rules

$$\frac{}{\Gamma \vdash \top}\ \top\mathbf{I} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash F}\ \bot\mathbf{E}$$

$$\frac{\Gamma \vdash F \qquad \Gamma \vdash G}{\Gamma \vdash F \wedge G}\ \wedge\mathbf{I}$$

$$\frac{\Gamma \vdash F \wedge G}{\Gamma \vdash F}\ \wedge\mathbf{E}_1 \qquad \frac{\Gamma \vdash F \wedge G}{\Gamma \vdash G}\ \wedge\mathbf{E}_2$$

$$\frac{\Gamma \vdash F}{\Gamma \vdash F \vee G}\ \vee\mathbf{I}_1 \qquad \frac{\Gamma \vdash G}{\Gamma \vdash F \vee G}\ \vee\mathbf{I}_2$$

$$\frac{\Gamma \vdash F \vee G \qquad \Gamma, F \vdash H \qquad \Gamma, G \vdash H}{\Gamma \vdash H}\ \vee\mathbf{E}$$

$$\frac{\Gamma, F \vdash G}{\Gamma \vdash F \to G}\ \to\mathbf{I} \qquad \frac{\Gamma \vdash F \to G \qquad \Gamma \vdash F}{\Gamma \vdash G}\ \to\mathbf{E}$$

Counting Logic

$$\frac{\Gamma, \mathscr{b} \vdash F \qquad \mu([\![\mathscr{b}]\!]) \geq q}{\Gamma \vdash \mathbf{C}_a^q F}\ \mathbf{CI}$$

where $a \notin \Gamma$

$$\frac{\Gamma \vdash \mathbf{C}_a^q F}{\Gamma \vdash F}\ \mathbf{CE}_1$$

where $a \notin F, q > 0$

$$\frac{\Gamma \vdash \mathbf{C}_a^q F \qquad \Gamma, F \vdash G}{\Gamma \vdash \mathbf{C}_a^{qs} G}\ \mathbf{CE}_2$$

where $a \notin \Gamma$

Figure 7.1: Rules of $\mathbf{ND}_{\mathbf{iCPL}}$

Example 7.1.3. For example, let us consider $\mathbf{CI}$,

$$\frac{\Gamma, \mathcal{b} \vdash F \qquad \mu([\![\mathcal{b}]\!]) \geq q}{\Gamma \vdash \mathbf{C}_a^q F}\ \mathbf{CI}$$

with the proviso that a does not occur in Γ and is the only name occurring in $\mathcal{b}$. This rule intuitively says that if there is a proof of F under assumptions Γ and $\mathcal{b}$, and if a randomly chosen valuation of a has chance at least q of being a model of $\mathcal{b}$, then a proof of $\mathbf{C}_a^q F$ from Γ can be built.

A Digression on $\mathbf{ND}_{\mathbf{iCPL}^-}$. It is also possible to introduce a variation of $\mathbf{ND}_{\mathbf{iCPL}}$, called $\mathbf{ND}_{\mathbf{iCPL}^-}$, which is defined as the system above, but does not include the axiom schemas $\mathbf{C}\vee$ and $\mathbf{C}\bot$.

Remark 7.1.1. *The distinction between these two systems corresponds to the one usually defined in the context of intuitionistic modal logic (**IML**, for short). Indeed, standard axiomatizations of **IML** include the axioms:*

$$\diamond(F \vee G) \to \diamond F \vee \diamond G \qquad (\diamond\vee)$$
$$\neg \diamond \bot, \qquad (\diamond\bot)$$

*which do not have a clear computational interpretation. Instead, a CHC can be defined for an axiomatization of **IML**, usually referred to as* constructive modal logic*, which does not include these two axioms* $(\diamond\vee)$ *and* $(\diamond\bot)$.

In a similar way, we show that provability in $\mathbf{ND}_{\mathbf{iCPL}^-}$ corresponds, under the decomposition provided by Lemma 7.2.1 below to provability in the Curry-Howard proof system $\mathbf{ND}_{\mathbf{iCPL}}$, while axioms $(\mathbf{C}\vee)$ and $(\mathbf{C}\bot)$ cannot be interpreted in a similar way.

7.2 The Computational Fragment of iCPL

From the perspective of the CHC, the system $\mathbf{ND}_{\mathbf{iCPL}}$ is still not appropriate. As **iCPL** contains Boolean logic, in order to relate proofs and programs, one should first choose among the several existing constructive interpretations of classical logic. Yet, in our previous examples, Boolean formulae are not proved, but used as *semantic constraints* that programs may or may not satisfy. For instance, we say that a program depending on some event ω, yields a proof of F *when* ω *satisfies* $\mathcal{b}$, or that the program has q chances of yielding a proof of F *when* $\mathcal{b}$ *has measure at least* q. It would then be desirable to somehow separate purely *constructive reasoning* from Boolean *semantic reasoning* within formulae and proofs of **iCPL**. The following lemma suggests that this is indeed possible.[2]

[2]Further details and a syntactic version of the following proof sketch can be found in [12, 11, Appendix A.2].

Lemma 7.2.1 (Decomposition Lemma)**.** *For any formula F of* **i*CPL****, there exist Boolean formulae $\mathscr{b}_v$ and purely intuitionistic formulae F_v – i.e. formulae containing no Boolean variables – where v varies over all possible valuations of the Boolean variables in F, such that:*

$$\vDash F \leftrightarrow \bigvee_v \mathscr{b}_v \wedge F_v$$

Proof Sketch. Let (i.) $\mathscr{b}_v$ be the formula characterizing v, i.e. the conjunction of all variables true in v and of all negations of variables false in v, and (ii.) F_v be obtained from F by replacing each Boolean variable by either $\bot$ or $\top$, depending on its value under v. □

By Lemma 7.2.1, any sequent $\Gamma \vdash F$ of **iCPL** can be associated with a *family* of intuitionistic sequents of the form $\Gamma_v, \mathscr{b}_v \vdash F_v$, where v ranges over all valuations of the Boolean variables of Γ and F, with $\Gamma_v, \mathscr{b}_v$, and F_v as in Lemma 7.2.1.

Notation 7.2.1. These special sequents $\Gamma_v, \mathscr{b}_v \vdash F_v$ are denoted as $\Gamma \vdash \mathscr{b} \rightarrowtail F$.

Notably, sequents of the form $\Gamma \vdash \mathscr{b} \rightarrowtail F$ have a natural computational interpretation, as expressing program specifications of the form "Π yields a proof of F from Γ whenever its sampled function satisfies $\mathscr{b}$". By the way, Lemma 7.2.1 ensures that, *modulo* Boolean reasoning, logical arguments in **iCPL** can be reduced to (families of) arguments of this kind.

7.2.1 Syntax and Semantics of iCPL$_0$

Let us now define **iCPL**$_0$, a fragment of **iCPL** made of *purely intuitionistic* formulae of **iCPL**. Observe that for simplicity's sake, we take implication as the only connective. Indeed, it is enough to define our CHC, but all other propositional connectives could have been added.

Definition 7.2.1 (Formulae of **iCPL**$_0$)**.** *Formulae of* **i*CPL***$_0$ are defined by the following grammar:

$$F ::= P \;\Big|\; F \to F \;\Big|\; \mathbf{C}^q F,$$

with $q \in \mathbb{Q} \cap [0,1]$.

As formulae do not contain Boolean variables, counting quantifiers in **iCPL**$_0$ are not named.

Definition 7.2.2 (Semantics for **iCPL**$_0$)**.** Given a structure $\mathcal{M} = (W, \preceq, \mathcal{J})$, we define by induction the relation $w, \omega \Vdash_{\mathcal{M}} F$, where $w \in W$ and $\omega \in 2^{\mathbb{N}}$:

$$\begin{array}{rll} w, \omega \Vdash_{\mathcal{M}} P & \text{iff} & w \in \mathcal{J}(P) \\ w, \omega \Vdash_{\mathcal{M}} F \to G & \text{iff} & \text{for all } w \preceq w',\ w', \omega \Vdash_{\mathcal{M}} F \text{ implies } w', \omega \Vdash_{\mathcal{M}} G \\ w, \omega \Vdash_{\mathcal{M}} \mathbf{C}^q F & \text{iff} & \mu\big(\{\omega \in 2^{\mathbb{N}} \mid w, \omega \vDash_{\mathcal{M}} F\}\big) \geq q. \end{array}$$

Notation 7.2.2. We use $\mathbf{C}^{q_1 * \cdots * q_n} F$ (or even $\mathbf{C}^{\vec{q}}$, for simplicity) as an abbreviation for $\mathbf{C}^{q_1} \dots \mathbf{C}^{q_n} F$.

Observe that **iCPL**$_0$ is a (very limited) fragment of **iCPL** and, in particular, its semantics is trivial – in the sense that it is "not quantitative" anymore.[3] **iCPL**$_0$ is not very interesting in terms of formulae provability: for $q > 0$, $\mathbf{C}^q F$ is provable whenever F is. Nonetheless, it is relevant because of its proof-theoretic behavior, namely cut-elimination. Otherwise said, due to **iCPL**$_0$, one can turn the system **iCPL** – which is interesting in terms of provability – into a constructive system – which is interesting in relation to proof theory.

7.2.2 Proof Theory of iCPL$_0$

We define a natural deduction system for **iCPL**$_0$, called $\mathbf{ND}_{\mathbf{iCPL}_0}$, obtained by enriching $\mathbf{ND}_{\mathbf{iCPL}}$ with standard rules for intuitionistic connectives adapted to the (labelled) language for sequents of $\mathbf{ND}_{\mathbf{iCPL}_0}$, that is by adding a fixed number of Boolean formulae to the consequent.[4] Indeed, this proof system is still made of natural deduction rules, as $\mathbf{ND}_{\mathbf{iCPL}}$ of Section 7.1.2, but in this case the language is labelled, as for sequent calculi $\mathbf{LK}_{\mathbf{CPL}_0}$ and $\mathbf{LK}_{\mathbf{CPL}_0}$ of Chapters 3 and 4.

Definition 7.2.3 (Sequents of $\mathbf{ND}_{\mathbf{iCPL}_0}$)**.** A *sequent of* $\mathbf{ND}_{\mathbf{iCPL}_0}$ is an expression of the form $\Gamma \vdash \mathcal{b} \rightarrowtail F$, where Γ is a set of **iCPL**$_0$-formulae, $\mathcal{b}$ is a Boolean formula and F is a formula of **iCPL**$_0$.

As in the proof system for **iCPL**, some rules of $\mathbf{ND}_{\mathbf{iCPL}_0}$ involve semantic premisses of the form $\mathcal{b} \vDash \mathcal{c}$ and $\mu([\![\mathcal{b}]\!]) \geq q$. On the other hand, beyond standard intuitionistic rules, $\mathbf{ND}_{\mathbf{iCPL}_0}$ includes structural rules to manipulate Boolean formulae.

Definition 7.2.4 (Proof System $\mathbf{ND}_{\mathbf{iCPL}}$)**.** The *proof system* $\mathbf{ND}_{\mathbf{iCPL}}$ is made of the rules defined in Figure 7.2, where it is assumed that in $\mathbf{CI}$, $\mathsf{FN}(\mathcal{b}) \cap \mathsf{FN}(\mathcal{c}) = \emptyset$.

Intuitively, the rule $\bot\mathbf{R}$ yields *dummy* proofs of any formula, namely proofs which are correct for no possible event. The rule $\mathbf{M}$ combines two proofs Π_1, Π_2 of the same formula into a single proof Π, with the choice depending on the value of some Boolean variable x_a^i (Π is thus something like $\Pi_1 \oplus_a \Pi_2$). The introduction rule for the counting quantifier $\mathbf{CI}$ is similar to the corresponding one in $\mathbf{ND}_{\mathbf{iCPL}}$, saying that if Π in the "environment" $\omega + \omega' \in (2^{\mathbb{N}})^{X \cup \{a\}} \equiv (2^{\mathbb{N}})^X \times 2^{\mathbb{N}}$ yields a proof of F whenever $\omega + \omega'$ satisfies the two *independent* constraints $\mathcal{b}$ and $\mathcal{c}$ (i.e. $\omega \vDash \mathcal{b}$ and $\omega' \vDash \mathcal{c}$), then, by randomly choosing $\omega' \in 2^{\mathbb{N}}$, we have at least $q \geq \mu_{\mathscr{C}}([\![\mathcal{c}]\!])$ chances of getting a proof of F (something like $\nu a.\Pi$). Finally, $\mathbf{CE}$ turns a proof of $F \to G$ into a proof of $\mathbf{C}^q F \to \mathbf{C}^{qs} G$. As we shall see in Chapter 9, this rule captures CbV function application.

[3] In particular, if $q > 0$, $\mathbf{C}^q F$ is equivalent to F (as usual, if $q = 0$, the $\mathbf{C}^0 F$ is equivalent to $\top$).
[4] For further details on the relation between $\mathbf{ND}_{\mathbf{iCPL}}$ and $\mathbf{ND}_{\mathbf{iCPL}_0}$, see [12, 11, Appendix A].

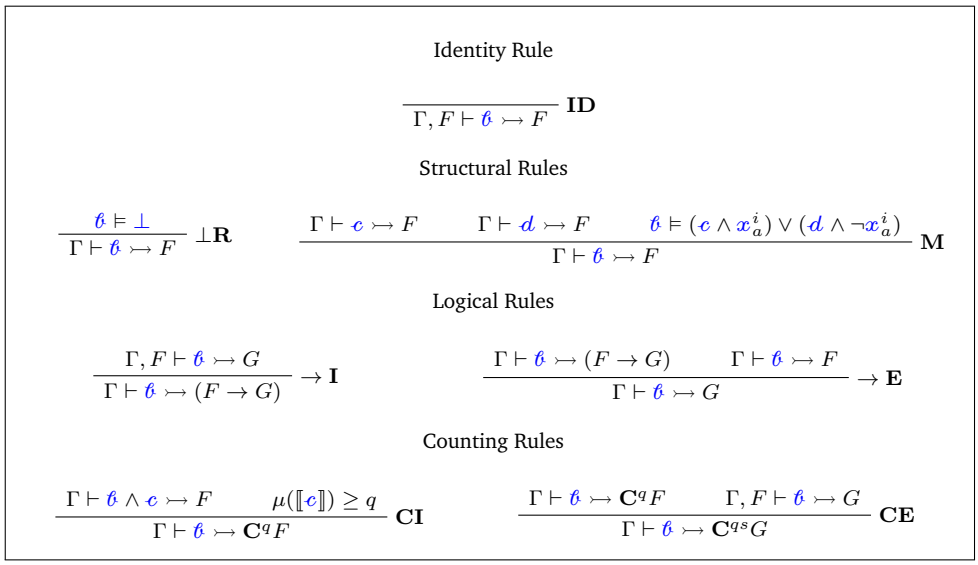


Figure 7.2: Rules of $\mathbf{ND}_{\mathbf{iCPL}_0}$

Example 7.2.1. The following derivation is a proof $\Pi_{\frac{1}{2}\mathrm{id}}$ of $\mathbf{C}^{1/2}(F \to F)$. It is obtained by first "mixing" an exact proof of $F \to F$ with a dummy one. Then, a counting quantifier is introduced.

$$\dfrac{\dfrac{\dfrac{F \vdash x_a^i \rightarrowtail F}{\vdash x_a^i \rightarrowtail (F \to F)} \to \mathbf{I} \qquad \dfrac{}{\vdash \bot \rightarrowtail (F \to F)} \bot\mathbf{R} \qquad x_a^i \vDash x_a^i \vee \bot}{\vdash x_a^i \rightarrowtail (F \to F)}\ \mathbf{M} \qquad \mu([\![x_a^i]\!]) \geq \frac{1}{2}}{\vdash \top \rightarrowtail \mathbf{C}^{1/2}(F \to F)}\ \mathbf{CI}$$

Example 7.2.2. Let us consider the derivation of $\mathbf{C}^q(F \to F) \to F \to \mathbf{C}^{q*q}F$:

$$\dfrac{\dfrac{\dfrac{\Pi' \qquad \Pi''}{\mathbf{C}^q(F \to F), F \vdash \mathbf{C}^q F \qquad \mathbf{C}^q(F \to F), F, F \vdash \mathbf{C}^q F}\ \mathbf{CE}}{\mathbf{C}^q(F \to F), F \vdash \mathbf{C}^{q*q} F}}{\vdash \mathbf{C}^q(F \to F) \to F \to \mathbf{C}^{q*q} F} \to \mathbf{I}$$

where,

$$\Pi'$$

$$\dfrac{\mathbf{C}^q(F \to F), F \vdash \mathbf{C}^q(F \to F) \qquad \dfrac{F \to F, F \vdash F \to F \qquad F \to F, F \vdash F}{F \to F, F \vdash F} \to \mathbf{E}}{\mathbf{C}^q(F \to F), F \vdash \mathbf{C}^q F}\ \mathbf{CE}$$

and

$$\dfrac{\mathbf{C}^q(F \to F), F, F \vdash \mathbf{C}^q(F \to F) \qquad \dfrac{\Pi''}{\dfrac{F \to F, F, F \vdash F \to F \qquad F \to F, F, F \vdash F}{F \to F, F, F \vdash F} \to \mathbf{E}}}{\mathbf{C}^q(F \to F), F, F \vdash \mathbf{C}^q F}\ \mathbf{CE}$$

Observe that, for simplicity's sake, labels $\top$ have been omitted.

7.2.3 Normalization of $\mathbf{ND}_{\mathbf{iCPL}_0}$

Going back to the CHC perspective, natural deduction proofs correspond to programs, and normalization corresponds to execution. So, we now need to establish the normalization for $\mathbf{ND}_{\mathbf{iCPL}_0}$. Concretely, this amounts at dealing with two normalization steps, for detours ($\to \mathbf{I}/\to \mathbf{E}$) and ($\mathbf{CI}/\mathbf{CE}$). In particular, the conversion for ($\to \mathbf{I}/ \to \mathbf{E}$) is standard and relies on the admissible rule of substitution **Sub**:

$$\dfrac{\Gamma \vdash \mathscr{b} \rightarrowtail F \qquad \Gamma, F \vdash \mathscr{b} \rightarrowtail G}{\Gamma \vdash \mathscr{b} \rightarrowtail G}\ \mathbf{Sub}$$

On the other hand, detour conversion for ($\mathbf{CI}/\mathbf{CE}$) is defined in Figure 7.3 below. First, the rule **CI** is permuted downwards. Then, as you can see, the

$$\dfrac{\dfrac{\overset{\Sigma'}{\Gamma \vdash \mathscr{b} \wedge \mathscr{c} \rightarrowtail F} \qquad \mu([\![\mathscr{c}]\!]) \geq q}{\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F}\ \mathbf{CI} \qquad \dfrac{\Sigma''}{\Gamma, F \vdash \mathscr{b} \rightarrowtail G}}{\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^{qs} G}\ \mathbf{CE}$$

$$\rightsquigarrow$$

$$\dfrac{\dfrac{\overset{\Sigma'}{\Gamma \vdash \mathscr{b} \wedge \mathscr{c} \rightarrowtail F} \qquad \overset{\Sigma''_{[\mathscr{b} \mapsto \mathscr{b} \wedge \mathscr{c}]}}{\Gamma, F \vdash \mathscr{b} \wedge \mathscr{c} \rightarrowtail G}}{\Gamma \vdash \mathscr{b} \wedge \mathscr{c} \rightarrowtail G}\ \mathbf{Sub} \qquad \mu([\![\mathscr{c}]\!]) \geq qs}{\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^{qs} G}\ \mathbf{CI}$$

Figure 7.3: Normalization Step for ($\mathbf{CI}/\mathbf{CE}$)

normalization step consists in applying the rule **Sub** to the premiss of **CI** and the minor premiss of **CE**. For instance, given Π defined as in Example 7.2.2, if one cuts it with $\Pi_{\frac{1}{2}\mathrm{id}}$ (letting $q = \frac{1}{2}$), normalization duplicates Π, that is it

duplicates the choice between the correct and dummy proof of $F \to F$, yielding a normal proof of $F \to \mathbf{C}^{1/2*1/2}F$. The other normalization steps permute $\mathbf{M}$ with other rules. Examples of such detour conversions are displayed in Figure 7.4.[5] As we shall see, these correspond to reduction in probabilistic event λ-calculus introduced in Chapter 8.[6]

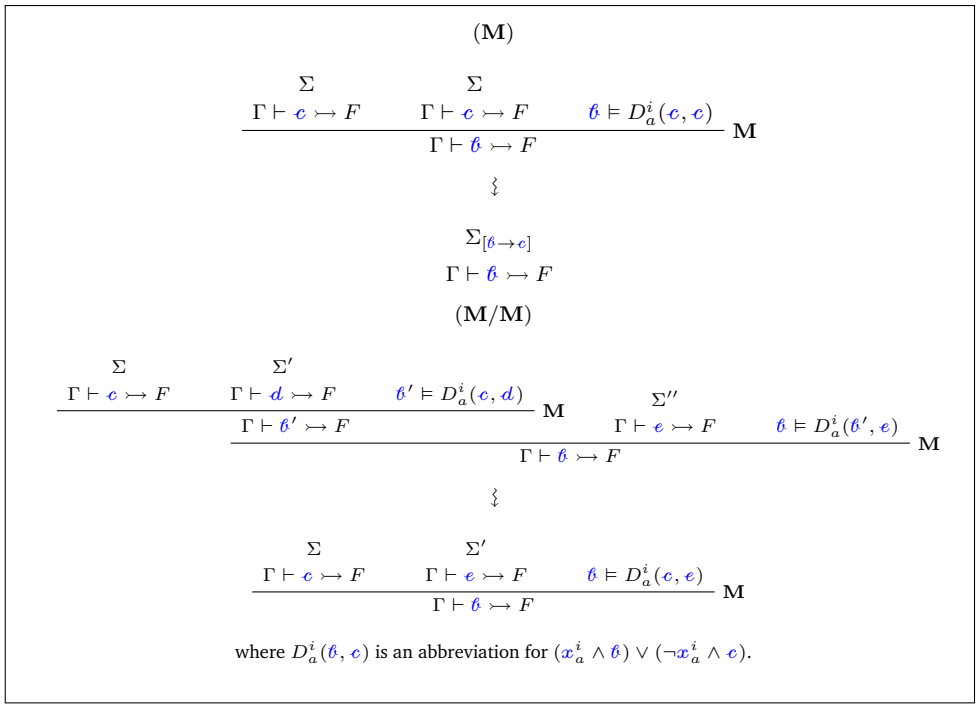


Figure 7.4: Two Examples of Normalization $\mathbf{M}$

7.3 A "CbN Proof System"

Finally, we introduce an alternative proof system, called $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$, which is nothing but a CbN version of $\mathbf{ND}_{\mathbf{iCPL}_0}$. Indeed, as we shall see in Chapter 9, this calculus provides the logical counterpart for a correspondence with the type system $\mathbf{C}\lambda^{\{\}}_{\to}$.

The Rule System $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$. In the perspective of defining a probabilistic CHC involving computation based on CbN, we restrict the rule $\mathbf{CE}$ and pass through

[5] Full details can be found in [12, 11].

[6] For simplicity, we do not consider a "multiplication rule" to pass from $\mathbf{C}^q\mathbf{C}^sF$ to $\mathbf{C}^{qs}$, as this would introduce other normalization steps.

the following definition of sequents. The language for $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$ is exactly the same as that for $\mathbf{ND}_{\mathbf{iCPL}_0}$.

Definition 7.3.1 (Sequent of $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$)**.** A *sequent of* $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$ is an expression of the form $\Phi, \Gamma \vdash \mathscr{b} \rightarrowtail F$, where Φ and Γ are two sets of formulae of $\mathbf{iCPL}_0$ and Φ contains *at most* one of them.

The fundamental intuition is that the formula in Φ (if any) is used *linearly* in the proof.

Definition 7.3.2 (Proof System $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$)**.** Rules for the so-called CbN-system are illustrated in Figure 7.5.

Identity Rules

$$\frac{}{;\Gamma, F \vdash \mathscr{b} \rightarrowtail F}\ \mathbf{ID}^{\mathsf{CbN}} \qquad \frac{}{F;\Gamma \vdash \mathscr{b} \rightarrowtail F}\ \mathbf{ID}_{\mathbf{lin}}$$

Structural Rules

$$\frac{\mathscr{b} \vDash \bot}{;\Gamma \vdash \mathscr{b} \rightarrowtail F}\ \bot\mathbf{R}^{\mathsf{CbN}}$$

$$\frac{;\Gamma \vdash \mathscr{c} \rightarrowtail F \qquad ;\Gamma \vdash \mathscr{d} \rightarrowtail F \qquad \mathscr{b} \vDash (\mathscr{c} \wedge x^i_a) \vee (\mathscr{d} \wedge \neg x^i_a)}{;\Gamma \vdash \mathscr{b} \rightarrowtail F}\ \mathbf{M}^{\mathsf{CbN}}$$

Logical Rules

$$\frac{\Phi;\Gamma, F \vdash \mathscr{b} \rightarrowtail G}{\Phi;\Gamma \vdash \mathscr{b} \rightarrowtail (F \to G)} \to \mathbf{I}^{\mathsf{CbN}} \qquad \frac{\Phi;\Gamma \vdash \mathscr{b} \rightarrowtail (F \to G) \qquad ;\Gamma \vdash \mathscr{b} \rightarrowtail F}{\Phi;\Gamma \vdash \mathscr{b} \rightarrowtail G} \to \mathbf{E}^{\mathsf{CbN}}$$

Counting Rules

$$\frac{;\Gamma \vdash \mathscr{b} \wedge \mathscr{c} \rightarrowtail F \qquad \mu([\![\mathscr{c}]\!]) \geq q}{;\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F}\ \mathbf{CI}^{\mathsf{CbN}} \qquad \frac{\Phi;\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^q F \qquad F;\Gamma \vdash \mathscr{b} \rightarrowtail G}{\Phi;\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^q G}\ \mathbf{CE}^{\mathsf{CbN}}$$

$$\frac{\Phi;\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^q \mathbf{C}^s F}{\Phi;\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^{qs} F}\ \mathbf{C}^{\mathsf{CbN}}_{\times}$$

Figure 7.5: Rules of $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$

Observe that, differently from $\mathbf{ND}_{\mathbf{iCPL}_0}$, this system includes a "multiplication rule" $\mathbf{C}^{\mathsf{CbN}}_{\times}$ permitting to derive $\mathbf{C}^{qs}F$ from $\mathbf{C}^q\mathbf{C}^s F$.

This system proves *less* formulae than $\mathbf{ND}_{\mathbf{iCPL}_0}$ as its restricted $\mathbf{CE}^{\mathsf{CbN}}$-rule allows to deduce $\mathbf{C}^q G$ from $\mathbf{C}^q F$ only when G can be deduced from F *linearly*. For instance, one cannot derive $\mathbf{C}^q(F \to F) \to (F \to \mathbf{C}^q F)$ as

done in Example 7.2.2, since the hypothesis $\mathbf{C}^q(F \to F)$ – which is a major premiss of a $\mathbf{CE}$-rule – should be used *twice*. From the programming perspective, this means that one cannot encode in $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$ the non-linear "CbV-function" $\lambda y.\lambda x\{\lambda f.f(fx)\}y$. For similar reasons, it seems that one cannot prove $\mathbf{C}^q F \to (F \to F \to G) \to \mathbf{C}^q G$ in $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$, while it is derivable in $\mathbf{ND}_{\mathbf{iCPL}_0}$ as shown in Figure 7.6. This means that one cannot encode the non-linear "CbV-function" $\lambda x.\lambda y.\{\lambda y.yxx\}x$.

$$\dfrac{\dfrac{\mathbf{C}^q F, H \vdash \mathbf{C}^q F}{\vdash \mathbf{C}^q F \to H \to \mathbf{C}^q G}\,\to \mathbf{Is}}{\dfrac{\dfrac{F, H \vdash F \to F \to G \qquad F, H \vdash F}{F, H \vdash F \to G}\,\to \mathbf{E} \qquad F, H \vdash F}{F, H \vdash G}\,\to \mathbf{E}}\,\mathbf{CE}$$

where $H = F \to F \to F$.

Figure 7.6: Derivation of $\mathbf{C}^q F \to (F \to F \to G) \to \mathbf{C}^q G$ in $\mathbf{ND}_{\mathbf{iCPL}_0}$

Normalization of $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$. Normalization steps are as in $\mathbf{ND}_{\mathbf{iCPL}_0}$, except for the case ($\mathbf{CI}^{\mathsf{CbN}}/\mathbf{CE}^{\mathsf{CbN}}$). In this case, to easily define such a detour-conversion we use the admissible rule $\mathbf{Sub}^{\mathsf{CbN}}$:

$$\dfrac{;\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^{s_1 * \cdots * s_n} F \qquad F; \Gamma \vdash \mathscr{b} \rightarrowtail G}{;\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^{s_1 * \cdots * s_n} G}\,\mathbf{Sub}^{\mathsf{CbN}}$$

Due to this rule, the desired normalization step is obtained as shown in Figure 7.7 below. Notice also that, in this case, the ($\mathbf{CI}^{\mathsf{CbN}}/\mathbf{CE}^{\mathsf{CbN}}$)-step includes a finite number of internal instances of $\mathbf{C}^{\mathsf{CbN}}_{\times}$.

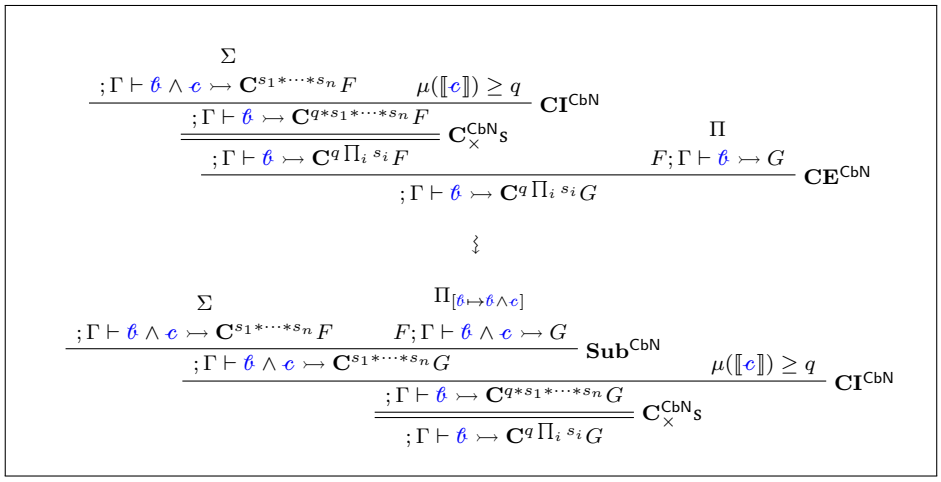


Figure 7.7: Normalization step ($\mathbf{CI}^{\mathsf{CbN}}/\mathbf{CE}^{\mathsf{CbN}}$)

Chapter 8

The Computational Side: $\Lambda_{\mathbf{PE}}$ and $\Lambda_{\mathbf{PE}}^{\{\}}$

In this chapter we focus on the computational side of our probabilistic CHC. This is defined in the form of a variant of the probabilistic event λ-calculus Λ_{PE} by Dal Lago, Guerrieri and Heijltjes [57], in which choices depend on events from the Cantor space, and terms yield distributions of values. So, we start by briefly recapping salient features of Λ_{PE}, in Section 8.1. Then, in Section 8.2, we focus on our calculus $\Lambda_{\mathbf{PE}}$.

8.1 The Probabilistic Event λ-Calculus

The randomized calculus we will introduce in Section 8.2 is strongly inspired by the probabilistic event λ-calculus Λ_{PE} [57]. So, we first outline its crucial features. Starting with motivations, we have already seen in Section 6.1.3, that one of the main undesirable features of standard probabilistic λ-calculi concerns (non-)confluence and duplication in relation with different evaluation strategies. Then, a notion of probabilistic λ-calculus usually comes with a prescribed reduction strategy, typically CbN or CbV, as the calculus is non-confluent and these strategies yield different results.

Example 8.1.1 (Non Confluence [57])**.** Let us consider the term

$$(\lambda x.x == x)(\top \oplus \bot),$$

where $==$ tests equality of Boolean values, yields different results basing on the evaluation strategy one takes into account:

$$\top \;{}_{\mathsf{CbV}} \twoheadleftarrow \; (\lambda x.x == x)(\top \oplus \bot) \;\; \twoheadrightarrow_{\mathsf{CbN}} \;\; \top \oplus \bot.$$

In Λ_{PE} this issue is handled in an elegant way by decomposing the probabilistic operator. This makes the calculus both randomized and confluent.

The Language of $\Lambda_{\mathbf{PE}}$. So, in the probabilistic event λ-calculus, non-confluence is handled by decomposing the probability operator into two syntactic constructs: a *generator* of the form $\boxed{a}$ and a *choice operator* of the form $\overset{a}{\oplus}$ (for a Boolean).

Definition 8.1.1 (Terms of Λ_{PE})**.** Terms of the probabilistic event λ-calculus are defined by the grammar below:

$$\mathsf{t} := x \;\Big|\; \lambda x\mathsf{t} \;\Big|\; \mathsf{tt} \;\Big|\; \mathsf{t} \overset{a}{\oplus} \mathsf{t} \;\Big|\; \boxed{a}\mathsf{t}$$

for a being a Boolean

Syntactically, $\boxed{a}$ acts as a quantifier, binding a. Semantically, the generator $\boxed{a}$ represents a probabilistic event, that generates a Boolean value recorded as a. Otherwise said, $\boxed{a}$ flips a coin setting a to either 0 or 1.

Notation 8.1.1. Observe also that the standard probabilistic operator $\oplus$, can be defined in terms of generator and choice operator as:

$$\mathsf{t} \oplus \mathsf{u} := \boxed{a}\mathsf{t} \overset{a}{\oplus} \mathsf{u}.$$

Confluence and Reduction. This technical novelty is crucial to obtain a calculus, which is actually *confluent*. In fact, decomposing the probabilistic operator allows Dal Lago, Guerrieri and Heijltjes to syntactically distinguish between duplicating an event, and duplicating its outcome. In this way, Λ_{PE} can express both CbN and CbV strategies.

Example 8.1.2. Let us now (re-)consider Example 8.1.1 in the context of Λ_{PE}:

$$(\lambda x.x = x)(\boxed{a_1}\top \overset{a_1}{\oplus} \bot) \to_\beta (\boxed{a_1}\top \overset{a_1}{\oplus} \bot) = (\boxed{a_2}\top \overset{a_2}{\oplus} \bot) \twoheadrightarrow \top \oplus \bot \qquad \text{CbN}$$

$$\boxed{a_1}(\lambda x.x = x)(\top \overset{a_1}{\oplus} \bot) \to_\beta \boxed{a_1}(\top \overset{a_1}{\oplus} \bot) = (\top \overset{a_1}{\oplus} \bot) \twoheadrightarrow \top. \qquad \text{CbV}$$

Clearly, in Λ_{PE}, CbN and CbV strategies are expressed by *different* terms and confluence is guaranteed.

Reduction in $\Lambda_{\mathbf{PE}}$ consists of standard β-reduction $\to_\beta$ plus an evaluation mechanism for generators and choice operator, which implements probabilistic choice.[1] In particular, we consider permutative reduction, $\to_p$.

Definition 8.1.2. Reduction $\to$ in $\Lambda_{\mathbf{PE}}$ is made of β-reduction $\to_\beta$ plus *permutative reduction* $\to_p$, as illustrated in Figure 8.1.

[1] Full details can be found in [57].

$$(\lambda x.\mathsf{t})\mathsf{u} \to_\beta \mathsf{t}[\mathsf{u}/x] \qquad (\beta)$$

$$\mathsf{t} \overset{a}{\oplus} \mathsf{t} \twoheadrightarrow_p \mathsf{t} \qquad (\mathsf{i})$$

$$(\mathsf{t} \overset{a}{\oplus} \mathsf{u}) \overset{a}{\oplus} \mathsf{v} \twoheadrightarrow_p \mathsf{t} \overset{a}{\oplus} \mathsf{v} \qquad (\mathsf{c}_1)$$

$$\mathsf{t} \overset{a}{\oplus} (\mathsf{u} \overset{a}{\oplus} \mathsf{v}) \twoheadrightarrow_p \mathsf{t} \overset{a}{\oplus} \mathsf{v} \qquad (\mathsf{c}_2)$$

$$\lambda x(\mathsf{t} \overset{a}{\oplus} \mathsf{u}) \twoheadrightarrow_p (\lambda x.\mathsf{t}) \overset{a}{\oplus} (\lambda x.\mathsf{u}) \qquad (\oplus\lambda)$$

$$(\mathsf{t} \overset{a}{\oplus} \mathsf{u})\mathsf{v} \twoheadrightarrow_p (\mathsf{tv}) \overset{a}{\oplus} (\mathsf{uv}) \qquad (\oplus\mathsf{f})$$

$$\mathsf{t}(\mathsf{u} \overset{a}{\oplus} \mathsf{v}) \twoheadrightarrow_p (\mathsf{tu}) \overset{a}{\oplus} (\mathsf{tv}) \qquad (\oplus\mathsf{a})$$

$$(\mathsf{t} \overset{a}{\oplus} \mathsf{u}) \overset{b}{\oplus} \mathsf{v} \twoheadrightarrow_p (\mathsf{t} \overset{b}{\oplus} \mathsf{v}) \overset{a}{\oplus} (\mathsf{u} \overset{b}{\oplus} \mathsf{v}) \quad (a < b) \qquad (\oplus\oplus_1)$$

$$\mathsf{t} \overset{b}{\oplus} (\mathsf{u} \overset{a}{\oplus} \mathsf{v}) \twoheadrightarrow_p (\mathsf{t} \overset{b}{\oplus} \mathsf{u}) \overset{a}{\oplus} (\mathsf{t} \overset{b}{\oplus} \mathsf{v}) \quad (a < b) \qquad (\oplus\oplus_2)$$

$$\boxed{a}(\mathsf{t} \overset{b}{\oplus} \mathsf{u}) \twoheadrightarrow_p (\boxed{a}\mathsf{t}) \overset{b}{\oplus} (\boxed{a}\mathsf{u}) \quad (b \neq a) \qquad (\oplus\Box)$$

$$\boxed{a}\mathsf{t} \twoheadrightarrow_p \mathsf{t} \quad (a \notin \mathsf{t}) \qquad (\not\Box)$$

$$\lambda x.\boxed{a}\mathsf{t} \twoheadrightarrow_p \boxed{a}\lambda x.\mathsf{t} \qquad (\Box\lambda)$$

$$(\boxed{a}\mathsf{t})\mathsf{u} \twoheadrightarrow_p \boxed{a}\mathsf{tu} \qquad (\Box\mathsf{f})$$

Figure 8.1: Reduction rules for $\Lambda_{\mathbf{PE}}$

As its crucial feature, the choice operator $\overset{a}{\oplus}$ does permute out the argument position of an application, but generator $\boxed{a}$ does not:

$$\mathsf{t}(\mathsf{u} \overset{a}{\oplus} \mathsf{v}) \twoheadrightarrow_p (\mathsf{tu}) \overset{a}{\oplus} (\mathsf{tv})$$

but

$$\mathsf{t}(\boxed{a}\mathsf{u}) \not\twoheadrightarrow_p \boxed{a}\mathsf{tu}.$$

This expresses the difference between the outcome of a probabilistic event, the duplicates of which can be identified, and the event itself, the duplicates of which may yield different outcomes. In [57, Sec. 3-4], $\twoheadrightarrow_p$ is also proved strongly normalizing and confluent.

Theorem 8.1.1 ([57]). *$\twoheadrightarrow_p$ is confluent and strongly normalizing. Full reduction $\twoheadrightarrow := \twoheadrightarrow_\beta \cup \twoheadrightarrow_p$ is confluent.*

Then, Λ_{PE} is as desired: it is both confluent and able to interpret both CbN and CbV strategies through different interpretations of the probabilistic operator.

8.2 A λ-Calculus Sampling from the Cantor Space

In Section 8.2.1 we introduce a variant of Λ_{PE}, called $\Lambda_{\mathbf{PE}}$, in which choices depend on events from the Cantor space and terms yield distributions of values.

We also define two notions of probabilistic normalizations for such distributions. Finally, in Section 8.2.3, we present another calculus, called $\Lambda^{\{\}}_{\mathbf{PE}}$, which provides a straightforward representation of CbV functions.[2]

8.2.1 Introducing the (Untyped) Calculus $\Lambda_{\mathbf{PE}}$

The grammar for terms of $\Lambda_{\mathbf{PE}}$ is slightly different from that of Definition 8.1.1. In particular, the choice operator becomes dependent from an index $i \in \mathbb{N}$, and the generator is now expressed as νa.

Definition 8.2.1 (Terms of $\Lambda_{\mathbf{PE}}$)**.** *Terms of* $\Lambda_{\mathbf{PE}}$ are defined as follows

$$\mathsf{t} := x \;\Big|\; \lambda x.\mathsf{t} \;\Big|\; \mathsf{tu} \;\Big|\; \mathsf{t} \oplus^i_a \mathsf{u} \;\Big|\; \nu a.\mathsf{t}$$

with $a \in \mathscr{A}$ and $i \in \mathbb{N}$.

The intuition is that $\nu a.\mathsf{t}$ samples some function ω from the Cantor space, and $\mathsf{t} \oplus^i_a \mathsf{u}$ yields either t or u depending on the value of $\omega(a)(i) \in \{0,1\}$.

Notation 8.2.1. In the following, let $\mathsf{t} \oplus^i \mathsf{u}$ be an abbreviation for $\nu a.\mathsf{t} \oplus^i_a \mathsf{u}$, supposing a does not occur free in either t or u.

As desirable $\Lambda_{\mathbf{PE}}$ maintains all the nice confluence properties of Λ_{PE} concerning confluence. For example, going back to the Example 6.2.2 in Chapter 6, it is clear that the CbN and CbV applications of 2 to $\mathsf{I} \oplus_a \Omega$ are here encoded by two *distinct* term: $2(\nu a.\mathsf{I} \oplus_a \Omega)$ and $\nu a.2(\mathsf{I} \oplus_a \Omega)$, crucially distinguishing between generating a probabilistic choice *before* or *after* a duplication takes place.

In usual randomized λ-calculi, program execution is defined so to be inherently probabilistic. For example, a term $\mathsf{t} \oplus \mathsf{u}$ can reduce to either t or u with the same probability $\frac{1}{2}$. In this way, chains of reduction can be described as stochastic Markovian sequences [175] and help formalizing the idea of *normalization with probability* $q \in [0,1]$, see [30]. By contrast, reduction in $\Lambda_{\mathbf{PE}}$ is fully deterministic.

Definition 8.2.2 (Reduction in $\Lambda_{\mathbf{PE}}$)**.** Reduction $\twoheadrightarrow$ in $\Lambda_{\mathbf{PE}}$ is defined by the usual (and un-restricted) β-rule $(\lambda x.\mathsf{t})\mathsf{u} \twoheadrightarrow_\beta \mathsf{t}[\mathsf{u}/x]$, together with *permutative reduction* $\mathsf{t} \twoheadrightarrow_p \mathsf{u}$ made of rules in Figure 8.2.

Intuitively, permutative reductions implement probabilistic choices by computing the full tree of possible choices. For example, given terms $\mathsf{t}_1, \mathsf{t}_2, \mathsf{u}_1, \mathsf{u}_2$,

$$\nu a.(\mathsf{t}_1 \oplus^0_a \mathsf{t}_2)(\mathsf{u}_1 \oplus^1_a \mathsf{u}_2) \twoheadrightarrow_p \nu a.(\mathsf{t}_1\mathsf{u}_1 \oplus^1_a \mathsf{t}_1\mathsf{u}_2) \oplus^0_a (\mathsf{t}_2\mathsf{u}_1 \oplus^1_a \mathsf{t}_2\mathsf{u}_2),$$

hence displaying all possible alternatives.[3]

[2] For several results presented in the following sections, the corresponding (Λ_{PE}) version has already been established in [57]. Since, in these cases, proofs are equivalent to the one in [57], they are only sketched or omitted. For further details, see [11, 12] or [57].

[3] To compare $\Lambda_{\mathbf{PE}}$ with Λ_{PE} [57], observe that, given a bijection $\varphi : \mathbb{N}^2 \to \mathbb{N}$, one can define an invertible embedding $\mathsf{t} \to \mathsf{t}^\varphi$ from Λ_{PE} by replacing $\mathsf{t} \oplus^i_a \mathsf{u}$ by $\mathsf{t}^\varphi \oplus^{\varphi(a,i)} \mathsf{u}^\varphi$ and $(\nu a.\mathsf{t})^\varphi = \nu\varphi(a,0)\ldots\nu\varphi(a,\mathsf{ord}_a(\mathsf{t})).\mathsf{t}^\varphi$, where $\mathsf{ord}_a(\mathsf{t})$ is the maximum i such that $\oplus^i_a$ occurs in t. In this way, permutation rules of Λ_{PE} are translated into those in Figure 8.2 and, then, results from [57] can be transposed into our language of $\Lambda_{\mathbf{PE}}$.

$$
\begin{array}{rlll}
\mathsf{t} \oplus^i_a \mathsf{t} \twoheadrightarrow_p \mathsf{t} & & & \text{(i)} \\
(\mathsf{t} \oplus^i_a \mathsf{u}) \oplus^i_a \mathsf{v} \twoheadrightarrow_p \mathsf{t} \oplus^i_a \mathsf{v} & & & (\mathrm{c}_1) \\
\mathsf{t} \oplus^i_a (\mathsf{u} \oplus^i_a \mathsf{v}) \twoheadrightarrow_p \mathsf{t} \oplus^i_a \mathsf{v} & & & (\mathrm{c}_2) \\
\lambda x.(\mathsf{t} \oplus^i_a \mathsf{u}) \twoheadrightarrow_p (\lambda x.\mathsf{t}) \oplus^i_a (\lambda x.\mathsf{u}) & & & (\oplus\lambda) \\
(\mathsf{t} \oplus^i_a \mathsf{u})\mathsf{v} \twoheadrightarrow_p (\mathsf{tu}) \oplus^i_a (\mathsf{uv}) & & & (\oplus\mathrm{f}) \\
\mathsf{t}(\mathsf{u} \oplus^i_a \mathsf{v}) \twoheadrightarrow_p (\mathsf{tu}) \oplus^i_a (\mathsf{tv}) & & & (\oplus\mathrm{a}) \\
(\mathsf{t} \oplus^i_a \mathsf{u}) \oplus^j_b \mathsf{v} \twoheadrightarrow_p (\mathsf{t} \oplus^j_b \mathsf{v}) \oplus^i_a (\mathsf{u} \oplus^j_b \mathsf{v}) & (a,i) < (b,j) & & (\oplus\oplus_1) \\
\mathsf{t} \oplus^j_b (\mathsf{u} \oplus^i_a \mathsf{v}) \twoheadrightarrow_p (\mathsf{t} \oplus^j_b \mathsf{u}) \oplus^i_a (\mathsf{t} \oplus^j_b \mathsf{v}) & (a,i) < (b,j) & & (\oplus\oplus_2) \\
\nu b.(\mathsf{t} \oplus^i_a \mathsf{u}) \twoheadrightarrow_p (\nu b.\mathsf{t}) \oplus^i_a (\nu b.\mathsf{u}) & a \neq b & & (\oplus\nu) \\
\nu a.\mathsf{t} \twoheadrightarrow_p \mathsf{t} & a \notin \mathsf{FN}(\mathsf{t}) & & (\neg\nu) \\
\lambda x.\nu a.\mathsf{t} \twoheadrightarrow_p \nu a.\lambda x.\mathsf{t} & & & (\nu\lambda) \\
(\nu a.\mathsf{t})\mathsf{u} \twoheadrightarrow_p \nu a.(\mathsf{tu}) & & & (\nu\mathrm{f})
\end{array}
$$

where $(a,i) < (b,j)$ if either νb occurs in the scope of νa or $a = b$ and $i < j$

Figure 8.2: Permutative Reductions

Let us now introduce the following technical notion, which will be crucial in relation to normal form.

Definition 8.2.3. For any term t, finite set X and $\omega \in (2^{\mathbb{N}})^X$, let the *projection of ω to t through X*, $\pi^\omega_X(\mathsf{t})$, be defined as follows:

$$
\begin{aligned}
\pi^\omega_X(x) &= x \\
\pi^\omega_X(\lambda x.\mathsf{t}) &= \lambda x.\pi^\omega_X(\mathsf{t}) \\
\pi^\omega_X(\mathsf{tu}) &= \pi^\omega_X(\mathsf{t})\pi^\omega_X(\mathsf{u}) \\
\pi^\omega_X(\mathsf{t} \oplus^i_a \mathsf{u}) &= \begin{cases} \pi^\omega_X(\mathsf{t}) & \text{if } a \in X \text{ and } \omega(a)(i) = 1 \\ \pi^\omega_X(\mathsf{u}) & \text{if } a \in X \text{ and } \omega(a)(i) = 0 \\ \pi^\omega_X(\mathsf{t}) \oplus^i_a \pi^\omega_X(\mathsf{u}) & \text{otherwise} \end{cases} \\
\pi^\omega_X(\nu b.\mathsf{t}) &= \nu b.\pi^\omega_X(\mathsf{t}).
\end{aligned}
$$

where $b \notin X$. The existence and unicity of normal forms for $\twoheadrightarrow_p$, the so-called *permutative normal forms* (**PNF**, for short) leads to natural questions on what these normal forms represent. Let $\mathcal{T}$ denote the set of **PNF**s containing no free name occurrence. For any $\mathsf{t} \in \mathcal{T}$, the **PNF** of t in one of two possible forms: (i.) either t starts with a generator, i.e. $\mathsf{t} = \nu a.\mathsf{t}'$ and t' is a tree of a-labelled choices $\oplus^i_a$, in which leaves form a finite set of $\mathcal{T}$, the *support* of t', $\mathsf{supp}(\mathsf{t})$, or (ii.) t is of the form $\lambda x_1 \ldots \lambda x_n.\mathsf{t}'\mathsf{t}_1 \ldots \mathsf{t}_p$, where t' is either a variable or a λ-abstraction. We call these last terms *pseudo-values* and let $\mathcal{V} \subseteq \mathcal{T}$ denote the

set formed by them. Using this decomposition, any $\mathsf{t} \in \mathcal{T}$ can be associated in a unique way with a sub-distribution of pseudo-values $\mathcal{D}_{\mathsf{v}} : \mathcal{V} \to [0,1]$ by letting $\mathcal{D}_{\mathsf{v}}(\mathsf{u}) = \delta_{\mathsf{t}}$, and $\mathcal{D}_{\mathsf{t}} := \sum_{\mathsf{u} \in \mathsf{supp}(\mathsf{t}')} \mathcal{D}_{\mathsf{u}}(\mathsf{v}) \cdot \mu\big(\{\omega \in 2^{\mathbb{N}} \mid \pi^{\omega}_{\{a\}}(\mathsf{t}') = \mathsf{u}\}\big)$, when $\mathsf{t} = \nu a.\mathsf{u}'$.[4] Intuitively $\mathcal{D}_{\mathsf{t}}(\mathsf{v})$ measures the probability of finding v by iteratively applying random choices of events from the Cantor space to t any time a v is found.

8.2.2 Probabilistic (Head) Normalization

Given a term $\mathsf{t} \in \mathcal{T}$, questions such as "is t in normal form?" or "does t reduce to a normal form?" receive univocal yes/no answers. Indeed $\to_{\triangleright}$ is deterministic. Nevertheless, if one thinks of t as $\mathcal{D}_{\mathsf{t}}$, the relevant questions become "what is the probability for t to be in normal form?" and "what is the probability for t to reduce to normal form?". To answer these new kinds of questions we introduce functions $\mathrm{HNV}_{\to_{\triangleright}}(\mathsf{t})$ and $\mathrm{NF}_{\to_{\triangleright}}(\mathsf{t})$ *measuring the probability* that t reduces to a normal form.

Let us start with randomized contexts and head-normal forms. The randomized context R is defined by the grammar below:

$$\mathsf{R}[\,] := [\,] \;\Big|\; \mathsf{R}[\,] \oplus^{i}_{a} \mathsf{u} \;\Big|\; \mathsf{t} \oplus^{i}_{a} \mathsf{R}[\,] \;\Big|\; \nu a.\mathsf{R}[\,].$$

Then, the *head-reduction* $\mathsf{t} \to_{h} \mathsf{u}$ is either a $\to_{p}$-reduction or a $\to_{\beta}$-reduction of the form:

$$\mathsf{R}[\lambda\vec{x}.(\lambda y.\mathsf{t})\mathsf{u}\mathsf{u}_1 \dots \mathsf{u}_n] \to_{\beta} \mathsf{R}[\lambda\vec{x}.\mathsf{t}[\mathsf{u}/x]\mathsf{u}_1 \dots \mathsf{u}_n],$$

where R is a randomized context.

Definition 8.2.4. A *head normal value* (HNV, for short) is a $\to_{h}$-normal term which is also a pseudo-value, i.e. it is of the form $\lambda\vec{x}.y\mathsf{t}_1 \dots \mathsf{t}_n$. Let HNV indicate the set of such terms. For any $t \in \mathcal{T}$,

$$\mathrm{HNV}(\mathsf{t}) := \sum_{\mathsf{v} \in \mathrm{HNV}} \mathcal{D}_{\mathsf{t}}(\mathsf{v})$$

and

$$\mathrm{HNV}_{\to_{\triangleright}}(\mathsf{t}) := \sup\{\mathrm{HNV}(\mathsf{u}) \mid \mathsf{t} \to^{*}_{h} \mathsf{u}\}.$$

When $\mathrm{HNV}_{\to_{\triangleright}}(\mathsf{t}) \geq q$, we say that t yields an HNV with probability at least q.

Example 8.2.1. Let us now consider the term

$$\mathsf{t} = \nu a.(\lambda x\lambda y.(y \oplus^{i}_{a} \mathsf{I})x)\mathsf{u},$$

where $\mathsf{u} = \nu b(\mathsf{I} \oplus^{j}_{b} \Omega)$. Then, $\mathrm{HNV}_{\to_{\triangleright}}(\mathsf{t}) = \frac{3}{4}$. Indeed,

$$\mathsf{t} \to^{*}_{h} \nu a.\big(\lambda y.y(\nu b.\mathsf{I} \oplus^{j}_{b} \Omega)\big) \oplus^{i}_{a} (\nu b'.\mathsf{I} \oplus^{j}_{b'} \Omega)$$

[4] Observe that the measurability of $\{\omega \in 2^{\mathbb{N}} \mid \pi^{\omega}_{\{a\}}(\mathsf{t}') = \mathsf{u}\}$ could be checked by induction on t'. We thank Raphaëlle Crubillé for pointing out to us the relevance of this measurability condition.

and three out of four possible choices (corresponding to choosing between either left or right for both νa and $\nu b'$) yield a HNV. Notice that the choice for νb does not matter, since $\lambda y.y\mathsf{u}$ is already a HNV.

Let us now consider normal forms. The first idea might be to define a similar function $\mathrm{NF}(\mathsf{t}) = \sum_{\mathsf{v}\text{ normal form}} \mathscr{D}_{\mathsf{t}}(\mathsf{v})$. Yet, following this definition, for example a term $\lambda x.x(\nu a.\mathsf{I}\oplus_a^0\Omega)$ would have probability 0 of yielding a normal form, while it should yield a normal form with probability $\frac{1}{2}$, i.e. depending on a choice for a. So, we introduce the definition below:

Definition 8.2.5. For any $\mathsf{t} \in \mathscr{T}$, $\mathrm{NF}(\mathsf{t})$ is defined by:

- if $\mathsf{t} = \lambda\vec{x}.y\mathsf{u}_1 \dots \mathsf{u}_n \in \mathrm{HNV}$, then
$$\mathrm{NF}(\mathsf{t}) := \prod_{i=1}^{n} \mathrm{NF}(\mathsf{u}_i)$$
- otherwise,
$$\mathrm{NF}(\mathsf{t}) := \sum_{\mathsf{u}\in\mathrm{HNV}} \mathrm{NF}(\mathsf{u}) \cdot \mathscr{D}_{\mathsf{t}}(\mathsf{u}).$$

Let $\mathrm{NF}_{\twoheadrightarrow}(\mathsf{t}) = \mathsf{supp}\{\mathrm{NF}(\mathsf{u}) \mid \mathsf{t} \twoheadrightarrow^* \mathsf{u}\}$ and if $\mathrm{NF}_{\twoheadrightarrow}(\mathsf{t}) \geq q$, we say that t *yields a normal form with probability at least* q.

Now, consider again the term t of Example 8.2.1: $\mathrm{NF}_{\twoheadrightarrow}(\mathsf{t}) = \frac{4}{8} = \frac{1}{2}$ as four out of eight possible choices for νa, νb and $\nu b'$ yield a normal form, i.e. either choose left for νa and νb and choose anything for $\nu b'$ or choose right for νa, left for $\nu b'$ and choose anything for νb.

8.2.3 Extending $\Lambda_{\mathbf{PE}}$ with CbV Functions

As said, in $\Lambda_{\mathbf{PE}}$ it is possible to encode a CbV redex like $\nu a.2(\mathsf{I}\oplus_a^0\Omega)$. Nevertheless, in the perspective of the functional interpretation of **iCPL**$_0$, we would be able to represent the CbV *functions* mapping $\nu a(\mathsf{t}\oplus_a\mathsf{v})$ onto $\nu a.2(\mathsf{t}\oplus_a^0\mathsf{v})$. To do so, we enrich the language of $\Lambda_{\mathbf{PE}}$ with a "CbV application" operator $\{\mathsf{t}\}\mathsf{u}$, with suitable permutative rules. Let $\Lambda_{\mathbf{PE}}^{\{\}}$ denote such extension of the syntax of $\Lambda_{\mathbf{PE}}$ with $\{\}$.

Reduction in $\Lambda_{\mathbf{PE}}^{\{\}}$ is similar to $\Lambda_{\mathbf{PE}}$, that is β-reduction is as standard and $\twoheadrightarrow_{p\{\}}$ is defined as in Figure 8.2, except for $(\neg\nu)$ and the three new permutation rules below:

$$\{\mathsf{t}\}\nu a.\mathsf{u} \twoheadrightarrow_{p\{\}} \nu a.\mathsf{tu} \qquad (\{\}\nu)$$
$$\{\mathsf{t}\}(\mathsf{u}\oplus_a^i\mathsf{u})\mathsf{v} \twoheadrightarrow_{p\{\}} \{\mathsf{t}\}\mathsf{v}\oplus_a^i\{\mathsf{t}\}\mathsf{v} \qquad (\{\}\oplus_1)$$
$$\{\mathsf{t}\}(\mathsf{u}\oplus_a^i\mathsf{v}) \twoheadrightarrow_{p\{\}} \{\mathsf{t}\}\mathsf{u}\oplus_a^i\{\mathsf{t}\}\mathsf{v} \qquad (\{\}\oplus_2)$$

Let us consider a clarifying example showing how to represent CbV into our calculus.

Example 8.2.1. A CbV Church numeral can be encoded in $\Lambda^{\{\}}_{\mathbf{PE}}$ as

$$2^{\mathsf{CbV}} := \lambda f.\{2\}f.$$

Indeed,

$$2^{\mathsf{CbV}}(\nu a.\mathsf{t} \oplus^i_a \mathsf{v}) \to_\beta \{2\}\nu a.\mathsf{t} \oplus^i_a \mathsf{v} \to_{p\{\}} \nu a.2(\mathsf{t} \oplus^i_a \mathsf{v}).$$

The fundamental properties of $\Lambda_{\mathbf{PE}}$ hold also when moving to $\Lambda^{\{\}}_{\mathbf{PE}}$. In particular, confluence and strong normalizations are proved as in [57].[5]

Proposition 8.2.1. *$\to_{p\{\}}$ is confluent and strongly normalizing. Full reduction $\to_{\{\}} := \to_\beta \cup \to_{p\{\}}$ is confluent.*

The definitions of $\mathscr{D}_\mathsf{t}$ and $\mathrm{HNV}_{\to}(\mathsf{t})$ scale to $\Lambda^{\{\}}_{\mathbf{PE}}$ in the natural way.

[5] For further details, see [12].

Chapter 9

Probabilistic Curry-Howard Correspondence

In this Chapter we introduce the core of our probabilistic CHC. First, in Section 9.1, we define two type systems $\mathbf{C}\lambda_{\to}$ and $\mathbf{C}\lambda^{\{\}}_{\to}$ for (resp.) $\Lambda_{\mathbf{PE}}$ and $\Lambda^{\{\}}_{\mathbf{PE}}$, which extend standard grammars for types with counting quantifiers. In Section 9.2 we focus on static and dynamic aspects of the correspondence between $\mathbf{ND}_{\mathbf{iCPL}_0}$ and $\mathbf{C}\lambda^{\{\}}_{\to}$. In particular, we show that each proof Π in the quoted calculus for $\mathbf{iCPL}_0$ can be associated with a typing derivation in $\mathbf{C}\lambda^{\{\}}_{\to}$ of some probabilistic term t^{Π}, in such a way that normalization of Π corresponds to reduction of t^{Π}. Remarkably, in this case, translating $\mathbf{CE}$ requires the CbV application operator $\{\}$. So, in Section 9.3, we also consider the alternative CbN proof-system for $\mathbf{iCPL}_0$, namely $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$, and provide the corresponding translation with respect to $\mathbf{C}\lambda^{\{\}}_{\to}$.

9.1 Introducing Types with Counting

We introduce two type systems, called $\mathbf{C}\lambda_{\to}$ and $\mathbf{C}\lambda^{\{\}}_{\to}$, which both extend standard grammars for types with counting quantifiers.

Counting Types and Judgements Counting types of $\mathbf{C}\lambda_{\to}$ are of the form $\mathbf{C}^{\mathfrak{q}}\mathfrak{s}$, where $\mathfrak{s}$ is prefixed by *exactly one* counting quantifier, while in $\mathbf{C}\lambda^{\{\}}_{\to}$, these types are of the form $\mathbf{C}^{\vec{\mathfrak{q}}}\mathfrak{s}$, so $\mathfrak{s}$ is prefixed by a (possibly empty) list of quantifiers.

Definition 9.1.1 (Types for $\mathbf{C}\lambda_{\to}$ and $\mathbf{C}\lambda^{\{\}}_{\to}$)**.** The grammar for types of $\mathbf{C}\lambda_{\to}$,

denoted as $\mathfrak{s}, \mathfrak{t}, \ldots$, is generated as follows:

$$\sigma := o \mid \mathfrak{s} \Rightarrow \sigma$$

$$\mathfrak{s} := \mathbf{C}^{\mathfrak{q}}\sigma.$$

and that for types of $\mathbf{C}\lambda^{\{\}}_{\to}$ is defined below:

$$\sigma := o \mid \mathfrak{s} \Rightarrow \sigma$$

$$\mathfrak{s} := \mathbf{C}^{\vec{\mathfrak{q}}}\sigma,$$

where, in both cases, $q \in \mathbb{Q}_{(0,1]}$.

In both systems, judgments are labelled and, of course, include counting quantifiers.

Definition 9.1.2 (Judgments in $\mathbf{C}\lambda_{\to}$ and $\mathbf{C}\lambda^{\{\}}_{\to}$)**.** *Judgments in* $\mathbf{C}\lambda_{\to}$ *and* $\mathbf{C}\lambda^{\{\}}_{\to}$ are of the form:

$$\Gamma \vdash^{X} \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s},$$

where Γ is a set of type declarations $x_i : \mathfrak{s}_i$ with pairwise distinct variables, t is a term of $\Lambda_{\mathbf{PE}}$ (resp. $\Lambda^{\{\}}_{\mathbf{PE}}$), $\mathscr{b}$ is a Boolean formula, and X is a finite set of names, with $\mathsf{FN}(\mathsf{t}), \mathsf{FN}(\mathscr{b}) \subseteq X$.

The intuitive meaning of $\Gamma \vdash^{X} \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s}$ is that, whenever $\omega \in (2^{\mathbb{N}})^{X}$ satisfies $\mathscr{b}$, $\pi^{\omega}_{X}(\mathsf{t})$ correctly maps programs of type Γ into programs of type $\mathfrak{s}$.

Typing Rules. The *typing rules* of $\mathbf{C}\lambda^{\{\}}_{\to}$ are essentially obtained by decorating those of $\mathbf{ND}_{\mathbf{iCPL}_0}$.

Definition 9.1.3 (Typing Rules of $\mathbf{C}\lambda^{\{\}}_{\to}$)**.** Typing rules of $\mathbf{C}\lambda^{\{\}}_{\to}$ are illustrated in Figure 9.1.

Rule $(\vee)$ allows one to merge n typing derivations for the same term.[1] Rules $(\oplus), (\mu)$ and $(\{\})$ are reminiscent of rules $\mathbf{M}, \mathbf{CI}$ and $\mathbf{CE}$ of $\mathbf{ND}_{\mathbf{iCPL}_0}$, respectively; for example,

$$\frac{\Gamma \vdash \mathscr{b} \wedge \mathscr{c} \rightarrowtail F \qquad \mu(\mathscr{c}) \geq q}{\Gamma \vdash \mathscr{b} \rightarrowtail \mathbf{C}^{q} F}\ \mathbf{CI} \qquad \frac{\Gamma \vdash^{X \cup \{a\}} \mathsf{t} : \mathscr{b} \wedge \mathscr{c} \rightarrowtail \mathfrak{s} \qquad \mu(\mathscr{c}) \geq q}{\vdash \Gamma^{X} \nu a.\mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{q}}\mathfrak{s}}\ (\mu)$$

Observe that we consider arrow types of the form $\mathbf{C}^{\vec{\mathfrak{q}}}(\mathfrak{s} \Rightarrow \sigma)$, so never having a counting quantifier *to the right of* $\Rightarrow$ (as in $\mathfrak{s} \Rightarrow \mathbf{C}^{q}\sigma$). The reason is that we need rule (λ) to be permutable over (μ), as required by the permuting rule $(\nu\lambda)$ (Figure 8.2). The typing rule of $\mathbf{C}\lambda_{\to}$ coincide with those of $\mathbf{C}\lambda^{\{\}}_{\to}$ where $\mathbf{C}^{\vec{\mathfrak{q}}}$ replaced by $\mathbf{C}^{\mathfrak{q}}$, except for the rule $(\{\})$, which is absent, and the rule (μ), which is adapted as follows:

[1] In particular, when $n = 0$, $\Gamma \vdash^{X} \mathsf{t} : \bot \rightarrowtail \mathfrak{s}$ holds for *any* term t.

Identity Rules

$$\frac{\mathsf{FN}(\mathscr{b}) \subseteq X}{\Gamma, x : \mathfrak{s} \vdash^X x : \mathscr{b} \rightarrowtail \mathfrak{s}} \ (\mathsf{id})$$

Structural Rule

$$\frac{\left\{\Gamma \vdash^X \mathsf{t} : \mathscr{b}_i \rightarrowtail \mathfrak{s}\right\}_{i\in\{1,\ldots,n\}} \qquad \mathscr{b} \vDash^X \bigvee_i \mathscr{b}_i}{\Gamma \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s}} \ (\vee)$$

Plus Rule

$$\frac{\Gamma \vdash^{X\cup\{a\}} \mathsf{t} : \mathscr{c} \rightarrowtail \mathfrak{s} \qquad \Gamma \vdash^{X\cup\{a\}} \mathsf{u} : \mathscr{d} \rightarrowtail \mathfrak{s} \qquad \mathscr{b} \vDash^X (\mathscr{c} \wedge x_a^i) \vee (\mathscr{d} \wedge \neg x_a^i)}{\Gamma \vdash^{X\cup\{a\}} \mathsf{t} \oplus_a^i \mathsf{u} : \mathscr{b} \rightarrowtail \mathfrak{s}} \ (\oplus)$$

Arrow Rule

$$\frac{\Gamma, x : \mathfrak{s} \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\vec{\mathfrak{q}}}\sigma}{\Gamma \vdash^X \lambda x.\mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\vec{q}}(\mathfrak{s} \Rightarrow \sigma)} \ (\lambda)$$

$$\frac{\Gamma \vdash^X \mathsf{t} : \mathscr{c} \rightarrowtail \mathbf{C}^{\vec{\mathfrak{q}}}(\mathfrak{s} \Rightarrow \sigma) \qquad \Gamma \vdash^X \mathsf{u} : \mathscr{d} \rightarrowtail \mathfrak{s} \qquad \mathscr{b} \vDash^X \mathscr{c} \wedge \mathscr{d}}{\Gamma \vdash^X \mathsf{tu} : \mathscr{b} \rightarrowtail \mathbf{C}^{\vec{\mathfrak{q}}}\sigma} \ (@)$$

$$\frac{\Gamma \vdash^X \mathsf{t} : \mathscr{c} \rightarrowtail \mathbf{C}^{\vec{\mathfrak{q}}}(\mathfrak{s} \Rightarrow \sigma) \qquad \Gamma \vdash^X \mathsf{u} : \mathscr{d} \rightarrowtail \mathbf{C}^{\mathfrak{r}}\mathfrak{s} \qquad \mathscr{b} \vDash^X \mathscr{c} \wedge \mathscr{d}}{\Gamma \vdash^X \{\mathsf{t}\}\mathsf{u} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{r}\mathfrak{s}*\vec{\mathfrak{q}}}\sigma} \ (\{\})$$

Counting Rule

$$\frac{\Gamma \vdash^{X\cup\{a\}} \mathsf{t} : \mathscr{b} \wedge \mathscr{c} \rightarrowtail \mathfrak{s} \qquad \mu(\mathscr{c}) \geq q}{\Gamma \vdash^X \nu a.\mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{q}}\mathfrak{s}} \ (\mu)$$

where $\mathsf{FN}(\mathscr{b}) \subseteq X, \mathsf{FN}(\mathscr{c}) \subseteq \{a\}, a \notin X$

Figure 9.1: Rules of $\mathbf{C}\lambda_{\rightarrow}^{\{\}}$

$$\frac{\Gamma \vdash^{X\cup\{a\}} \mathsf{t} : \mathscr{b} \wedge \mathscr{c} \rightarrowtail \mathbf{C}^{\mathfrak{q}}\sigma \qquad \mu(\mathscr{c}) \geq s}{\Gamma \vdash^X \nu a.\mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{q}\mathfrak{s}}\sigma} \ (\mu')$$

with $\mathsf{FN}(\mathscr{b}) \subseteq X, \mathsf{FN}(\mathscr{c}) \subseteq \{a\}$ and $a \notin X$

Let us now compare the two typing systems $\mathbf{C}\lambda_{\rightarrow}$ and $\mathbf{C}\lambda_{\rightarrow}^{\{\}}$ by means of the example below.

Example 9.1.1 (Typing Church Numerals)**.** Let us consider the typing derivation in $\mathbf{C}\lambda_{\rightarrow}^{\{\}}$ of the CbN Church numeral $2^{\mathsf{CbN}} = \lambda y\lambda x.\{y\}(yx)$ with type $\mathbf{C}^{\mathfrak{q}*\mathfrak{q}}(\mathbf{C}^{\mathfrak{q}}(o \Rightarrow o) \Rightarrow (o \Rightarrow o))$:

$$\dfrac{\dfrac{y : \mathbf{C}^{\mathsf{q}}(o \Rightarrow o), x : o \vdash y : \top \rightarrowtail \mathbf{C}^{\mathsf{q}}(o \Rightarrow o) \qquad \mathcal{D}}{y : \mathbf{C}^{q}(o \Rightarrow o), x : o \vdash \{y\}(yx) : \top \rightarrowtail \mathbf{C}^{\mathsf{q}*\mathsf{q}} o}\,(\{\})}{\vdash 2^{\mathsf{CbN}} : \top \rightarrowtail \mathbf{C}^{\mathsf{q}*\mathsf{q}}(\mathbf{C}^{\mathsf{q}}(o \Rightarrow o) \Rightarrow (o \Rightarrow o))}\,(\lambda)$$

where $\mathcal{D}$ is the derivation,

$$\dfrac{y : \mathbf{C}^{\mathsf{q}}(o \Rightarrow o), x : o \vdash y : \top \rightarrowtail \mathbf{C}^{\mathsf{q}}(o \Rightarrow o) \qquad y : \mathbf{C}^{\mathsf{q}}(o \Rightarrow o), x : o \vdash x : \top \rightarrowtail o}{y : \mathbf{C}^{\mathsf{q}}(o \Rightarrow o), x : o \vdash yx : \top \rightarrowtail \mathbf{C}^{\mathsf{q}} o}\,(@)$$

On the other hand, the derivation in $\mathbf{C}\lambda_{\rightarrow}$ of the CbV Church numeral $2^{\mathsf{CbV}} = \lambda f.\{2\}f$ with type $\mathbf{C}^{\mathsf{q}}(\mathbf{C}^{\mathsf{q}}(o \Rightarrow o) \Rightarrow (o \Rightarrow o))$

$$\dfrac{\dfrac{\vdots}{y : \mathbf{C}^{\mathsf{q}}(o \Rightarrow o) \vdash 2 : \top \rightarrowtail (o \Rightarrow o) \Rightarrow (o \Rightarrow o)} \qquad \dfrac{}{y : \mathbf{C}^{\mathsf{q}}(o \Rightarrow o) \vdash y : \top \rightarrowtail \mathbf{C}^{\mathsf{q}}(o \Rightarrow o)}}{\dfrac{y : \mathbf{C}^{\mathsf{q}}(o \Rightarrow o) \vdash \{2\}y : \top \rightarrowtail \mathbf{C}^{\mathsf{q}}(o \Rightarrow o)}{\vdash 2^{\mathsf{CbV}} \rightarrowtail \mathbf{C}^{\mathsf{q}}(\mathbf{C}^{\mathsf{q}}(o \Rightarrow o) \Rightarrow (o \Rightarrow o))}\,(\lambda)}\,(\{\})$$

Notice also that these systems enjoy the subject reduction property.[2]

Proposition 9.1.1 (Subject Reduction)**.** *If* $\Gamma \vdash^{X} \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s}$ *in* $\mathbf{C}\lambda_{\rightarrow}$ *(resp., in* $\mathbf{C}\lambda_{\rightarrow}^{\{\}}$*) and* $\mathsf{t} \twoheadrightarrow \mathsf{u}$ *(resp.,* $\mathsf{t} \twoheadrightarrow_{\{\}} \mathsf{u}$*), then* $\Gamma \vdash^{X} \mathsf{u} : \mathscr{b} \rightarrowtail \mathfrak{s}$*.*

Normalization. Both $\mathbf{C}\lambda_{\rightarrow}$ and $\mathbf{C}\lambda_{\rightarrow}^{\{\}}$ can type *non-normalizable* terms. For instance, one can type terms of the form $\mathsf{I} \oplus^{i} \Omega$ in $\mathbf{C}\lambda_{\rightarrow}^{\{\}}$ with $\mathbf{C}^{1/2}(o \Rightarrow o)$ and in $\mathbf{C}\lambda_{\rightarrow}$ with $\mathbf{C}^{1/2}(\mathbf{C}^{1} o \Rightarrow o)$. The failure of normalization for typable programs can be ascribed to the rule $(\vee)$, as shown by Theorem 9.1.1.

Notation 9.1.1. Let $\Gamma \vdash_{\neg\vee} \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s}$ indicate that $\Gamma \vdash \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s}$ is derived without using the rule $(\vee)$.

Theorem 9.1.1 (Deterministic Normalization)**.** *In both* $\mathbf{C}\lambda_{\rightarrow}$ *and* $\mathbf{C}\lambda_{\rightarrow}^{\{\}}$*, if* $\Gamma \vdash_{\neg\vee} \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s}$*, then* t *is strongly normalizing.*

As observed before, for a term of $\Lambda_{\mathbf{PE}}$ to have a normal form excludes the most interesting part of the calculus, which is made of terms for which normalization is inherently probabilistic. Similarly, the restriction to type derivations without $(\vee)$ trivializes the most interesting features of $\mathbf{C}\lambda_{\rightarrow}$ and $\mathbf{C}\lambda_{\rightarrow}^{\{\}}$, that is, the possibility to estimate probabilities of termination.

[2] For further details, see [12].

9.2 Relating $\mathrm{ND}_{\mathbf{iCPL}_0}$ and $\mathrm{C}\lambda^{\{\}}_{\rightarrow}$

In this section we show how derivations in $\mathbf{iCPL}_0$ translate into typing derivations in $\mathbf{C}\lambda^{\{\}}_{\rightarrow}$.

Definition 9.2.1 (Static Translation)**.** For any formula of $\mathbf{iCPL}_0$ F, we define a corresponding type $\mathfrak{s}_F$ by letting:

$$\begin{aligned}\mathfrak{s}_P &:= o\\ \mathfrak{s}_{F\rightarrow \mathbf{C}^{\vec{\mathfrak{q}}}G} &:= \mathbf{C}^{\vec{\mathfrak{q}}}(\mathfrak{s}_F \Rightarrow \mathfrak{s}_G)\\ \mathfrak{s}_{\mathbf{C}^{\mathfrak{q}}F} &:= \mathbf{C}^{\mathfrak{q}}\mathfrak{s}_F.\end{aligned}$$

The translation of a derivation Π of $\Gamma \vdash \mathscr{b} \rightarrowtail F$ in $\mathbf{iCPL}_0$ into a typing derivation $\mathcal{D}^{\Pi}$ of $\mathfrak{s}_\Gamma \vdash \mathsf{t}^{\Pi} : \mathscr{b} \rightarrowtail \mathfrak{s}_F$ in $\mathbf{C}\lambda^{\{\}}_{\rightarrow}$, with $\mathsf{FN}(\mathsf{t}^{\Pi}) \subseteq \mathsf{FN}(\mathscr{b})$ is defined by induction on the height of Π as illustrated in Figure 9.2. and 9.3.

Notice that the rule $\rightarrow \mathbf{E}$ translates as the CbN application tu, while the rule $\mathbf{CE}$ translates as the CbV application $\{\mathsf{t}\}\mathsf{u}$.

Furthermore, as required by the CHC, normalization steps of $\mathbf{iCPL}_0$ are simulated by $\twoheadrightarrow_{\{\}}$-reductions:

Proposition 9.2.1 (Stability Under Normalization)**.** *If* $\Pi \rightsquigarrow \Pi'$*, then* $\mathsf{t}^{\Pi} \twoheadrightarrow^{*}_{\{\}} \mathsf{t}^{\Pi'}$.

Proof Sketch. The normalization step $(\rightarrow \mathbf{I}/\rightarrow \mathbf{E})$ translates into β-reduction:

$$(\lambda x^{\mathbf{C}^{\mathfrak{q}}\mathfrak{s}_F}.\mathsf{t}^{\Pi})\nu a.\mathsf{t}^{\Pi'} \rightarrow_{\beta} \mathsf{t}^{\pi}[\nu a.\mathsf{t}^{\Pi'}/x].$$

The normalization step $(\mathbf{CI}/\mathbf{CE})$ translates into the chain of reductions:

$$\{\lambda^{\mathfrak{s}_F}.\mathsf{t}^{\Pi}\}\nu a.\mathsf{t}^{\Pi'} \twoheadrightarrow_{p\{\}} \nu a.(\lambda x^{\mathfrak{s}_F}.\mathsf{t}^{\Pi})\mathsf{t}^{\Pi'} \rightarrow_{\beta} \nu a.\mathsf{t}^{\Pi}[\mathsf{t}^{\Pi'}/x].$$

All other normalization steps translate into $\twoheadrightarrow_{p\{\}}$-reductions. □

Observe that the detour conversion for $(\rightarrow \mathbf{I}/\rightarrow \mathbf{E})$ translates into CbN reduction, that is plain β-reduction: so, the "choice" $\nu a.\mathsf{t}^{\Pi'}$ is directly substituted, and thus possibly duplicated. On the contrary, the normalization step $(\mathbf{CI}/\mathbf{CE})$ translates into CbV reduction, that is $(\nu\mathsf{f})$ followed by β-reduction: so, the generator νa is first permuted down and only $\mathsf{t}^{\Pi'}$ is substituted.

Since only $(\vee)$ introduces the constant $\mathbf{c}$, from Theorem 9.1.1, we conclude the as follows.

Corollary 9.2.1. $\mathbf{iCPL}_0$ *is strongly normalizing.*

9.3 Relating $\mathrm{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$ and $\mathrm{C}\lambda_{\rightarrow}$

In this section we define a CHC between the CbN variant of the proof system for $\mathbf{iCPL}_0$, namely $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$, and the type system $\mathbf{C}\lambda_{\rightarrow}$ above.

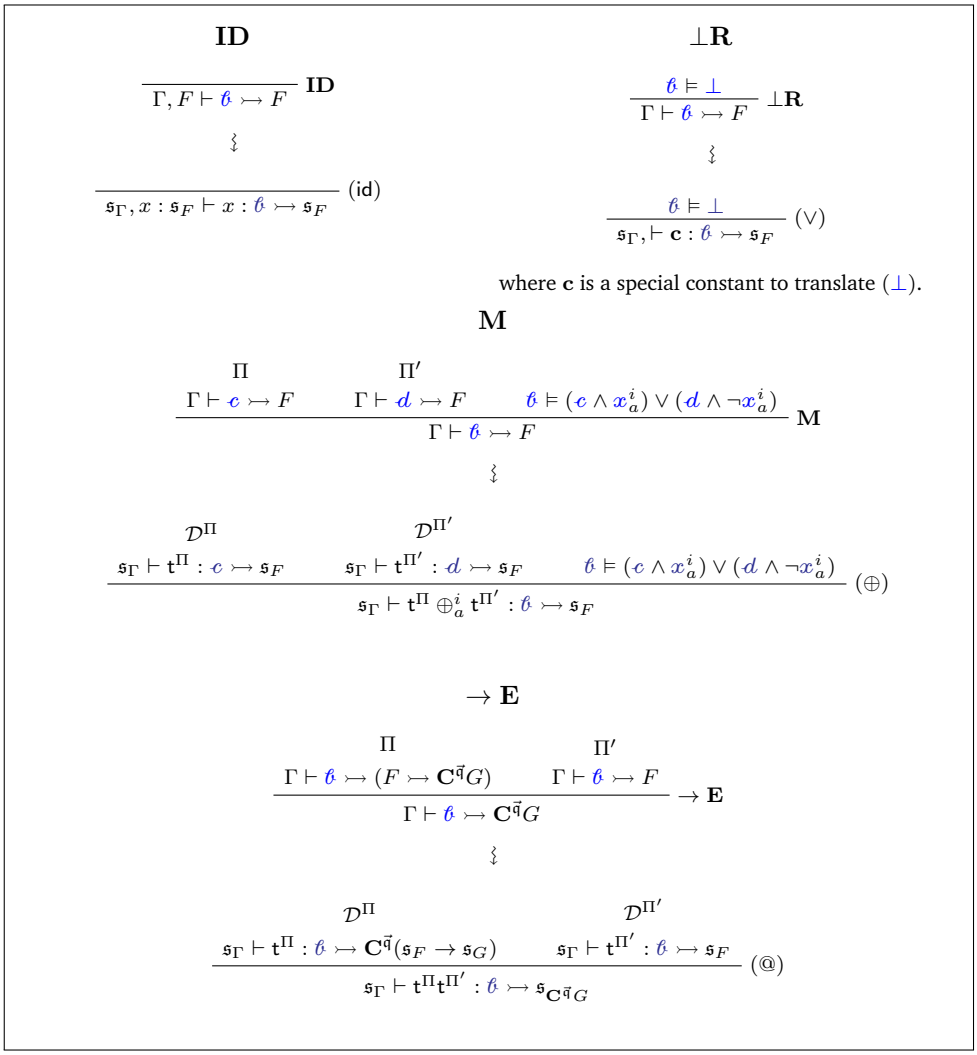


Figure 9.2: Translation $\Pi \rightsquigarrow \mathcal{D}^\Pi$ from $\mathbf{ND}_{\mathbf{iCPL}}$ to $\mathbf{C}\lambda^{\{\}}_{\rightarrow}$

Head Context. The translation from $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$ to $\mathbf{C}\lambda_\rightarrow$ relies on properties of head context in $\Lambda_{\mathbf{PE}}$.

Definition 9.3.1 (Head Context)**.** *Head contexts* $\mathsf{H}[\,]$ are defined by the grammar below:

$$\mathsf{H}[\,] := [\,] \ \mid\ \lambda x.\mathsf{H} \ \mid\ \mathsf{H}[\,]\mathsf{u}.$$

The fundamental property of head contexts is that they naturally behave as CbV functions, due to the following Lemma 9.3.1.

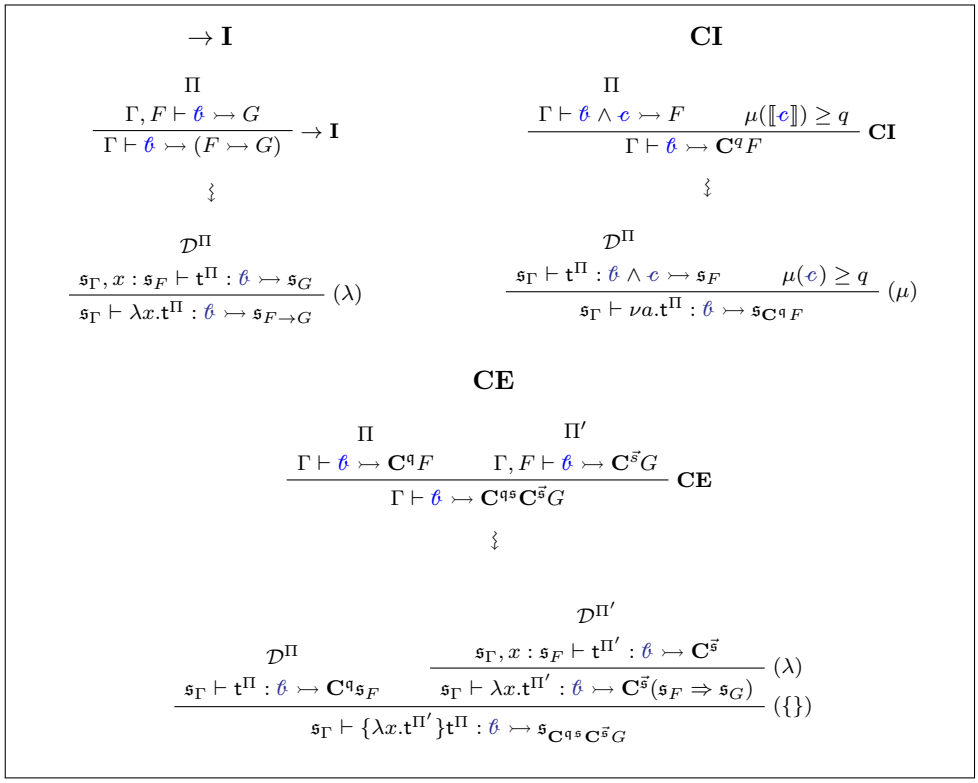


Figure 9.3: Translation $\Pi \rightsquigarrow \mathcal{D}^{\Pi}$ from $\mathbf{ND}_{\mathbf{iCPL}}$ to $\mathbf{C}\lambda^{\{\}}_{\rightarrow}$ (continuation)

Lemma 9.3.1. *For any head context* $\mathsf{H}[\,]$ *and term* t*,*

$$\mathsf{H}[\nu a.\mathsf{t}] \twoheadrightarrow_p^* \nu a.\mathsf{H}[\mathsf{t}].$$

Proof. The proof is by induction on the structure of $\mathsf{H}[\,]$:

- $\mathsf{H}[\,] = [\,]$. Then, the claim is immediate.
- $\mathsf{H}[\,] = \lambda x.\mathsf{H}'[\,]$. Then,

$$\mathsf{H}[\nu a.\mathsf{t}] = \lambda x.\mathsf{H}'[\nu a.\mathsf{t}] \overset{IH}{\twoheadrightarrow}_p^* \twoheadrightarrow_p \nu a.\lambda x.\mathsf{H}'[\mathsf{t}] = \nu a.\mathsf{H}[\mathsf{t}].$$

- $\mathsf{H}[\,] = \mathsf{H}'[\,]\mathsf{t}$. Then,

$$\mathsf{H}[\nu a.\mathsf{t}] = \mathsf{H}'[\nu a.\mathsf{t}]\mathsf{u} \overset{IH}{\twoheadrightarrow}_p^* (\nu a.\mathsf{H}'[\mathsf{t}])\mathsf{u} \twoheadrightarrow_p \nu a.\mathsf{H}'[\mathsf{t}]\mathsf{u} = \nu a.\mathsf{H}[\mathsf{t}].$$

□

Otherwise said, whenever t is a function of the form $\lambda x.\mathsf{H}[x]$ for some head context $\mathsf{H}[\ ]$, CbN and CbV application of t coincide, since $\mathsf{t}(\nu a.\mathsf{u})$ and $\nu a.\mathsf{t}\mathsf{u}$ have the same normal form.

Lemma 9.3.2. *For any head context H, the following rule of head-substitution is derivable in $\mathbf{C}\lambda_{\to}$:*

$$\frac{\Gamma \vdash \mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{q}\mathfrak{s}}\sigma \qquad x : \mathbf{C}^{\mathfrak{q}}\,\sigma, \Gamma \vdash \mathsf{H}[x] : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{r}}\tau}{\Gamma \vdash \mathsf{H}[\mathsf{t}] : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{r}\mathfrak{s}}\tau}\ (\mathsf{hs})$$

Proof. The proof is by induction on $\mathsf{H}[\ ]$:

- $\mathsf{H}[\ ] = [\ ]$. Then, $q = r$ and the claim is immediate.
- $\mathsf{H}[\ ] = \lambda y.\mathsf{H}'[\ ]$. Then, $\tau = \mathfrak{t} \Rightarrow \tau'$ and $x : \mathbf{C}^{\mathfrak{q}}\sigma, \Gamma, y : \mathfrak{t} \vdash \mathsf{H}'[x] : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{q}}\tau'$. Then, by IH, we deduce $\Gamma, y : \mathfrak{t} \vdash \mathsf{H}'[\mathsf{t}] : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{r}\mathfrak{s}}\tau'$ and conclude $\Gamma \vdash \mathsf{H}[\mathsf{t}] : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{r}\mathfrak{s}}\tau$.
- $\mathsf{H}[\ ] = \mathsf{H}'[\ ]\mathsf{u}$. Then, $x : \mathbf{C}^{\mathfrak{q}}\sigma, \Gamma \vdash \mathsf{H}'[x] : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{r}}(\mathfrak{t} \Rightarrow \tau)$ and $x : \mathbf{C}^{\mathfrak{q}}\sigma, \Gamma \vdash \mathsf{u} : \mathscr{b} \rightarrowtail \mathfrak{t}$. So, by IH, we deduce $\Gamma \vdash \mathsf{H}'[\mathsf{t}] : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{r}\mathfrak{s}}(\mathfrak{t} \Rightarrow \tau)$ and conclude $\Gamma \vdash \mathsf{H}[\mathsf{t}] : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathfrak{r}\mathfrak{s}}\,\tau$.

□

Static and Dynamic Translation. It is now possible to show how derivations in $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}_0}$ translate into type derivations in $\mathbf{C}\lambda_{\to}$.

Definition 9.3.2 (Static Translation)**.** Any formula of $\mathbf{iCPL}_0$ F, is associated with a non-quantified type σ_F and a positive real $|F| \in (0,1] \cap \mathbb{Q}$ of $\mathbf{C}\lambda_{\to}$ as follows:

$$\begin{aligned} \sigma_P &:= o & |o| &:= 1 \\ \sigma_{F\to G} &:= (\mathbf{C}^{|F|}\sigma_F) \Rightarrow \sigma_G & |F \to G| &:= |G| \\ \sigma_{\mathbf{C}^{\mathfrak{q}}F} &:= \sigma_F & |\mathbf{C}^{q}F| &:= q \cdot |F|. \end{aligned}$$

Then, let:

$$\mathfrak{s}_F := \mathbf{C}^{|F|}\sigma_F.$$

Observe that $\mathfrak{s}_{\mathbf{C}^{\mathfrak{q}}F} = \mathbf{C}^{\mathfrak{q}\cdot|F|}\sigma_F$.

The translation of a derivation Π of $\Phi; \Gamma \vdash \mathscr{b} \rightarrowtail F$ into a typing derivation $\mathcal{D}^{\Pi}$ of $\mathfrak{s}_\Phi, \mathfrak{s}_\Gamma \vdash \mathsf{t}^{\Pi} : \mathscr{b} \rightarrowtail \mathfrak{s}_F$ in $\mathbf{C}\lambda_{\to}$ is illustrated in Figure 9.3 and 9.5, where we exploit the fact that if $\Phi = \{F\}$ is non-empty, then $\mathsf{t}^{\Pi} = \mathsf{t}^{\Pi}[x : \mathfrak{s}_F]$ is a head context (as it can be checked by induction on the construction).[3]

[3]The case of the rule $(\mathbf{C}_{\times})$ is omitted as it follows immediately from the IH, since $\mathfrak{s}_{\mathbf{C}^q\mathbf{C}^s F} = \mathfrak{s}_{\mathbf{C}^{qs}F}$. Here, the stability of the translation under normalization is easily checked using Lemma 9.3.1.

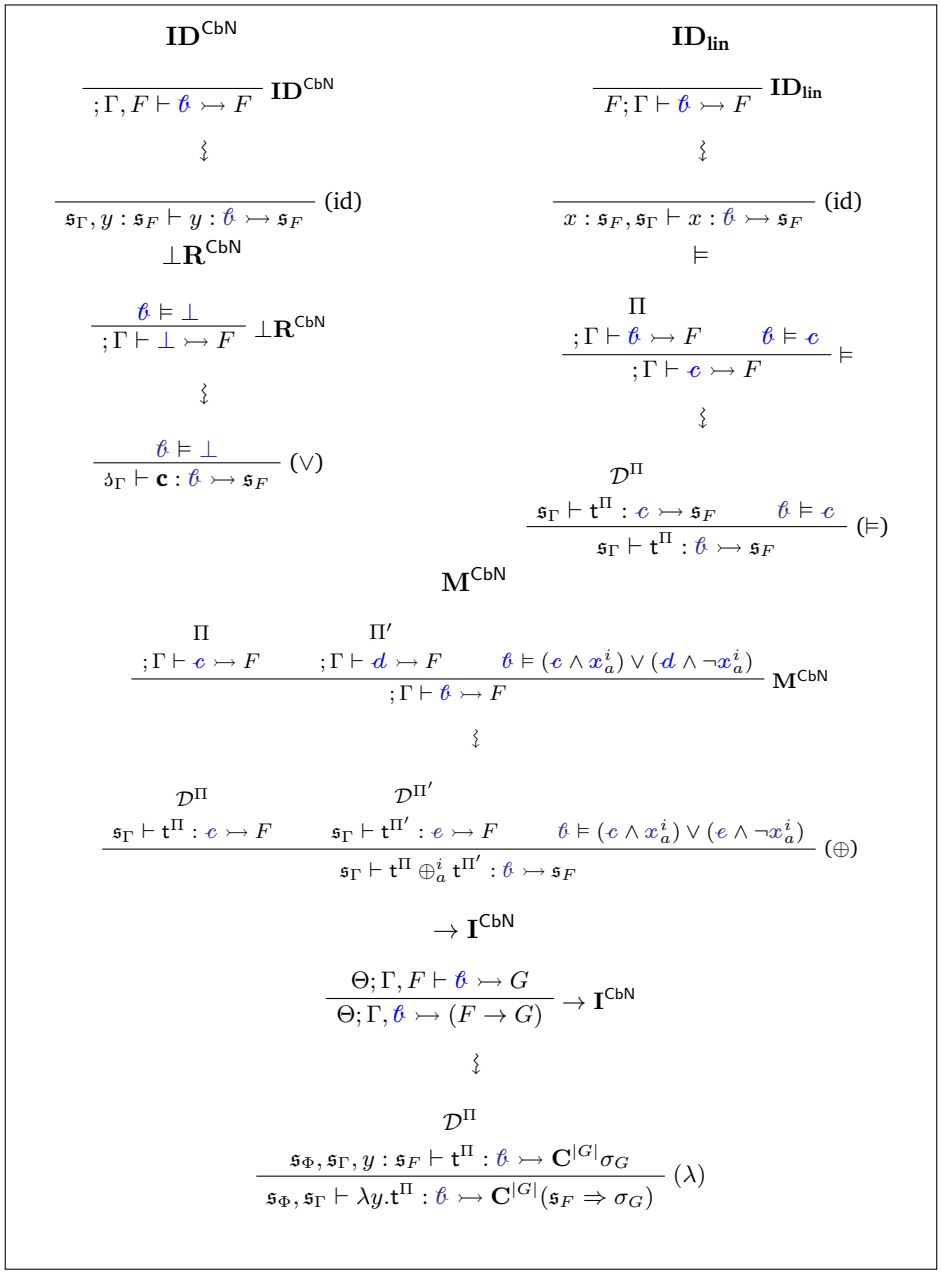


Figure 9.4: Translation $\Pi \rightsquigarrow \mathcal{D}^\Pi$ from $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}}$ to $\mathbf{C}\lambda_\rightarrow$ (continuation)

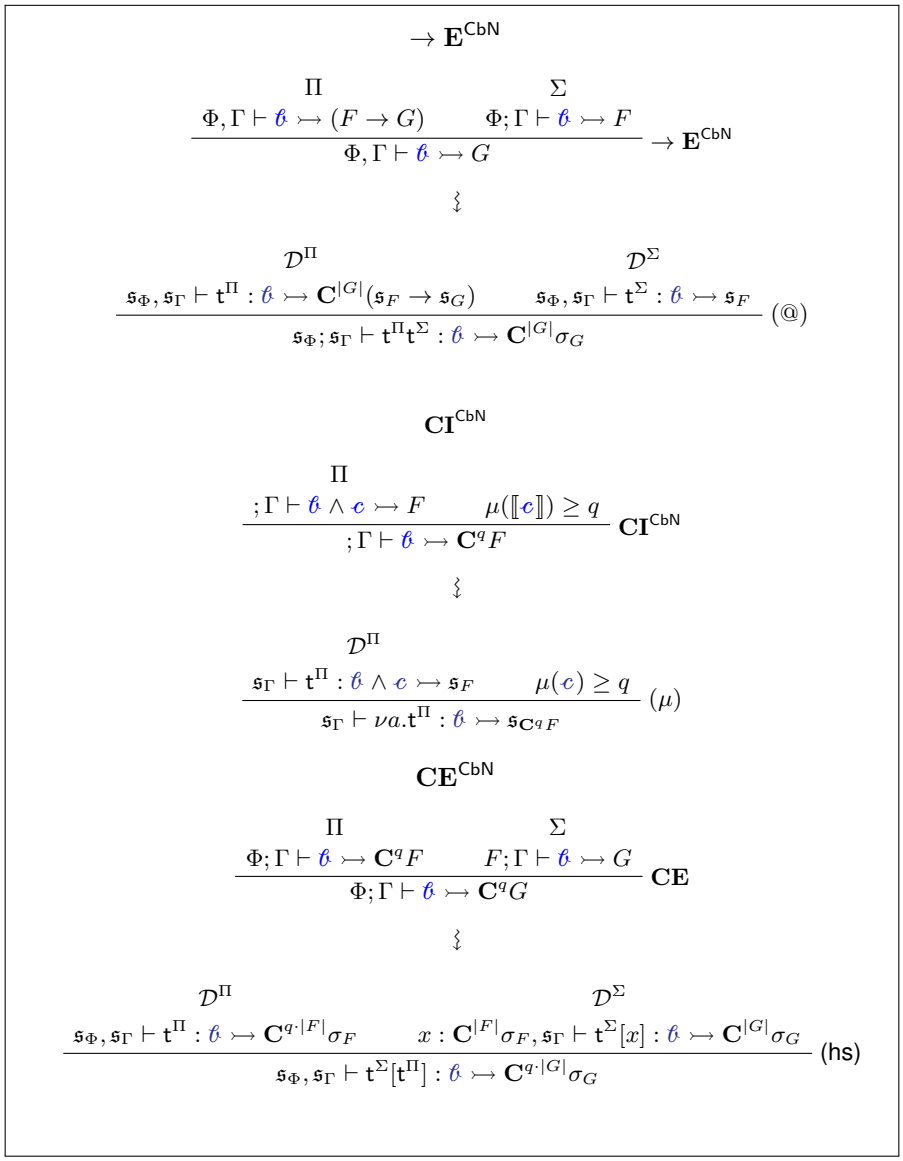


Figure 9.5: Translation $\Pi \rightsquigarrow \mathcal{D}^{\Pi}$ from $\mathbf{ND}^{\mathsf{CbN}}_{\mathbf{iCPL}}$ to $\mathbf{C}\lambda_{\rightarrow}$ (continuation)

Chapter 10

From Type Soundness to Type Completeness

In this chapter we show that derivations in $\mathbf{C}\lambda_{\to}$ and $\mathbf{C}\lambda^{\{\}}_{\to}$ provide sound approximations of HNV(t) and NF(t). In order to achieve completeness, in Section 10.2, we introduce an extension of $\mathbf{C}\lambda_{\to}$ with intersection types, called $\mathbf{C}\lambda_{\to,\cap}$. This type system is shown expressive enough to capture both deterministic and probabilistic notions of termination for $\Lambda_{\mathbf{PE}}$. We conclude by comparing our systems and result with the ones presented in the literature in Section 10.3.

10.1 From Types to Probability

As seen, if a term t has type $\mathbf{C}^{1/2}(o \Rightarrow o)$, then t has one chance out of two of yielding a "correct" program for $o \Rightarrow o$. The result below makes this intuition precise, by showing that the probabilities derived in $\mathbf{C}\lambda_{\to}$ and $\mathbf{C}\lambda^{\{\}}_{\to}$ are *lower bounds* for the function $\mathrm{HNV}_{\to}(\mathsf{t})$, that is for the actual probability of finding a head normalizable term in the distribution $\mathscr{D}_{\mathsf{t}}$.[1]

Theorem 10.1.1. *i. If* $\vdash_{\mathbf{C}\lambda_{\to}} \mathsf{t} : \bot \rightarrowtail \mathbf{C}^{\mathfrak{q}}\sigma$, *then* $\mathit{HNV}_{\to}(\mathsf{t}) \geq q$.

ii. If $\vdash_{\mathbf{C}\lambda^{\{\}}_{\to}} \mathsf{t} : \top \rightarrowtail \mathbf{C}^{\mathfrak{q}_1 * \cdots * \mathfrak{q}_n}\sigma$, *then* $\mathit{HNV}_{\to}(\mathsf{t}) \geq \prod_{i=1}^{n} q_i$.

Yet, what about reduction to normal form, namely the function $\mathrm{NF}_{\to}(\mathsf{t})$? A result like Theorem 10.1.1 cannot hold in this case, as shown by the example below.

Example 10.1.1. Consider the term:

$$\mathsf{t} = \lambda y.y(\mathsf{I} \oplus^{i} \Omega).$$

While $\mathrm{NF}(\mathsf{t}) = \frac{1}{2}$, $\mathbf{C}\lambda^{\{\}}_{\to}$ types t with $\mathfrak{s} = \mathbf{C}^{1}(\mathbf{C}^{1}(\mathbf{C}^{1/2}\sigma \Rightarrow \sigma) \Rightarrow \sigma)$, with $\sigma = o \Rightarrow o$. Then, type $\mathfrak{s}$ contains the "unbalanced" assumption $\mathbf{C}^{1}(\mathbf{C}^{1/2}\sigma \Rightarrow \sigma)$

[1] For further details and the proof of Theorem 10.1.1, see [11, 12].

– corresponding in logical terms to the formula $\mathbf{C}^{1/2}F \to \mathbf{C}^{1}F$ – that is, exploits the assumption of the existence of a function turning a $\frac{1}{2}$-correct input into a 1-correct output. Notice that such a function f can only be one that erases its input, and these are the only functions such that $\mathtt{t}f$ can reduce to a normal form

Nevertheless, soundness with respect to $\mathrm{NF}(\mathtt{t})$ can be proved for $\mathbf{C}\lambda_{\to}$, by restricting to types not containing "unbalanced" assumptions, that is to types corresponding to programs not increasing probabilities.

Definition 10.1.1. For any type $\mathfrak{s}$ of $\mathbf{C}\lambda_{\to}$ of the form $\mathbf{C}^{q}\sigma$, let $\lceil \mathfrak{s} \rceil = q$. A type $\mathbf{C}^{q}(\mathfrak{s}_1 \Rightarrow \ldots \Rightarrow \mathfrak{s}_i \Rightarrow o)$ of $\mathbf{C}\lambda_{\to}$ is *balanced* if all $\mathfrak{s}_i$ are balanced and $q \leq \prod_{i=1}^{n} \lceil \mathfrak{s}_i \rceil$.

Theorem 10.1.2. *If* $\vdash \mathtt{t} : \top \rightarrowtail \mathfrak{s}$ *is derivable in* $\mathbf{C}\lambda_{\to}$*, where* $\mathfrak{s}$ *is balanced, then* $NF_{\to}(\mathtt{t}) \geq \lceil \mathfrak{s} \rceil$.

Both Theorem 10.1.1 and 10.1.2 are proved by adapting the standard technique of *reducibility predicates* to the quantitative notion of probabilistic normal form.[2]

10.2 From Probability to (Intersection) Types

To achieve a type-theoretic characterization of $\mathrm{HNV}_{\to}(\mathtt{t})$ and $\mathrm{NF}_{\to}(\mathtt{t})$, we introduce an extension of $\mathbf{C}\lambda_{\to}$ with intersection types, called $\mathbf{C}\lambda_{\to,\cap}$.

Adding Intersection Types. As for $\mathbf{C}\lambda_{\to}$, types are of the form $\mathfrak{s} = \mathbf{C}^{\mathsf{q}}\sigma$, but the grammar for σ is richer than that of Definition 9.1.1.

Definition 10.2.1 (Types for $\mathbf{C}\lambda_{\to,\cap}$)**.** The grammar for types of $\mathbf{C}\lambda_{\to,\cap}$ is as follows:

$$\sigma := o \;\Big|\; \mathsf{n} \;\Big|\; \mathsf{hn} \;\Big|\; \mathfrak{M} \Rightarrow \sigma$$

$$\mathfrak{M} := [\mathfrak{s}, \ldots, \mathfrak{s}]$$

$$\mathfrak{s} := \mathbf{C}^{\mathsf{q}}\sigma,$$

where $[a_1, \ldots, a_n]$ denotes a finite set.

Observe that $\mathfrak{M}$ intuitively stands for a finite intersection of types, and the new ground types n and hn correspond to the types of normalizable and head-normalizable programs. We also introduce a preorder $\sigma \preceq \tau$ over types by $\alpha \preceq \alpha$, for $\alpha = o, \mathsf{n}, \mathsf{hn}$, $\mathbf{C}^{q}\sigma \preceq \mathbf{C}^{p}\tau$ if $q \leq p$ and $\sigma \preceq \tau$, and $(\mathfrak{M} \Rightarrow \sigma) \preceq (\mathfrak{N} \Rightarrow \tau)$ if $\sigma \preceq \tau$ and $\mathfrak{N} \preceq^{*} \mathfrak{M}$, where $[\mathfrak{s}_1, \ldots, \mathfrak{s}_n] \preceq^{*} [\mathfrak{t}_1, \ldots, \mathfrak{t}_m]$ holds if there is an injective function $f : \{1, \ldots, m\} \mapsto \{1, \ldots, n\}$ such that $\mathfrak{s}_{f(i)} \preceq \mathfrak{t}_i$.

[2] For further details, see [12].

Typing Rules of $\mathbf{C}\lambda_{\to,\cap}$. Type judgments are defined as expected.

Definition 10.2.2 (Judgments of $\mathbf{C}\lambda_{\to,\cap}$)**.** A type judgment of $\mathbf{C}\lambda_{\to,\cap}$ is of the form $\Gamma \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s}$, where Γ is made of declarations of the form $x_i : \mathfrak{M}_i$.

We also introduce typing rules for $\mathbf{C}\lambda_{\to,\cap}$.

Definition 10.2.3 (Typing Rules of $\mathbf{C}\lambda_{\to,\cap}$)**.** Typing rules of $\mathbf{C}\lambda_{\to,\cap}$ are illustrated in Figure 10.1, except for $(\vee)$- and $(\oplus)$-rules which are omitted. They are the same as those for $\mathbf{C}\lambda^{\{\}}_{\to}$.

Identity Rules

$$\frac{\mathsf{exists}\ i.\mathfrak{s}_i \preceq \mathsf{t} \qquad \mathsf{FN}(\mathscr{b}) \subseteq X}{\Gamma, x : [\mathfrak{s}_1, \ldots, \mathfrak{s}_n] \vdash^X x : \mathscr{b} \rightarrowtail \mathsf{t}}\ (\mathsf{id}_{\preceq})$$

Ground Types Rules

$$\frac{\Gamma \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathsf{q}}\sigma}{\Gamma \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathsf{q}}\mathsf{hn}}\ (\mathsf{hn}) \qquad \frac{\Gamma \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathsf{q}}\sigma \qquad \sigma\ \mathsf{safe}}{\Gamma \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathsf{q}}\mathsf{n}}\ (\mathsf{n})$$

Arrow Rules

$$\frac{\Gamma, x : \mathfrak{M} \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathsf{q}}\sigma}{\Gamma \vdash^X \lambda x.\mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathsf{q}}(\mathfrak{M} \Rightarrow \sigma)}\ (\lambda)$$

$$\frac{\Gamma \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathsf{q}}(\mathfrak{M} \Rightarrow \sigma) \qquad \left\{\Gamma \vdash^X \mathsf{u} : \mathscr{b} \rightarrowtail \mathfrak{s}_i\right\}_{i=1,\ldots,n}}{\Gamma \vdash^X \mathsf{tu} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathsf{q}}\sigma}\ (@_{\cap})$$

$\mathfrak{M} = [\mathfrak{s}_1, \ldots, \mathfrak{s}_n]$

Counting Rules

$$\frac{\left\{\Gamma \vdash^{X\cup\{a\}} \mathsf{t} : \mathscr{b} \wedge \mathscr{d}_i \rightarrowtail \mathbf{C}^{\mathsf{q}_i}\sigma\right\}_{i=1,\ldots,n} \qquad \mu(\mathscr{d}_i) \geq p_i}{\Gamma \vdash^X \nu a.\mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\sum_i \mathsf{q}_i \mathsf{p}_i}\sigma}\ (\mu_{\Sigma})$$

Figure 10.1: Typing Rules of $\mathbf{C}\lambda_{\to,\cap}$

In the rule (μ_{Σ}) it is assumed that a does not occur in $\mathscr{b}$, is the only name in the $\mathscr{d}_i$, and that for $i \neq j$, $\mathscr{d}_i \wedge \mathscr{d}_j \vDash \bot$. The two rules (hn) and (n) are justified by Proposition 10.2.2 and Theorem 10.2.1 below. As rule (n) must warrant a bound on normal forms, following Theorem 10.1.2, σ has to be *safe*, that is balanced[3] and $\{[\,], \mathsf{hn}\}$-free. The rule $(@_{\cap})$ is a standard extension of rule $(@)$ of $\mathbf{C}\lambda_{\to}$ to finite intersections. The counting rule (μ_{Σ}) requires some discussion.

[3]Definition 10.1.1 extends to the types of $\mathbf{C}\lambda_{\to,\cap}$ by letting $\lceil \mathsf{hn} \rceil = \lceil [\,] \rceil = 0$, $\lceil \mathsf{n} \rceil = 1$ and $\lceil [\mathfrak{s}_1, \ldots, \mathfrak{s}_{n+1}] \rceil = max\{\lceil \mathfrak{s}_i \rceil\}$.

The rule admits $n+1$ major premisses expressing typings for t, which depend on pairwise disjoint events (the Boolean formulae $\mathscr{d}_i$). This is needed to cope with situations like the following one. Let

$$\mathsf{t}[a,b] = \Big(\big(I \oplus_b^i \Omega\big) \oplus_b^0 \Omega\Big) \oplus_a^0 \big(\Omega \oplus_b^0 \mathsf{I}\big),$$

$\mathsf{t}[a,b]$ can be given type $\sigma = \mathbf{C}^1(\mathbf{C}^1 o \Rightarrow o)$ under either of the two *disjoint* Boolean constraints $\mathscr{d} = x_a^0 \wedge (x_b^0 \wedge x_b^i)$ and $\mathscr{d}_2 = \neg x_a^0 \wedge \neg x_b^0$. Notice that the term $\nu a.\nu b.\mathsf{t}[a,b]$ has probability $\mu(x_a^0)\mu(x_b^0 \wedge x_b^1) + \mu(\neg x_a^0)\mu(\neg x_b^0) = \frac{1}{2} \cdot \frac{1}{4} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{8}$ of yielding a head normal value. Yet, in $\mathbf{C}\lambda_{\to}$, the best we can achieve is $\vdash \nu a.\nu b.\mathsf{t}[a,b] : \top \rightarrowtail \mathbf{C}^{1/4}\sigma$, that is a probability estimation of $\frac{1}{4} < \frac{3}{8}$. Indeed, the rule (μ') forces us to approximate $\mu(x_b^0 \wedge x_b^1)$ and $\mu(\neg x_b^1)$ to a *common* lower bound – namely $\frac{1}{4}$ – in order to apply a $(\vee)$-rule as illustrated in Figure 10.2.**a**. Instead, using (μ_Σ) we can reach the *actual* probability $\frac{3}{8}$, as illustrated in Figure 10.2.**b**. Observe that, without intersection types, (μ_Σ) is not sufficient to achieve completeness. For example, given a term like $\lambda x.xx$, it normalizes with probability 1 (but a system without intersection types assigns only probability 0 to it).

Achieving Completeness. Thanks to the rule (μ_Σ), the generalized counting rule (μ^*),

$$\frac{\Gamma \vdash^{\{a_1,\dots,a_n\}} \mathsf{t} : \mathscr{b} \rightarrowtail \mathbf{C}^{\mathsf{q}}\sigma}{\Gamma \vdash^{\emptyset} \nu a_1.\dots.\nu a_n.\mathsf{t} : \top \rightarrowtail \mathbf{C}^{\mathsf{q}\cdot\mu(\mathscr{b})}\sigma}\ (\mu^*)$$

becomes admissible in $\mathbf{C}\lambda_{\to,\cap}$. This rule – together with the standard result that both *subject reduction* and *subject expansion* hold for intersection types – plays an essential role in the proof of completeness.

Proposition 10.2.1 (Subject Reduction and Subject Expansion)**.** *If* $\Gamma \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s}$ *and either* $\mathsf{t} \twoheadrightarrow \mathsf{u}$ *or* $\mathsf{u} \twoheadrightarrow \mathsf{t}$*, then* $\Gamma \vdash^X \mathsf{t} : \mathscr{b} \rightarrowtail \mathfrak{s}$.

Remarkably, typing in $\mathbf{C}\lambda_{\to,\cap}$ captures both deterministic and probabilistic properties of terms. First, we show that types hn and n capture deterministic termination.

Proposition 10.2.2 (Deterministic Completeness)**.** *For any closed term* t*,*

i. t *is head-normalizable when* $\vdash_{\neg\vee} \mathsf{t} : \top \rightarrowtail \mathbf{C}^1\mathsf{hn}$

ii. t *is normalizable when* $\vdash_{\neg\vee} \mathsf{t} : \top \rightarrowtail \mathbf{C}^1\mathsf{n}$

iii. t *is strongly normalizable when* $\vdash_{\neg\vee} \mathsf{t} : \top \rightarrowtail \mathbf{C}^1\mathsf{n}$ *and all types in the derivation are safe.*

Proof Sketch. Using standard arguments for intersection types we show that $\vdash_{\neg\vee} \mathsf{t} : \top \rightarrowtail \mathbf{C}^1\mathsf{hn}$ holds for any head-normal t. The first half of (i.) is deduced using Proposition 10.2.1. The second half follows from a normalization argument similar to that of Theorem 9.1.1. Cases (ii.) and (iii.) are similar. □

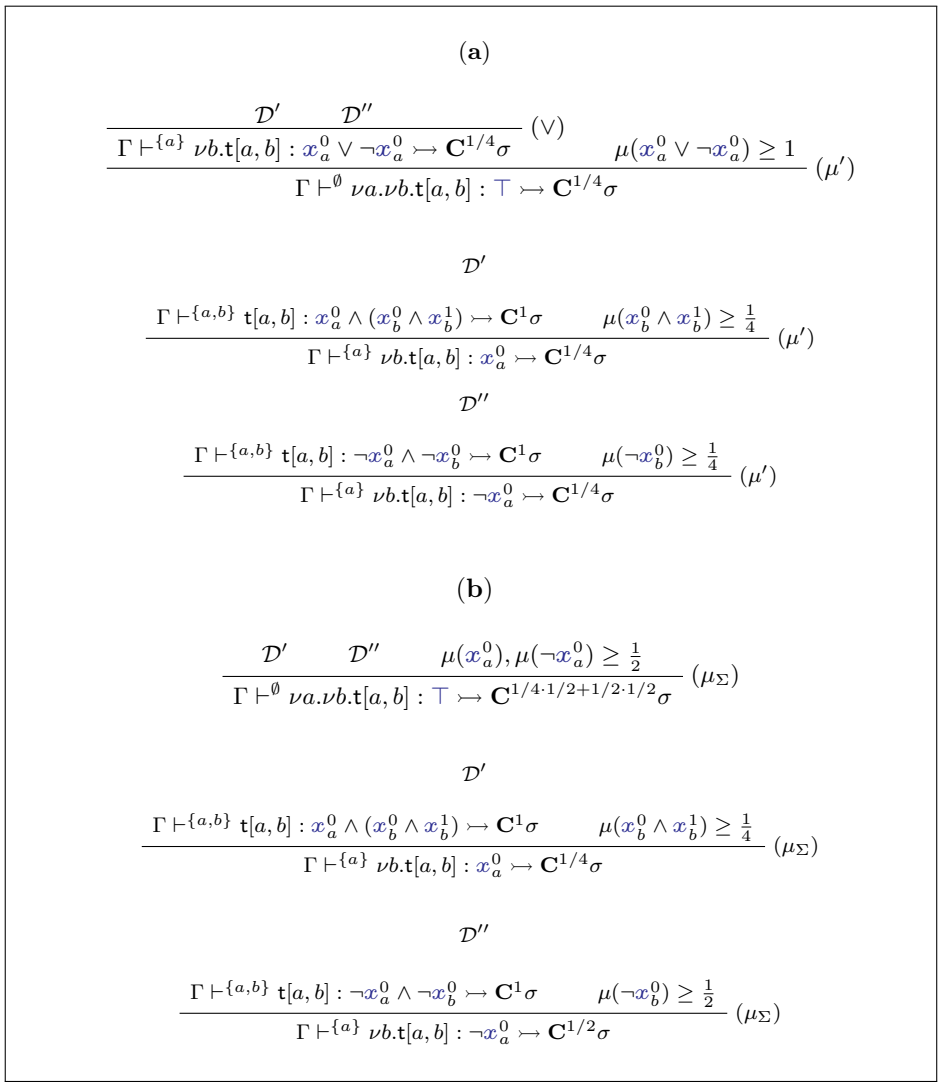


Figure 10.2: Comparing Probabilities Derived with the Rules (μ') and (μ_Σ)

The probabilistic normalization Theorems 10.1.1 and Theorem 10.1.2 extend smoothly to $\mathbf{C}\lambda_{\to,\cap}$. This ensures that if t has type $\mathbf{C}^q\mathsf{hn}$ (resp., $\mathbf{C}^q\mathbf{n}$), then $\mathrm{HNV}_{\twoheadrightarrow}(\mathsf{t}) \geq q$ (resp., $\mathrm{NF}_{\twoheadrightarrow}(\mathsf{t}) \geq q$). Conversely, $\mathrm{HNV}_{\twoheadrightarrow}(\mathsf{t})$ and $\mathrm{NF}_{\twoheadrightarrow}(\mathsf{t})$ can be bounded by means of derivations in $\mathbf{C}\lambda_{\to,\cap}$, in the following way.

Theorem 10.2.1 (Probabilistic Completeness)**.** *For any closed term* t*,*

$$HNV_{\to}(\mathsf{t}) = \mathit{supp}\{q \mid \vdash \mathsf{t} : \top \rightarrowtail \mathbf{C}^q\mathsf{hn}\}$$
$$NF_{\to}(\mathsf{t}) = \mathit{supp}\{q \mid \vdash \mathsf{t} : \top \rightarrowtail \mathbf{C}^q\mathsf{n}\}.$$

Proof Sketch. Without loss of generality, assume $\mathsf{t} = \nu a_1.\ldots.\nu a_k.\mathsf{t}'$. For any $\mathsf{u} \in \mathrm{HNV}$ such that $\mathscr{D}_{\mathsf{t}}(\mathsf{u}) > 0$, we deduce $\vdash \mathsf{u} : \top \rightarrowtail \mathsf{hn}$. The sequence of probabilistic choices leading to u is finite, and thus captured by a Boolean formula $\mathscr{b}_{\mathsf{t}\to\mathsf{u}}$. Using subject reduction/expansion we deduce $\vdash \mathsf{t}' : \mathscr{b}_{\mathsf{t}\to\mathsf{u}} \rightarrowtail \mathsf{hn}$. Hence, for any finite number of head normal forms $\mathsf{t}_1, \ldots, \mathsf{t}_n$ such that $\mathscr{D}_{\mathsf{t}}(\mathsf{u}_i) > 0$, we deduce $\vdash \mathsf{t}' : \mathscr{b}_{\mathsf{t}\to\mathsf{u}_i} \rightarrowtail \mathsf{hn}$. Using $(\vee)$ and the generalized counting rule (μ^*) we deduce $\vdash \mathsf{t} : \top \rightarrowtail \mathbf{C}^p\mathsf{hn}$, where $p = \sum_{i=1}^{n} \mu(\mathscr{b}_{\mathsf{t}\to\mathsf{u}_i}) = \mu(\bigvee_{i=1}^{n} \mathscr{b}_{\mathsf{t}\to\mathsf{u}_i})$. The argument for $\mathrm{NF}(\mathsf{t})$ is similar. □

10.3 Related Works

In our opinion, this proposal for a probabilistic CHC and our counting-type approach to capture probabilistic termination are new. On the logical side, intuitionistic **iCPL** is clearly inspired by our counting propositional logic **CPL**. As said, extensive studies on logical systems enabling – in various ways and for different purposes – some forms of probabilistic reasoning [158, 17, 81, 108, 137, 140, 87] have appeared in the literature, but not many of them tied logic to computational aspects.[4] Yet, to the best of our knowledge, these operators were not defined within an intuitionistic logical system.

On the other hand, intuitionistic modal logic has been related in the Curry and Howard sense to monadic extensions of the λ-calculus [1, 23, 49, 65, 234]. In these cases, modal operators are linked with *qualitative* properties of programs – typically, tracing algebraic effects – as opposed to the *quantitative* properties expressed by counting quantifiers. Observe that the Kripke-style structures we presented in Chapter 7 can be related to standard **IML** ones.[5] Moreover, quantitative semantics arising from linear logic have been largely used to study probabilistic λ-calculi, [54, 73, 77]. Notably, *probabilistic coherence spaces* provide a fully abstract model of probabilistic PCF, [73, 75, 93]. While we are not aware of correspondence relating probabilistic programs with proofs in linear logic, it seems that the proof-theory of counting quantifiers could somehow be related to that of *bounded exponentials* [58, 94] and, more generally, to the theory of *graded* monads and co-monads [32, 88, 122, 123].

[4] A partial exception is offered by Wagner's counting operator [226] and Kontinen's quantifiers [134], which have indeed inspired our work.

[5] Indeed, these are based on the set W with *two* pre-order relations – $\leq$ and R – enjoying a suitable "diamond" property $R;\leq \subseteq \leq;R$. We obtain a similar structure by considering worlds to be pairs w, ω made of a world and an outcome from the Cantor space, with $(w,\omega) \preceq (w',\omega)$, whenever $w \leq w'$, and $(w,\omega)R(w,\omega+\omega')$. The clause for $\mathbf{C}^q F$ can then be seen as a quantitative variant of the corresponding clause for $\diamond F$. Actually, this is not very surprising, given the similarity between the introduction and elimination rules for $\mathbf{C}^q$ and those for $\diamond$. For further details, see, for example, [1, 23].

On the computational side, *probabilistic* λ-calculi were developed starting from the pioneering work by Saheb-Djaromi [180], where the syntax and (operational) semantics of a calculus with binary probabilistic choice was introduced for the first time. Denotational models for these calculi were presented, for instance, in [119, 120, 105], while operational semantics can be found in [63]. Furthermore, quantitative semantics arising from linear logic have been largely used in the study of λ-calculi with choice operators, as for example in [54, 73, 77], and *probabilistic coherence spaces* [75, 202] have been shown to provide a fully abstract model of probabilistic PCF. Remarkably, probabilistic λ-calculi are often (implicitly) associated with either CbN or CbV strategies.[6] As said, our calculus $\Lambda_{\mathbf{PE}}$ is strongly inspired by Λ_{PE} [57] and is reminiscent of calculi with generic effects [172, 194]. The main difference between our approach and the one developed in [57] concerns type systems, as Λ_{PE} is associated with simple types.[7]

Generally speaking, in the last decades, several type systems for probabilistic λ-calculi were introduced, for instance in the context of size [56], intersection [31], and refinement type disciplines [15]. Type systems for probabilistic λ-calculi focused on capturing genuinely probabilistic properties of normalization, have been recently introduced as well. Among these we can certainly mention systems based on *type distributions* [56] – where a single derivation assigns several types to a term, each with some probability – and systems based on *oracle intersection types* [31] – where type derivations capture single evaluations as determined by an oracle. One can see our type systems as sitting in between these two approaches. Like the former (and unlike the latter), typing derivations can capture a finite number of different evaluations, although without using distributions of types. Similarly to the latter ones, our typings reflect the dependency of evaluation on oracles, although the latter are manipulated in an aggregate way by means of Boolean constraints. Another noteworthy work in this area is [229], where dependent type theory is enriched with a probabilistic choice operator, yielding a calculus with both term and type distributions. Interestingly, a fragment of this system enjoys a sort of CHC with so-called *Markov Logic Networks* [178], a class of probabilistic graphical models specified by means of first-order logic formulae.

[6] Some remarkable exceptions are [54, 73, 77]. Observe that differently from $\Lambda^{!}_{\oplus}$ [77] (in which the operator ! is both a marker for duplicity and a checkpoint for any "fired" probabilistic choice), in Λ_{PE} and $\Lambda_{\mathbf{PE}}$ duplication is not controlled and checkpoints are used.

[7] In [57], typings ensure strong normalization, so these systems do not provide information about probability of termination for non-normalizable terms.

Part III

Randomized Bounded Arithmetic

Chapter 11

Characterizing Probabilistic Complexity

In this part of the thesis, we introduce a minimal extension of first-order **PA**, via second-order measure quantifiers. These quantifiers are strongly inspired by counting ones and associated with a quantitative interpretation. We show that this language is capable of formalizing simple results from probability theory which cannot be expressed in standard arithmetic and of representing every recursive random function. We also introduce a new randomized *bounded* theory and prove that the class of formulae which are Σ_1^b-representable in it precisely corresponds to that of *poly-time* random functions. This result, together with the notion of measure quantifier, is at the basis of our *arithmetical* characterization of relevant probabilistic complexity classes, like $\mathbf{BPP}$, so generalizing classic results by Buss [34] and Ferreira [82] to the randomized realm.

11.1 On Arithmetic and (Randomized) Computation

As anticipated, interactions between first-order arithmetic and the theory of computation are plentiful and deep. The language of arithmetic is able to express interesting properties of algorithms and several problems in computation theory can be investigated in the framework of arithmetic. For example, proof systems for arithmetic can be used to prove *termination* of certain classes of algorithms [198] or to establish complexity bounds [45, 34], while higher-order programming languages capture the computational content of arithmetical proofs. These insightful results have then been pushed further, giving rise to logical and type theories of various strength, at the basis of which lies the tight connection between the concept of *totality* (of functions) and *termination* (of algorithms).

Yet, in the probabilistic setting, behavioral properties – like termination –

have a *quantitative* nature – any computation terminates *with a given probability*. In this Part, we want to develop a logic to study such quantitative properties *within a logical system*:

$$\frac{\textbf{PA}}{\text{properties of computation}} = \frac{x}{\text{properties of randomized computation}}$$

Of course, logics dealing with set-theory and second-order logic can be expressive enough to represent measure theory and to talk about randomized computation [194]. Yet, we want to define a *minimal* extension of first-order arithmetic capable of describing probabilistic computation.

To do so, we take inspiration from the notion of counting quantifiers as defined in Chapter 3. We introduce a system extending first-order **PA** with measure quantifiers and associate it with a quantitative, measure-theoretic semantics. As $\mathbf{CPL}_0$ and **CPL**, this language offers a very natural model for stochastic events, but its expressive power is far more extended. Indeed, it allows us to formally express (and analyze) basic results from probability theory – for instance, the infinite monkey theorem or the random walk theorem – which are not expressible in **PA**. This new language is at the basis of our definition of a randomized bounded theory *á la Buss*, and of our *arithmetical* characterization of probabilistic complexity classes. Indeed, we show that the class of formulae which are Σ_1^b-representable in our bounded theory precisely captures the notion of poly-time random functions. Then, by internalizing the error-bound check within our logical system, we provide an arithmetical characterization of *semantic* classes, like $\mathbf{BPP}$.

11.2 A Brief Overview of Bounded Arithmetic

Arithmetic theories are related to computable functions in several ways. In the 1970s-80s, also fragments of **PA** started to receive attention. Indeed, these sub-theories not only support a satisfactory axiomatization – somehow escaping proof-theoretical drawbacks coming from incompleteness [100] – but are also deeply connected with interesting complexity classes [166, 45]. In particular, in 1986 Buss introduced a bounded theory of arithmetic able to characterize the class of *poly-time* computable functions [34]. Inspired by this work, also Ferreira defined a theory equivalent to the corresponding one by Buss, but expressed in a *word language*, and corresponding to the class of poly-time functions *over strings* [83].

11.2.1 Sub-Theories of Arithmetic and Complexity

Very Weak, Weak and Strong Fragments. Sub-theories of **PA** have been obtaining increasing interest for their intimate connection with complexity classes. Buss divided them into three main categories: strong, weak and very weak fragments [35]. *Very weak* theories do not admit any induction axioms. Among them there is well-known Robinson arithmetic **Q**, introduced in the 1950s by

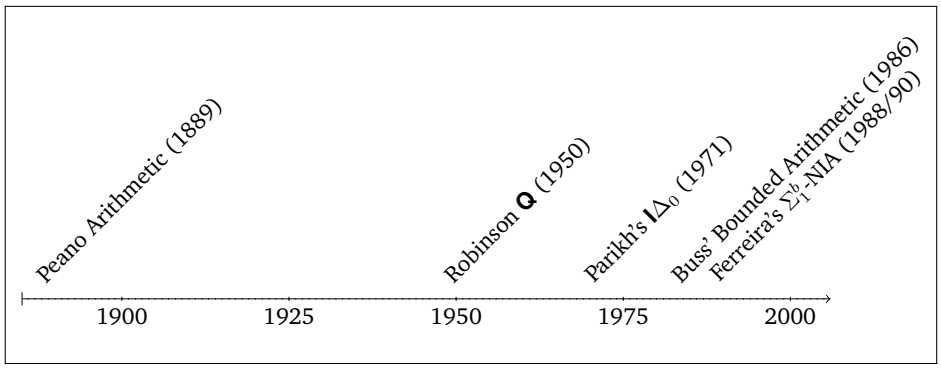


Figure 11.1: From Peano to Bounded Arithmetic

Robinson, Tarski, and Mostowski [179, 204]. *Weak theories* are defined by a language which extends that of **PA** by additional symbols of specific growth rate (sometimes together with explicit bounded quantifiers) and by limited induction schemas. In *strong theories* the language is enlarged so to include symbols for all primitive recursive functions.[1]

Bounded Theories. In particular, bounded theories of arithmetic are weak fragments of **PA**, typically including bounded quantifiers and in which induction is limited. Buss defined a bounded arithmetic as a theory axiomatized by Π_1-formulae [35]. The potential strength of such theories and their ability to characterize complexity, depends on the (sub-exponential) growth rate of the function symbols in the language. The study of **BA** was initiated by Parikh in 1971 [166], who introduced **I**Δ_0 to give an appropriate proof theory to linear bounded automata, namely to predicates computable by linear space-bounded TMs. Then, other bounded theories were introduced by Buss [34] and extensively studied.

11.2.2 Buss' Bounded Arithmetic

Buss' Ph.D. thesis [34] provided a groundbreaking result in the study of the arithmetical characterization of complexity classes. He started by considering the *definability* of a function in a theory: an arithmetic theory T defines a function f when there is a formula F in the language of T such that f satisfies $F(x, f(x))$ for any x and $T \vdash (\forall x)(\exists! y)F(x, y)$. The constructive proof of $(\forall x)(\exists y)F(x, y)$ also provides an algorithm to compute f. So, the given procedure is effectively computable but not necessarily *feasible*, that is *computable in polynomial time*. Due to his bounded theories, Buss was able to arithmetically

[1] Examples of strong theories are **I**Σ_n, that is $\mathbf{Q}_{\leq}$ (the conservative extension of **Q** with $x \leq y \leftrightarrow (\exists z)(x + z = y)$) plus Σ_n-IND, and **I**Δ obtained adding Δ_0-IND to $\mathbf{Q}_{\leq}$.

characterize functions computable with given resource bound, thus to characterize interesting complexity classes.

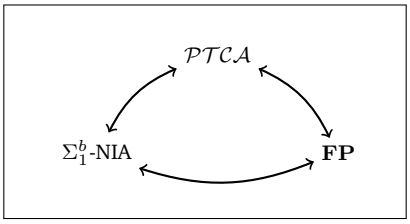


Figure 11.2: Ferreira's Proof Schema [82]

Language. The language of Buss' fragments extends that of **PA** with three special predicate symbols: $\llcorner \frac{1}{2} \cdot \lrcorner$ which divides for two and rounds down the argument, $|\cdot|$ returning the length of the binary representation the argument, and Nelson's function $\#$, such that $x\#y = 2^{|x|\cdot|y|}$. In the language of **BA**, standard quantifiers are called *unbounded*, while quantifiers of the form $(\forall x \leq t)$ or $(\exists x \leq t)$ are called *bounded* and are such that $(\forall x \leq t)F$ is an shorthand of $(\forall x)(x \leq t \rightarrow F)$, and $(\exists x)F$ abbreviates $(\exists x)(x \leq t \wedge F)$.[2] A special kind of bounded quantifiers are *sharply bounded* ones, namely $(\forall x \leq |t|)$ and $(\exists x \leq |t|)$. Bounded formulae (converted into PNF) are classified in a hierarchy of classes, Σ_k^b and Π_k^b, by counting alternations of bounded quantifiers (ignoring sharply ones).

Axiomatization. Bounded theories are then defined by adding 32 basic axioms [34, pp. 30-31] to the ones for **PA** and restricting the induction schema. Buss himself noticed that there is a certain amount of flexibility in the choice of basic axioms and the ones he introduced in his thesis are not optimal [35, p. 101]. Alternative sets were proposed for example by Cook and Urquhart [46] and Buss and Ignjatović [36].[3] In particular, Buss introduced the class $\mathbf{S}_2^i$ as axiomatized by basic axioms plus Σ_i^b-PIND:

$$F(x) \wedge (\forall x)(F(\llcorner \frac{1}{2} x \lrcorner) \rightarrow F(x)) \rightarrow (\forall x)F(x),$$

where F is a Σ_i^b-formula, while $\mathbf{T}_2^i$ are defined by the same set of basic axioms together with the induction schema Σ_i^b-IND,

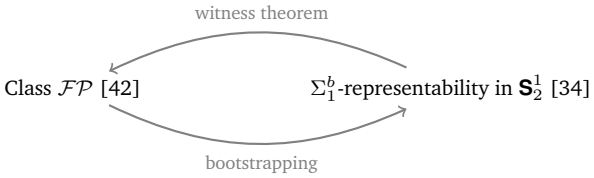
$$F(0) \wedge (\forall x)(F(x) \rightarrow F(\mathsf{S}(x)) \rightarrow (\forall x)F(x),$$

where F is a Σ_i^b-formula.

[2] Otherwise, the syntax can be expanded so to directly include them. In this case the calculus must be extended accordingly.

[3] For further details, see [35]

Relating $\mathbf{S}^1_2$ and poly-time computable functions. The main result of Buss' Thesis is the proof that these arithmetic theories actually provide a logical characterization of complexity classes in $\mathbf{PH}$. In particular, he proved that poly-time computable functions are Σ^b_1-definable in $\mathbf{S}^1_2$ [34, Cor. 8, p. 99]. On the one hand, a function is poly-time computable when there is a TM $\mathscr{M}$ and a polynomial $p(n)$ such that $\mathscr{M}$ computes the given function and runs in time smaller or equal than $p(n)$, for any input of length n. In 1964, this notion was made precise by Cobham in the form of a function algebra [42], on which Buss' proof relies.[4] On the other, a function is Σ^b_1-definable in $\mathbf{S}^1_2$ when there is a Σ^b_1-formula F such that conditions above hold. The link between complexity of computing and quantified formulae is then established in two main steps:



That every poly-time function is Σ^b_1-definable in $\mathbf{S}^1_2$ is proved via so-called *bootstrapping*, that is a series of coding functions was introduced. The proof of the converse direction is more difficult. Buss presented a sequent calculus, extending **LK** with rules for limited induction and bounded quantifiers. Then, due to cut elimination, he proved the "witness" theorem, showing that proofs in this calculus contain explicit algorithms to compute the output of the function from the input *in polynomial time*.

11.2.3 Ferreira's Bounded Arithmetic

In [82, 83], Ferreira introduced a "(supposedly) more natural" [83, p. 2] bounded theory defined in a word language, instead of the standard language of arithmetic $\mathcal{L}_{\mathbb{N}}$. This arithmetic characterizes poly-time computable functions – this time defined over strings – and, indeed, can interpret Buss' $\mathbf{S}^1_2$ [84]. For clarity's sake, we sum up its salient aspects following notation and axiomatization by [84], which is slightly different from [82, 83] in the surface but essentially equivalent.

The Function Algebra $\mathcal{PTCA}$. Ferreira defined an algebra of functions *over strings* $\mathcal{PTCA}$ (poly-time computable arithmetic) analogous to Cobham's one but made of:

- initial functions:
  - $E_{\mathcal{F}}(x) = \emptyset$
  - $P^{n,i}_{\mathcal{F}}(x_1, \ldots, x_n) = x_i$, with $1 \leq i \leq n$
  - $C^b_{\mathcal{F}}(x) = x\mathbb{b}$, where if $b = 1$, then $\mathbb{b} = \mathbb{1}$ and if $b = 0$, then $\mathbb{b} = \mathbb{0}$

[4] Buss also noticed that a proof "directly" based on the machine definition is also possible.

–

$$Q_{\mathcal{F}}(x,y) = \mathbb{1} \leftrightarrow x \subseteq y$$
$$Q_{\mathcal{F}}(x,y) = \mathbb{0} \vee Q_{\mathcal{F}}(x,y) = \mathbb{1}$$

where $\emptyset$ denotes the empty string, for any two strings x and y, xy is, as usual, (a shorthand for) their concatenation, and $\subseteq$ indicates the subword relation between strings, i.e. given two strings x and y, $x \subseteq y$ expresses that x is an initial or prefix substring of y.

- functions obtained by:
  - composition, i.e. f is obtained from $g, h_1, \ldots, h_k$ as $f = (x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_k(x_1, \ldots, x_n))$
  - bounded iteration, i.e. f is obtained from g, h_0, h_1 as,

$$f(x_1, \ldots, x_n, \emptyset) = g(x_1, \ldots, x_n)$$
$$f(x_1, \ldots, x_n, y\mathtt{b}) = h_b(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))|_{t(x_1,\ldots,x_n,y)}$$

where t is a term called *bound* and $\cdot|_{\cdot}$ denote truncation. Then, the function f is said to be defined from g, h_0, h_1 by bounded iteration, with bound t.

The Word Language $\mathcal{L}_{\mathbb{W}}$. As anticipated, Ferreira's theory is defined in a word language $\mathcal{L}_{\mathbb{W}}$, which is basically a first-order language with equality endowed with three constants $\epsilon, \mathtt{0}, \mathtt{1}$, two function symbols $\frown, \times$ and a relation symbol $\subseteq$.[5] Interpretation is as predictable: ϵ denotes the empty word, $\mathtt{0}$ and $\mathtt{1}$ the bits $\mathbb{0}$ and $\mathbb{1}$ (resp.), $\frown$ word concatenation, $\times$ the binary product (i.e. $x \times y = x \frown \ldots \frown x$, $|y|$-times), and $\subseteq$ the *initial* subword relation. *Bounded quantifiers* in $\mathcal{L}_{\mathbb{W}}$ are of the form $\forall x \preceq t$ and $\exists x \preceq t$ (with t term), where $x \preceq t$ intuitively means that the length of x is smaller or equal than that of t. Bounded-quantified formulae $(\forall x \preceq t)F$ and $(\exists x \preceq t)F$ abbreviate respectively $(\forall x)(\mathtt{1} \times x \subseteq \mathtt{1} \times t \rightarrow F)$ and $(\exists x)(\mathtt{1} \times x \subseteq \mathtt{1} \times t \wedge F)$. *Subword quantifiers* are of the form $\forall x \subseteq^* t$ and $\exists x \subseteq^* t$, so defined that $(\forall x \subseteq^* t)F$ is a shorthand for $(\forall x)(\exists w \subseteq t(wx \subseteq t) \rightarrow F)$ and $(\exists x \subseteq^* t)F$ for $(\exists x)(\exists w \subseteq t(wx \subseteq t) \wedge F)$.

The Theory Σ_1^b-NIA. Then, Σ_1^b-NIA is a first-order theory in $\mathcal{L}_{\mathbb{W}}$ defined by the following axioms:[6]

- Basic axioms:

$$x\epsilon = x \qquad x(y\mathtt{b}) = (xy)\mathtt{b} \qquad x \times \epsilon = \epsilon \qquad x \times y\mathtt{b} = (x \times y)x$$
$$x \subseteq \epsilon \leftrightarrow x = \epsilon \qquad x \subseteq y\mathtt{b} \leftrightarrow x \subseteq y \vee x = y\mathtt{b}$$
$$x\mathtt{b} = y\mathtt{b} \rightarrow x = y \qquad x\mathtt{0} \neq y\mathtt{1} \qquad x\mathtt{b} \neq \epsilon.$$

[5] Observe that Ferreira used different symbols for the empty string and for concatenation.
[6] Again, the name Σ_1^b-NIA is not original by [82], but was introduced in [84].

- Axiom schema for induction on notation:

$$F(\epsilon) \wedge (\forall x)(F(x) \rightarrow F(x\mathbb{0}) \wedge F(x\mathbb{1})) \rightarrow (\forall x)F(x),$$

  where F is a Σ_1^b-formula in $\mathcal{L}_{\mathbb{W}}$.

11.3 Towards Randomized Bounded Arithmetic

Describing complexity classes within logical and arithmetical languages may offer a better understanding of the nature of such classes, and approaches inspired by descriptive complexity [78, 45, 116, 138], made it possible to consider complexity from viewpoint less dependent on concrete machine models and on explicit resource bounds. Yet, randomized classes are not among the ones which has received such a characterization. Our goal is to generalize classic results by Buss and Ferreira to the probabilistic realm. To do so, we introduce a new *randomized* bounded theory and show that formulae which are Σ_1^b-definable in it precisely capture poly-time random functions. Then, we provide an arithmetical characterization of the probabilistic class $\mathbf{BPP}$ by internalizing the error-bound check within logic. Observe that this "encoding" essentially relies on the use of measure-sensitive quantifiers.

11.3.1 Semantic, All Too Semantic

Before delving into technical details, we spend a few words on the dichotomy between syntactic and semantic classes, and on the intrinsic difficulty of characterizing the latter ones. Although this distinction appears in many popular textbooks – for instance in [14, 162] – the literature does not offer a precise definition. Generally speaking, syntactic classes are defined imposing limitations on the *amount of resources* the underlying algorithm is allowed to use. Semantic classes require an additional condition, typically that the underlying algorithm returns the correct answer *often enough*. Otherwise said, in semantic classes being resource bounded is not sufficient for an algorithm to solve some problems in the class, since there can well be algorithms getting it wrong too often. This distinction between semantic classes – as $\mathbf{BPP}$ and $\mathbf{ZPP}$ – and syntactic ones – as $\mathbf{P}$, $\mathbf{NP}$, and $\mathbf{PSPACE}$ – refers to *how a class is defined* and not to the underlying set of problems. It is thus of *intensional* nature.

It is difficult to verify resource bounds on *arbitrary* algorithms, but it is surprisingly easy to define an enumeration of resource-bounded algorithms containing at least *one* algorithm for any problem in the class. Suppose we want to characterize a syntactic class like $\mathbf{P}$. On the one hand, the class of *all* algorithms working in polynomial time is recursion-theoretically very hard, actually Σ_0^2-complete. On the other, the class of those algorithms consisting of a `for` loop executed a polynomial number of times, the body of which itself consists of conditionals and simple enough instructions manipulating string variables, is both easy to enumerate and big enough to characterize $\mathbf{P}$, at least in an extensional sense: every problem in $\mathbf{P}$ is decided by at least one algorithm in

the class and vice versa. Many characterizations of $\mathbf{P}$ (and of other syntactic classes) – e.g. based on safe-recursion [22], on light and soft linear logic [94], or through bounded theories [34] – are basically instances of the above pattern, where the precise class of poly-time algorithms varies, leaving the underlying classes unchanged. In semantic classes – that is in presence of conditions about the error rate – the enumeration strategy just sketched is not applicable, as we need to isolate a simple enough subclass of algorithms which are not only resource-bounded, but also not too erratic.

11.3.2 An Arithmetical Theory to Characterize Probabilistic Complexity

As seen, one of the original motivations for the development of **BA** was their connection with computational complexity [34, 35], and informally a first-order theory of arithmetic T defines a numerical function f when there is a formula F such that: (i) for every x, $F(x, f(x))$ and (ii.) $T \vdash (\forall x)(\exists y!)F(x, y)$. This implies the existence of a proof in T providing an algorithm to compute f, but, of course, not all computable functions are computable in an effective way. Concretely, we are often interested in functions computable *with some given amount of resources* and, specifically, in restricting analysis to *feasibly computable* ones. We have also seen that, to do so, Buss defined a family of formal theories, called *bounded arithmetics*, which are fragments of **PA** including function symbols with specific growth-rate. Due to *bounded* quantifiers and theories, he managed to characterize complexity classes in terms of arithmetical formulae. In particular, he proved that every poly-time computable function corresponds to a function which is Σ^b_1-definable in the corresponding bounded theory $\mathbf{S}^1_2$.

This result was very insightful. However, no similar achievement exists when switching to the probabilistic framework. Our goal is to generalize this approach to obtain such an arithmetical characterization for *probabilistic* classes:

$$\frac{\textbf{BA}}{\text{“traditional” complexity classes}} = \frac{x}{\text{probabilistic complexity classes}}$$

Our core idea is to generalize standard conditions of definability for functions in a theory to a quantitative setting and to define a new *randomized* bounded theory.

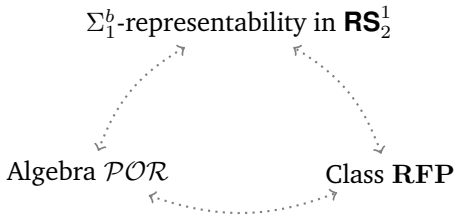
Concretely, the first step to characterize probabilistic classes consists of obtaining a class of arithmetical formulae corresponding to functions computable by poly-time PTM. To this aim, we introduce three classes of functions:

1. The class of *poly-time oracle recursive functions* $\mathcal{POR}$ is a class of functions from (finite and infinite) strings to strings defined extending Ferreira's class of poly-time functions [82, 83] – itself being a word version of the corresponding class by Cobham [42] – with a *query* function, accessing an oracle from the Cantor space
2. The class of functions which are Σ^b_1-representable in a *randomized bounded theory*, called $\mathbf{RS}^1_2$, expressed in a probabilistic first-order word language

with equality [84], augmented by a unary predicate $\texttt{Flip}(\cdot)$ similar to the one defined in Chapter 12.

3. The class $\mathbf{RFP}$ of functions which are computable by a poly-time PTM. Actually we pass through a second computational model, considering stream Turing machines (STM, for short). An STM is basically TM with $k+1$-tapes, one of which is treated as a read-only *oracle* tape. Such machines access randomness in a way which is closer to that of functions in $\mathcal{POR}$.

Then, we show that functions which are Σ^b_1-representable in $\mathbf{RS}^1_2$ are precisely those computable by poly-time PTMs:



Due to this result, together with the notion of measure quantifiers, we provide an arithmetical characterization of probabilistic (semantic) classes. Indeed, we rely on the power of our measure-quantifiers to reason about error bounds *from within* the logic.

11.4 Outline of Part III

As seen, this part of the thesis focusses on the relation between the theory of arithmetic and randomized computation. It is bipartite:

- In Chapter 12 we introduce of the language **MQPA** – i.e. the language extending that of **PA** with quantifiers inspired by those introduced in Chapter 3. This language has a noteworthy expressive power: in it we can formalize interesting results from probability theory. We also show that every random function computed by a PTM can be expressed as a formula of **MQPA**, thus providing a probabilistic version of Gödel's arithmetization.
- Most of Chapter 13 is devoted to the proof that the class of functions which are Σ^b_1-representable in $\mathbf{RS}^1_2$ is precisely the class of random functions computed by poly-time PTM. Given the distance between the way in which randomness is accessed by poly-time random functions and expressed by formulae of $\mathbf{RS}^1_2$, we need to pass through a series of auxiliary notions, the central one being the class of poly-time oracle recursive functions $\mathcal{POR}$. This result is fundamental to provide an *arithmetical* characterization of probabilistic complexity classes. Indeed, it is relying

on the correspondence $\mathbf{RFP}$ and $\mathbf{RS}_2^1$ together with the notion of measure quantifiers of **MQPA** that, in Section 13.5.2, we provide a (semantic) arithmetical characterization of $\mathbf{BPP}$.

Chapter 12

On Measure Quantifiers in First-Order Arithmetic

We introduce a language, called **MQPA**, which is a "minimal" extension of first-order Peano Arithmetic via second-order measure quantifiers. This language is shown to be capable of formalizing a few basic and well-known results from probability theory and of representing every recursive random function. In Section 12.1, we introduce the grammar and semantics of **MQPA**. Then, in Section 12.2, we show that some non-trivial results from measure theory – for instance, the infinite monkey theorem – can be expressed in this non-standard language. Finally, in Section 12.3, we generalize Gödel's arithmetization to the probabilistic setting, proving that every random function which is computable by a PTM can be expressed in **MQPA**.

12.1 Measure-Quantified Peano Arithmetic

The language **MQPA** is obtained by enriching that of **PA** with two elements: (i.) the special unary predicate $\texttt{Flip}(\cdot)$, the interpretation of which is an element of the Cantor space, and (ii.) measure-quantified formulae, the interpretation of which is similar to counting-quantified ones. Observe that the appeal to the Cantor space is essential here, since there is no *a priori* bound on the amount of random bits a given computation might need. At the same time, as for **CPL**, also this more expressive language yields a natural measure-theoretic semantics.

Preliminaries. The standard model $\mathcal{N} = (\mathbb{N}, +, \times)$ has nothing probabilistic in itself. Nevertheless, it can be naturally extended due to the probability space. Again, arithmetic being discrete, we can deal with $2^{\mathbb{N}}$ – namely the set of all infinite sequences of elements from $2 = \{0, 1\} (= \mathbb{B})$ – as the underlying sample space.

Notation 12.1.1. We use metavariables, $\omega_1, \omega_2, \ldots$ for the elements of $2^{\mathbb{N}}$.

As seen in Section 3.1, there are standard ways of building well-behaved σ-algebra and probability space on $2^{\mathbb{N}}$.

Notation 12.1.2. In particular, we call an *n-cylinder* a subset of $2^{\mathbb{N}}$ of the form

$$\mathsf{C}_X = \{s \cdot \omega \mid s \in X \ \& \ \omega \in 2^{\mathbb{N}}\},$$

where $X \subseteq 2^n$ and $\cdot$ denotes sequence concatenation.

Specifically, we are interested in Xs defined as

$$X_n^b = \{s \cdot b \mid s \in 2^n \ \& \ b \in \{0,1\}\} \subseteq 2^{n+1},$$

with $n \in \mathbb{N}$ and often deal with cylinders of the form $\mathsf{C}_{X_n^1}$. The canonical probability measure $\mu_{\mathscr{C}}$ on $\mathscr{C}$ assigns to any C_X measure $\frac{|X|}{2^n}$. Then, the standard model $\mathcal{N}$ can be generalized to:

$$\mathscr{P} = (\mathbb{N}, +, \times, \sigma(\mathscr{C}), \mu_{\mathscr{C}}),$$

which is the standard model for **MQPA**. Clearly, when interpreting sequences in $2^{\mathbb{N}}$ as infinite supplies of random bits, the set such that the k-th coin flip results in 1 (for any fixed k) is assigned measure $\frac{1}{2}$, meaning that each random bit is uniformly distributed and independent from the others.

Syntax. Terms of **MQPA** are defined as in standard first-order arithmetic, while formulae are obtained by endowing the language of **PA** with *flipcoin formulae* of the form $\mathtt{Flip}(t)$, and *measure-quantified formulae* of the form $\mathbf{C}^{t/s}F$ and $\mathbf{D}^{t/s}F$, with t and s terms. Specifically, $\mathtt{Flip}(\cdot)$ is a special unary predicate with an intuitive computational meaning: it basically provides an infinite supply of independent and uniformly distributed bits. Intuitively, given a closed term t, $\mathtt{Flip}(t)$ holds when the n-th tossing returns 1, where n is the denotation of $t+1$.

Definition 12.1.1 (Formulae of **MQPA**)**.** Let $\mathcal{G}$ be a denumerable set of ground variables, the elements of which are indicated by metavariables $x, y, \ldots$. *Terms of **MQPA***, denoted by $t, s, \ldots$, are defined as:

$$t ::= x \ \mid \ \mathtt{0} \ \mid \ \mathtt{S}(t) \ \mid \ t + t \ \mid \ t \times t.$$

*formulae of **MQPA*** are defined by the following grammar:

$$F ::= \mathtt{Flip}(t) \ \mid \ (t = s) \ \mid \ \neg F \ \mid \ F \triangle F \ \mid \ (\exists x)F \ \mid \ (\forall x)F \ \mid \ \mathbf{C}^{t/s}F \ \mid \ \mathbf{D}^{t/s}F,$$

where t, s are terms and $\triangle \in \{\vee, \wedge\}$.

Semantics. Given an environment $\xi : \mathcal{G} \to \mathbb{N}$, the interpretation for terms is standard.

Definition 12.1.2 (Semantics for Terms of **MQPA**)**.** An *environment* $\xi : \mathcal{G} \to \mathbb{N}$ is a mapping that assigns to each ground variable a natural number. Given a term t and an environment ξ, the *interpretation of t in* ξ is the natural number $[\![t]\!]_\xi \in \mathbb{N}$ inductively defined as follows:

$$\begin{aligned} [\![x]\!]_\xi &:= \xi(x) \in \mathbb{N} & [\![t+s]\!]_\xi &:= [\![t]\!]_\xi + [\![s]\!]_\xi \\ [\![\mathtt{0}]\!]_\xi &:= 0 & [\![t \times s]\!]_\xi &:= [\![t]\!]_\xi \times [\![s]\!]_\xi. \\ [\![\mathtt{S}(t)]\!]_\xi &:= [\![t]\!]_\xi + 1 \end{aligned}$$

On the contrary, the interpretation for formulae is, as for **CPL**, inherently quantitative. Indeed, any formula is associated with a *measurable* set.

Definition 12.1.3 (Semantics for formulae of **MQPA**)**.** Given a formula F and an environment ξ, the *interpretation of* F *in* ξ is the measurable set of sequences $[\![F]\!]_\xi \in \sigma(\mathscr{C})$ defined below:

$$\begin{aligned} [\![\mathtt{Flip}(t)]\!]_\xi &:= \mathsf{C}_{X^1_{[\![t]\!]_\xi}} & [\![G \vee H]\!]_\xi &:= [\![G]\!]_\xi \cup [\![H]\!]_\xi \\ [\![t = s]\!]_\xi &:= \begin{cases} 2^{\mathbb{N}} & \text{if } [\![t]\!]_\xi = [\![s]\!]_\xi \\ \emptyset & \text{otherwise} \end{cases} & [\![G \wedge H]\!]_\xi &:= [\![G]\!]_\xi \cap [\![H]\!]_\xi \\ & & [\![(\exists x)G]\!]_\xi &:= \bigcup_{i \in \mathbb{N}} [\![G]\!]_{\xi\{x \leftarrow i\}} \\ [\![\neg G]\!] &:= 2^{\mathbb{N}} - [\![G]\!]_\xi & [\![(\forall x)G]\!]_\xi &:= \bigcap_{i \in \mathbb{N}} [\![G]\!]_{\xi\{x \leftarrow i\}} \end{aligned}$$

$$[\![\mathbf{C}^{t/s} G]\!]_\xi := \begin{cases} 2^{\mathbb{N}} & \text{if } [\![s]\!]_\xi > 0 \text{ and } \mu_{\mathscr{C}}([\![G]\!]_\xi) \geq \frac{[\![t]\!]_\xi}{[\![s]\!]_\xi} \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![\mathbf{D}^{t/s} G]\!]_\xi := \begin{cases} 2^{\mathbb{N}} & \text{if } [\![s]\!]_\xi = 0 \text{ or } \mu_{\mathscr{C}}([\![G]\!]_\xi) < \frac{[\![t]\!]_\xi}{[\![s]\!]_\xi} \\ \emptyset & \text{otherwise.} \end{cases}$$

The semantics is well-defined. Indeed, atomic formulae $[\![\mathtt{Flip}(t)]\!]_\xi$ and $[\![t = s]\!]_\xi$ are measurable, and measurability is preserved by all the logical operators. It is not difficult to see that any n-cylinder is captured by some formula of **MQPA**. In fact, this measure-quantified language allows us to express more and more complex measurable sets, as illustrated in Section 12.2.

The notions of validity and logical equivalence are again standard.

Definition 12.1.4. A formula of **MQPA** F, is *valid* when for every ξ, $[\![F]\!]_\xi = 2^{\mathbb{N}}$ and *invalid* when $[\![F]\!]_\xi = \emptyset$. Two formulae of **MQPA**, say F ad G, are *logically equivalent* $F \equiv G$, when for every ξ, $[\![F]\!]_\xi = [\![G]\!]_\xi$.

Notably, the two measure quantifiers are inter-derivable, $[\![\mathbf{C}^{t/s} F]\!] = [\![\neg \mathbf{D}^{t/s} F]\!]_\xi$.

Lemma 12.1.1. *For every formula of* ***MQPA*** F*,*

$$\mathbf{C}^{t/s} F \equiv \neg \mathbf{D}^{t/s} F.$$

Proof. The proof is based on Definition 12.1.3,

$$\begin{aligned}
[\![\neg \mathbf{D}^{t/s} F]\!]_\xi &= 2^{\mathbb{N}} - [\![\mathbf{D}^{t/s} F]\!]_\xi \\
&= 2^{\mathbb{N}} - \begin{cases} 2^{\mathbb{N}} & \text{if } \mu_{\mathscr{C}}([\![F]\!]_\xi) < [\![t]\!]_\xi / [\![s]\!]_\xi \\ \emptyset & \text{otherwise} \end{cases} \\
&= \begin{cases} \emptyset & \text{if } \mu_{\mathscr{C}}([\![F]\!]_\xi) < [\![t]\!]_\xi / [\![s]\!]_\xi \\ 2^{\mathbb{N}} & \text{otherwise} \end{cases} \\
&= [\![\mathbf{C}^{t/s} F]\!]_\xi.
\end{aligned}$$

□

The following examples further illustrate the "meaning" of measure-quantifiers $\mathbf{C}^{t/s}$ and $\mathbf{D}^{t/s}$ and, in particular, the role of probabilities of the form $\frac{t}{s}$.

Example 12.1.1. The formula $F = \mathbf{C}^{1/1}(\exists x)\texttt{Flip}(x)$ states that a true random bit (values) will almost surely be met. It is valid, as the set of constantly 0 sequences forms a singleton, which has measure 0.

Example 12.1.2. The formula $F = (\forall x)\mathbf{C}^{1/2^x}(\forall_{y \leq x})\texttt{Flip}(y)$ states that the probability for the first x random bits to be true is at least $\frac{1}{2^x}$. Actually, for readability's sake, F is written with a little abuse of notation, the effective formula being $(\forall x)\mathbf{C}^{1/z}(\text{EXP}(z,x) \wedge (\forall y)\big((\exists w)(y+w=x) \rightarrow \texttt{Flip}(y)\big)$, where $\text{EXP}(z,x)$ is an arithmetical formula expressing $z = 2^x$ and $(\exists w)y+w=x$ expresses $y \leq x$. The formula F is valid.

Brief Digression on Terminology. To the best of our knowledge, the term "measure quantifier" was first introduced by Morgenstern in 1979 to formalize the idea that a formula $F(x)$ is true *for almost all x* [151]. This definition was inspired by Mostowski's notion of generalized quantifier [152], which was "introduced to specify that a given formula was true for *many x's*" [151, p. 103].[1] In particular, Morgenstern defined a language $\mathcal{L}_\mu$, obtained by adding the measure quantifier $\mathbf{Q}_\mu$ to the standard first-order grammar:

> DEFINITION 2.1 A *measure structure* $\mathcal{U}$ is a pair $\mathcal{U} = (\mathcal{U}, \mu^{\mathcal{U}})$, where $\mathcal{U}'$ is a first-order structure, $card|\mathcal{U}| = k$, a measurable cardinal, and $\mu^{\mathcal{U}}$ is a non-trivial k-additive measure on $|\mathcal{U}'|$ which satisfies the partition property. [...]
> DEFINITION 2.2 Define a language L_μ to be a first-order language together with a quantifier $\mathbf{Q}_\mu$, binding one free variable, where a measure $\mathcal{U} \vDash \mathbf{Q}_\mu v_0 \varphi(v_0)$ iff $\{x \in |\mathcal{U}| \mid \mathcal{U}' \vDash \varphi[x]\} \in \mu^{\mathcal{U}}$. [151, pp. 103-104]

In the same years, similar quantifiers were investigated from a model-theoretic perspective by H. Friedman.[2] More recently, Mio et al. [148, 149] investigated

[1] Generalized quantifiers were first introduced by Mostowski, as "operators which represent a natural generalization of the logical quantifiers" [152, p. 13] and have then been extensively studied in the context of finite-model theory [142, 134]. Second-order generalized quantifiers have been recently defined as well [3].

[2] See [199] for a survey.

the possibility for such quantifiers to define extensions of MSO. These works were strongly influenced by generalized quantifiers [152] as well.

12.2 On the Expressive Power of MQPA

As anticipated, the language of **MQPA** allows us to express some basic results from probability theory, and to check their validity in the structure $\mathcal{P}$. In this section, we sketch a couple of examples.

The Infinite Monkey Theorem. Our first example is the so-called *infinite monkey theorem* ($\mathbf{IMT}$). It is a classic result stating that a monkey randomly typing on a keyboard has probability 1 of ending up writing *Macbeth* (or any other fixed string), sooner or later. To express it in **MQPA**, let us consider start by considering two formulae of **PA**, namely $G(x, y)$ and $H(x, y)$ expressing respectively that "the length of y is strictly smaller than the length of (the binary sequence coded by) x" and that "the y+1-th bit of x is 1". We formalize $\mathbf{IMT}$ through the following formula:

$$F_{\mathbf{IMT}} : (\forall x)\mathbf{C}^{1/1}(\forall y)(\exists z)(\forall w)\big(G(x, w) \to (H(x, w) \leftrightarrow \texttt{Flip}(y + z + w))\big).$$

Indeed, if x is the binary encoding of the *Macbeth*, $F_{\mathbf{IMT}}$ says that for all choice of start time y, there is a time $y + z$ after which the random sequence defined by the "`Flip`-formula" will evolve exactly like x with probability 1.

Due to Definition 12.1.3, we can even (semantically) justify $F_{\mathbf{IMT}}$. Let $F'(x, y, z, w)$ indicate the formula $G(x, w) \to \big(H(x, w) \leftrightarrow \texttt{Flip}(y + z + w)\big)$. We show that for any $n \in \mathbb{N}$, there is a measurable set $S^n \subseteq 2^{\mathbb{N}}$ of measure 1 such that any sequence in S^n satisfies the formula $(\forall y)(\exists z)(\forall w)F'(n, y, z, w)$. This fact is now proved relying on the second Borel-Cantelli Lemma, a well-known result from measure theory.

Theorem 12.2.1 ([25], Thm. 4.4, p. 55)**.** *If $(U_y)_{y\in\mathbb{N}}$ is a sequence of independent events in $2^{\mathbb{N}}$, and $\sum_y^{\infty} \mu_{\mathscr{C}}(U_y)$ diverges, then*

$$\mu_{\mathscr{C}}\Big(\bigcap_{y}\bigcup_{z>y} U_z\Big) = 1.$$

Let us fix $n \in \mathbb{N}$ and let $\ell(n)$ indicate the length of the binary string encoded by n. For simplicity suppose $\ell(n) > 0$ (the case $\ell(n) = 0$ is trivial). We construct S^n in a few steps:

1. For all $p \in \mathbb{N}$, let U_p^n be the cylinder of sequences which, after p steps, agree with n. Observe that sequences in U_p^n satisfy the formula $(\forall w)F'(n, p, 0, w)$, that is if the length of n is (strictly) greater than that of w, the w-th bit of n is exactly as $\texttt{Flip}(p + w)$.
2. For all $p \in \mathbb{N}$, let $V_p^n = U_{p\cdot\ell(n)+1}^n$. Observe that the sets V_p^n are pairwise independent and $\mu_{\mathscr{C}}\big(\sum_p^{\infty} V_p^n\big) = \infty$.

3. For all $p \in \mathbb{N}$, let

$$S_p^n = \bigcup \{U_{p+q}^n \mid (\exists_{s>p}) p + q = s \cdot \ell(n) + 1\}.$$

Observe that any sequence in S_p^n satisfies $(\exists z)(\forall w)F'(n,p,z,w)$. Moreover, one can check that $S_p^n = \bigcup_{q>p} V_q^n$.

4. Finally, let

$$S^n = \bigcap_p S_p^n.$$

Now, any sequence in S^n satisfies $(\forall y)(\exists z)(\forall w)F'(n,y,z,w)$. Furthermore, by Theorem 12.2.1,

$$\mu_{\mathscr{C}}(S^n) = \mu_{\mathscr{C}}\big(\bigcap_p \bigcup_{q>p} V_q^n\big) = 1.$$

Thus, for any choice of $n \in \mathbb{N}$,

$$\mu_{\mathscr{C}}\big([\![(\forall y)(\exists z)(\forall w)F'(x,p,z,w)]\!]_{\{x \leftarrow n\}}\big) \geq \mu_{\mathscr{C}}(S^n) \geq 1,$$

and we conclude $[\![F_{\mathbf{IMT}}]\!]_\xi = 2^{\mathbb{N}}$. In this way, we proved that, for any (binary encoding of a) word or finite string, "there would have been a time for such a word" [189, Act V, Scene V] or string to be typed by the monkey.

The Random Walk Theorem. A second example we consider is the *random walk theorem* (**RW**) stating that any simple random walk over $\mathbb{Z}$ starting from 1 will pass through 1 infinitely many times with probability 1. Formally, any $\omega \in 2^{\mathbb{N}}$ induces a simple random walk starting from 1, by letting the n-th move be right if $\omega(n) = 1$ holds and left if $\omega(n) = 0$ holds.

Theorem 12.2.2 ([25], Thm. 8.3, p. 117)**.** *Let $U_{i,j}^{(n)} \subseteq 2^{\mathbb{N}}$ be the set of sequences for which the simple random walk starting from i leads to j in n steps. Then, $\mu_{\mathscr{C}}\big(\bigcap_x \bigcup_{y \geq x} U_{11}^{(y)}\big) = 1$.*

The random predicate $\texttt{Flip}(n)$ induces a simple random walk starting from 1, by letting the n-th move be right if $\texttt{Flip}(n)$ holds and left if $\neg\texttt{Flip}(n)$ holds. Again we formalize **RW** in **MQPA** relying on two arithmetical formulae:

- $H(y,z)$ expresses that y is even and z is the code of a sequence of length $\frac{y}{2}$, such that for all $i, j < \frac{y}{2}$, $z_i < y$, and if $z_i = z_j$ then $i = j$, that is z codes a subset of $\{0, \ldots, y-1\}$ of cardinality $\frac{y}{2}$,
- let $K(y,z,v) = H(y,z) \wedge (\exists i)(i < \frac{y}{2} \wedge z_i = v)$.

Then, the measure-quantified formula expressing **RW** is as follows,

$$F_{\mathbf{RW}} : \mathbf{C}^{1/1}(\forall x)(\exists y)(\exists z)\big(y \leq x \wedge H(y,z) \wedge (\forall v)\big(v < y \rightarrow \big(K(y,z,v) \leftrightarrow \texttt{Flip}(v)\big)\big)\big).$$

The formula $F_{\mathbf{RW}}$ intuitively says that for any fixed x we can find $y \geq x$ and subset z of $\{0, \ldots, y-1\}$ of cardinality $\frac{y}{2}$, containing all and only the values

$v < y$ such that $\texttt{Flip}(v)$ holds, so that the number of $v < y$ with $\texttt{Flip}(v)$ holding coincides with the number of $v < y$ with $\neg\texttt{Flip}(v)$ holds. This is the case precisely when the simple random walk goes back to 1 after exactly y steps.

We can also show $F_{\mathbf{RW}}$ valid, considering the measurable set $S = \bigcap_n \bigcup_{p \geq n} U_{11}^{(p)}$. Let $F'(y, z, v)$ be the formula $(v < y \to (K(y, z, v) \leftrightarrow \texttt{Flip}(v)))$ and observe that any sequence in $U_{11}^{(n)}$ satisfies the formula $(\exists z)\big(H(n, z) \wedge (\forall v)F'(y, z, v, w)\big)$. Then, any sequence in S satisfies $(\forall x)(\exists y)(\exists z)\big(y \geq x \wedge H(y, z) \wedge (\forall v)F'(y, z, v)\big)$. Since by Theorem 12.2.2, $\mu_{\mathscr{C}}(S) = 1$, we conclude $\mu_{\mathscr{C}}([\![F_{\mathbf{RW}}]\!]_\xi) \geq \mu_{\mathscr{C}}(S) \geq 1$ and, thus, $[\![F_{\mathbf{RW}}]\!]_\xi = 2^{\mathbb{N}}$.

12.3 Randomized Arithmetizaion

The language of arithmetic can express interesting algorithmic properties, and several important questions in computation theory can be studied through arithmetic theory. It is a classic result in computability theory [97, 195, 198] that all computable functions are arithmetical, that is for any recursive function $f : \mathbb{N}^m \rightharpoonup \mathbb{N}$, there is formula of **PA** F_f, such that for any $n_1, \ldots, n_m, l \in \mathbb{N}$,

$$f(n_1, \ldots, n_m) = l \quad \text{iff} \quad \mathcal{N} \vDash F_f(n_1, \ldots, n_m, l).$$

In this section, we show that with **MQPA**, this fundamental result can be generalized to computable *random* functions. We start in Section 12.3.1 with a brief historical overview, recapping classic notions and achievements in recursion theory. Then, in Section 12.3.2, we introduce our main result, namely randomized arithmetization. Given the conceptual distance between the notions involved in the proof, we present two auxiliary classes of functions: probabilistic random functions $\mathcal{PR}$ [55] and oracle recursive functions $\mathcal{OR}$, the latter inspired by oracle machines. Finally, we put the two classes in relation, concluding our proof:

$$\frac{\text{computable functions}}{\text{formulae of } \textbf{PA}} = \frac{\text{probabilistic computable functions}}{\text{formulae of } \textbf{MQPA}}$$

12.3.1 Historical Background

Groundbreaking results in arithmetic and recursion theory were inspired by Hilbert's program: in his *Grundlagen der Geometrie* (1899) and, later, during the 1900 Paris congress, Hilbert listed 23 problems open problems in the foundation of mathematics, including the axiomatization of arithmetic, the proof of its consistency, and the *Entscheidungsproblem* [111].[3] Indeed, at the end of the XIX century, first formalizations of arithmetic appeared [169] and, in 1928, an (axiomatic) deductive system for FOL was neatly presented by Hilbert and

[3] Indeed, a recent discovery made by Thiele in Göttingen shown that a 24th problem – concerning the development of theories of proof methods in mathematics – had to be added to the Paris list [206].

Bernays. It was in this context that Gödel developed his theorems. Meanwhile, in his 1933 doctoral thesis, Gentzen introduced new proof-theoretical tools to represent the structure of mathematical arguments.[4]

Simultaneously, Hilbert's foundational work stimulated the development of a precise notion of *computable function*. In the 1930s on, new models appeared, from Church's λ-calculus to recursive functions, from Markov's algorithms to Turing machines. In particular, today definitions of (primitive) recursive functions come from Gödel, Church and Kleene's formalizations. Actually, the origin of the modern term "recursion" may be traced back to Dedekind's and Peano's works on natural numbers, see [129], but the development of the discipline towards nowadays formulation took some years.

The Axiomatization of Arithmetic

The first formulation of modern arithmetic is commonly attributed to either Peano – from whom it has taken its name – and Dedekind.[5] In particular, in his treatise *Arithmetices Principia* (1889), Peano formulated nine axioms: four for equality plus five properly arithmetical ones. In particular, he introduced the language and axiomatization in the section *De numeris et de additione*:[6]

$Explicationes.$

Signo N significatur *numerus (integer positivus)*.

- ► 1 ► *unitas*,
- ► $a+1$ ► *sequens* a sive a *plus* 1,
- ► $=$ ► *est aequalis*. Hoc ut novum signum considerandum est, etsi logicae signi figuram habeat.

$Axiomata.$

1. $1 \epsilon N$.
2. $a \epsilon N . \supset . a = a$.
3. $a, b \epsilon N . \supset : a = b . = . b = a$.
4. $a, b, c \epsilon N . \supset :: a = b . b = c : \supset . a = c$.
5. $a = b . b \epsilon N : \supset . a \epsilon N$.
6. $a \epsilon N . \supset . a + 1 \epsilon N$.
7. $a, b \epsilon N . \supset : a = b . = . a + 1 = b + 1$.

[4] Gentzen in fact obtained his natural deduction and sequent calculi as "byproducts" [221]. His main goal was to extend this approach to derivations in **PA**, so to study their (meta-)properties. His work can be interpreted as a continuation of Hilbert' s program in response to Gödel's results. For further details see [157].

[5] Of course, other mathematicians – from Hankel to Schr̈oder, from Skolem to Bernays – contributed to the development of recursive arithmetic and to the formulation of (what is today known as) **PA**. In his *Lehrbuch der Arithmetic für höhere Lehranstalten* (1861) Grassmann defined an abstract approach to natural numbers, with explicit recursive definition of arithmetical operations, without separating successor and addition. In was with Schröder and Dedekind that these operators gained independent definition and notation (using '). For further details see [222].

[6] The logical basis of Peano arithmetic mostly came from Boole. The notation is standard for the XIX century, when $\supset$ (abbreviating *consequentia*) stood for implication $\supset$.

8. $a \in N . \supset a + 1 - = 1$

9. $k \in K :: 1 \in k :: x \in N . x \in k : \supset_x . x + 1 \epsilon k :: \supset . N \supset k$. [169, p. 1]

So, Axioms (2.)-(5.) concerns identity, (1.) and (6.) define (resp.) 1 and the successor of a number both as numbers, (7.) expresses that different numbers has different successors, (8.) there is no number of which 1 is the successor, and (9.) is a formulation of the induction principle. In *Sul concetto del numero* (1891), Peano reformulated these axioms in terms of Dedekind's theory of operations and slightly modified (9.). It is a common idea – at least from van Heijenoort on [218] – that Peano owed his axioms and recursive definition to Dedekind, but their authorship is actually controversial.[7] Being by Dedekind or Peano, this formulation was the basis on which most foundational studies of the beginning of the XX century relied.

Recursion Theory

As Kleene wrote, the history of recursion theory is long, and can be seen as starting in the 1880s with Kronecker's, Peano's and Dedekind's works:

> "[t]he theory of recursive functions is nearly one hundred years old" [129, p. 43].

Nevertheless, initially, recursion theory was not a free-standing field, but, little by little, has become independent. In parallel, the notion of "recursion" obtained a clear and formal definition (relatively) recently [196, 197]. In the XIX century, the term was often used without being precisely introduced (sometimes even interchangeably with "computable"), and its occurrences showed the connection with the idea of *recurring* and induction.[8] To the best of our knowledge, primitive recursive functions were first defined by Gödel in 1931 [97], while the notion of general recursion first appeared in 1934/36 works by (resp.) Gödel [99] and Kleene [126, 125].

In [97], Gödel introduced primitive recursive function as a preliminary notion for his incompleteness proof. Actually, he used the term *recursive* function – "rekursive Function" – to define what is today called *primitive recursive*. Specifically, in 1931, Gödel took as basic functions, the constant and successor only [100, p. 46], while in 1934, he added the so-called identity function (corresponding to projection):

> The function $\varphi(x_1, \ldots, x_n)$ shall be compound with respect to $\psi(x_1, \ldots, x_m)$ and $\xi_i(x_1, \ldots, x_n)$ $(i = 1, \ldots, m)$ if, for all natural numbers $x_1, \ldots, x_n$,
>
> $$\varphi(x_1, \ldots, x_n) = \psi(\xi_1(x_1, \ldots, x_n), \ldots, \xi_m(x_1, \ldots, x_n)). \quad (1)$$

[7] According to von Plato, these axioms were invented by Peano [223] and the mentioning of Dedekind's 1888 script in [169] was just a late addition to the preface.

[8] In Grassman, Dedekind, and Peano's works arithmetical operations on numbers were *recursively* defined (at least implicitly). In 1919, Skolem used recursive definitions for functions, e.g. successor, then incorporated in Gödel's formalization.

$\varphi(x_1, \ldots, x_n)$ shall be said to be recursive with respect to $\psi(x_1, \ldots, x_{n-1})$ and $\xi(x_1, \ldots, x_{n+1})$ if, for all natural numbers, $k, x_2, \ldots, x_n$,

$$\varphi(0, x_2, \ldots, x_n) = \psi(x_2, \ldots, x_n) \tag{12.1}$$

$$\varphi(x+1, x_2, \ldots, x_n) = \xi(k, \varphi(x_2, \ldots, x_n), x_2, \ldots, x_n). \tag{2}$$

[...] We define the class of recursive functions to be the totality of function which can be generated by substitution, according to the scheme (1), and recursion, according to the schema (2), from the successor function $x+1$, constant function $f(x_1, \ldots, x_n) = x$, and identity function $\bigcup_j^n(x_1, \ldots, x_n) = x_j$ $(1 \leq j \leq n)$. In other words, a function φ shall be recursive if there is a finite sequence of functions $\varphi_1, \ldots, \varphi_n$ which terminates with φ such that each function of the sequence is either the successor function $x+1$ or a constant function $f(x_1, \ldots, x_n) = x$, or an identity function $\bigcup_j^n(x_1, \ldots, x_n) = x_j$, or is compound with respect to preceding functions, or is recursive with respect to preceding functions. [99, p. 43]

In 1934, basing on early suggestions by Herbrand, Gödel introduced the wider class of general recursive functions. The distinction between these two classes was made precise in [126, p. 727], thus leading to modern terminology.[9]

Gödel's Arithmetization

The so-called arithmetization theorem is also rooted in Gödel's celebrated article [97], where it is proved that every recursive function is arithmetical.

Theorem 12.3.1 (Arithmetization)**.** *All recursive functions are arithmetical.*

A formula is said to be *arithmetical* if it can be expressed in the language of arithmetic. Otherwise said, for any m-ary function $f : \mathbb{N}^m \to \mathbb{N}$ there is an arithmetical formula F_f such that, for every $n_1, \ldots, n_m, l \in \mathbb{N}$, $f(n_1, \ldots, n_m) = l$ when $\mathcal{N} \vDash F_f(\overline{n_1}, \ldots, \overline{n_m}, \overline{l})$, (where, for any $n \in \mathbb{N}$, $\overline{n}$ is the symbol in $\mathcal{L}_{\mathbb{N}}$ corresponding to n). So, first-order arithmetic can express the fundamental notions related to computation (including termination of programs).

As said it was in his 1931 paper that Gödel introduced arithmetization as a preliminary step to establish the incompleteness (and incompletability) of **PA**. In particular, after defining the notion of primitive recursion in § 3, he proceeded with his incompleteness proof by defining *arithmetical* formulae.

> A relation (class) is called *arithmetical* if it can be defined solely by means of the concepts +, ·, and the logical constants ∨, -, (x), =, where (x) and = are to relate only to natural numbers. The concept of "arithmetical proposition" is defined in a corresponding way. [100, p. 63]

Then, (so-called) Proposition VII, stating that every primitive recursive relation is arithmetical, is established:

> Proposition VII: Every recursive relation is arithmetical. [...] According to Proposition VII there corresponds to every problem of the form $(x)F(x)$ (F recursive) an equivalent arithmetical problem... [100, p. 65-66]

[9] Observe that Kleene's analysis was also important for the systematization of recursion theory, also in relation to other formalizations of algorithms [129].

12.3.2 Making Arithmetization Randomized

Furthermore, the recursion-theoretic characterization of computable functions was shown as particularly fruitful in the context of arithmetization. In 2014, an analogous recursive class was also introducedd for *probabilistic* functions. In what follows, we show that the language **MQPA** is expressive enough to define *random* functions.

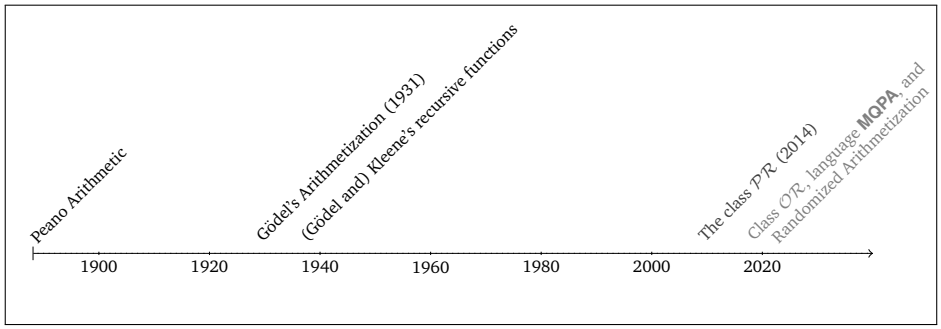


Figure 12.1: From Standard to Randomized Arithmetization

Introducing Randomized Arithmetization. In order to generalize Gödel's arithmetization to the computable random functions, we start with a notion notion of arithmetical random function.

Definition 12.3.1 (Arithmetical Random Function)**.** Let $\mathbb{D}(\mathbb{N})$ indicate the set of *pseudo-distributions on* $\mathbb{N}$.[10] A *random function* $f : \mathbb{N}^m \to \mathbb{D}(\mathbb{N})$ is said to be *arithmetical* when there is a formula of **MQPA** F_f, with free variables $x_1, \ldots, x_m, y$, such that for any $n_1, \ldots, n_m, l \in \mathbb{N}$,

$$\mu_{\mathscr{C}}\big([\![F_f(n_1, \ldots, n_m, l)]\!]\big) = f(n_1, \ldots, n_m)(l).$$

Then, the arithmetization theorem below relates random functions and formulae of **MQPA**.

Theorem 12.3.2 (Randomized Arithmetization)**.** *All computable random functions are arithmetical.*

As we shall see, we actually establish a stronger fact, proving that any computable random function is arithmetized – in the sense of Definition 12.3.1 – by a Σ_1^0-formula of **MQPA**, where a measure-quantified formula F is Σ_1^0 when there is an equivalent formula in the form $(\exists x_1) \ldots (\exists x_n) F'$ such that F' contains neither first-order or measure quantifiers.

[10] Further details related to the machine models will be given in Section 12.3.2.

The Structure of the Proof. Given the conceptual distance existing between TMs and **MQPA**, a direct proof of Theorem 12.3.2 would be cumbersome. So, we follow an alternative route. We formally consider the notion of computable *random* function, which was defined due to the equivalent class of probabilistic recursive function $\mathcal{PR}$, defined in [64, 55]. This class provides a proper, probabilistic counterpart of standard formalizations of recursive functions [126], but is inspired by a precise computational model, namely PTM,[11] and the source of randomness of such machines is quite different from that of our language **MQPA**. In order to fill the gap between the two notions we pass through a second class – that of oracle recursive function $\mathcal{OR}$ – this time inspired by oracle machines. We conclude our proof showing the correspondence between these two classes.

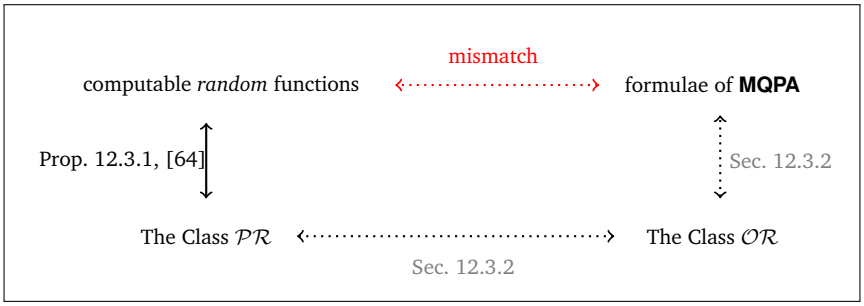


Figure 12.2: The Structure of the Proof

Probabilistic Function Algebras

In this section we introduce two classes of functions. The class of *probabilistic* or random recursive function $\mathcal{PR}$ was introduced in [64, 55] and shown equivalent to that of functions computable by PTMs. It provides the first recursion-theoretic characterization of computable random functions. Concretely, the approach of Dal Lago, Gabbrielli and Zuppiroli consists of generalizing classic results on (partial) computable functions by Church and Kleene [41, 126], adding an initial function able to "tossing a (fair) coin".

On the contrary, the class of *oracle recursive function* $\mathcal{OR}$ is new. It is inspired by the oracle machine model: these functions take natural numbers plus an infinite sequence of (random) bits – intuitively corresponding to the oracle tape – and return a number. In this case, the probabilistic element is enucleated by the so-called query function, which, given a number n and an infinite sequence of bit(-value)s ω, returns the n-th value of ω. Remarkably, the way in which this function accesses its source of randomness is similar to that associated with the interpretation of **MQPA**-formulae.

[11] For further details, see Section 2.1.

The Class $\mathcal{PR}$. The machine model on which the definition of $\mathcal{PR}$ relies is that of PTM [183, 91].[12] Any PTM can be seen as computing a *random* function [183, pp. 706-707]. As said, the set of pseudo-distributions $\mathbb{D}(\mathbb{N})$ is the set of functions $f : \mathbb{N} \to \mathbb{R}_{[0,1]}$, such that $\sum_{n\in\mathbb{N}} f(n) \leq 1$. So, given a PTM $\mathscr{M}_{\mathscr{P}}$, a random function is a function $\langle \mathscr{M}_{\mathscr{P}} \rangle : \mathbb{N} \to \mathbb{D}(\mathbb{N})$ that for each $n \in \mathbb{N}$ returns the pseudo-distribution $\mathbb{D}(\mathbb{N})$ of all possible outcomes $\mathscr{M}_{\mathscr{P}}$ produces when fed with (an encoding of) n in input together with the corresponding probability. Coherently, the class $\mathcal{PR}$ is defined by Dal Lago, Gabbrielli and Zuppiroli generalizing the intuition of Church and Kleene's standard one [41, 126, 125, 128]. Indeed, $\mathcal{PR}$ is characterized as the smallest class of functions, which (i.) contains some basic functions including one expressing (fair) coin tossing,[13] and (ii.) is closed under composition, primitive recursion and minimization.

For all this to make sense, composition and primitive recursion are defined following the *monadic* structure of $\mathbb{D}(\cdot)$. So, we preliminarily introduce the notion of *Kleisli extension* of a function with values in $\mathbb{D}(\mathbb{N})$.

Definition 12.3.2 (Kleisli Extension)**.** Given a function $f : \mathbb{N} \to \mathbb{D}(\mathbb{N})$, its *(simple) Kleisli extension* $f^{\mathbf{K}} : \mathbb{D}(\mathbb{N}) \to \mathbb{D}(\mathbb{N})$ is defined as follows:

$$f^{\mathbf{K}}(d)(n) = \sum_{i\in\mathbb{N}} d(i) \cdot f(i)(n).$$

In general, given a k-ary function $f : X_1 \times \cdots \times X_{i-1} \times \mathbb{N} \times X_{i+1} \times \cdots \times X_k \to \mathbb{D}(\mathbb{N})$, its *$i$-th Kleisli extension* $f_i^{\mathbf{K}} : X_1 \times \ldots X_{i-1} \times \mathbb{D}(\mathbb{N}) \times X_{i+1} \times \ldots X_k \to \mathbb{D}(\mathbb{N})$ is defined as,

$$f_i^{\mathbf{K}}(x_1, \ldots, x_{i-1}, d, x_{i+1}, \ldots, x_k)(n) = \sum_{j\in\mathbb{N}} d(j) \cdot f(x_1, \ldots, x_{i-1}, j, x_{i+1}, \ldots, x_k)(n).$$

Observe that the construction at the basis of the Kleisli extension ($\mathbf{K}$-extension, for short) can be applied more than once. Given a function $f : \mathbb{N}^k \to \mathbb{D}(\mathbb{N})$, its *total* $\mathbf{K}$-extension $f^{\mathbf{K}} : (\mathbb{D}(\mathbb{N}))^k \to \mathbb{D}(\mathbb{N})$ is defined as follows:

$$f^{\mathbf{K}}(d_1, \ldots, d_k)(n) = \sum_{i_1,\ldots,i_k\in\mathbb{N}} f(i_1, \ldots, i_k)(n) \cdot \prod_{1\leq j\leq k} d_j(i_j).$$

Example 12.3.1. Let us consider a binary function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{D}(\mathbb{N})$, its *total*

[12] For further details, see Section 2.1.2.

[13] In fact, as we shell see, this is concretely implemented by the fair coin function $r(\cdot)(\cdot)$ behaving either as the identity or as the successor, with the same probability $\frac{1}{2}$. For further details, see [55].

K-extension is as follows:

$$\begin{aligned} f^{\mathbf{K}}(d_1, d_2)(y) &= \sum_{i_1 \in \mathbb{N}} d_1(i_1) \cdot f^{\mathbf{K}}(i_1, d_2)(y) \\ &= \sum_{i_1 \in \mathbb{N}} d_1(i_1) \cdot \sum_{i_2 \in \mathbb{N}} d_2(i_2) \cdot f(i_1, i_2)(y) \\ &= \sum_{i_1, i_2 \in \mathbb{N}} f(i_1, i_2)(y) \cdot d_1(i_1) \cdot d_2(i_2) \\ &= \sum_{i_1, i_2 \in \mathbb{N}} f(i_1, i_2)(y) \cdot \prod_{k \in \{1,2\}} d_k(i_k). \end{aligned}$$

We now define the class $\mathcal{PR}$ in a formal way.

Definition 12.3.3 (The Class $\mathcal{PR}$ [64])**.** The class of *probabilistic recursive functions* $\mathcal{PR}$, is the smallest class of probabilistic functions containing:

- The *zero function* $z : \mathbb{N} \to \mathbb{D}(\mathbb{N})$, such that for every $x \in \mathbb{N}$,

$$z(x)(0) = 1$$

- The *successor function* $s : \mathbb{N} \to \mathbb{D}(\mathbb{N})$, such that for every $x \in \mathbb{N}$,

$$s(x)(x+1) = 1$$

- The *projection function* $\pi^n_m : \mathbb{N}^n \to \mathbb{D}(\mathbb{N})$, such that for $1 \leq m \leq n$,

$$\pi^n_m(x_1, \ldots, x_n)(x_m) = 1$$

- The *fair coin function* $r : \mathbb{N} \to \mathbb{D}(\mathbb{N})$ such that,

$$r(x)(y) = \begin{cases} \frac{1}{2} & \text{if } y = x \\ \frac{1}{2} & \text{if } y = x+1 \\ 0 & \text{otherwise} \end{cases}$$

and closed under:

- *Probabilistic composition*. Given $f : \mathbb{N}^n \to \mathbb{D}(\mathbb{N})$ and $g_1, \ldots, g_n : \mathbb{N}^k \to \mathbb{D}(\mathbb{N})$, their composition is a function $f \odot (g_1, \ldots, g_n) : \mathbb{N}^k \to \mathbb{D}(\mathbb{N})$ defined

as:[14]

$$(f \odot (g_1, \ldots, g_n))(\mathbf{x}) = f^{\mathbf{K}}(g_1(\mathbf{x}), \ldots, g_n(\mathbf{x}))$$

- *Probabilistic Primitive Recursion*. Given $f : \mathbb{N}^k \to \mathbb{D}(\mathbb{N})$ and $g : \mathbb{N}^{k+2} \to \mathbb{D}(\mathbb{N})$, the function $h : \mathbb{N}^{k+1} \to \mathbb{D}(\mathbb{N})$ obtained from them by primitive recursion is:

$$h(\mathbf{x}, 0) = f(\mathbf{x})$$
$$h(\mathbf{x}, y+1) = g^{\mathbf{K}}_{k+2}(\mathbf{x}, y, h(\mathbf{x}, y))$$

- *Probabilistic minimization.* Given $f : \mathbb{N}^{k+1} \to \mathbb{D}(\mathbb{N})$, the function $h : \mathbb{N}^k \to \mathbb{D}(\mathbb{N})$ obtained from minimization is as follows:

$$\mu f(\mathbf{x})(y) = f(\mathbf{x}, y)(0) \cdot \prod_{z<y} \sum_{k>0} f(\mathbf{x}, z)(k).$$

The following Proposition has been proved in [64].

Proposition 12.3.1 ([64])**.** *$\mathcal{PR}$ coincides with the class of computable random functions.*

The Class $\mathcal{OR}$. The class $\mathcal{PR}$ is still conceptually far from **MQPA**. While measure-quantified formulae access randomness in the form of a global supply of random bits, probabilistic recursive functions fire random choices locally, through the dedicated initial function. To bridge this gap, we introduce a third characterization of computable random functions, which is better-suited for our purposes. We define the class of oracle recursive functions $\mathcal{OR}$, as loosely inspired by *oracle* TMs, i.e. deterministic TMs the transition function of which can query a *random-bit tape* $\omega \in \mathbb{B}^{\mathbb{N}}$.

The class of *oracle recursive functions* $\mathcal{OR}$, is the smallest class of partial functions of the form $f : \mathbb{N}^m \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N}$, which (i.) contains the class of *oracle basic functions* and (ii.) is closed under composition, primitive recursion, and minimization. Formally,

[14] That is,

$$\begin{aligned}((f \odot (g_1, \ldots, g_n))(\mathbf{x}))(y) &= (f^{\mathbf{K}}(g_1(\mathbf{x}), \ldots, g_n(\mathbf{x})))(y) \\ &= \sum_{i_1, \ldots, i_n} f(i_1, \ldots, i_k)(y) \cdot \prod_{1 \leq j \leq n} g_j(\mathbf{x})(i_j).\end{aligned}$$

The simplest case is that of unary composition: given $f : \mathbb{N} \to \mathbb{D}(\mathbb{N})$ and $g : \mathbb{N} \to \mathbb{D}(\mathbb{N})$, the function $h : \mathbb{N} \to \mathbb{D}(\mathbb{N})$ obtained by composition from f and g is:

$$(f \odot g)(x)(y) = f^{\mathbf{K}}(g(x))(y).$$

Otherwise said,

$$((f \odot g)(x))(y) = \sum_{z \in \mathbb{N}} g(x)(z) \cdot f(z)(y).$$

Definition 12.3.4 (The Class $\mathcal{OR}$)**.** The class of *oracle recursive functions* $\mathcal{OR}$, is the smallest class of probabilistic functions containing:

- The *zero function* f_0, such that

$$f_0(x_1, \ldots, x_k, \omega) = 0$$

- The *successor function* f_s, such that

$$f_s(x, \omega) = x + 1$$

- The *projection function* f_{π_i}, such that for $1 \leq i \leq k$

$$f_{\pi_i}(x_1, \ldots, x_k, \omega) = x_i$$

- The *query function* f_q, such that

$$f_q(x, \omega) = \omega(x)$$

and closed under:

- *Oracle composition*. Given the oracle functions h from $\mathbb{N}^n \times \mathbb{B}^{\mathbb{N}}$ to $\mathbb{N}$, and $g_1, \ldots, g_n$ from $\mathbb{N}^m$, the function f obtained by composition from them is defined as:

$$f(x_1, ..., x_m, \omega) = h(g_1(x_1, ..., x_m, \omega), ..., g_n(x_1, ..., x_m, \omega), \omega)$$

- *Oracle primitive recursion.* Given two oracle functions h and g from respectively $\mathbb{N}^n \times \mathbb{B}^{\mathbb{N}}$ and $\mathbb{N}^{n+2} \times \mathbb{B}^{\mathbb{N}}$ to $\mathbb{N}$, the function f obtained by primitive recursion from them is as follows:

$$f(x, x_1, ..., x_n) = \begin{cases} f(0, x_1, ..., x_n, \omega) = h(x_1, ..., x_n, \omega) \\ f(x+1, x_1, ..., x_n, \omega) = g(f(x, x_1, ..., x_n, \omega), x, x_1, ..., x_n, \omega) \end{cases}$$

- *Oracle minimization.* Given the oracle function g from $\mathbb{N}^{n+1} \times \mathbb{B}^{\mathbb{N}}$ to $\mathbb{N}$, the function f obtained by minimization from g is defined as:

$$f(x_1, ..., x_n, \omega) = \mu x(g(x_1, ..., x_n, x, \omega) = 0).$$

Notice that the only basic function depending on ω is the query function and that all the closure schemes are independent from it.

Relating $\mathcal{OR}$ and $\mathcal{PR}$

Intuitively, the class $\mathcal{OR}$ corresponds to that of computable random functions, but, given their form, in what sense do oracle functions *represent* random ones? In order to clarify the relationship between $\mathcal{OR}$ and $\mathcal{PR}$, we need to introduce a series of auxiliary notions and lemmas.

The Auxiliary Function f^*. We start with the introduction of the auxiliary function f^*, associated with the oracle function f.

Definition 12.3.5 (Auxiliary Function)**.** Given an oracle function $f : \mathbb{N}^m \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N}$, the corresponding *auxiliary function* $f^* : \mathbb{N}^m \times \mathbb{N} \to \mathcal{P}(\mathbb{B}^{\mathbb{N}})$ is defined as:

$$f^*(x_1, \ldots, x_m, y) = \{\omega \mid f(x_1, \ldots, x_m, \omega) = y\}.$$

The following Lemma 12.3.1 ensures that the value of f^* is always a measurable set.

Lemma 12.3.1. *For any oracle recursive function $f : \mathbb{N}^m \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N} \in \mathcal{OR}$ and $x_1, \ldots, x_m, y \in \mathbb{N}$, the set $f^*(x_1, \ldots, x_m, y)$ is measurable.*

Proof. We show for each $f \in \mathcal{OR}$ that $f^* \in \sigma(\mathscr{C})$ by induction on the structure of oracle recursive functions:

Base Case. Let $f \in \mathcal{OR}$ be an oracle basic function. There are four possible sub-cases:

- *Zero Function.* Let f_0 be the zero function $f_0(x_1, \ldots, x_n, \omega) = 0$. Then,

  $$f_0^*(x_1, \ldots, x_n, 0) = \{\omega \mid f_0(x_1, \ldots, x_n, \omega) = 0\} = \mathbb{B}^{\mathbb{N}}.$$

  For the first axiom of σ-algebras, $\mathbb{B}^{\mathbb{N}} \in \sigma(\mathscr{C})$, and so also $f_0^* \in \sigma(\mathscr{C})$.
- *Successor Function.* Let f_s be the successor function $f_s(x, \omega) = x+1$. Then,

  $$f_s^*(x, x+1) = \{\omega \mid f_s(x, \omega) = x+1\} = \mathbb{B}^{\mathbb{N}}.$$

  As before, $f_s^* \in \sigma(\mathscr{C})$.
- *Projection Function.* Let f_{π_i} be the projection function $f_{\pi_i}(x_1, \ldots, x_n, \omega) = x_i$, with $1 \leq i \leq n$. Then,

  $$f_{\pi_i}^*(x_1, \ldots, x_n, x_i) = \{\omega \mid f_{\pi_i}(x_1, \ldots, x_n, \omega) = x_i\} = \mathbb{B}^{\mathbb{N}}.$$

  Again we conclude $f_{\pi_i}^* \in \sigma(\mathscr{C})$.
- *Query Function.* Let f_q be the query function $f_q(x, \omega') = \omega'(x)$. Then,

  $$f_q^*(x, \omega'(x)) = \{\omega \mid f_q(x, \omega) = \omega'(x)\} = \{\omega \mid \omega(x) = \omega'(x)\},$$

  that is

  $$f_q^*(x, \omega') = \begin{cases} \{\omega \mid \omega(x) = 1\} & \text{if } \omega'(x) = 1 \\ \{\omega \mid \omega(x) = 0\} & \text{if } \omega'(x) = 0. \end{cases}$$

  In both cases, $f_q^*(x, \omega')$ is a (thin) cylinder, and so $f_q^*(x, 0) \in \sigma(\mathscr{C})$.

Inductive Case. Let $f \in \mathcal{OR}$ be obtained by oracle composition, recursion, or minimization from oracle recursive functions. Since the three cases are proved in a similar way, let us take into account (simple) composition only. Let $f : \mathbb{N}^n \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N}$ be obtained by (unary) composition from $h : \mathbb{N} \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N}$ and

$g : \mathbb{N}^n \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N}$. Assume $f(x_1, \ldots, x_n, \omega) = h(g(x_1, \ldots, x_n, \omega), \omega) = v$. By Definition 12.3.5,

$$f^*(x_1, \ldots, x_n, v) = \bigcup_{z \in \mathbb{N}} \{\omega \mid h(z, \omega) = v\} \cap \{\omega \mid g(x_1, \ldots, x_n, \omega) = z\}$$
$$= \bigcup_{z \in \mathbb{N}} h^*(z, v) \cap g^*(x_1, \ldots, x_n, z).$$

By IH, $h^*, g^* \in \sigma(\mathscr{C})$. So, by the third axiom of σ-algebras, $h^* \cap g^* \in \sigma(\mathscr{C})$ as well. Therefore, since f^* is a countable union of measurable sets, again for the third axiom, we conclude $f^* \in \sigma(\mathscr{C})$. □

The Auxiliary Function $f^\sharp$. Due to Lemma 12.3.1, we can associate any oracle recursive function $f : \mathbb{N}^m \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N}$ with a random function $f^\sharp : \mathbb{N}^m \to \mathbb{D}(\mathbb{N})$, defined as

$$f^\sharp(x_1, \ldots, x_m)(y) = \mu_{\mathscr{C}}(f^*(x_1, \ldots, x_m, y)).$$

This defines a close correspondence between the classes $\mathcal{PR}$ and $\mathcal{OR}$.

Proposition 12.3.2. *For any $f \in \mathcal{PR}$, there is an oracle function $g \in \mathcal{OR}$, such that $f = g^\sharp$. Symmetrically, for any $f \in \mathcal{OR}$, $f^\sharp \in \mathcal{PR}$.*

Actually, for our purpose only the first part of Proposition 12.3.2 is necessary, namely the proof that for any $f \in \mathcal{PR}$, there is a $g \in \mathcal{OR}$ such that $g^\sharp = f$. This is established by means of a few intermediate steps.

First, some preliminary notions are introduced. We fix a *computable bijection* between $\mathbb{N}$ and $\mathbb{N} \times \mathbb{N}$, the corresponding maps $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $\pi_1, \pi_2 : \mathbb{N} \to \mathbb{N}$. A *tree* is defined as a subset X of the finite set of strings $\mathbb{B}^*$, such that if $t \in X$ and $v \sqsubset t$ ($\sqsubseteq$ being the prefix relation), then $v \notin X$. Given $t \in \mathbb{B}^*, \omega \in \mathbb{B}^{\mathbb{N}}$ is said to be an *n-extension of t* when $\omega = v \cdot t \cdot \omega'$, with $|v| = n$ and $\omega' \in \mathbb{B}^{\mathbb{N}}$. The set of all n-extensions of t is indicated as EXT_t^n and is measurable. Moreover, $\mu(\mathsf{EXT}_t^n) = \frac{1}{2^{|t|}}$ for every n and t. Given a tree X, every function $f : X \to \mathbb{N}$ is said to be an *X-function*. Thus, an oracle function $f : \mathbb{N}^{n+1} \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N} \in \mathcal{OR}$ returns a tree X and an X-function g on input $(m_1, \ldots, m_n)$ if and only if for every $k \in \mathbb{N}$:

i. $f(m_1, \ldots, m_n, k, \omega)$ is defined when $\omega \in \mathsf{EXT}_t^k$, where $t \in X$

ii. if $\omega \in \mathsf{EXT}_t^k$ and $t \in X$, then

$$f(m_1, \ldots, m_n, k, \omega) = q$$

where $\pi_1(q) = g(t)$ and $\pi_2(q) = |t|$.

Then, the following Lemma can be proved.

Lemma 12.3.2. *For every $f : \mathbb{N}^n \to \mathbb{D}(\mathbb{N}) \in \mathcal{PR}$, there is an oracle recursive function $g : \mathbb{N}^{n+1} \times \mathbb{B} \rightharpoonup \mathbb{N} \in \mathcal{OR}$, such that for any $m_1, \ldots, m_n$, g returns a tree $X_{m_1,\ldots,m_n}$ and an $X_{m_1,\ldots,m_n}$-function $h_{m_1,\ldots,m_n}$ on input $m_1, \ldots, m_n$ and*

$$f(m_1, \ldots, m_n)(y) = \sum_{h_{m_1,\ldots,m_n}(t)=y} \frac{1}{2^{|t|}}.$$

Proof. The proof is by induction on the structure of $f \in \mathcal{PR}$.

Base Case. Let $f \in \mathcal{PR}$ be a basic recursive function. There are four possible sub-cases:

- *Zero Function.* Let $z \in \mathcal{PR}$ be the zero function. Then, $f \in \mathcal{OR}$ is an oracle function so defined that on inputs m_1, k, ω it returns $\langle 0, 0 \rangle$. Indeed, g returns the tree $X_{m_1} = \{\epsilon\}$ and the X_{m_1}-function h_{m_1} always returning 0, as it can be easily checked. Moreover,

$$\begin{aligned} z(m_1)(y) &= \begin{cases} 1 & \text{if } y = 0 \\ 0 & \text{otherwise} \end{cases} \\ &= \sum_{h_{m_1}(s)=y} \frac{1}{2^{|t|}}. \end{aligned}$$

- *Successor Function.* Let $s \in \mathcal{PR}$ be the successor function. Then, $g \in \mathcal{OR}$ is an oracle function so defined that on inputs m_1, k, and ω, it returns the value $\langle m_1 + 1, 0 \rangle$. So, g returns the tree $X_{m_1} = \{\epsilon\}$ and the X_{m_1}-function h_{m_1} always returning $m_1 + 1$. Therefore,

$$\begin{aligned} s(m_1)(y) &= \begin{cases} 1 & \text{if } y = m_1 + 1 \\ 0 & \text{otherwise} \end{cases} \\ &= \sum_{h_{m_1}(t)=y} \frac{1}{2^{|t|}}. \end{aligned}$$

- *Projection Function.* Let $\pi_i^n \in \mathcal{PR}$, with $1 \leq i \leq n$, be the projection function. Then, $g \in \mathcal{OR}$ is an oracle function so defined that, on inputs $m_1, \ldots, m_n, k$, and ω it returns the value $\langle m_1, 0 \rangle$. Indeed, g returns the tree $X_{m_1,\ldots,m_n} = \{\epsilon\}$ and the $X_{m_1,\ldots,m_n}$-function $h_{m_1,\ldots,m_n}$ always returning m_i. Moreover,

$$\begin{aligned} \pi_i^n(m_1, \ldots, m_n)(y) &= \begin{cases} 1 & \text{if } y = m_i \\ 0 & \text{otherwise} \end{cases} \\ &= \sum_{h_{m_1,\ldots,m_n}(t)=y} \frac{1}{2^{|t|}}. \end{aligned}$$

- *Fair Coin Function.* Let $r \in \mathcal{PR}$ be the fair coin function. Then, $g \in \mathcal{OR}$ is an oracle function so defined that $g(m_1, k, \omega) = \langle l, 1 \rangle$ where,

$$l = \begin{cases} m_1 & \text{if } \omega[k] = 0 \\ m_{1+1} & \text{if } \omega[k] = 1. \end{cases}$$

Inductive Case. If f is obtained by either composition, primitive recursion or minimization, the argument is a bit more involved. Indeed, to define the

function g we must take into account how the bits of the oracle accessed by g are distributed in an independent way to each of the component functions. We only illustrate how this works in the case of composition. Let f be obtained by composition from $f_1, \ldots, f_p : \mathbb{N}^n \to \mathbb{D}(\mathbb{N})$ and $f' : \mathbb{N}^p \to \mathbb{D}(\mathbb{N})$, that is

$$f(m_1, \ldots, m_n)(y) = \sum_{i_1, \ldots, i_p} f'(i_1, \ldots, i_p)(y) \cdot \prod_{j=1}^{p} f_j(m_1, \ldots, m_n)(i_j).$$

By IH, there exist functions $g_1, \ldots, g_p : \mathbb{N}^{n+1} \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N}$ and $g' : \mathbb{N}^{p+1} \times \mathbb{B}^{\mathbb{N}} \to \mathbb{N}$ such that:

1. for all $m_1, \ldots, m_n \in \mathbb{N}$, each g_i returns a tree $C^i_{m_1,\ldots,m_n}$ and an $X^i_{m_1,\ldots,m_n}$-function $h^i_{m_1,\ldots,m_n}$ and

$$f_i(m_1, \ldots, m_n)(y) = \sum_{h^i_{m_1,\ldots,m_n}(t)=y} \frac{1}{2^{|t|}}$$

2. for all $m_1, \ldots, m_p \in \mathbb{N}$, g' returns a tree $Y_{m_1,\ldots,m_n}$ and a $Y_{m_1,\ldots,m_p}$-function $h'_{m_1,\ldots,m_p}$, and

$$f'(m_1, \ldots, m_p)(y) = \sum_{h'_{m_1,\ldots,m_p}(t)=y} \frac{1}{2^{|t|}}.$$

Thus, we have:

$$\begin{aligned} f(m_1, \ldots, m_n)(y) &= \sum_{i=1,\ldots,i_p} \left(\sum_{h'_{i_1,\ldots,i_p}(s)=y} \frac{1}{2^{|s|}} \right) \cdot \left(\prod_{j=1}^{p} \sum_{h^j_{m_1,\ldots,m_n}(t)=i_j} \frac{1}{2^{|t|}} \right) \\ &= \sum_{i=1,\ldots,i_p} \left(\sum_{h'_{i_1},\ldots,h'_{i_p}(s)=y, h^j_{m_1,\ldots,m_n}(t_j)=i_j} \frac{1}{2^{|s|+\sum_{j=1}^{p}|t_j|}} \right) \end{aligned}$$

For all $m_1, \ldots, m_n \in \mathbb{N}$, let $X_{m_1,\ldots,m_n} = \{t_1 \cdots t_p \mid s \in t_j \in X^j_{m_1,\ldots,m_n}, s \in Y_{h^1_{\vec{m}}(t_1),\ldots,h^p_{\vec{m}(t_p)}}\}$. Observe that any $v \in X_{m_1,\ldots,m_n}$ can be decomposed in a unique way as $v = t_0 \cdot t_1 \cdots t_p$.[15] Using this fact, we show that $X_{m_1,\ldots,m_n}$ is also a tree.[16] Let $h_{m_1,\ldots,m_n} : X_{m_1,\ldots,m_n} \to \mathbb{N}$ be defined by $h_{m_1,\ldots,m_n}(v) = y$, where v uniquely decomposes as $s \cdot t_1 \cdots t_p, h^j_{\vec{m}}(t_j) = i_j$ and $h'_{i_1,\ldots,i_p}(s) = y$. We can

[15] Assume that $t'_0 \cdot t'_1 \cdots t'_p$ is any other decomposition and $j \leq p$ is minimum such that $t_j \neq t'_j$. Then, it must be either $t_j \sqsubset t'_j$ or $t'_j \sqsubset t_j$, which contradicts the fact that $X^j_{m_1,\ldots,m_n}$ and $Y_{h^1_{\vec{m}}(t_1),\cdots,h^p_{\vec{m}}(t_p)}$ are all trees.

[16] Suppose $v = t_0 \cdot t_1 \cdots t_p \in X_{m_1,\ldots,m_n}$ and $v' \in X_{m_1,\ldots,m_n}$, where $v' \sqsubseteq v$. Then, v' has a unique decomposition $t'_0 \cdots t'_p$ and, as easily shown by induction on $j \leq p$, $t'_j = t'_j$ holds. Hence, $v' = v$ holds, against the assumption.

finally define,

$$g(m_1,...,m_n,k,\omega) = \Big\langle \pi_1\Big(g'\big(\pi_1(L_1),...,\pi_1(L_p),k+\sum_{j=1}^{p} L_j,\omega\big)\Big), \\ \pi_2\Big(g'\big(\pi_1(L_1),\ldots,\pi_1(L_p),k+\sum_{j=1}^{p} L_j,\omega\big)\Big) + R_p\Big\rangle$$

where L_j, R_j are defined by induction as:

$$\begin{aligned} L_1 &= g_1(\vec{m},k,\omega) & R_1 &= 0 \\ L_{j+1} &= g_{j+1}(\vec{m},k+R_{j+1},\omega) & R_{j+1} &= R_j + \pi_2(L_j) \end{aligned}$$

It can be checked that, by construction, $g(m_1,\ldots,m_n,k,\omega) = \langle h_{m_1,\ldots,m_n}(v), |v|\rangle$, where $v \in \mathsf{EXT}^k_{v'}$ and v' uniquely decomposes as $s \cdot t_1 \cdots t_p$.

Using the equations above we conclude,

$$f(m_1,\ldots,m_n)(y) = \sum_{h_{m_1,\ldots,m_n}(v)=y} \frac{1}{2^{|v|}}.$$

□

Then, the desired proof of Proposition 12.3.2 is a straightforward corollary of Lemma 12.3.2.

Proof of Proposition 12.3.2. It is a consequence of Lemma 12.3.2. Indeed, if g' is obtained from f by Lemma 12.3.2 and $g(m_1,\ldots,m_n,\omega) = \pi_2(g'(m_1,\ldots,m_n,0,\omega))$, then $f = g^\sharp$. □

Arithmetizing Oracle Functions. The last ingredient to prove Theorem 12.3.2 is Lemma 12.3.3 below, which is established by induction on the structure of functions in $\mathcal{OR}$.

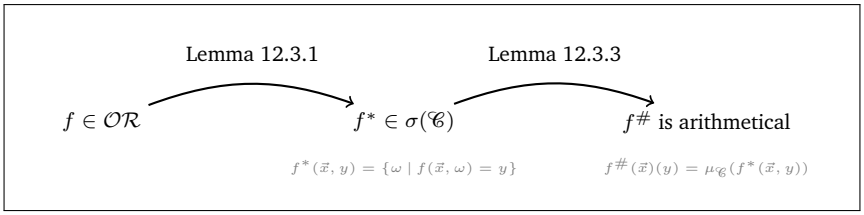


Figure 12.3: From $f \in \mathcal{OR}$ to arithmetical $f^\#$

Preliminarily observe that for both functions functions in $\mathcal{OR}$ and formulae of **MQPA**, randomness is defined in terms of denumerable amount of random bits. As a consequence, the following proof is straightforward.

Lemma 12.3.3. *For any oracle function $f \in \mathcal{OR}$, the random function $f^\sharp$ is arithmetical.*

Proof. By construction for any oracle function $f \in \mathcal{OR}$, the corresponding $f^\sharp$ is defined as,

$$f^\sharp = \mu_{\mathscr{C}}(f^*(x_1, \ldots, x_m, y)) = \mu_{\mathscr{C}}(\{\omega \mid f(x_1, \ldots, x_m, \omega) = y\}).$$

We prove that $f^\sharp$ is arithmetical, i.e. that there is a formula of **MQPA** $F_{f^\sharp}$ such that for any $n_1, \ldots, n_m, l \in \mathbb{N}$,

$$\mu_{\mathscr{C}}([\![F_{f^\sharp}(n_1, \ldots, n_m, l)]\!]) = f^\sharp(n_1, \ldots, n_m)(l).$$

In fact, we will establish something stronger, namely that for any $f \in \mathcal{OR}$, there is an F_f, such that for any $n_1, \ldots, n_m, l \in \mathbb{N}$, $[\![F_f(n_1, \ldots, n_m, l)]\!] = \{\omega \mid f(n_1, \ldots, n_m, \omega) = l\}$. The proof is by induction on the structure of oracle recursive functions. Actually, the only case which is worth considering is that of query functions, as all other cases are obtained by trivial generalizations of standard ones [97].

Base Case. For any basic oracle function $f \in \mathcal{OR}$, the corresponding random function $f^\sharp$ is arithmetical. There are four possible sub-cases:

- *Oracle Zero Function.* Let $f_0 \in \mathcal{OR}$ be the oracle zero function. Then, $f_0(x_1, \ldots, x_n, \omega) = y$, with $y = 0$, is such that the corresponding $f_0^\sharp$ is defined by the following formula of **MQPA**,

$$F_{f_0^\sharp} : y = \mathtt{0}.$$

- *Oracle Successor Function.* Let $f_s \in \mathcal{OR}$ be the successor function. By definition, $f_s(x, \omega) = y$, with $y = x + 1$ is such that $f_s^\sharp$ is defined by the following formula of **MQPA**,

$$F_{f_s^\sharp} : \mathtt{S}(x) = y.$$

- *Oracle Projection Function.* Let $f_{\pi_i} \in \mathcal{OR}$ be the projection function. By Definition, $f_{\pi_i}(x_1, \ldots, x_k, \omega) = y$ with $y = x$ is such that $f_{\pi_i}^\sharp$ is defined by the following formula of **MQPA**,

$$F_{\pi_i^k} : x_i = y.$$

- *Oracle Query Function*. Let $q \in \mathcal{OR}$ be the query function. By Definition, $q(x, \omega) = \omega(x)$ is such that $f_q^\sharp$ is defined by the following formula of **MQPA**,

$$F_q : x = (y = \mathtt{1} \wedge \mathtt{Flip}(x)) \vee (y = \mathtt{0} \wedge \neg \mathtt{Flip}(x)).$$

Inductive Case. For each oracle function $f \in \mathcal{OR}$ obtained by composition, primitive recursion, or minimization from oracle recursive functions, the corresponding random function $f^\sharp$ is arithmetical. The proof is very similar to

standard ones, so we consider the case of (simple) composition only. Let f be obtained by composition from h and g, that is:

$$f(x_1, \ldots, x_n, \omega) = h(g(x_1, \ldots, x_n, \omega), \omega) = v.$$

It is possible to show that $f^\sharp(x_1, \ldots, x_n)(v)$ is arithmetical, namely that for every $x_1, \ldots, x_n, v \in \mathbb{N}$, there is a formula of **MQPA** $F_{f^\sharp}(x_1, \ldots, x_n, v)$, such that $\mu_{\mathscr{C}}([\![F_{f^\sharp}(x_1, \ldots, x_n, v)]\!]) = f^\sharp$ (and for any $m_1, \ldots, m_n, l \in \mathbb{N}$ $[\![F_{f^\sharp}(m_1, \ldots, m_n, l)]\!] = \{\omega \mid f(m_1, \ldots, m_n) = l\}$). Indeed, the desired formula is

$$F_{f^\sharp} = (\exists v)\big(F_{h^\sharp}(v, y) \wedge F_{g^\sharp}(x_1, \ldots, x_n, v)\big),$$

where by IH $\mu_{\mathscr{C}}([\![F_{h^\sharp}]\!]) = h^\sharp$ and $\mu_{\mathscr{C}}([\![F_{g^\sharp}]\!]) = g^\sharp$. □

Concluding the Proof. Finally, Theorem 12.3.2 is proved relying on Lemma 12.3.3 above together with Proposition 12.3.1.

Proof of Theorem 12.3.2. Any computable random function is in $\mathcal{PR}$, by Proposition 12.3.1, and each $\mathcal{PR}$ function is arithmetical by Lemma 12.3.3 and Proposition 12.3.2. Indeed, by Proposition 12.3.2, for any $f \in \mathcal{PR}$, there is a $g \in \mathcal{OR}$ such that $f = g^\sharp$ and, since $g \in \mathcal{OR}$, by Lemma 12.3.3, $g^\sharp (= f)$ is arithmetical. □

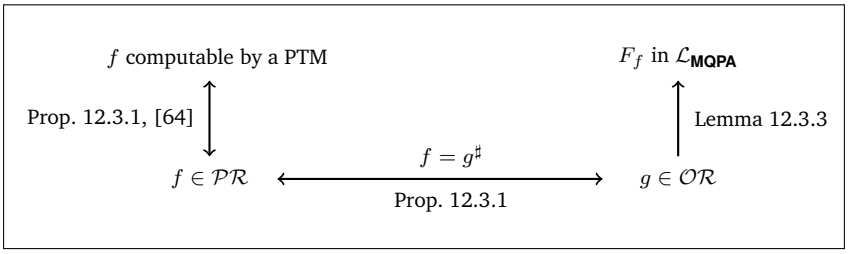


Figure 12.4: The Structure of the Proof

Chapter 13

An Arithmetic to Characterize Probabilistic Classes

We introduce a new bounded theory $\mathbf{RS}^1_2$, and show that the functions which are Σ^b_1-representable in it are precisely random functions which can be computed in polynomial time. Concretely, we pass through the class of oracle functions over strings $\mathcal{POR}$, which is introduced in Section 13.2, together with $\mathbf{RS}^1_2$. Then, we show that functions computed by poly-time PTMs are *arithmetically* characterized by a class of probabilistic bounded formulae: in Section 13.3.2, we prove that the class of poly-time oracle functions is equivalent to that of functions which are Σ^b_1-representable in $\mathbf{RS}^1_2$, and in Section 13.4 we establish the converse. This result, together with the notion of measure quantifier, allows us to internalize probabilistic computation with resource- and error-bound checks *within the logical system* and to provide an *arithmetical* characterization of $\mathbf{BPP}$. This is done in Section 13.5.2.

13.1 Overview

Usual characterizations of poly-time (deterministic) functions in bounded arithmetic are obtained by two "macro" results [34, 83]. Some Cobham-style algebra for poly-time functions is introduced and shown equivalent to (1) that of functions computed by TMs running in polynomial time, and (2) that of functions which are Σ^b_1-representable in the proper bounded theory. The global structure of our proof follows a similar path, with an algebra of oracle recursive function, called $\mathcal{POR}$, playing the role of our Cobham-style function algebra. In our case, functions are poly-time computable by PTMs and the theory is randomized $\mathbf{RS}^1_2$. After introducing these classes, we show that the random functions which are Σ^b_1-representable in $\mathbf{RS}^1_2$ are precisely those in $\mathcal{POR}$, and that $\mathcal{POR}$ is equivalent (in a very specific sense) to the class of functions computed by PTMs running in polynomial time. While the first part is established due to

standard arguments [83, 46], the presence of randomness introduced a delicate ingredient to be dealt with in the second part. Indeed, functions in $\mathcal{POR}$ access randomness in a rather different way with respect to PTMs, and relating these models requires some effort, that involves long chains of intermediate simulations.

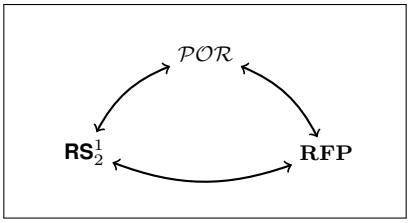


Figure 13.1: Proof Schema

Concretely, we start by defining the class of oracle functions over strings, the new theory $\mathbf{RS}_2^1$, strongly inspired by [84], but over a "probabilistic word language", and considering a slightly modified notion of Σ_i^b-representability, fitting the domain of our peculiar oracle functions. Then, we prove that the class of random functions computable in polynomial time, called $\mathbf{RFP}$, is precisely the class of functions which are Σ_1^b-representable in $\mathbf{RS}_2^1$ in three steps:

1. We prove that functions in $\mathcal{POR}$ are Σ_1^b-representable in $\mathbf{RS}_2^1$ by induction on the structure of oracle functions (and relying on the encoding machinery presented in [34, 83]).
2. We show that all functions which are Σ_1^b-representable in $\mathbf{RS}_2^1$ are in $\mathcal{POR}$ by realizability techniques similar to Cook and Urquhart's one [46].
3. We generalize Cobham's result to probabilistic models, showing that functions in $\mathcal{POR}$ are precisely those in $\mathbf{RFP}$.

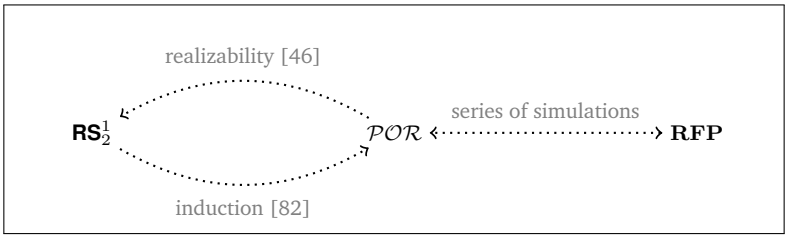


Figure 13.2: Our Proof in a Nutshell

13.2 Introducing $\mathcal{POR}$ and $\mathbf{RS}_2^1$

In this section, we introduce a Cobham-style function algebra for poly-time oracle recursive functions $\mathcal{POR}$, and a randomized bounded arithmetic $\mathbf{RS}_2^1$. As Ferreira's ones [82, 83], these classes are both associated with binary strings rather than natural positive integer: $\mathcal{POR}$ is a class of oracle functions over sequences of bits, while $\mathbf{RS}_2^1$ is defined in a probabilistic word language $\mathcal{RL}$. Strings support a natural notion of term-size and make it easier to deal with bounds and time-complexity. Observe that working with strings is not crucial and all results below could be spelled out in terms of natural numbers. Indeed, theories have been introduced in both formulations – Ferreira's Σ_1^b-NIA and Buss' $\mathbf{S}_2^1$ – and proved equivalent [84].

13.2.1 The Function Algebra $\mathcal{POR}$

We introduce a function algebra for poly-time *oracle* recursive functions inspired by Ferreira's $\mathcal{PTCA}$ [82, 83], and defined over strings.

Notation 13.2.1. Let $\mathbb{B} = \{\mathbb{0}, \mathbb{1}\}$, $\mathbb{S} = \mathbb{B}^*$ be the set of binary strings of finite length, and $\mathbb{O} = \mathbb{B}^{\mathbb{S}}$ be the set of binary strings of infinite length. Metavariables $\eta', \eta'', \ldots$ are used to denote the elements of $\mathbb{O}$.

Let $|\cdot|$ denote the length-map, so that for any string x, $|x|$ indicates the length of x. Given two binary strings x, y we use $x \subseteq y$ to express that x is an *initial* or *prefix substring* of y, $x \frown y$ (abbreviated as xy) for concatenation, and $x \times y$ obtained by self-concatenating x for $|y|$-times. Given an infinite string of bits η, and a finite string x, $\eta(x)$ denotes *one* specific bit of η, the so-called x-th bit of x.

A fundamental difference between oracle functions and those of $\mathcal{PTCA}$ is that the latter ones are of the form $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$, carrying an additional argument to be interpreted as the underlying source of random bits. Furthermore, $\mathcal{PR}$ includes the basic function *query*, $Q(x, \eta) = \eta(x)$, which can be used to observe any bit in η.

Definition 13.2.1 (The Class $\mathcal{POR}$)**.** The *class* $\mathcal{POR}$ is the smallest class of functions $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$, containing:

- The *empty (string) function* $E(x, \eta) = \boldsymbol{\epsilon}$
- The *projection (string) functions* $P_i^n(x_1, \ldots, x_n, \eta) = x_i$, for $n \in \mathbb{N}$ and $1 \leq i \leq n$
- The *word-successor* $S_b(x, \eta) = x\mathbb{b}$, where $b = 0$ if $\mathbb{b} = \mathbb{1}$ and $b = 1$ if $\mathbb{b} = \mathbb{1}$
- The *conditional (string) function*

$$C(\boldsymbol{\epsilon}, y, z_0, z_1, \eta) = y$$
$$C(x\mathbb{b}, y, z_0, z_1, \eta) = z_b,$$

  where $b = 0$ if $\mathbb{b} = \mathbb{0}$ and $b = 1$ if $\mathbb{b} = \mathbb{1}$

- The *query (string) function* $Q(x, \eta) = \eta(x)$

and closed under the following schemas:

- *Composition*, where f is defined from $g, h_1, \ldots, h_k$ as

$$f(\vec{x}, \eta) = g(h_1(\vec{x}, \eta), \ldots, h_k(\vec{x}, \eta), \eta)$$

- *Bounded recursion on notation*, where f is defined from g, h_0, and h_1 as

$$\begin{aligned} f(\vec{x}, \boldsymbol{\epsilon}, \eta) &= g(\vec{x}, \eta) \\ f(\vec{x}, y\mathbb{0}, \eta) &= h_0(\vec{x}, y, f(\vec{x}, y, \eta), \eta)|_{t(\vec{x},y)} \\ f(\vec{x}, y\mathbb{1}, \eta) &= h_1(\vec{x}, y, f(\vec{x}, y, \eta), \eta)|_{t(\vec{x},y)} \end{aligned}$$

  and t is obtained from $\boldsymbol{\epsilon}, \mathbb{1}, \mathbb{0}, \frown$, and $\times$ by explicit definition, that is t can be obtained applying $\frown$ and $\times$ on the constants $\boldsymbol{\epsilon}, \mathbb{0}, \mathbb{1}$, and the variables $\vec{x}$ and y.[1]

Actually, the conditional function C could be defined by bounded recursion.

Remark 13.2.1. *Neither the query function or the conditional function appear in Ferreira's characterization [83], which instead contains the "substring-conditional" function:*

$$S(x, y, \eta) = \begin{cases} \mathbb{1} & \text{if } x \subseteq y \\ \mathbb{0} & \text{otherwise,} \end{cases}$$

which can be defined in $\mathcal{POR}$ by bounded recursion.

13.2.2 Randomized Bounded Arithmetics

First, we introduce a probabilistic word language for our bounded arithmetic, together with its quantitative interpretation.

The Language $\mathcal{RL}$. Following [84], we consider a first-order signature for natural numbers in binary notation endowed with a special predicate symbol $\texttt{Flip}(\cdot)$. Consequently, formulae are interpreted over $\mathbb{S}$ rather than $\mathbb{N}$.

Definition 13.2.2 (Terms and Formulae of $\mathcal{RL}$)**.** *Terms* and *formulae of $\mathcal{RL}$* are defined by the grammar below:

$$t ::= x \mid \epsilon \mid \texttt{0} \mid \texttt{1} \mid t \frown t \mid t \times t$$

$$F ::= \texttt{Flip}(t) \mid t = s \mid \neg F \mid F \wedge F \mid F \vee F \mid (\exists x)F \mid (\forall x)F.$$

[1] Notice that there is a clear correspondence with the grammar for terms in $\mathcal{RL}$, Definition 13.2.5.

Notation 13.2.2 (Truncation)**.** For readability, we adopt the following abbreviations: ts for $t \frown s$, $\mathtt{1}^t$ for $\mathtt{1} \times t$, and $t \preceq s$ for $\mathtt{1}^t \subseteq \mathtt{1}^s$, expressing that the length of t is smaller than that of s. Given three terms $t, r,$ and s, the abbreviation $t|_r = s$ denotes the following formula,

$$(\mathtt{1}^r \subseteq \mathtt{1}^t \wedge s \subseteq t \wedge \mathtt{1}^r = \mathtt{1}^s) \vee (\mathtt{1}^t \subseteq \mathtt{1}^r \wedge s = t),$$

saying that s is the *truncation* of t at the length of r.

Every string $\sigma \in \mathbb{S}$ can be seen as a term of $\mathcal{RL}$ $\overline{\sigma}$, such that $\overline{\epsilon} = \epsilon$, $\overline{\sigma \mathbb{b}} = \overline{\sigma}\mathtt{b}$, where $\mathbb{b} \in \mathbb{B}$ and $\mathtt{b} \in \{\mathtt{0}, \mathtt{1}\}$, e.g. $\overline{\mathbb{0}\mathbb{0}\mathbb{1}} = \mathtt{001}$.

A central feature of bounded arithmetic is the presence of bounded quantification.

Notation 13.2.3 (Bounded Quantifiers)**.** In $\mathcal{RL}$, *bounded quantified expressions* are expressions of either the form $(\forall x)(\mathtt{1}^x \subseteq \mathtt{1}^t \rightarrow F)$ or $(\exists x)(\mathtt{1}^x \subseteq \mathtt{1}^t \wedge F)$, usually abbreviated as $(\forall x \preceq t)F$ and $(\exists x \preceq t)F$ respectively.

Notation 13.2.4. We call *subword quantifications*, quantifications of the form $(\forall x \subseteq^* t)F$ and $(\exists x \subseteq^* t)F$, abbreviating $(\forall x)\big((\exists w \subseteq t)(wx \subseteq t) \rightarrow F\big)$ and $(\exists x)(\exists w \subseteq t)(wx \subseteq t \wedge F)$. Furthermore, we abbreviate so-called *initial subword quantification* $(\forall x)(x \subseteq t \rightarrow F)$ as $(\forall x \subseteq t)F$ and $(\exists x)(x \subseteq t \wedge F)$ as $(\exists x \subseteq t)F$.

The distinction between bounded and subword quantification is important for complexity reasons. If $\sigma \in \mathbb{S}$ is a string of length k, the witness of a subword existentially quantified formula $(\exists x \subseteq^* \overline{\sigma})F$ is to be looked for among all possible sub-strings of σ, that is within a space of size $\mathcal{O}(k)$. On the contrary, the witness of a bounded formula $(\exists x \preceq \overline{\sigma})F$ is to be looked for among all possible strings of length k, namely within a space of size $\mathcal{O}(2^k)$.

Remark 13.2.2. *In order to avoid misunderstanding let us briefly sum up the different notions and symbols used for subword relations. We use $\subseteq$ to express a relation between strings, that is $x \subseteq y$ expresses that x is an initial substring or prefix of y. We use $\subseteq$ as a relation symbol in the language $\mathcal{RL}$. We use $\preceq$ as an auxiliary symbol in the language $\mathcal{RL}$; in particular, as seen, $t \preceq s$ is syntactic sugar for $\mathtt{1}^t \subseteq \mathtt{1}^s$. We use $\subseteq^*$ as an auxiliary symbol in the language $\mathcal{RL}$ to denote subword quantification. We also use $(\exists w \subseteq t)F$ as an abbreviation of $(\exists w)(w \subseteq t \wedge F)$ and similarly for $(\forall w \subseteq t)$.*

Definition 13.2.3 (Σ_1^b-Formulae)**.** A Σ_0^b-formula is a subword quantified formula, i.e. a formula belonging to the smallest class of $\mathcal{RL}$ containing atomic formulae and closed under Boolean operations and subword quantification. A formula is said to be a *Σ_1^b-formula*, if it is of the form $(\exists x_1 \preceq t) \ldots (\exists x_n \preceq t_n)F$, where the only quantifications in F are subword ones. We call Σ_1^b the class containing all and only the Σ_1^b-formulae.

An *extended Σ_1^b-formula* is any formula of $\mathcal{RL}$ that can be constructed in a finite number of steps, starting with subword quantifications and bounded existential quantifications.

The Bounded Theory $\mathbf{RS}^1_2$. We introduce the bounded theory $\mathbf{RS}^1_2$, which can be seen as a probabilistic version to Ferreira's Σ^b_1-NIA [82]. It is expressed in the language $\mathcal{RL}$.

Definition 13.2.4 (Theory $\mathbf{RS}^1_2$)**.** The theory $\mathbf{RS}^1_2$ is defined by axioms belonging to two classes:

- *Basic axioms*:

  1. $x\epsilon = x$
  2. $x(y\mathtt{b}) = (xy)\mathtt{b}$
  3. $x \times \epsilon = \epsilon$
  4. $x \times x\mathtt{b} = (x \times y)x$
  5. $x \subseteq \epsilon \leftrightarrow x = \epsilon$
  6. $x \subseteq y\mathtt{b} \leftrightarrow x \subseteq y \vee x = y\mathtt{b}$
  7. $x\mathtt{b} = y\mathtt{b} \rightarrow x = y$
  8. $x\mathtt{0} \neq y\mathtt{1}$,
  9. $x\mathtt{b} \neq \epsilon$

  with $\mathtt{b} \in \{\mathtt{0}, \mathtt{1}\}$

- *Axiom schema for induction on notation*,

$$B(\epsilon) \wedge (\forall x)\big(B(x) \rightarrow B(x\mathtt{0}) \wedge B(x\mathtt{1})\big) \rightarrow (\forall x)B(x),$$

  where B is a Σ^b_1-formula in $\mathcal{RL}$.

Induction on notation adapts the usual induction schema of **PA** to the binary representation. Of course, as in Buss' and Ferreira's approach, the restriction of this schema to Σ^b_1-formulae is essential to characterize algorithms computed *with bounded resources*. Indeed, more general instances of the schema would extend representability to random functions which are not poly-time (probabilistic) computable.

Proposition 13.2.1 ([82])**.** *In $\mathbf{RS}^1_2$ any extended Σ^b_1-formula is logically equivalent to a Σ^b_1-formula.*[2]

Semantics for Formulae in $\mathcal{RL}$. We introduce a *quantitative* semantics for formulae of $\mathcal{RL}$, which is strongly inspired by that for **MQPA**. In particular, function symbols of $\mathcal{RL}$ as well as the predicate symbols "=" and "$\subseteq$" have a standard interpretation as relations over $\mathbb{S}$ in the canonical model $\mathscr{W} = (\mathbb{S}, \frown, \times)$, while, as we shall see, $\mathtt{Flip}(t)$ can be interpreted either in a standard way or following Definition 12.1.3.

[2] Actually, Ferreira proved this result for the theory Σ^b_1-NIA [82, pp. 148-149], but it clearly holds for $\mathcal{RL}$ as well.

Definition 13.2.5 (Semantics for Terms in $\mathcal{RL}$)**.** Given a set of term variables $\mathcal{G}$, an environment $\xi : \mathcal{G} \to \mathbb{S}$ is a mapping that assigns to each variable a string. Given a term t in $\mathcal{RL}$ and an environment ξ, the *interpretation of t in ξ* is the string $[\![t]\!]_\xi \in \mathbb{S}$ inductively defined as follows:

$$\begin{aligned} [\![\epsilon]\!]_\xi &:= \boldsymbol{\epsilon} & [\![x]\!]_\xi &:= \xi(x) \in \mathbb{S} \\ [\![\mathtt{0}]\!]_\xi &:= \mathbb{0} & [\![t \frown s]\!]_\xi &:= [\![t]\!]_\xi [\![s]\!]_\xi \\ [\![\mathtt{1}]\!]_\xi &:= \mathbb{1} & [\![t \times s]\!]_\xi &:= [\![t]\!]_\xi \times [\![s]\!]_\xi. \end{aligned}$$

As in Chapter 12, we extend the canonincal model $\mathscr{W}$ with the probability space $\mathscr{P}_{\mathscr{W}} = (\mathbb{0}, \sigma(\mathscr{C}), \mu_{\mathscr{C}})$, where $\sigma(\mathscr{C}) \subseteq \mathcal{P}(\mathbb{0})$ is the Borel σ-algebra generated by cylinders $\mathsf{C}^{\mathbb{b}}_\sigma = \{\eta \mid \eta(\sigma) = \mathbb{b}\}$, for $\mathbb{b} \in \mathbb{B}$, and such that $\mu_{\mathscr{C}}(\mathsf{C}^{\mathbb{b}}_\sigma) = \frac{1}{2}$. To formally define cylinders over $\mathbb{B}^{\mathbb{S}}$, we slightly modify Billingsley's notion of *cylinder of rank n*.

Definition 13.2.6 (Cylinder over S)**.** For any countable set S, finite $K \subset S$ and $H \subseteq \mathbb{B}^K$,

$$\mathsf{C}(H) = \{\eta \in \mathbb{B}^S \mid \eta|_K \in H\},$$

is a *cylinder* over S.

Then, $\mathscr{C}$ and $\sigma(\mathscr{C})$ are defined in the standard way and the probability measure over it is defined as follows:[3]

Definition 13.2.7 (Cylinder Measure)**.** For any countable set S, $K \subseteq S$ and $H \subseteq \mathbb{B}^K$ such that $\mathsf{C}(H) = \{\eta \in \mathbb{B}^S \mid \eta|_K \in H\}$,

$$\mu_{\mathscr{C}}(\mathsf{C}(H)) = \frac{|H|}{2^{|K|}}.$$

This is a measure over $\sigma(\mathscr{C})$.

Then, formulae of $\mathcal{RL}$ are interpreted as sets *measurable* sets.

Definition 13.2.8 (Semantics for Formulae in $\mathcal{RL}$)**.** Given a term t, a formula F, and an environment $\xi : \mathcal{G} \to \mathbb{S}$, where $\mathcal{G}$ is the set of term variables, the *interpretation of F under ξ* is the measurable set of sequences $[\![F]\!]_\xi \in \sigma(\mathscr{C})$ inductively defined as follows:

$$[\![\mathtt{Flip}(t)]\!]_\xi := \{\eta \mid \eta([\![t]\!]_\xi) = \mathbb{1}\}$$

$$[\![t = s]\!]_\xi := \begin{cases} \mathbb{0} & \text{if } [\![t]\!]_\xi = [\![s]\!]_\xi \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![t \subseteq s]\!]_\xi := \begin{cases} \mathbb{0} & \text{if } [\![t]\!]_\xi \subseteq [\![s]\!]_\xi \\ \emptyset & \text{otherwise} \end{cases}$$

$$\begin{aligned} [\![\neg G]\!]_\xi &:= \mathbb{0} - [\![G]\!]_\xi \\ [\![G \vee H]\!]_\xi &:= [\![G]\!]_\xi \cup [\![H]\!]_\xi \\ [\![G \wedge H]\!]_\xi &:= [\![G]\!]_\xi \cap [\![H]\!]_\xi \\ [\![G \to H]\!]_\xi &:= (\mathbb{0} - [\![G]\!]_\xi) \cup [\![H]\!]_\xi \\ [\![(\exists x)G]\!]_\xi &:= \bigcup_{i \in \mathbb{S}} [\![G]\!]_{\xi\{x \leftarrow i\}} \\ [\![(\forall x)G]\!]_\xi &:= \bigcap_{i \in \mathbb{S}} [\![G]\!]_{\xi\{x \leftarrow i\}}. \end{aligned}$$

[3] For further details, see [68].

As anticipated, this semantics is well-defined. Indeed, the sets $[\![\texttt{Flip}(t)]\!]_\xi$, $[\![t = s]\!]_\xi$ and $[\![t \subseteq s]\!]_\xi$ are measurable and measurability is preserved by all the logical operators.

A interpretation of the language $\mathcal{RL}$ in the usual sense is given due to an environment ξ plus the choice of an interpretation η for $\texttt{Flip}(x)$.

Definition 13.2.9 (Standard Semantics for Formulae in $\mathcal{RL}$)**.** Given a $\mathcal{RL}$-formula F, and an interpretation $\rho = (\xi, \eta^{\texttt{FLIP}})$, where $\xi : \mathcal{G} \to \mathbb{S}$ and $\eta^{\texttt{FLIP}} \subseteq \mathbb{O}$, the *interpretation of F in ρ* $[\![F]\!]_\rho$, is inductively defined as follows:

$$[\![\texttt{Flip}(t)]\!]_\rho := \begin{cases} 1 & \text{if } \eta^{\texttt{FLIP}}([\![t]\!]_\rho) = \mathbb{1} \\ 0 & \text{otherwise} \end{cases}$$

$$[\![t = s]\!]_\rho := \begin{cases} 1 & \text{if } [\![t]\!]_\rho = [\![s]\!]_\rho \\ 0 & \text{otherwise.} \end{cases}$$

$$[\![t \subseteq s]\!]_\rho := \begin{cases} 1 & \text{if } [\![t]\!]_\rho \subseteq [\![s]\!]_\rho \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} [\![\neg G]\!]_\rho &:= 1 - [\![G]\!]_\rho \\ [\![G \wedge H]\!]_\rho &:= min\{[\![G]\!]_\rho, [\![H]\!]_\rho\} \\ [\![G \vee H]\!]_\rho &:= max\{[\![G]\!]_\rho, [\![H]\!]_\rho\} \\ [\![G \to H]\!]_\rho &:= max\{(1 - [\![G]\!]_\rho), [\![H]\!]_\rho\} \\ [\![(\forall x)G]\!]_\rho &:= min\{[\![G]\!]_{\rho\{x \leftarrow \sigma\}} \mid \sigma \in \mathbb{S}\} \\ [\![(\exists x)G]\!]_\rho &:= max\{[\![G]\!]_{\rho\{x \leftarrow \sigma\}} \mid \sigma \in \mathbb{S}\}. \end{aligned}$$

Notation 13.2.5. For readability's sake, we abbreviate $[\![\cdot]\!]_\rho$ simply as $[\![\cdot]\!]_\eta$, and $[\![\cdot]\!]_\xi$ as $[\![\cdot]\!]$.

Observe that *quantitative* and *qualitative* semantics for $\mathcal{RL}$ are mutually related, as can be proved by induction on the structure of formulae [68].

Proposition 13.2.2. *For any formula F in $\mathcal{RL}$, environment ξ, function $\eta \in \mathbb{O}$ and $\rho = (\eta, \xi)$,*

$$[\![F]\!]_{\xi,\eta} = 1 \ \textit{iff} \ \ \eta \in [\![F]\!]_\rho.$$

13.3 $\mathbf{RS}^1_2$ characterizes $\mathcal{POR}$

As said, our proof follows a so-to-say standard path [34, 82]. The first step consists in showing that functions in $\mathcal{POR}$ are precisely those which are Σ^b_1-representable in $\mathbf{RS}^1_2$. To do so, we extend Buss' representability conditions by adding a constraint to link the quantitative semantics of formulae in $\mathbf{RS}^1_2$ with the additional functional parameter η of oracle recursive functions.

Definition 13.3.1 (Σ^b_1-Representability)**.** A function $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$ is Σ^b_1-*representable in* $\mathbf{RS}^1_2$ if there is a Σ^b_1-formula $F(\vec{x}, y)$ of $\mathcal{RL}$ such that:

1. $\mathbf{RS}_2^1 \vdash (\forall \vec{x})(\exists y)F(\vec{x}, y)$
2. $\mathbf{RS}_2^1 \vdash (\forall \vec{x})(\forall y)(\forall z)\big(F(\vec{x}, y) \wedge F(\vec{x}, z) \to y = z\big)$
3. for all $\sigma_1, \ldots, \sigma_j, \tau \in \mathbb{S}$ and $\eta \in \mathbb{O}$,

$$f(\sigma_1, \ldots, \sigma_j, \eta) = \tau \quad \text{iff} \quad \eta \in [\![F(\overline{\sigma_1}, \ldots, \overline{\sigma_j}, \overline{\tau})]\!].$$

We recall that the language $\mathcal{RL}$ allows us to associate the formula F with both a *qualitative* – namely, when dealing with 1. and 2. – and a *quantitative* interpretation – namely, in 3. Then, in Section 13.3.1, we prove the following theorem.

Theorem 13.3.1 ($\mathcal{POR}$ and $\mathbf{RS}_2^1$)**.** *For any function $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$, f is Σ_1^b-representable in $\boldsymbol{RS}_2^1$ when $f \in \mathcal{POR}$.*

In particular, that any function in $\mathcal{POR}$ is Σ_1^b-representable in $\mathbf{RS}_2^1$ is proved in Section 13.3.1 by a straightforward induction on the structure of probabilistic oracle functions. The other direction is established in Section 13.3.2 by a realizability argument very close to the one offered in [46].

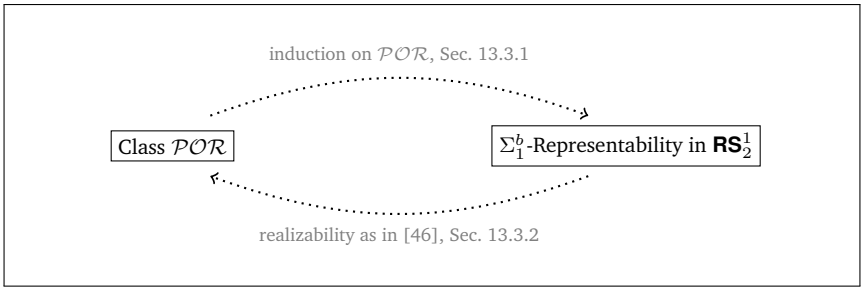


Figure 13.3: Relating $\mathcal{POR}$ and $\mathbf{RS}_2^1$

13.3.1 Functions in $\mathcal{POR}$ are Σ_1^b-Representable in $\mathbf{RS}_2^1$

We prove that any function in $\mathcal{POR}$ is Σ_1^b-representable in $\mathbf{RS}_2^1$ by constructing the desired formula by induction on the structure of oracle functions. Preliminarily notice that, for example, the formula $(\forall \vec{x})(\exists y)G(\vec{x}, y)$ occurring in condition 1. is *not* in Σ_1^b, since its existential quantifier is not bounded. Hence, in order to prove the inductive steps of Theorem 13.3.2 – namely, composition and bounded recursion on notation – we need to adapt Parikh's theorem [166] to $\mathbf{RS}_2^1$.[4]

[4]The theorem is usually presented in the context of Buss' bounded theories, as stating that given a bounded formula F in $\mathcal{L}_{\mathbb{N}}$ such that $\mathbf{S}_2^1 \vdash (\forall \vec{x})(\exists y)F$, then there is a term $t(\vec{x})$ such that also $\mathbf{S}_2^1 \vdash (\forall \vec{x})(\exists y \leq t(\vec{x}))F(\vec{x}, y)$ [34, 35]. Furthermore, due to [84], Buss' syntactic proof can be adapted to Σ_1^b-NIA in a natural way. The same result holds for $\mathbf{RS}_2^1$, which does not contain any specific rule concerning $\texttt{Flip}(\cdot)$.

Proposition 13.3.1 ("Parikh" [166]). *Let $F(\vec{x}, y)$ be a bounded formula in $\mathcal{RL}$ such that $\boldsymbol{RS}^1_2 \vdash (\forall \vec{x})(\exists y) F(\vec{x}, y)$. Then, there is a term t such that,*

$$\boldsymbol{RS}^1_2 \vdash (\forall \vec{x})(\exists y \preceq t(\vec{x}))F(\vec{x}, y).$$

Theorem 13.3.2. *Every $f \in \mathcal{POR}$ is Σ^b_1-representable in $\boldsymbol{RS}^1_2$.*

Proof Sketch. The proof is by induction on the structure of functions in $\mathcal{POR}$.[5]

Base Case. Each basic function is Σ^b_1-representable in $\mathbf{RS}^1_2$. There are five possible sub-cases:

- *Empty (String) Function.* $f = E$ is Σ^b_1-represented in $\mathbf{RS}^1_2$ by the formula:

$$F_E(x, y) : x = x \wedge y = \epsilon.$$

  1. Existence is proved considering $y = \epsilon$. For the reflexivity of identity both $\mathbf{RS}^1_2 \vdash x = x$ and $\mathbf{RS}^1_2 \vdash \epsilon = \epsilon$ hold. So, by rules for conjunction, we obtain $\mathbf{RS}^1_2 \vdash x = x \wedge \epsilon = \epsilon$, and conclude:

$$\mathbf{RS}^1_2 \vdash (\forall x)(\exists y)(x = x \wedge y = \epsilon).$$

  2. Uniqueness is proved assuming $\mathbf{RS}^1_2 \vdash x = x \wedge z = \epsilon$. By rules for conjunction, in particular $\mathbf{RS}^1_2 \vdash z = \epsilon$, and since $\mathbf{RS}^1_2 \vdash y = \epsilon$, by the transitivity of identity, we conclude

$$\mathbf{RS}^1_2 \vdash y = z.$$

  3. Assume $E(\sigma, \eta^*) = \tau$. If $\tau = \boldsymbol{\epsilon}$, then:

$$\begin{aligned} [\![\overline{\sigma} = \overline{\sigma} \wedge \overline{\tau} = \epsilon]\!] &= [\![\overline{\sigma} = \overline{\sigma}]\!] \cap [\![\overline{\tau} = \epsilon]\!] \\ &= \mathbb{0} \cap \mathbb{0} \\ &= \mathbb{0}. \end{aligned}$$

     So, in this case, for any η^*, $\eta^* \in [\![\overline{\sigma} = \overline{\sigma} \wedge \overline{\tau} = \epsilon]\!]$, as clearly $\eta^* \in \mathbb{0}$. If $\tau \neq \boldsymbol{\epsilon}$, then

$$\begin{aligned} [\![\overline{\sigma} = \overline{\sigma} \wedge \overline{\tau} = \epsilon]\!] &= [\![\overline{\sigma} = \overline{\sigma}]\!] \cap [\![\overline{\tau} = \epsilon]\!] \\ &= \mathbb{0} \cap \emptyset \\ &= \emptyset. \end{aligned}$$

     So, for any η^*, $\eta^* \notin [\![\overline{\sigma} = \overline{\sigma} \vee \overline{\tau} = \epsilon]\!]$, as clearly $\eta^* \notin \emptyset$.

- *Projection (String) Function.* $f = P^n_i$, for $1 \leq i \leq n$, is Σ^b_1-represented in $\mathbf{RS}^1_2$ by the formula:

$$F_{P^n_i}(x, y) : \bigwedge_{j \in J} (x_j = x_j) \wedge y = x_i,$$

  where $J = \{1, \ldots, n\} \setminus \{i\}$.

[5] For further details, see [6].

- *Word-Successor Function.* $f = S_b$ is Σ^b_1-represented in $\mathbf{RS}^1_2$ by the formula:

$$F_{S_b}(x, y) : y = x\mathtt{b}$$

where $\mathtt{b} = \mathtt{0}$ if $b = 0$ and $\mathtt{b} = \mathtt{1}$ if $b = 1$.

- *Conditional (String) Function.* $f = C$ is Σ^b_1-represented in $\mathbf{RS}^1_2$ by the formula:

$$F_C(x, v, z_0, z_1, y) : (x = \epsilon \wedge y = v) \vee (\exists x' \preceq x)(x = x'\mathtt{0} \wedge y = z_0) \\ \vee (\exists x' \preceq x)(x = x'\mathtt{1} \wedge y = z_1).$$

- *Query (String) Function.* $f = Q$ is Σ^b_1-represented in $\mathbf{RS}^1_2$ by the formula:

$$F_Q(x, y) : (\mathtt{Flip}(x) \wedge y = \mathtt{1}) \vee (\neg\mathtt{Flip}(x) \wedge y = \mathtt{0}).$$

Notice that, in this case, the proof crucially relies on the fact that oracle functions invoke *exactly one* oracle.

1. Existence is proved by cases.[6] Since our underlying logic is classical $\mathbf{RS}^1_2 \vdash \mathtt{Flip}(x) \vee \neg\mathtt{Flip}(x)$ holds. When $\mathbf{RS}^1_2 \vdash \mathtt{Flip}(x)$, let $y = \mathtt{1}$. By the reflexivity of identity, $\mathbf{RS}^1_2 \vdash \mathtt{1} = \mathtt{1}$ holds, so also $\mathbf{RS}^1_2 \vdash \mathtt{Flip}(x) \wedge \mathtt{1} = \mathtt{1}$. By rules for disjunction, we conclude $\mathbf{RS}^1_2 \vdash (\mathtt{Flip}(x) \wedge \mathtt{1} = \mathtt{1}) \vee (\neg\mathtt{Flip}(x) \wedge \mathtt{1} = \mathtt{0})$ and so,

$$\mathbf{RS}^1_2 \vdash (\exists y)\big((\mathtt{Flip}(x) \wedge y = \mathtt{1}) \vee (\neg\mathtt{Flip}(x) \wedge y = \mathtt{0})\big).$$

Then, when $\mathbf{RS}^1_2 \vdash \neg\mathtt{Flip}(x)$, let $y = \mathtt{0}$. By the reflexivity of identity $\mathbf{RS}^1_2 \vdash \mathtt{0} = \mathtt{0}$ holds. Thus, by the rules for conjunction, $\mathbf{RS}^1_2 \vdash \neg\mathtt{Flip}(x) \wedge \mathtt{0} = \mathtt{0}$ and for disjunction, we conclude $\mathbf{RS}^1_2 \vdash (\mathtt{Flip}(x) \wedge \mathtt{0} = \mathtt{1}) \vee (\neg\mathtt{Flip}(x) \wedge \mathtt{0} = \mathtt{0})$ and so,

$$\mathbf{RS}^1_2 \vdash (\exists y)\big((\mathtt{Flip}(x) \wedge y = \mathtt{1}) \vee (\neg\mathtt{Flip}(x) \wedge y = \mathtt{0})\big).$$

[6] More formally, the proof (basing on an extension of standard **G3K** [155, 156]) is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{Flip}(x) \vdash (\exists y)F_Q(z,y), \mathtt{Flip}(z)}\,Ax}{\vdash (\exists y)F_Q(z,y), \mathtt{Flip}(z), \neg\mathtt{Flip}(z)}\,\neg R \qquad \mathcal{D}_{\exists Q} \vdash (\exists y)F_Q(z,y), \mathtt{Flip}(z), \mathtt{1} = \mathtt{0}}{(\exists y)F_Q(z,y), \mathtt{Flip}(z), \neg\mathtt{Flip}(z) \wedge \mathtt{1} = \mathtt{0}}\,\wedge R \qquad \dfrac{\dfrac{}{\mathtt{1} = \mathtt{1} \vdash \ldots, \mathtt{1} = \mathtt{1}}\,Ax}{\vdash \ldots, \mathtt{1} = \mathtt{1}}\,Ref}{\vdash (\exists y)F_Q(z,y), (\mathtt{Flip}(z) \wedge \mathtt{1} = \mathtt{1}), (\neg\mathtt{Flip}(z) \wedge \mathtt{1} = \mathtt{0})}\,\wedge R}{\vdash (\exists y)F_Q(z,y), (\mathtt{Flip}(z) \wedge \mathtt{1} = \mathtt{1}) \vee (\neg\mathtt{Flip}(z) \wedge \mathtt{1} = \mathtt{0})}\,\vee R}{\vdash (\exists y)((\mathtt{Flip}(z) \wedge y = \mathtt{1}) \vee (\neg\mathtt{Flip}(z) \wedge y = \mathtt{0}))}\,\exists R}{\vdash (\forall x)(\exists y)((\mathtt{Flip}(x) \wedge y = \mathtt{1}) \vee (\neg\mathtt{Flip}(x) \wedge y = \mathtt{0}))}\,\forall R$$

where $\mathcal{D}_{\exists Q}$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{Flip}(z) \vdash \ldots, \mathtt{Flip}(z)}\,Ax}{\vdash \ldots, \mathtt{Flip}(z), \neg\mathtt{Flip}(z)}\,\neg R \qquad \dfrac{\dfrac{}{\mathtt{0} = \mathtt{0} \vdash \ldots, \mathtt{Flip}(z), \mathtt{0} = \mathtt{0}}\,Ax}{\vdash \ldots, \mathtt{Flip}(z), \mathtt{0} = \mathtt{0}}\,Ref}{\vdash \ldots, \mathtt{Flip}(z), \mathtt{Flip}(z) \wedge \mathtt{0} = \mathtt{1}, \neg\mathtt{Flip}(z) \wedge \mathtt{0} = \mathtt{0}}\,\wedge R}{\vdash (\exists y)F_Q(z,y), \mathtt{1} = \mathtt{0}, \mathtt{Flip}(z), F_Q(z,\mathtt{0})}\,\vee R}{\vdash (\exists y)F_Q(z,y), \mathtt{Flip}(z), \mathtt{1} = \mathtt{0}}\,\exists R$$

For further details, see [6].

2. Uniqueness is established relying on the transitivity of identity.
3. Finally, it is shown that for every $\sigma, \tau \in \mathbb{S}$ and $\eta^* \in \mathbb{O}$, $Q(\sigma, \eta^*) = \tau$ when $\eta^* \in [\![F_Q(\overline{\sigma}, \overline{\tau})]\!]$. Assume $Q(\sigma, \eta^*) = \mathbb{1}$, which is $\eta^*(\sigma) = \mathbb{1}$,

$$\begin{aligned}
[\![F_Q(\overline{\sigma}, \overline{\tau})]\!] &= [\![\mathtt{Flip}(\overline{\sigma}) \wedge \overline{\tau} = \mathtt{1}]\!] \cup [\![\neg\mathtt{Flip}(\overline{\sigma}) \wedge \overline{\tau} = \mathtt{0}]\!] \\
&= ([\![\mathtt{Flip}(\overline{\sigma})]\!] \cap [\![\mathtt{1} = \mathtt{1}]\!]) \cup ([\![\neg\mathtt{Flip}(\overline{\sigma})]\!] \cap [\![\mathtt{1} = \mathtt{0}]\!]) \\
&= ([\![\mathtt{Flip}(\overline{\sigma})]\!] \cap \mathbb{O}) \cup ([\![\neg\mathtt{Flip}(\overline{\sigma})]\!] \cap \emptyset) \\
&= [\![\mathtt{Flip}(\overline{\sigma})]\!] \\
&= \{\eta \mid \eta(\sigma) = \mathbb{1}\}.
\end{aligned}$$

Clearly, $\eta^* \in [\![(\mathtt{Flip}(\overline{\sigma}) \wedge \overline{\tau} = \mathtt{1}) \vee (\neg\mathtt{Flip}(\overline{\sigma}) \wedge \overline{\tau} = \mathtt{0})]\!]$.
The case $Q(\sigma, \eta^*) = \mathbb{0}$ and the opposite direction are proved in a similar way.

Inductive Case. If f is defined by composition or bounded recursion from Σ^b_1-representable functions, then f is Σ^b_1-representable in $\mathbf{RS}^1_2$:

- *Composition.* Assume that f is defined by composition from functions $g, h_1, \ldots, h_k$ so that

$$f(\vec{x}, \eta) = g(h_1(\vec{x}, \eta), \ldots, h_k(\vec{x}, \eta), \eta)$$

and that $g, h_1, \ldots, h_k$ are represented in $\mathbf{RS}^1_2$ by the Σ^b_1-formulae $F_g, F_{h_1}, \ldots, F_{h_k}$, respectively. By Proposition 13.3.1, there exist suitable terms $t_g, t_{h_1}, \ldots, t_{h_k}$ such that condition 1. of Definition 13.3.1 can be strengthened to $\mathbf{RS}^1_2 \vdash (\forall \vec{x})(\exists y \preceq t_i) F_i(\vec{x}, y)$ for each $i \in \{g, h_1, \ldots, h_k\}$. We conclude that $f(\vec{x}, \eta)$ is Σ^b_1-represented in $\mathbf{RS}^1_2$ by the following formula:

$$\begin{aligned}
F_f(x, y) : (\exists z_1 \preceq t_{h_1}(\vec{x})) \ldots (\exists z_k \preceq t_{h_k}(\vec{x})) \big(& F_{h_1}(\vec{x}, z_1) \wedge \ldots F_{h_k}(\vec{x}, z_k) \\
& \wedge F_g(z_1, ..., z_k, y) \big).
\end{aligned}$$

Indeed, by IH, $F_g, F_{h_1}, \ldots, F_{h_k}$ are Σ^b_1-formulae. Then, also F_f is in Σ^b_1. Conditions 1.-3. are proved to hold by slightly modifying standard proofs.

- *Bounded Recursion.* Assume that f is defined by bounded recursion from g, h_0, and h_1 so that:

$$\begin{aligned}
f(\vec{x}, \boldsymbol{\epsilon}, \eta) &= g(\vec{x}, \eta) \\
f(\vec{x}, y\mathbb{b}, \eta) &= h_i(\vec{x}, y, f(\vec{x}, y, \eta), \eta)|_{t(\vec{x}, y)},
\end{aligned}$$

where $i \in \{0, 1\}$ and $\mathbb{b} = \mathbb{0}$ when $i = 0$ while and $\mathbb{b} = \mathbb{1}$ when $i = 1$. Let g, h_0, h_1 be represented in $\mathbf{RS}^1_2$ by, respectively, the Σ^b_1-formulae F_g, F_{h_0}, and F_{h_1}. Moreover, by Proposition 13.3.1, there exist suitable terms t_g, t_{h_0}, and t_{h_1} such that condition 1. of Definition 13.3.1 can be

strengthened to its "bounded version". Then, it can be proved that $f(\vec{x}, y)$ is Σ_1^b-represented in $\mathbf{RS}_2^1$ by the formula below:

$$\begin{aligned} F_f(x,y) : (\exists v \preceq t_g(\vec{x})t_f(\vec{x})(y \times t(\vec{x},y)t(\vec{x},y)\mathtt{11}))\big(&F_{lh}(v, \mathtt{1} \times y\mathtt{1}) \\ &\wedge (\exists z \preceq t_g(\vec{x}))(F_{eval}(v, \epsilon, z) \wedge F_g(\vec{x}, z)) \\ &\wedge (\forall u \subset y)(\exists z)\big(\tilde{z} \preceq t(\vec{x},y)\big)\big(F_{eval}(v, \mathtt{1} \times u, z) \wedge F_{eval}(v, \mathtt{1} \times u\mathtt{1}, \tilde{z}) \\ &\wedge (u\mathtt{0} \subseteq y \rightarrow (\exists z_0 \preceq t_{h_0}(\vec{x}, u, z))(F_{h_0}(\vec{x}, u, z, z_0) \wedge z_0|_{t(\vec{x},u)} = \tilde{z})) \\ &\wedge (u\mathtt{1} \subseteq y \rightarrow (\exists z_1 \preceq t_{h_1}(\vec{x}, u, z))(F_{h_1}(\vec{x}, u, z, z_1) \wedge z_1|_{t(\vec{x},u)} = \tilde{z})))\big), \end{aligned}$$

where F_{lh} and F_{eval} are Σ_1^b-formulae defined as in [83]. Intuitively, $F_{lh}(x, y)$ states that the number of 1s in the encoding of x is yy, while $F_{eval}(x, y, z)$ is a "decoding" formula (strongly resembling Gödel's β-formula), expressing that the "bit" encoded in x as its y-th bit is z. Moreover $x \subset y$ is an abbreviation for $x \subseteq y \wedge x \neq y$. Then, this formula F_f satisfies all the requirements to Σ_1^b-represent in $\mathbf{RS}_2^1$ the function f, obtained by bounded recursion from $g, h_0,$ and h_1. In particular, conditions 1. and 2. concerning existence and uniqueness, have already been proved to hold by Ferreira [83]. Furthermore, F_f expresses that, given the desired encoding sequence v: (i.) the ϵ-th bit of v is (the encoding of) z' such that $F_g(\vec{x}, z')$ holds, where (for IH) F_g is the Σ_1^b-formula representing the function g, and (ii.) given that for each $u \subset y$, z denotes the " bit" encoded in v at position $1 \times u$ and, similarly, $\tilde{z}$ is the next "bit", encoded in v at position $\mathtt{1} \times u\mathtt{1}$, then if $u\mathtt{b} \subseteq y$ (that is, if we are considering the initial substring of y the last bit of which corresponds to $\mathtt{b}$), then there is a z_b such that $F_{h_b}(\vec{x}, y, z, z_b)$, where F_{h_b} Σ_1^b-represents the function f_{h_b} and the truncation of z_b at $t(\vec{x}, u)$ is precisely $\tilde{z}$, with $b = 0$ when $\mathtt{b} = \mathtt{0}$ and $b = 1$ when $\mathtt{b} = \mathtt{1}$.[7]

□

13.3.2 The functions which are Σ_1^b-Representable in $\mathbf{RS}_2^1$ are in $\mathcal{POR}$

Here, we consider the opposite direction. Our proof is obtained by adapting that by Cook and Urquhart for IPV^ω [46]. It passes through a *realizability interpretation* of the intuitionistic version of $\mathbf{RS}_2^1$, called $\mathbf{IRS}_2^1$ and is structured as follows:

1. First, we define $\mathcal{POR}^\lambda$, a basic equational theory for a simply typed λ-calculus endowed with primitives corresponding to functions of $\mathcal{POR}$.

[7] Otherwise said, if $u\mathtt{0} \subseteq y$, there is a z_0 such that the Σ_1^b-formula $F_{h_0}(\vec{x}, u, z, z_0)$ represents the function h_0 and, in this case, $\tilde{z}$ corresponds to the truncation of z_0 at $t(\vec{x}, u)$, that is the "bit" encoded by v at the position $\mathtt{1} \times u\mathtt{1}$ (i.e. corresponding to $u\mathtt{0} \subseteq y$) is precisely such $\tilde{z}$. Equally, if $u\mathtt{1} \subseteq y$, there is a z_0 such that the Σ_1^b-formula $F_{h_1}(\vec{x}, u, z, z_1)$ represents now the function h_1 and $\tilde{z}$ corresponds to the truncation of z_1 at $t(\vec{x}, u)$, that is the "bit" encoded by v at position $\mathtt{1} \times u\mathtt{1}$ (i.e. corresponding to $u\mathtt{1} \subseteq y$) is precisely such $\tilde{z}$. For further details, see [6].

2. Second, we introduce a first-order *intuitionistic* theory $\mathcal{IPOR}^\lambda$, which extends $\mathcal{POR}^\lambda$ with the usual predicate calculus as well as an $\mathbf{NP}$-induction schema. It is shown that $\mathcal{IPOR}^\lambda$ is strong enough to prove all theorems of $\mathbf{IRS}^1_2$, the intuitionistic version of $\mathbf{RS}^1_2$.
3. Then, we develop a realizability interpretation of $\mathcal{IPOR}^\lambda$ (inside itself), showing that from any derivation of $(\forall x)(\exists y)F(x,y)$ (where F is a Σ^b_0-formula) one can extract a λ-term t of $\mathcal{POR}^\lambda$, such that $(\forall x)F(x,\mathrm{t}x)$ is provable in $\mathcal{IPOR}^\lambda$. From this we deduce that every function which is Σ^b_1-representable in $\mathbf{IRS}^1_2$ is in $\mathcal{POR}$.
4. Finally, we extend this result to classical $\mathbf{RS}^1_2$ showing that any Σ^b_1-formula provable in $\mathcal{IPOR}^\lambda$ + Excluded Middle ($\mathbf{EM}$, for short) is already provable in $\mathcal{IPOR}^\lambda$.

The System $\mathcal{POR}^\lambda$

We define an equational theory for a simply typed λ-calculus augmented with primitives for functions of $\mathcal{POR}$. Actually, these do not exactly correspond to the ones of $\mathcal{POR}$, although the resulting function algebra is proved equivalent.[8]

The Syntax of $\mathcal{POR}^\lambda$. We start by considering the syntax of $\mathcal{POR}^\lambda$.

Definition 13.3.2 (Types of $\mathcal{POR}^\lambda$)**.** *Types of $\mathcal{POR}^\lambda$* are defined by the grammar below:

$$\sigma := s \;\Big|\; \sigma \Rightarrow \sigma.$$

Definition 13.3.3 (Terms of $\mathcal{POR}^\lambda$)**.** *Terms of $\mathcal{POR}^\lambda$* are standard, simply typed λ-terms plus the constants below:

$$\begin{aligned}
0, 1, \epsilon &: s \\
\circ &: s \Rightarrow s \Rightarrow s \\
\mathsf{Tail} &: s \Rightarrow s \\
\mathsf{Trunc} &: s \Rightarrow s \Rightarrow s \\
\mathsf{Cond} &: s \Rightarrow s \Rightarrow s \Rightarrow s \Rightarrow s \\
\mathsf{Flipcoin} &: s \Rightarrow s \\
\mathsf{Red} &: s \Rightarrow (s \Rightarrow s \Rightarrow s) \Rightarrow (s \Rightarrow s \Rightarrow s) \Rightarrow (s \Rightarrow s) \Rightarrow s \Rightarrow s.
\end{aligned}$$

Intuitively, $\mathsf{Tail}(x)$ computes the string obtained by deleting the first digit of x; $\mathsf{Trunc}(x,y)$ computes the string obtained by truncating x at the length of y; $\mathsf{Cond}(x,y,z,w)$ computes the function that yields y when $x = \epsilon$, z when $x = x'\mathbb{0}$, and w when $x = x'\mathbb{1}$; $\mathsf{Flipcoin}(x)$ indicates a random $\mathbb{0}/\mathbb{1}$ generator; Rec is the operator for bounded recursion on notation.

[8] Our choice follows the principle that the defining equations for the functions different from the recursion operator should not depend on it.

Notation 13.3.1. We usually abbreviate $x \circ y$ as xy. Moreover, for readability's sake, being T any constant $\mathsf{Tail}, \mathsf{Trunc}, \mathsf{Cond}, \mathsf{Flipcoin}, \mathsf{Rec}$ of arity n, we indicate $\mathsf{T}\mathsf{u}_1, \ldots, \mathsf{u}_n$ as $\mathsf{T}(\mathsf{u}_1, \ldots, \mathsf{u}_n)$.

$\mathcal{POR}^\lambda$ is reminiscent of PV^ω by Cook and Urquhart [46] without the induction rule (R5) that we do not need. The main difference being the constant $\mathsf{Flipcoin}$, which, as said, intuitively denotes a function which randomly generates either $\mathbb{0}$ or $\mathbb{1}$ when reads a string.[9]

Remark 13.3.1. *In the following, we often define terms implicitly using bounded recursion on notation. Otherwise said, we define new terms, say* $\mathsf{F} : s \Rightarrow \ldots \Rightarrow s$*, by equations of the form:*

$$\begin{aligned} \mathsf{F}\vec{x}\epsilon &:= \mathsf{G}\vec{x} \\ \mathsf{F}\vec{x}(y\mathsf{0}) &:= \mathsf{H}_0\vec{x}y(\mathsf{F}\vec{x}y) \\ \mathsf{F}\vec{x}(y\mathsf{1}) &:= \mathsf{H}_1\vec{x}y(\mathsf{F}\vec{x}y), \end{aligned}$$

where $\mathsf{G}, \mathsf{H}_0, \mathsf{H}_1$ *are already-defined terms, and the second and third equations satisfy a length bound given by some term* K *(which is usually* $\lambda\vec{x}\lambda y.\mathsf{0}$*). The term* F *can be explicitly defined as follows:*

$$\mathsf{F} := \lambda\vec{x}.\lambda y.\mathsf{Rec}(\mathsf{G}\vec{x}, \lambda yy'.\mathsf{H}_0\vec{x}yy', \lambda yy'.\mathsf{H}_1\vec{x}yy', \mathsf{K}\vec{x}, y).$$

We also introduce the following abbreviations for composed functions:

- $\mathsf{B}(x) := \mathsf{Cond}(x, \epsilon, \mathsf{0}, \mathsf{1})$ denotes the function that computes the last digit of x, i.e. coerces x to a Boolean value
- $\mathsf{BNeg}(x) := \mathsf{Cond}(x, \epsilon, \mathsf{1}, \mathsf{0})$ denotes the function that computes the Boolean negation of $\mathsf{B}(x)$
- $\mathsf{BOr}(x, y) := \mathsf{Cond}(\mathsf{B}(x), \mathsf{B}(y), \mathsf{B}(y), \mathsf{1})$ denotes the function that coerces x and y to Booleans and then performs the OR operation
- $\mathsf{BAnd}(x, y) := \mathsf{Cond}(\mathsf{B}(x), \epsilon, \mathsf{0}, \mathsf{B}(y))$ denotes the function that coerces x and y to Booleans and then performs the AND operation
- $\mathsf{Eps}(x) := \mathsf{Cond}(x, \mathsf{1}, \mathsf{0}, \mathsf{0})$ denotes the characteristic function of the predicate "$x = \epsilon$"
- $\mathsf{Bool}(x) := \mathsf{BAnd}(\mathsf{Eps}(\mathsf{Tail}(x)), \mathsf{BNeg}(\mathsf{Eps}(x)))$ denotes the characteristic function of the predicate "$x = \mathsf{0} \vee x = \mathsf{1}$"
- $\mathsf{Zero}(x) := \mathsf{Cond}(\mathsf{Bool}(x), \mathsf{0}, \mathsf{Cond}(x, \mathsf{0}, \mathsf{0}, \mathsf{1}), \mathsf{0})$ denotes the characteristic function of the predicate "$x = \mathsf{0}$"
- $\mathsf{Conc}(x, y)$ denotes the concatenation function defined by the equations below:

$$\mathsf{Conc}(x, \epsilon) := x \qquad \mathsf{Conc}(x, y\mathsf{b}) := \mathsf{Conc}(x, y)\mathsf{b},$$

with $\mathsf{b} \in \{\mathsf{0}, \mathsf{1}\}$.

[9] These interpretations will be made clear by Definition 13.3.6 below.

- $\mathsf{Eq}(x, y)$ denotes the characteristic function of the predicate "$x = y$" and is defined by double recursion by the equation below:

$$\mathsf{Eq}(\epsilon, \epsilon) := 1 \qquad \mathsf{Eq}(\epsilon, y\mathsf{b}) := 0$$
$$\mathsf{Eq}(x\mathsf{b}, \epsilon) = \mathsf{Eq}(x\mathsf{0}, y\mathsf{1}) = \mathsf{Eq}(x\mathsf{1}, y\mathsf{0}) := 0 \qquad \mathsf{Eq}(x\mathsf{b}, y\mathsf{b}) := \mathsf{Eq}(x, y),$$

  with $\mathsf{b} \in \{\mathsf{0}, \mathsf{1}\}$

- $\mathsf{Times}(x, y)$ denotes the function for self-concatenation, $x, y \mapsto x \times y$, and is defined by the equations below:

$$\mathsf{Times}(x, \epsilon) := \epsilon \qquad \mathsf{Times}(x, y\mathsf{b}) := \mathsf{Conc}(\mathsf{Times}(x, y), x),$$

  with $\mathsf{b} \in \{\mathsf{0}, \mathsf{1}\}$.

- $\mathsf{Sub}(x, y)$ denotes the initial-substring functions, $x, y \mapsto S(x, y)$, and is defined by bounded recursion as follows:

$$\mathsf{Sub}(x, \epsilon) := \mathsf{Eps}(x) \qquad \mathsf{Sub}(x, y\mathsf{b}) := \mathsf{BOr}(\mathsf{Sub}(x, y), \mathsf{Eq}(x, y\mathsf{b})),$$

  with $\mathsf{b} \in \{\mathsf{0}, \mathsf{1}\}$.

Let us now define formulae of $\mathcal{POR}^\lambda$.

Definition 13.3.4 (Formulae of $\mathcal{POR}^\lambda$)**.** *Formulae of* $\mathcal{POR}^\lambda$ are equations $\mathsf{t} = \mathsf{u}$, where t and u are terms of type s.

The Theory $\mathcal{POR}^\lambda$. We now introduce the theory $\mathcal{POR}^\lambda$.

Definition 13.3.5 (Theory $\mathcal{POR}^\lambda$)**.** Axioms of $\mathcal{POR}^\lambda$ are the following ones:

- Defining equations for the constants of $\mathcal{POR}^\lambda$:

$$\epsilon x = x\epsilon = x \qquad x(y\mathsf{b}) = (xy)\mathsf{b}$$

$$\mathsf{Tail}(\epsilon) = \epsilon \qquad \mathsf{Tail}(x\mathsf{b}) = x$$

$$\mathsf{Trunc}(x, \epsilon) = \mathsf{Trunc}(\epsilon, x) = \epsilon$$
$$\mathsf{Trunc}(x\mathsf{b}, y\mathsf{0}) = \mathsf{Trunc}(x\mathsf{b}, y\mathsf{1}) = \mathsf{Trunc}(x, y)\mathsf{b}$$

$$\mathsf{Cond}(\epsilon, y, z, w) = y \qquad \mathsf{Cond}(x\mathsf{0}, y, z, w) = z \qquad \mathsf{Cond}(x\mathsf{1}, y, z, w) = w$$

$$\mathsf{Bool}(\mathsf{Flipcoin}(x)) = \mathsf{1}$$

$$\mathsf{Rec}(x, h_0, h_1, k, \epsilon) = x$$
$$\mathsf{Rec}(x, h_0, h_1, k, y\mathsf{b}) = \mathsf{Trunc}(h_b y(\mathsf{Rec}(x, h_0, h_1, k, y)), ky),$$

  where $\mathsf{b} \in \{\mathsf{0}, \mathsf{1}\}$ and $b \in \{0, 1\}$.[10]

[10] When if $\mathsf{b} = \mathsf{0}$, then $b = 0$ and $\mathsf{b} = \mathsf{1}$, then $b = 1$.

- The (β)- and (ν)-axioms:

$$\mathsf{C}\big[(\lambda x.\mathsf{t})\mathsf{u}\big] = \mathsf{C}\big[\mathsf{t}\{\mathsf{u}/x\}\big] \qquad (\beta)$$
$$\mathsf{C}\big[\lambda x.\mathsf{t}x\big] = \mathsf{C}\big[\mathsf{t}\big]. \qquad (\nu)$$

  where $\mathsf{C}\big[\,\cdot\,\big]$ indicates a context with a unique occurrence of the hole $\big[\,\big]$, so that $\mathsf{C}\big[\mathsf{t}\big]$ denotes the variable capturing replacement of $\big[\,\big]$ by t in $\mathsf{C}\big[\,\big]$.

The inference rules of $\mathcal{POR}^\lambda$ are the following ones:

$$\mathsf{t} = \mathsf{u} \vdash \mathsf{u} = \mathsf{t} \qquad \text{(R1)}$$
$$\mathsf{t} = \mathsf{u}, \mathsf{u} = \mathsf{v} \vdash \mathsf{t} = \mathsf{v} \qquad \text{(R2)}$$
$$\mathsf{t} = \mathsf{u} \vdash \mathsf{v}\{\mathsf{t}/x\} = \mathsf{v}\{\mathsf{u}/x\} \qquad \text{(R3)}$$
$$\mathsf{t} = \mathsf{u} \vdash \mathsf{t}\{\mathsf{v}/x\} = \mathsf{u}\{\mathsf{v}/x\}. \qquad \text{(R4)}$$

As predictable, $\vdash_{\mathcal{POR}^\lambda} \mathsf{t} = \mathsf{u}$ expresses that the equation $\mathsf{t} = \mathsf{u}$ is deducible using instances of the axioms above plus inference rules (R1)-(R4). Similarly, given any set T of equations, $T \vdash_{\mathcal{POR}^\lambda} \mathsf{t} = \mathsf{u}$ expresses that the equation $\mathsf{t} = \mathsf{u}$ is deducible using instances of the quoted axioms and rules together with equations from T.

Relating $\mathcal{POR}$ and $\mathcal{POR}^\lambda$. For any string $\sigma \in \mathbb{S}$, let $\overline{\overline{\sigma}} : s$ denote the term of $\mathcal{POR}^\lambda$ corresponding to it, that is:

$$\overline{\overline{\epsilon}} = \epsilon \qquad \overline{\overline{\sigma\mathbb{0}}} = \overline{\overline{\sigma}}\mathsf{0} \qquad \overline{\overline{\sigma\mathbb{1}}} = \overline{\overline{\sigma}}\mathsf{1}.$$

For any $\eta \in \mathbb{O}$, let T_η be the set of all equations of the form $\mathsf{Flipcoin}(\overline{\overline{\sigma}}) = \overline{\overline{\eta(\sigma)}}$.

Definition 13.3.6 (Provable Representability)**.** Let $f : \mathbb{O} \times \mathbb{S}^j \to \mathbb{S}$. A term $\mathsf{t} : s \Rightarrow \ldots \Rightarrow s$ of $\mathcal{POR}^\lambda$ *provably represents* f when for all strings $\sigma_1, \ldots, \sigma_j, \sigma \in \mathbb{S}$, and $\eta \in \mathbb{O}$,

$$f(\sigma_1, \ldots, \sigma_j, \eta) = \sigma \quad \text{iff} \quad T_\eta \vdash_{\mathcal{POR}^\lambda} \mathsf{t}\overline{\overline{\sigma_1}} \ldots \overline{\overline{\sigma_j}} = \overline{\overline{\sigma}}.$$

Example 13.3.1. The term $\mathsf{Flipcoin} : s \Rightarrow s$ provably represents the query function $Q(x, \eta) = \eta(x)$ of $\mathcal{POR}$, since for any $\sigma \in \mathbb{S}$ and $\eta \in \mathbb{O}$,

$$\mathsf{Flipcoin}(\overline{\overline{\sigma}}) = \overline{\overline{\eta(\sigma)}} \vdash_{\mathcal{POR}^\lambda} \mathsf{Flipcoin}(\overline{\overline{\sigma}}) = \overline{\overline{Q(\sigma, \eta)}}.$$

We now consider some of the terms described above and show them to provably represent the intended functions. Let $Tail(\sigma, \eta)$ indicate the string obtained by chopping the first digit of σ, and $Trunc(\sigma_1, \sigma_2, \eta) = \sigma_1|_{\sigma_2}$.

Lemma 13.3.1. *Terms* $\mathsf{Tail}, \mathsf{Trunc}$ *and* Cond *provably represent the functions* $Tail$, $Trunc$, *and* C, *respectively.*

Proof Sketch. For Tail and Cond, the claim follows immediately from the defining axioms of the corresponding constants. For example, if $\sigma_1 = \sigma_2\mathbb{0}$, then $Tail(\sigma_1, \eta) = \sigma_2$ (and $\overline{\overline{\sigma_1}} = \overline{\overline{\sigma_2\mathbb{0}}} = \overline{\overline{\sigma_2}}\mathsf{0}$). Using the defining axioms of Tail,

$$\vdash_{\mathcal{POR}^\lambda} \mathsf{Tail}(\overline{\overline{\sigma_1}}) = \mathsf{Tail}(\overline{\overline{\sigma_2\mathbb{0}}}) = \overline{\overline{\sigma_2}}.$$

For $Trunc$ by double induction on $\sigma_1, \sigma_2 \in \mathbb{S}$ we conclude:

$$\vdash_{\mathcal{POR}^\lambda} \mathsf{Trunc}(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}}) = \overline{\overline{\sigma_1}}|_{\overline{\overline{\sigma_2}}}.$$

□

We generalize this result by Theorem 13.3.3 below.

Theorem 13.3.3. *1. Any function $f \in \mathcal{POR}$ is provably represented by a term $\mathsf{t} \in \mathcal{POR}^\lambda$.*

2. For any term $\mathsf{t} \in \mathcal{POR}^\lambda$, there is a function $f \in \mathcal{POR}$ such that f is provably represented by t.

Proof Sketch. 1. The proof is by induction on the structure of $f \in \mathcal{POR}$:

Base Case. Each base function is provably representable. Let us consider two examples only:

- *Empty Function.* $f = E$ is provably represented by the term $\lambda x.\epsilon$. For any string $\sigma \in \mathbb{S}$, $\overline{\overline{E(\sigma, \eta)}} = \overline{\overline{\epsilon}} = \epsilon$ holds. Moreover, $\vdash_{\mathcal{POR}^\lambda} (\lambda x.\epsilon)\overline{\overline{\sigma}} = \epsilon$ is an instance of the (β)-axiom. So, we conclude:

$$\vdash_{\mathcal{POR}^\lambda} (\lambda x.\epsilon)\overline{\overline{\sigma}} = \overline{\overline{E(\sigma, \eta)}}.$$

- *Query Function.* $f = Q$ is provably represented by the term Flipcoin, as observed in Example 13.3.1 above.

Inductive case. Each function defined by composition or bounded recursion from provably represented functions is provably represented as well. We consider the case of bounded recursion. Let f be defined as:

$$\begin{aligned}
f(\sigma_1, \ldots, \sigma_n, \boldsymbol{\epsilon}, \eta) &= g(\sigma_1, \ldots, \sigma_n, \eta) \\
f(\sigma_1, \ldots, \sigma_n, \sigma\mathbb{0}, \eta) &= h_0(\sigma_1, \ldots, \sigma_n, \sigma, f(\sigma_1, \ldots, \sigma_n, \sigma, \eta), \eta)|_{k(\sigma_1, \ldots, \sigma_n, \sigma)} \\
f(\sigma_1, \ldots, \sigma_n, \sigma\mathbb{1}, \eta) &= h_1(\sigma_1, \ldots, \sigma_n, \sigma, f(\sigma_1, \ldots, \sigma_n, \sigma, \eta), \eta)|_{k(\sigma_1, \ldots, \sigma_n, \sigma)}.
\end{aligned}$$

By IH, g, h_0, h_1, and k are provably represented by the corresponding terms $\mathsf{t_g}, \mathsf{t_{h_1}}, \mathsf{t_{h_2}}, \mathsf{t_k}$, respectively. So, for any $\sigma_1, \ldots, \sigma_{n+2}, \sigma \in \mathbb{S}$ and $\eta \in \mathbb{O}$, we derive:

$$T_\eta \vdash_{\mathcal{POR}^\lambda} \mathsf{t_g}\overline{\overline{\sigma_1}} \ldots \overline{\overline{\sigma_n}} = \overline{\overline{g(\sigma_1, \ldots, \sigma_n, \eta)}} \qquad (\mathsf{t_g})$$

$$T_\eta \vdash_{\mathcal{POR}^\lambda} \mathsf{t_{h_0}}\overline{\overline{\sigma_1}} \ldots \overline{\overline{\sigma_{n+2}}} = \overline{\overline{h_0(\sigma_1, \ldots, \sigma_{n+2}, \eta)}} \qquad (\mathsf{t_{h_0}})$$

$$T_\eta \vdash_{\mathcal{POR}^\lambda} \mathsf{t_{h_1}}\overline{\overline{\sigma_1}} \ldots \overline{\overline{\sigma_{n+2}}} = \overline{\overline{h_1(\sigma_1, \ldots, \sigma_{n+2}, \eta)}} \qquad (\mathsf{t_{h_1}})$$

$$T_\eta \vdash_{\mathcal{POR}^\lambda} \mathsf{t_k}\overline{\overline{\sigma_1}} \ldots \overline{\overline{\sigma_n}} = \overline{\overline{k(\sigma_1, \ldots, \sigma_n, \eta)}}. \qquad (\mathsf{t_k})$$

We can prove by induction on σ that,

$$T_\eta \vdash_{\mathcal{POR}^\lambda} \mathsf{t_f}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \overline{\overline{f(\sigma_1, \dots, \sigma_n, \eta)}},$$

where

$$\mathsf{t_f} : \lambda x_1 \dots \lambda x_n . \lambda x . \mathsf{Rec}(\mathsf{t_g} x_1 \dots x_n, \mathsf{t_{h_0}} x_1 \dots x_n, \mathsf{t_{h_1}} x_1 \dots x_n, \mathsf{t_k} x_1 \dots x_n, x).$$

Then,

- if $\sigma = \epsilon$, then $f(\sigma_1, \dots, \sigma_n, \sigma, \eta) = g(\sigma_1, \dots, \sigma_n, \eta)$. Using the (β)-axiom, we deduce,

  $$\vdash_{\mathcal{POR}^\lambda} \mathsf{t_f}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \mathsf{Rec}(\mathsf{t_g}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t_{h_0}}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t_{h_1}}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t_k}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \overline{\overline{\sigma}})$$

  and using the axiom $\mathsf{Rec}(\mathsf{t_g} x_1 \dots x_n, \mathsf{t_{h_0}} x_1 \dots x_n, \mathsf{t_{h_1}} x_1 \dots x_n, \mathsf{t_k} x_1 \dots x_n, \epsilon) = \mathsf{t_g} x_1 \dots x_n$, we obtain,

  $$\vdash_{\mathcal{POR}^\lambda} \mathsf{t_f}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \mathsf{t_g}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}},$$

  by (R2) and (R3). We conclude using $(\mathsf{t_g})$ together with (R2).
- if $\sigma = \sigma_m \mathbb{0}$, then $f(\sigma_1, \dots, \sigma_n, \sigma, \eta) = h_0(\sigma_1, \dots, \sigma_n, \sigma_m, f(\sigma_1, \dots, \sigma_n, \sigma, \eta), \eta)|_{k(\sigma_1, \dots, \sigma_n, \sigma_m)}$. By IH, we suppose,

  $$T_\eta \vdash_{\mathcal{POR}^\lambda} \mathsf{t_f}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma_m}} = \overline{\overline{f(\sigma_1, \dots, \sigma_n, \sigma', \eta)}}.$$

  Then, using the (β)-axiom $\mathsf{t_f}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \mathsf{Rec}(\mathsf{t_g}\ \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t_{h_0}}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t_{h_1}}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t_k}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \overline{\overline{\sigma}})$ the axiom $\mathsf{Rec}(g, h_0, h_1, k, x\mathsf{0}) = \mathsf{Trunc}(h_0 x(\mathsf{Rec}(g, h_0, h_1, k, \mathsf{0})), kx)$ and IH we deduce,

  $$\vdash_{\mathcal{POR}^\lambda} \mathsf{t_f}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \mathsf{Trunc}(\mathsf{t_{h_0}}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma_m}}\,\overline{\overline{f(\sigma_1, \dots, \sigma_n, \sigma_m, \eta)}}, \mathsf{t_k}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}})$$

  by (R2) and (R3). Using $(\mathsf{t_{h_0}})$ and $(\mathsf{t_k})$ we conclude using (R3) and (R2):

  $$\vdash_{\mathcal{POR}^\lambda} \mathsf{t_f}\overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \overline{\overline{h_0(\sigma_1, \dots, \sigma_n, \sigma_m, f(\sigma_1, \dots, \sigma_n, \sigma_m, \eta))|_{k(\sigma_1, \dots, \sigma_n, \sigma_m)}}}.$$

- the case $\sigma = \sigma_m \mathbb{1}$ is proved in a similar way.

2. It is a consequence of the normalization property for the simply typed λ-calculus: a β-normal term $\mathsf{t} : s \Rightarrow \dots \Rightarrow s$ cannot contain variables of higher types. By exhaustively inspecting possible normal forms one can check that these all represent functions in $\mathcal{POR}$. □

Corollary 13.3.1. *For any function $f : \mathbb{S}^j \times \mathbb{0} \to \mathbb{S}$, $f \in \mathcal{POR}$ when f is provably represented by some term $\mathsf{t} : s \Rightarrow \dots \Rightarrow s \in \mathcal{POR}^\lambda$.*

The Theory $\mathcal{IPOR}^\lambda$

We introduce a first-order *intuitionistic* theory, called $\mathcal{IPOR}^\lambda$, which extends $\mathcal{POR}^\lambda$ with basic predicate calculus and a restricted induction principle. We also define $\mathbf{IRS}^1_2$ as a variant of $\mathbf{RS}^1_2$ having the intuitionistic – rather than classical –predicate calculus as its logical basis. All theorems of $\mathcal{POR}^\lambda$ and $\mathbf{IRS}^1_2$ are provable in $\mathcal{IPOR}^\lambda$. In fact, $\mathcal{IPOR}^\lambda$ can be seen as an extension of $\mathcal{POR}^\lambda$, and provides a language to associate derivations in $\mathbf{IRS}^1_2$ with poly-time computable functions (corresponding to terms of $\mathcal{IPOR}^\lambda$).

The Syntax of $\mathcal{IPOR}^\lambda$. The equational theory $\mathcal{POR}^\lambda$ is rather weak. In particular, even simple equations, such as $x = \mathsf{Tail}(x)\mathsf{B}(x)$, cannot be proved in it. Indeed, induction is needed:

$$\begin{aligned} &\vdash_{\mathcal{POR}^\lambda} \epsilon = \epsilon\epsilon = \mathsf{Tail}(\epsilon)\mathsf{B}(\epsilon) \\ x = \mathsf{Tail}(x)\mathsf{B}(x) &\vdash_{\mathcal{POR}^\lambda} x0 = \mathsf{Tail}(x0)\mathsf{B}(x0) \\ x = \mathsf{Tail}(x)\mathsf{B}(x) &\vdash_{\mathcal{POR}^\lambda} x1 = \mathsf{Tail}(x1)\mathsf{B}(x1). \end{aligned}$$

From this we would like to deduce, by induction, that $x = \mathsf{Tail}(x)\mathsf{B}(x)$. Thus, we introduce $\mathcal{IPOR}^\lambda$, the language of which extends that of $\mathcal{POR}^\lambda$ with (a translation for) all expressions of $\mathbf{RS}^1_2$. In particular, the grammar for terms of $\mathcal{IPOR}^\lambda$ is precisely the same as that of Definition 13.3.3, while that for formulae is defined below.

Definition 13.3.7 (Formulae of $\mathcal{IPOR}^\lambda$)**.** *Formulae of $\mathcal{IPOR}^\lambda$* are defined as follows: (i.) all equations of $\mathcal{POR}^\lambda$ $\mathtt{t} = \mathtt{u}$, are formulae of $\mathcal{IPOR}^\lambda$; (ii.) for any (possibly open) term of $\mathcal{POR}^\lambda$, say $\mathtt{t}$ and $\mathtt{u}$, $\mathtt{t} \subseteq \mathtt{u}$ and $\mathtt{Flip}(\mathtt{t})$ are formulae of $\mathcal{IPOR}^\lambda$; (iii.) formulae of $\mathcal{IPOR}^\lambda$ are closed under $\wedge, \vee, \rightarrow, \forall, \exists$.

Notation 13.3.2. We adopt the standard conventions: $\bot := 0 = 1$ and $\neg F := F \rightarrow \bot$.

The notions of Σ^b_0- and Σ^b_1-formula of $\mathcal{IPOR}^\lambda$ are precisely those for $\mathbf{RS}^1_2$, as introduced in Definition 13.2.3.

Remark 13.3.2. *Any formula of $\mathbf{RS}^1_2$ can be seen as a formula of $\mathcal{IPOR}^\lambda$, where each occurrence of the symbol $\mathtt{0}$ is replaced by $\mathsf{0}$, of $\mathtt{1}$ by $\mathsf{1}$, of $\frown$ by $\circ$ (usually omitted), of $\times$ by Times. In the following, we assume that any formula of $\mathbf{RS}^1_2$ is a formula of $\mathcal{IPOR}^\lambda$, modulo the substitutions defined above.*

Definition 13.3.8 (The Theory $\mathcal{IPOR}^\lambda$)**.** The axioms of $\mathcal{IPOR}^\lambda$ include standard rules of the intuitioinstic first-order predicate calculus, usual rules for the equality symbol, and axioms below:

1. All axioms of $\mathcal{POR}^\lambda$
2. $x \subseteq y \leftrightarrow \mathsf{Sub}(x, y) = 1$
3. $x = \epsilon \vee x = \mathsf{Tail}(x)0 \vee x = \mathsf{Tail}(x)1$

4. $0 = 1 \to x = \epsilon$
5. $\mathsf{Cond}(x, y, z, w) = w' \leftrightarrow (x = \epsilon \wedge w' = y) \vee (x = \mathsf{Tail}(x)0 \wedge w' = z) \vee (x = \mathsf{Tail}(x)1 \wedge w' = w)$
6. $\mathtt{Flip}(x) \leftrightarrow \mathsf{Flipcoin}(x) = 1$
7. Any formula of the form,

$$\big(F(\epsilon) \wedge (\forall x)(F(x) \to F(x0)) \wedge (\forall x)(F(x) \to F(x1))\big) \to (\forall y)F(y),$$

where F is of the form $(\exists z \preceq \mathsf{t})\mathsf{u} = \mathsf{v}$, with t containing only first-order open variables.

Notation 13.3.3 (**NP**-Predicate)**.** We refer to a formula of the form $(\exists z \preceq \mathsf{t})\mathsf{u} = \mathsf{v}$, with t containing only first-order open variables, as an **NP**-*predicate*.

Relating $\mathcal{IPOR}^\lambda$ with $\mathcal{POR}^\lambda$ and $\mathbf{IRS}^1_2$. Now that $\mathcal{IPOR}^\lambda$ has been introduced we show that theorems of both $\mathcal{POR}^\lambda$ and the intuitionistic version of $\mathbf{RS}^1_2$ are derived in it. First, Proposition 13.3.2 is easily easily established by inspecting all rules of $\mathcal{POR}^\lambda$.

Proposition 13.3.2. *Any theorem of $\mathcal{POR}^\lambda$ is a theorem of $\mathcal{IPOR}^\lambda$.*

Then, we consider $\mathbf{IRS}^1_2$ and establish that every theorem in it is derivable in $\mathcal{IPOR}^\lambda$. To do so, we prove a few properties concerning $\mathcal{IPOR}^\lambda$. In particular, its recursion schema differs from that of $\mathbf{IRS}^1_2$ as dealing with formulae of the form $(\exists y \preceq \mathsf{t})\mathsf{u} = \mathsf{v}$ and not with all the Σ^b_1-ones. The two schemas are related by Proposition 13.3.3, proved by induction on the structure of formulae.[11]

Proposition 13.3.3. *For any Σ^b_0-formula $F(x_1, \ldots, x_n)$ in $\mathcal{RL}$, there exists a term $\mathsf{t}_F(x_1, \ldots, x_n)$ of $\mathcal{POR}^\lambda$ such that:*

1. $\vdash_{\mathcal{IPOR}^\lambda} F \leftrightarrow \mathsf{t}_F = 0$
2. $\vdash_{\mathcal{IPOR}^\lambda} \mathsf{t}_F = 0 \vee \mathsf{t}_F = 1$.

This leads us to the following corollary and allows us to prove Theorem 13.3.4 relating $\mathcal{IPOR}^\lambda$ and $\mathbf{IRS}^1_2$.

Corollary 13.3.2. *For any Σ^b_0-formula F, $\vdash_{\mathcal{IPOR}^\lambda} F \vee \neg F$.*

Theorem 13.3.4. *Any theorem of $\mathbf{IRS}^1_2$ is a theorem of $\mathcal{IPOR}^\lambda$.*

Proof. First, observe that, as a consequence of Proposition 13.3.3, for any Σ^b_1-formula $F = (\exists x_1 \preceq t_1) \ldots (\exists x_n \preceq t_n)G$ in $\mathcal{RL}$,

$$\vdash_{\mathcal{IPOR}^\lambda} F \leftrightarrow (\exists x_1 \preceq \mathsf{t}_1) \ldots (\exists x_n \preceq \mathsf{t}_n)\mathsf{t}_G = 0,$$

any instance of the Σ^b_1-recursion schema of $\mathbf{IRS}^1_2$ is derivable in $\mathcal{IPOR}^\lambda$ from the **NP**-induction schema. Then, it suffices to check that all basic axioms of $\mathbf{IRS}^1_2$ are provable in $\mathcal{IPOR}^\lambda$. □

[11] For further details, see [6].

This result also leads to the following straightforward consequences.

Corollary 13.3.3. *For any closed Σ^b_0-formula of $\mathcal{RL}$ F and $\eta \in \mathbb{O}$, either* $\mathrm{T}_\eta \vdash_{\mathcal{IPOR}^\lambda} F$ *or* $\mathrm{T}_\eta \vdash_{\mathcal{IPOR}^\lambda} \neg F$.

Proof. It suffices to show that for all closed F, $T_\eta \vdash_{\mathcal{IPOR}^\lambda} \mathsf{t_F} = 0$ or $T_\eta \vdash_{\mathcal{IPOR}^\lambda} \mathsf{t_F} = 1$. This is proved by induction on F. We only consider non-trivial cases:

- $F = \mathtt{Flip}(u)$, then for hypothesis u is closed, so it corresponds to a string σ_u; then we deduce $T_\eta \vdash_{\mathcal{IPOR}^\lambda} \mathsf{t_F} = \overline{\omega(\sigma_u)}$, depending on some $\omega(\sigma_u) = 0, 1$.
- if $F = (\exists x \subseteq u)G$, then by IH, for any choice of a string σ, we know that $T_\eta \vdash_{\mathcal{IPOR}^\lambda} (\mathsf{t_G})^{\exists}(\overline{\sigma}) = 1$. The claim is proved by induction on σ.

□

Due to Corollary 13.3.3, we can even establish the following Lemma 13.3.2.

Lemma 13.3.2. *Let F be a closed Σ^b_0-formula of $\mathcal{RL}$ and $\eta \in \mathbb{O}$, then:*

$$\mathrm{T}_\eta \vdash_{\mathcal{IPOR}^\lambda} F \quad \textit{iff} \quad \eta \in [\![F]\!].$$

Proof. ($\Rightarrow$) This soundness result is established by induction on the structure of rules for $\mathcal{IPOR}^\lambda$.

($\Leftarrow$) For Corollary 13.3.3, we know that either $T_\eta \vdash_{\mathcal{IPOR}^\lambda} F$ or $T_\eta \vdash_{\mathcal{IPOR}^\lambda} \neg F$. Hence, if $\eta \in [\![F]\!]$, then it cannot be $T_\eta \vdash_{\mathcal{IPOR}^\lambda} \neg F$ (by soundness). So, we conclude $T_\eta \vdash_{\mathcal{IPOR}^\lambda} F$. □

Realizability

Here, we introduce realizability as internal to $\mathcal{IPOR}^\lambda$. As a corollary, we obtain that from any derivation in $\mathbf{IRS}^1_2$ – actually, in $\mathcal{IPOR}^\lambda$ – of a formula in the form $(\forall x)(\exists y)F(x, y)$, one can extract a functional term of $\mathcal{POR}^\lambda$ $\mathsf{f} : s \Rightarrow s$, such that $\vdash_{\mathcal{IPOR}^\lambda} (\forall x)F(x, \mathsf{f}x)$. This allows us to conclude that if a function f is Σ^b_1-representable in $\mathbf{IRS}^1_2$, then $f \in \mathcal{POR}$.

Notation 13.3.4. Let $\mathbf{x}, \mathbf{y}$ denote finite sequences of term variables, (resp.) $x_1, \ldots, x_n$ and $y_1, \ldots, y_k$, and $\mathbf{y}(\mathbf{x})$ be an abbreviation for $y_1(\mathbf{x}), \ldots, y_k(\mathbf{x})$. Let Λ be a shorthand for the empty sequence and $y(\Lambda) := y$.

Definition 13.3.9. Formulae $x \circledR F$ are defined by induction as follows:

$$\begin{aligned}
\Lambda \circledR F &:= F \quad (F \text{ atomic}) \\
\mathbf{x}, \mathbf{y} \circledR (G \wedge H) &:= (\mathbf{x} \circledR G) \wedge (\mathbf{y} \circledR H) \\
z, \mathbf{x}, \mathbf{y} \circledR (G \vee H) &:= (z = 0 \wedge \mathbf{x} \circledR G) \vee (z \neq 0 \wedge \mathbf{y} \circledR H) \\
\mathbf{y} \circledR (G \to H) &:= (\forall \mathbf{x})(\mathbf{x} \circledR G \to \mathbf{y}(\mathbf{x}) \circledR H) \wedge (G \to H) \\
z, \mathbf{x} \circledR (\exists y)G &:= \mathbf{x} \circledR G\{z/y\} \\
\mathbf{x} \circledR (\forall y)G &:= (\forall y)(\mathbf{x}(y) \circledR G),
\end{aligned}$$

where no variable in $\mathbf{x}$ occurs free in F. Given terms $\mathbf{t} = \mathtt{t}_1, \ldots, \mathtt{t}_n$, we let:

$$\mathbf{t} \circledR F := (\mathbf{x} \circledR F)\{\mathbf{t}/\mathbf{x}\}.$$

We relate the derivability of these new formulae with that of formulae of $\mathcal{IPOR}^\lambda$. Proofs below are by induction, respectively, on the structure of $\mathcal{IPOR}^\lambda$-formulae and on the height of derivations.

Theorem 13.3.5 (Soundness)**.** *If $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{t} \circledR F$, then $\vdash_{\mathcal{IPOR}^\lambda} F$.*

Notation 13.3.5. Given $\Gamma = F_1, \ldots, F_n$, let $\mathbf{x} \circledR \Gamma$ be a shorthand for $\mathbf{x}_1 \circledR F_1, \ldots, \mathbf{x}_n \circledR F_n$.

Theorem 13.3.6 (Completeness)**.** *If $\vdash_{\mathcal{IPOR}^\lambda} F$, then there exist terms $\mathbf{t}$, such that $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{t} \circledR F$.*

Proof Sketch. We prove that if $\Gamma \vdash_{\mathcal{IPOR}^\lambda} F$, then there exist terms $\mathbf{t}$ such that $\mathbf{x} \circledR \Gamma \vdash_{\mathcal{IPOR}^\lambda} \mathbf{t}\mathbf{x}_1 \ldots \mathbf{x}_n \circledR F$. The proof is by induction on the derivation of $\Gamma \vdash_{\mathcal{IPOR}^\lambda} F$. Let us consider just the case of rule $\vee R_1$ as an example:

$$\frac{\begin{array}{c}\vdots\\ \Gamma \vdash G\end{array}}{\Gamma \vdash G \vee H} \vee R_1$$

By IH, there exist terms $\mathbf{u}$, such that $\mathbf{t} \circledR \Gamma \vdash_{\mathcal{IPOR}^\lambda} \mathbf{t}\mathbf{u} \circledR G$. Since $x, y \circledR G \vee H$ is defined as $(x = 0 \wedge y \circledR G) \vee (x \neq 0 \wedge y \circledR H)$, we can take $\mathbf{t} = 0, \mathbf{u}$. □

Corollary 13.3.4. *Let $(\forall x)(\exists y)F(x, y)$ be a closed theorem of $\mathcal{IPOR}^\lambda$, where F is a Σ_1^b-formula. Then, there exists a closed term $\mathtt{t} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$ such that:*

$$\vdash_{\mathcal{IPOR}^\lambda} (\forall x)F(x, \mathtt{t}x).$$

Proof. By Theorem 13.3.6, there exist $\mathbf{t} = \mathtt{t}, w$ such that $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{t} \circledR (\forall x)(\exists y)F(x, y)$. So, by Definition 13.3.9,

$$\begin{aligned}\mathbf{t} \circledR (\forall x)(\exists y)F(x, y) &\equiv (\forall x)(\mathbf{t}(x) \circledR (\exists y)F(x, y))\\ &\equiv (\forall x)(w(x) \circledR F(x, \mathtt{t}x)).\end{aligned}$$

From this, by Theorem 13.3.5, we deduce,

$$\vdash_{\mathcal{IPOR}^\lambda} (\forall x)F(x, \mathtt{t}x).$$

□

Functions which are Σ^b_1-Representable in $\mathbf{IRS}^1_2$ are in $\mathcal{POR}$. Now, we have all the ingredients to prove that if a function is Σ^b_1-representable in $\mathbf{IRS}^1_2$, in the sense of Definition 13.3.1, then it is in $\mathcal{POR}$.

Corollary 13.3.5. *For any function $f : \mathbb{O} \times \mathbb{S} \to \mathbb{S}$, if there is a closed Σ^b_1-formula in $\mathcal{RL}$ $F(x, y)$, such that:*

1. $\mathbf{IRS}^1_2 \vdash (\forall x)(\exists! y)F(x, y)$
2. $[\![F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})]\!] = \{\eta \mid f(\sigma_1, \eta) = \sigma_2\}$,

then $f \in \mathcal{POR}$.

Proof. Since $\vdash_{\mathbf{IRS}^1_2} (\forall x)(\exists! y)F(x, y)$, by Theorem 13.3.4 $\vdash_{\mathcal{IPOR}^\lambda} (\forall x)(\exists! y)F(x, y)$. Then, from $\vdash_{\mathcal{IPOR}^\lambda} (\forall x)(\exists y)F(x, y)$ we deduce $\vdash_{\mathcal{IPOR}^\lambda} (\forall x)F(x, \mathrm{g}x)$ for some closed term $\mathrm{g} \in \mathcal{POR}^\lambda$, by Corollary 13.3.4. Furthermore, by Theorem 13.3.3.2, there is a $g \in \mathcal{POR}$ such that for any $\sigma_1, \sigma_2 \in \mathbb{S}$ and $\eta \in \mathbb{O}$, $g(\sigma_1, \eta) = \sigma_2$ when $T_\eta \vdash_{\mathcal{IPOR}^\lambda} \mathrm{g}\overline{\overline{\sigma_1}} = \overline{\overline{\sigma_2}}$. So, by Proposition 13.3.2, for any $\sigma_1, \sigma_2 \in \mathbb{S}$ and $\eta \in \mathbb{O}$ if $g(\sigma_1, \eta) = \sigma_2$, then $T_\eta \vdash_{\mathcal{IPOR}^\lambda} \mathrm{g}\overline{\overline{\sigma_1}} = \overline{\overline{\sigma_2}}$ and so $T_\eta \vdash_{\mathcal{IPOR}^\lambda} F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})$. By Lemma 13.3.2, $T_\eta \vdash_{\mathcal{IPOR}^\lambda} F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})$ when $\eta \in [\![F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})]\!]$, that is $f(\sigma_1, \eta) = \sigma_2$. But then $f = g$, so since $g \in \mathcal{POR}$ also $f \in \mathcal{POR}$. □

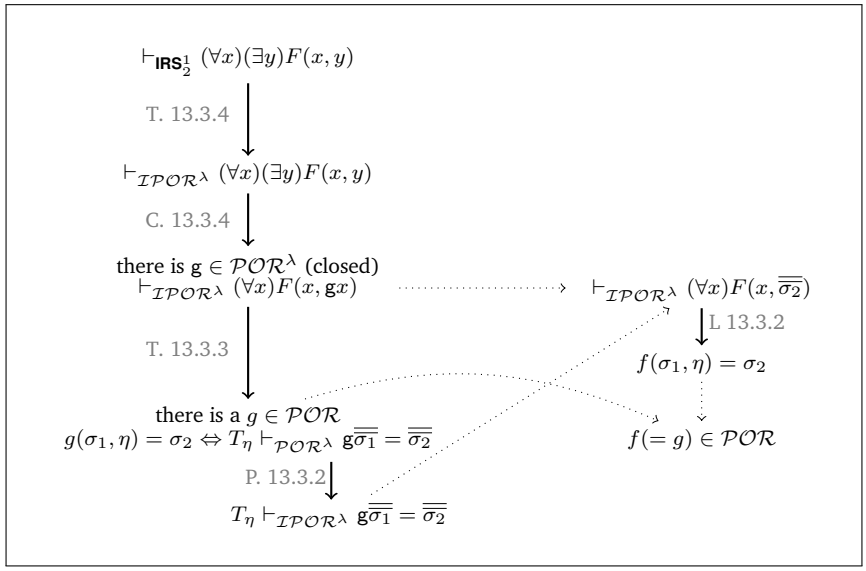


Figure 13.4: Proof Schema of Corollary 13.3.5

∀NP-Conservativity of $\mathcal{IPOR}^\lambda$ + EM over $\mathcal{IPOR}^\lambda$

Corollary 13.3.5 is already very close to the result we are looking for. The remaining step to conclude our proof is its extension from intuitioninstic $\mathbf{IRS}^1_2$

to classical $\mathbf{RS}^1_2$, showing that any function which is Σ^b_1-representable in $\mathbf{RS}^1_2$ is also in $\mathcal{POR}$. The proof is obtained by adapting the method from [46]. We start by considering an extension of $\mathcal{IPOR}^\lambda$ via **EM** and show that the realizability interpretation extends to it so that for any of its closed theorems $(\forall x)(\exists y \preceq \mathsf{t})F(x,y)$, being F a Σ^b_1-formula, there is a closed term $\mathsf{t} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$ such that $\vdash_{\mathcal{IPOR}^\lambda} (\forall x)F(x,\mathsf{t}x)$.

From $\mathcal{IPOR}^\lambda$ to $\mathcal{IPOR}^\lambda$+(Markov) Let **EM** be the excluded-middle schema, $F \vee \neg F$, and *Markov's principle* be defined as follows,

$$\neg\neg(\exists x)F \to (\exists x)F, \qquad \textbf{(Markov)}$$

where F is a Σ^b_1-formula.

Proposition 13.3.4. *For any Σ^b_1-formula F, if $\vdash_{\mathcal{IPOR}^\lambda+\mathbf{EM}} F$, then $\vdash_{\mathcal{IPOR}^\lambda+(\mathbf{Markov})} F$.*

Proof Sketch. The claim is proved by applying the double negation translation, with the following two remarks: (1) for any Σ^b_0-formula F, $\vdash_{\mathcal{IPOR}^\lambda} \neg\neg F \to F$; (2) using (**Markov**), the double negation of an instance of the **NP**-induction can be shown equivalent to an instance of the **NP**-induction schema. □

We conclude by showing that the realizability interpretation defined above extends to $\mathcal{IPOR}^\lambda$+(**Markov**), that is for any closed theorem $(\forall x)(\exists y \preceq \mathsf{t})F(x,y)$ with F Σ^b_1-formula, of $\mathcal{IPOR}^\lambda$ + (**Markov**), there is a closed term of $\mathcal{POR}^\lambda$ $\mathsf{t} : s \Rightarrow s$, such that $\vdash_{\mathcal{IPOR}^\lambda} (\forall x)F(x,\mathsf{t}x)$.

From $\mathcal{IPOR}^\lambda$ to $(\mathcal{IPOR}^\lambda)^*$. Let us assume given a subjective encoding $\sharp : (s \Rightarrow s) \Rightarrow s$ in $\mathcal{IPOR}^\lambda$ of first-order unary functions as strings, together with a "decoding" function $\mathsf{app} : s \Rightarrow s \Rightarrow s$ satisfying:

$$\vdash_{\mathcal{IPOR}^\lambda} \mathsf{app}(\sharp\mathsf{f}, x) = \mathsf{f}x.$$

Moreover, let

$$x * y := \sharp(\lambda z.\mathsf{BAnd}(\mathsf{app}(x,z), \mathsf{app}(y,z)))$$

and

$$T(x) := (\exists y)(\mathsf{B}(\mathsf{app}(x,y)) = \mathsf{0}).$$

There is a *meet semi-lattice* structure on the set of terms of type s defined by $\mathsf{t} \sqsubseteq \mathsf{u}$ when $\vdash_{\mathcal{IPOR}^\lambda} T(\mathsf{u}) \to T(\mathsf{t})$ with top element $\underline{\mathbf{1}} := \sharp(\lambda x.1)$ and meet given by $x * y$. Indeed, from $T(x * 1) \leftrightarrow T(x)$, $x \sqsubseteq \underline{\mathbf{1}}$ follows. Moreover, from $\mathsf{B}(\mathsf{app}(x,\mathsf{u})) = \mathsf{0}$, we obtain $\mathsf{B}(\mathsf{app}(x * y, \mathsf{u})) = \mathsf{BAnd}(\mathsf{app}(x,\mathsf{u}), \mathsf{app}(y,\mathsf{u})) = \mathsf{0}$, whence $T(x) \to T(x * y)$, i.e. $x * y \sqsubseteq x$. One can similarly prove $x * y \sqsubseteq y$. Finally, from $T(x) \to T(v)$ and $T(y) \to T(v)$, we deduce $T(x * y) \to T(v)$, by observing that $\vdash_{\mathcal{IPOR}^\lambda} T(x * y) \to T(y)$. Notice that the formula $T(x)$ is *not* a Σ^b_1-one, as its existential quantifier is not bounded.

Definition 13.3.10. For any formula of $\mathcal{IPOR}^\lambda$ F, and fresh variable x, we define formulae $x \Vdash F$ inductively:

$$\begin{aligned} x \Vdash F &:= F \vee T(x) \quad (F \; atomic) \\ x \Vdash G \wedge H &:= x \Vdash G \wedge x \Vdash H \\ x \Vdash G \vee H &:= x \Vdash G \vee x \Vdash H \\ x \Vdash G \to H &:= (\forall y)(y \Vdash G \to x * y \Vdash H) \\ x \Vdash (\exists y)G &:= (\exists y)x \Vdash G \\ x \Vdash (\forall y)G &:= (\forall y)x \Vdash G. \end{aligned}$$

The following Lemma 13.3.3 is established by induction on the structure of formulae in $\mathcal{IPOR}^\lambda$, as in [47].

Lemma 13.3.3. *If F is provable in $\mathcal{IPOR}^\lambda$ without using* **NP***-induction, then $x \Vdash F$ is provable in $\mathcal{IPOR}^\lambda$.*

Lemma 13.3.4. *Let $F = (\exists x \preceq \mathsf{t})G$, where G is a Σ^b_0-formula. Then, there exists a term $\mathsf{u_F} : s$ with $\mathsf{FV}(\mathsf{u_F}) = \mathsf{FV}(G)$ such that:*

$$\vdash_{\mathcal{IPOR}^\lambda} F \leftrightarrow T(\mathsf{u_F}).$$

Proof. Since $G(x)$ is a Σ^b_0-formula, for all terms $\mathsf{v} : s$, $\vdash_{\mathcal{IPOR}^\lambda} G(x) \leftrightarrow \mathsf{t}_{\mathsf{x} \preceq \mathsf{t} \wedge \mathsf{G}}(x) = \mathsf{0}$, where $\mathsf{t}_{\mathsf{x} \preceq \mathsf{t} \wedge \mathsf{G}}$ has the free variables of t and G. Let $H(x)$ be a Σ^b_0-formula, it is shown by induction on its structure that for any term $\mathsf{v} : s$, $\mathsf{t}_{\mathsf{H(v)}} = \mathsf{t}_{\mathsf{H}}(\mathsf{v})$. Then,

$$\vdash_{\mathcal{IPOR}^\lambda} F \leftrightarrow (\exists x)\mathsf{t}_{\mathsf{x} \preceq \mathsf{t} \wedge \mathsf{G}}(x) = \mathsf{0} \leftrightarrow (\exists x)T(\sharp(\lambda x.\mathsf{t}_{\mathsf{x} \preceq \mathsf{t} \wedge \mathsf{G}}(x))).$$

So, we let $\mathsf{u_F} = \sharp(\lambda x.\mathsf{t}_{\mathsf{x} \preceq \mathsf{t} \wedge \mathsf{G}}(x))$. □

From which we also deduce the following three properties:

i. $\vdash_{\mathcal{IPOR}^\lambda} (x \Vdash F) \leftrightarrow (F \vee T(x))$
ii. $\vdash_{\mathcal{IPOR}^\lambda} (x \Vdash \neg F) \leftrightarrow (F \to T(x))$
iii. $\vdash_{\mathcal{IPOR}^\lambda} (x \Vdash \neg\neg F) \leftrightarrow (F \vee T(x))$,

where F is a Σ^b_1-formula.

Corollary 13.3.6 (Markov's Principle)**.** *If F is a Σ^b_1-formula, then*

$$\vdash_{\mathcal{IPOR}^\lambda} x \Vdash \neg\neg F \to F.$$

To define the extension $(\mathcal{IPOR}^\lambda)^*$ of $\mathcal{IPOR}^\lambda$, we introduce PIND in a formal way.

Definition 13.3.11 (PIND)**.** Let PIND(F) indicate the formula:

$$\big(F(\epsilon) \wedge \big((\forall x)(F(x) \to F(x0)) \wedge (\forall x)(F(x) \to F(x1))\big)\big) \to (\forall x)F(x).$$

Observe that if $F(x)$ is a formula of the form $(\exists y \preceq \mathsf{t})\mathsf{u} = \mathsf{v}$, then $z \Vdash \text{PIND}(F)$ is of the form $\text{PIND}(F(x) \vee T(z))$, which is *not* an instance of the **NP**-induction schema (as the formula $T(z) = (\exists x)\mathsf{B}(\mathsf{app}(z, x)) = \mathsf{0}$ is not bounded).

Definition 13.3.12 (The Theory $(\mathcal{IPOR}^{\lambda})^{*}$)**.** Let $(\mathcal{IPOR}^{\lambda})^{*}$ indicate the theory extending $\mathcal{IPOR}^{\lambda}$ with all instances of the induction schema $\text{PIND}(F(x) \vee G)$, where $F(x)$ is of the form $(\exists y \preceq \mathsf{t})\mathsf{u} = \mathsf{v}$, and G is an arbitrary formula with $x \notin \mathsf{FV}(G)$.

We then deduce the following Proposition relating derivability in $\mathcal{IPOR}^{\lambda}$ and in $(\mathcal{IPOR}^{\lambda})^{*}$.

Proposition 13.3.5. *For any Σ_1^b-formula F, if $\vdash_{\mathcal{IPOR}^{\lambda}} F$, then $\vdash_{(\mathcal{IPOR}^{\lambda})^{*}} x \Vdash F$.*

Finally, we extend realizability to $(\mathcal{IPOR}^{\lambda})^{*}$ by constructing a realizer for $\text{PIND}(F(x) \vee G)$.

Lemma 13.3.5. *Let $F(x) : (\exists y \preceq \mathsf{t})\mathsf{u} = \mathsf{0}$ and G be any formula not containing free occurrences of x. Then, there exist terms $\mathbf{t}$ such that:*

$$\vdash_{\mathcal{IPOR}^{\lambda}} \mathbf{t} \circledR \textit{PIND}(F(x) \vee G).$$

So, by Theorem 13.3.5, we obtain that for any Σ_1^b-formula F and formula G, with $x \notin \mathsf{FV}(F)$,

$$\vdash_{\mathcal{IPOR}^{\lambda}} \text{PIND}(F(x) \vee G).$$

Proposition 13.3.6. *For any Σ_1^b-formula F and G with $x \notin \mathsf{FV}(F)$, $\vdash_{\mathcal{IPOR}^{\lambda}} \textit{PIND}(F(x) \vee G)$.*

Corollary 13.3.7 (∀**NP**-Conservativity of $\mathcal{IPOR}^{\lambda} + \mathbf{EM}$ over $\mathcal{IPOR}^{\lambda}$)**.** *Let F be a Σ_1^b-formula, if $\vdash_{\mathcal{IPOR}^{\lambda}+\mathbf{EM}} (\forall x)(\exists y \preceq \mathsf{t})F(x, y)$, then $\vdash_{\mathcal{IPOR}^{\lambda}} (\forall x)(\exists y \preceq \mathsf{t})F(x, y)$.*

Concluding the Proof. We conclude our proof establishing Proposition 13.3.7.

Proposition 13.3.7. *Let $(\forall x)(\exists y \preceq \mathsf{t})F(x, y)$ be a closed theorem of $\mathcal{IPOR}^{\lambda} + (\mathbf{Markov})$, where F is a Σ_1^b-formula. Then, there exists a closed term of $\mathcal{POR}^{\lambda}$ $\mathsf{t} : s \Rightarrow s$, such that:*

$$\vdash_{\mathcal{IPOR}^{\lambda}} (\forall x)F(x, \mathsf{t}x).$$

Proof. If $\vdash_{\mathcal{IPOR}^{\lambda}+(\mathbf{Markov})} (\forall x)(\exists y)F(x, y)$, then by Parikh's Proposition 13.3.1, also $\vdash_{\mathcal{IPOR}^{\lambda}+(\mathbf{Markov})} (\exists y \preceq \mathsf{t})F(x, y)$. Moreover, $\vdash_{(\mathcal{IPOR}^{\lambda})^{*}} z \Vdash (\exists y \preceq \mathsf{t})F(x, y)$. Then, let us consider $G = (\exists y \preceq \mathsf{t})F(x, y)$. By taking $\mathsf{v} = \mathsf{u}_G$, using Lemma 13.3.4, we deduce $\vdash_{(\mathcal{IPOR}^{\lambda})^{*}} G$ and, thus, by Lemma 13.3.3 and 13.3.5, we conclude that there exist $\mathbf{t}, \mathbf{u}$ such that $\vdash_{\mathcal{IPOR}^{\lambda}} \mathbf{t}, \mathbf{u} \circledR G$, which implies $\vdash_{\mathcal{IPOR}^{\lambda}} F(x, \mathbf{t}x)$, and so $\vdash_{\mathcal{IPOR}^{\lambda}} (\forall x)(F(x), \mathbf{t}x)$. □

So, by Proposition 13.3.4, if $\vdash_{\mathcal{IPOR}^\lambda+\mathbf{EM}} (\forall x)(\exists y \preceq \mathsf{t})F(x,y)$, being F a closed Σ_1^b-formula, then there is a closed term of $\mathcal{POR}$ $\mathsf{t} : s \Rightarrow s$, such that $\vdash_{\mathcal{IPOR}^\lambda} (\forall x)F(x,\mathsf{t}x)$. Finally, we conclude the desired Corollary 13.3.8 for classical $\mathbf{RS}_2^1$ arguing as above.

Corollary 13.3.8. *Let $\boldsymbol{RS}_2^1 \vdash (\forall x)(\exists y \preceq t)F(x,y)$, where F is a Σ_1^b-formula with only x and y free. For any function $f : \mathbb{S} \times \mathbb{O} \to \mathbb{S}$, if $(\forall x)(\exists y \preceq t)F(x,y)$ represents f so that:*

1. $\boldsymbol{RS}_2^1 \vdash (\forall x)(\exists! y)F(x,y)$
2. $[\![F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})]\!] = \{\eta \mid f(\sigma_1,\eta) = \sigma_2\}$,

then $f \in \mathcal{POR}$.

Now, putting Theorem 13.3.2 and Corollary 13.3.8 together, we conclude that Theorem 13.3.1 holds and that $\mathbf{RS}_2^1$ provides an *arithmetical* characterization of functions in $\mathcal{POR}$.

13.4 Relating $\mathcal{POR}$ and Poly-Time PTMs

Theorem 13.3.1 is still not enough to characterize probabilistic classes, which are defined in terms of functions computed by PTMs, and, as observed, there is a crucial difference between the ways in which these machines and oracle functions access randomness. So, our next goal consists in filling this gap, by relating these two classes in a precise way.[12]

13.4.1 Preliminaries

We start by defining (or re-defining) the classes of functions (over strings) computed by poly-time PTMs and of functions computed by poly-time stream machines, that is TMs with an extra oracle tape.

The Class RFP

We start by (re-)defining the class of functions computed by poly-time PTMs.[13]

Definition 13.4.1 (Class **RFP**)**.** Let $\mathbb{D}(\mathbb{S})$ denote the set of functions $f : \mathbb{S} \to [0,1]$ such that $\sum_{\sigma\in\mathbb{S}} f(\sigma) = 1$. The *class* **RFP** is made of all functions $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ such that, for some PTM $\mathscr{M}_{\mathscr{P}}$ running in polynomial time, and every $\sigma_1,\ldots,\sigma_k,\tau \in \mathbb{S}$, $f(\sigma_1,\ldots,\sigma_k)(\tau)$ coincides with the probability that $\mathscr{M}_{\mathscr{P}}(\sigma_1\sharp\ldots\sharp\sigma_k) \Downarrow \tau$.

[12] Actually, this proof is particularly convoluted so, for simplicity's sake, we present here just its skeleton. The interested reader can however find further details in [6], and the full proof was presented in [68].

[13] Clearly, there is a strong affinity with the standard definition of random functions [183] presented for example in Section 12.3.2. Here, pseudo-distributions and functions are over strings rather than numbers. Furthermore, we are now considering machines explicitly associated with a (polynomial-)time resource bound.

So – similarly to $\langle \mathscr{M}_{\mathscr{P}} \rangle$ by [183, 91] – the function computed by this machine associates each possible output with a probability corresponding to the actual probability that a run of the machine actually produces that output, and we need to adapt the notion of Σ_1^b-representability accordingly.

Definition 13.4.2. A function $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ is *Σ_1^b-representable in* $\mathbf{RS}_2^1$ if there is a Σ_1^b-formula of $\mathcal{RL}$ $F(x_1, \ldots, x_k, y)$, such that:

1. $\mathbf{RS}_2^1 \vdash (\forall \vec{x})(\exists! y) F(\vec{x}, y)$,
2. for all $\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{S}$, $f(\sigma_1, \ldots, \sigma_k, \tau) = \mu([\![F(\overline{\sigma_1}, \ldots, \overline{\sigma_k}, \tau)]\!])$.

The central result of this chapter can be re-stated as follows:

Theorem 13.4.1. *For any function $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$, f is Σ_1^b-representable in $\mathbf{RS}_2^1$ when $f \in \mathbf{RFP}$.*

The proof of Theorem 13.4.1 relies on Theorem 13.3.1, once we relate the function algebra $\mathcal{POR}$ with the class $\mathbf{RFP}$ by the Lemma 13.4.1 below.

Lemma 13.4.1. *For any functions $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$ in $\mathcal{POR}$, there exists $g : \mathbb{S} \to \mathbb{D}(\mathbb{S})$ in $\mathbf{RFP}$ such that for all $\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{S}$,*

$$\mu(\{\omega \mid f(\sigma_1, \ldots, \sigma_k, \eta) = \tau\}) = g(\sigma_1, \ldots, \sigma_k, \tau)$$

and vice versa.

However, the proof of Lemma 13.4.1 is convoluted, as it is based on a chain of language simulations.

Introducing the Class SFP

The core idea to relate $\mathcal{POR}$ and $\mathbf{RFP}$ is to introduce an intermediate class, called $\mathbf{SFP}$. This is the class of functions computed by a poly-time *stream Turing machine* (STM, for short), where an STM is a deterministic TM with one extra (read-only) tape intuitively accounting for probabilistic choices: at the beginning the extra tape is sampled from $\mathbb{B}^{\mathbb{N}}$; then, at each computation step, the machine reads one new bit from this tape, always moving to the right.

Definition 13.4.3 (Class $\mathbf{SFP}$)**.** The *class* $\mathbf{SFP}$ is made of functions $f : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$, such that there is an STM $\mathscr{M}_{\mathcal{S}}$ running in polynomial time such that for any $\sigma_1, \ldots, \sigma_k \in \mathbb{S}$ and $\omega \in \mathbb{B}^{\mathbb{N}}$,[14] $f(\sigma_1, \ldots, \sigma_k, \omega) = \tau$ when for inputs $\sigma_1 \sharp \ldots \sharp \sigma_k$ and tape ω, the machine $\mathscr{M}_{\mathcal{S}}$ outputs τ.

[14] By a slight abuse of notation, we use $\omega, \omega', \omega'', \ldots$ as meta-varibles for sequences in $\mathbb{B}^{\mathbb{N}}$. Indeed, as said in Section 3.1, the sets $\{0, 1\}$ and $\{\mathbb{0}, \mathbb{1}\}$ are basically equivalent.

13.4.2 Relating RFP and SFP

The global behavior of STMs and PTMs is similar, but the former access randomness in an explicit way: instead of flipping a coin at each step, the machine samples a stream of bits once, and then reads one new bit at each step. So, to prove the equivalence of the two models, we pass through the following Proposition 13.4.1.

Proposition 13.4.1 (Equivalence of PTMs and STMs)**.** *For any poly-time STM $\mathscr{M}_{\mathcal{S}}$, there is a poly-time PTM $\mathscr{M}_{\mathcal{S}}{}^{*}$ such that for all strings $\sigma, \tau \in \mathbb{S}$,*

$$\mu(\{\omega \mid \mathscr{M}_{\mathcal{S}}(\sigma, \omega) = \tau\}) = \Pr[\mathscr{M}_{\mathcal{S}}{}^{*}(\sigma) = \tau],$$

and vice versa.

Corollary 13.4.1 (Equivalence of **RFP** and **SFP**)**.** *For any $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ in* **RFP***, there is a $g : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$ in* **SFP***, such that for all $\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{S}$,*

$$f(\sigma_1, \ldots, \sigma_k, \tau) = \mu(\{\omega \mid g(\sigma_1, \ldots, \sigma_k, \omega) = \tau\}),$$

and vice versa.

13.4.3 Relating SFP and $\mathcal{POR}$

Finally, we need to prove the equivalence between $\mathcal{POR}$ and **SFP**:

Moving from PTMs to STMs, we obtain a machine model which accesses randomness in a way which is similar to that of functions in $\mathcal{POR}$: as seen, at the beginning of the computation an oracle is sampled, and computation proceeds querying it. Yet, there are still relevant differences in the way in which these families of machines treat randomness. While functions of $\mathcal{POR}$ access an oracle in the form of a function $\eta \in \mathbb{B}^{\mathbb{S}}$, the oracle for an STM is a stream of bits $\omega \in \mathbb{B}^{\mathbb{N}}$. Otherwise said, a function in $\mathcal{POR}$ is of the form $f_{\mathcal{POR}} : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$, whereas one in **SFP** is $f_{\mathbf{SFP}} : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$. Then, we cannot compare them *directly*, and provide an indirect comparison in two main steps.

From SFP to $\mathcal{POR}$

First, we show that any function computable by a poly-time STM is in $\mathcal{POR}$.

Proposition 13.4.2 (From **SFP** to $\mathcal{POR}$)**.** *For any $f : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$ in* **SFP***, there is a function $f^{\star} : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$ in $\mathcal{POR}$ such that for all $\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{S}$ and $\omega \in \mathbb{B}^{\mathbb{N}}$,*

$$\mu(\{\omega \in \mathbb{B}^{\mathbb{N}} \mid f(\sigma_1, \ldots, \sigma_k, \omega) = \tau\}) = \mu(\{\eta \in \mathbb{O} \mid f^{\star}(\sigma_1, \ldots, \sigma_k, \eta) = \tau\}).$$

The fundamental observation is that, given an input $\sigma \in \mathbb{S}$ and the extra tape $\omega \in \mathbb{B}^{\mathbb{N}}$, an STM running in polynomial time can access a *finite* portion of ω only, the length of which can be bounded by some polynomial $p(|\sigma|)$. Using this fact, we construct $f^\star$ as follows:

1. We introduce the new class **PTF**, made of functions $f : \mathbb{S}^k \times \mathbb{S} \to \mathbb{S}$ computed by a *finite stream Turing machine* (FSTM, for short), the extra tape of which is a finite string.
2. We define a function $h \in \mathbf{PTF}$ such that for any $f : \mathbb{S} \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$ with polynomial bound $p(x)$,
$$f(n, \omega) = h(x, \omega_{p(|x|)}).$$
3. We define $h' : \mathbb{S} \times \mathbb{S} \times \mathbb{O} \to \mathbb{S}$ such that,
$$h'(x, y, \eta) = h(x, y).$$
By an encoding of FSTMs we show that $h' \in \mathcal{POR}$. Moreover h' can be defined *without* using the query function, since the computation of h' never looks at η.
4. Finally, we define an *extractor function* $e : \mathbb{S} \times \mathbb{O} \to \mathbb{S} \in \mathcal{POR}$, which mimics the prefix extractor $\omega_{p(|x|)}$, having as its outputs *the same distributions* of all possible prefixes of ω, even though within a different space.[15] This is obtained by exploiting a bijection $dyad : \mathbb{S} \to \mathbb{N}$, ensuring that for each $\omega \in \mathbb{B}^{\mathbb{N}}$, there is an $\eta \in \mathbb{B}^{\mathbb{S}}$ such that any prefix of ω is an output of $e(y, \eta)$, for some y. Since $\mathcal{POR}$ is closed under composition, we finally define
$$f^\star(x, \eta) := h'(x, e(x, \eta), \eta).$$

From $\mathcal{POR}$ to SFP

In order to simulate functions of $\mathcal{POR}$ via STMs we observe not only that these two models invoke oracles of different shape, but also that the former can manipulate such oracles in a more liberal way:

- STMs query the oracle before each step is produced. By contrast functions of $\mathcal{POR}$ may invoke the query function $Q(x, \eta)$ freely during computation. We call this access policy *on demand*.
- STMs query a new bit of the oracle at each step of computation, and cannot access previously observed bits. We call this access policy *linear*. By contrast, functions of $\mathcal{POR}$ can query the same bits as many times as needed.

Consequently, a direct simulation of $\mathcal{POR}$ via STMs is challenging even for a basic function like $Q(x, \eta)$. So, again, we follow an indirect path: we pass through a chain of simulations, dealing with each of these differences separately.

[15] Recall that $\eta \in \mathbb{B}^{\mathbb{N}}$, while the second argument of e is in $\mathbb{O}$.

1. First, we translate $\mathcal{POR}$ into an imperative language SIFP_{RA} inspired by Winskel's IMP [230], with the same access policy as $\mathcal{POR}$. SIFP_{RA} is endowed with assignments, a `while` construct, and a command $\texttt{Flip}(e)$, which first evaluates e to a string σ and then stores the value $\eta(\sigma)$ in a register. The encoding of oracle functions in SIFP_{RA} is easily obtained by induction on the function algebra.

2. Then, we translate SIFP_{RA} into another imperative language, called SIFP_{LA}, associated with a *linear* policy of access. SIFP_{LA} is defined like SIFP_{RA} except for $\texttt{Flip}(e)$, which is replaced by the new command $\texttt{RandBit}()$ generating a random bit and storing it in a register. A weak simulation from SIFP_{RA} into SIFP_{LA} is defined by progressively constructing an *associative table* containing pairs in the form (string, bit) of past observations. Each time $\texttt{Flip}(e)$ is invoked, the simulation checks whether a pair (e, b) had already been observed. Otherwise it increments the table by producing a new pair $(e, \texttt{RandBit}())$. This is by far the most complex step of the whole simulation.

3. The language SIFP_{LA} can be translated into STMs. Observe that the access policy of SIFP_{LA} is still on-demand: $\texttt{RandBit}()$ may be invoked or not before executing the instruction. So, we first consider a translation from SIFP_{LA} into a variant of STMs admitting an on-demand access policy – that is, a computation step may or may not access a bit from the extra tape. Then, the resulting program is encoded into a regular STM. Observe that we cannot expect that the machine $\mathscr{M}_{\mathcal{S}}{}^{\dagger}$ simulating an on-demand machine $\mathscr{M}_{\mathcal{S}}$ will produce *the same* output and oracle. Rather, as in many other cases, we show that $\mathscr{M}_{\mathcal{S}}{}^{\dagger}$ can be defined so that, for any $\sigma, \tau \in \mathbb{S}$, the sets $\{\omega \mid \mathscr{M}_{\mathcal{S}}{}^{\dagger}(\sigma, \omega) = \tau\}$ and $\{\omega \mid \mathscr{M}_{\mathcal{S}}(\sigma, \omega) = \tau\}$ have the same measure.

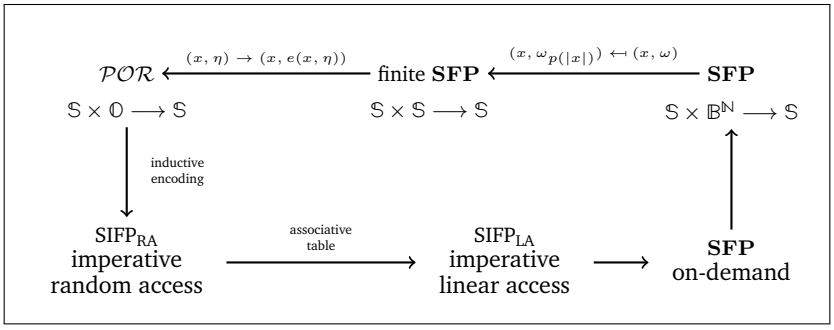


Figure 13.5: Equivalence between $\mathcal{POR}$ and **SFP** 13.4.1

Concluding the Proof.

These ingredients are enough to conclude the proof as outlined in Figure 13.4.3, and to relate poly-time random functions and Σ_1^b-formulae of $\mathbf{RS}_2^1$.

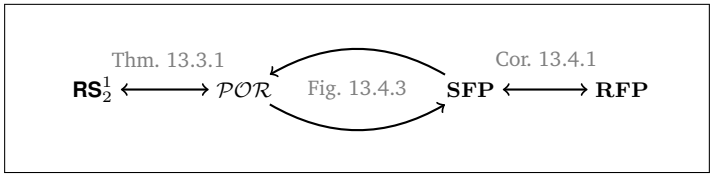


Figure 13.6: Proof Sketch of Theorem 13.4.1

13.5 Arithmetical Characterization of BPP

In this section we provide a logical characterization of the probabilistic (and semantic) class $\mathbf{BPP}$. This is done by putting together Theorem 13.4.1 with measure quantifiers inspired by those for **MQPA**. As seen, the theorem above shows that a class of arithmetical formulae – those which are Σ_1^b-representable in $\mathbf{RS}_2^1$ – precisely corresponds to that of poly-time computable ones, while a measure-quantified language allows us to express error bounds for probabilistic algorithms in a purely logical way.

13.5.1 From Standard to Randomized Classes

The possibility of describing complexity classes within the language of logic and mathematics certainly offers a better understanding of the nature of such classes. From the 1970s on – also inspired by studies in descriptive complexity [78, 45, 116, 138] – the logical characterization of several fundamental classes made it possible to consider them from a new viewpoint, less dependent on concrete machine models and explicit resource bounds. Indeed, characterizing a class via simple enough proof- or recursion-theoretical systems means being able to *enumerate* the problems in the class, thus devising a sound and complete language for it and, from this, also type systems and static analysis methodologies were derived [113].

Among the various classes of problems with which complexity has been concerned, the ones defined on the basis of *randomized* algorithms have appeared difficult to be logically captured. Nevertheless, probabilistic classes – for example $\mathbf{BPP}$ or $\mathbf{ZPP}$ – are nowadays very important and a better understanding of their nature and relations would be highly desirable. In particular, $\mathbf{BPP}$ can be seen as *the* class of feasible problems, and most complexity theorists conjecture that it actually coincides with $\mathbf{P}$. Yet, it appears as pretty different from $\mathbf{P}$. Indeed, it is a *semantic* class: for a randomized algorithm to be in $\mathbf{BPP}$, it is not

enough to be efficient, but it's also required to be not "too *entropic*", i.e., once an input is fixed, the two possible output values are returned with probabilities that are not "too similar" to one another.

Definition 13.5.1. Let $L \subseteq \mathbb{S}$ be a language and f_L its characteristic function, then $L \in \mathbf{BPP}$ when there is a poly-time PTM $\mathscr{M}_{\mathscr{P}}$ such that, for any $\sigma \in L$,

$$\textsc{Prob}[\mathscr{M}_{\mathscr{P}}(\sigma) = f_L(\sigma)] \geq \frac{2}{3}.$$

So, to check whether a language belongs to the class $\mathbf{BPP}$, one has to look for a randomized algorithm to satisfy *both* a polynomial *resource* bound and a (uniform) *error* bound.

As seen in Section 11.3.1, semantic classes, as $\mathbf{BPP}$ is, are more challenging to be *logically* captured than syntactic ones. Indeed, the sparse contributions offering characterizations of probabilistic classes via logical tools either are, so-to-say, semantical in nature [61, 76] – namely, they do not capture the limitations of the error probability *within* the logical *system*[16] – or they deal with classes like $\mathbf{PP}$, which are not semantic [59, 60]. In the following Section 13.5.2, we make a step forward to a proper logical characterization of these randomized, semantic classes, relying both on the power of measure quantifiers introduced in Chapter 12, together with Theorem 13.4.1

13.5.2 Characterizing BPP

The last ingredient consists in simply showing that reasoning about error bounds can be internalized into a logical language slightly extending that of $\mathcal{RL}$. Indeed, as seen, by Definition 13.2.8, any formula of $\mathcal{RL}$ F is associated with a measurable set, $[\![F]\!] \subseteq \mathbb{O}$. So, a natural idea is that of enriching $\mathcal{RL}$ with measure quantifiers inspired by those of Chapter 12. So, we introduce a new language $\mathcal{RL}^{\mathsf{MQ}}$, which is an the extension of $\mathcal{RL}$ with measure-quantified formulae of the form $\mathbf{C}^{t/s}F$, where t and s are terms of $\mathcal{RL}$. Formally, the grammar for terms is exactly as in Definition 13.2.2, while that for formulae is below.

Definition 13.5.2 (Formulae of $\mathcal{RL}$)**.** *Formulae of* $\mathcal{RL}^{\mathsf{MQ}}$ are defined as follows:

$$F ::= \mathtt{Flip}(t) \;\Big|\; t = s \;\Big|\; \neg F \;\Big|\; F \triangle F \;\Big|\; (\exists x)F \;\Big|\; (\forall x)F \;\Big|\; \mathbf{C}^{t/s}F,$$

where t, s are $\mathcal{RL}$-terms and $\triangle \in \{\vee, \wedge\}$.

As predictable, also semantics for terms is as in Definition 13.2.5, and formulae are interpreted extending Definition 13.2.8, with

$$[\![\mathbf{C}^{t/s}F]\!] := \begin{cases} \mathbb{O} & \text{if } \mu_{\mathscr{C}}([\![F]\!]) \geq \frac{\mathbb{1}^{[\![t]\!]}}{\mathbb{1}^{[\![s]\!]}} \\ \emptyset & \text{otherwise.} \end{cases}$$

[16] A partial exception was offered by Jerábek's proposal [118]. He defined a syntactic approach to probabilistic poly-time programs in the context of **BA** and introduced a notion of "definable $\mathbf{BPP}$-problem", relative to some bounded theory, and based on an arithmetical encoding of approximate counting problems.

Notation 13.5.1. For readability's sake, we abbreviate $\mathbf{C}^{1^n/1^m}F$ as $\mathbf{C}^{n/m}F$, with $n, m \in \mathbb{N}$.

Given that Theorem 13.4.1 allows us to logically internalize probabilistic algorithms computed by resource-bounded machines, now with $\mathcal{RL}^{\mathsf{MQ}}$ we can even express and "keep under control" the probability of error from within the logic itself. In particular, since $\mathbf{BPP} \subseteq \mathbf{PH}$, for any language $L \in \mathbf{BPP}$, the corresponding characteristic function $f_L : \mathbb{S} \to \mathbb{B}$ is represented by the formula $H_L(x, y)$ of **PA**. By suitably adapting results by Buss and Goldreich [34, 101], the formula $H_L(x, y)$ can be taken to be a Σ_3^b-formula of $\mathcal{RL}$ – namely, a formula of the form $(\exists y \preceq t(\vec{x}))(\forall z \preceq u(\vec{x}, y))H'(\vec{x}, y, z)$, where $H'(\vec{x}, y, z)$ is a Σ_1^b-formula – leading to the following characterization

Theorem 13.5.1 (Semantic Characterization of $\mathbf{BPP}$)**.** *For any language $L \subseteq \mathbb{S}$, $L \in \mathbf{BPP}$ when there is a Σ_1^b-formula $F(x, y)$ such that the following conditions hold:*

1. $\boldsymbol{RS}_2^1 \vdash (\forall x)(\exists! y)F(x, y)$
2. *for any $\sigma \in \mathbb{S}$ and $\mathbb{b} \in \mathbb{B}$,* $\vDash \mathbf{C}^{2/3}\big(F(\overline{\sigma}, \overline{\mathbb{b}}) \leftrightarrow H_L(\overline{\sigma}, \overline{\mathbb{b}})\big)$.

Proof. ($\Rightarrow$) Assume $L \in \mathbf{BPP}$ and that $f : \mathbb{S} \to \mathbb{D}(\mathbb{S})$ is a poly-time function computing L with a uniform error bound. By Theorem 13.4.1, there is a Σ_1^b-formula $F(x, y)$ such that for any $\sigma \in \mathbb{S}$ and $\mathbb{b} \in \mathbb{B}$, $\mu\big([\![F(\overline{\sigma}, \overline{\mathbb{b}})]\!]\big) \geq \frac{2}{3}$ when $f(\sigma)(\mathbb{b}) \geq \frac{2}{3}$. Then, for any $\sigma \in \mathbb{S}$, if $f_L(\sigma) = 0$, we deduce $[\![H_L(\overline{\sigma}, \mathsf{0})]\!] = \mathbb{0}$ and $f(\sigma)(\mathbb{b}) \geq \frac{2}{3}$. So, $\mu([\![G(\overline{\sigma}, \mathsf{0})]\!]) \geq \frac{2}{3}$ and we conclude, $\mu([\![F(\overline{\sigma}, \mathsf{0}) \leftrightarrow H_L(\overline{\sigma}, \mathsf{0})]\!]) \geq \frac{2}{3}$. If $f_L(\sigma) = 1$, we argue in a similar way.

($\Leftarrow$) Conversely, assume that (1.) and (2.) hold. Then, there is a function $f \in \mathbf{RFP}$, such that for any $\sigma \in \mathbb{S}$ and $\mathbb{b} \in \mathbb{B}$, $f(\sigma)(\mathbb{b}) = \mu\big([\![F(\overline{\sigma}, \overline{\mathbb{b}})]\!]\big)$. If $f_L(\sigma) = 0$, then $[\![H_L(\overline{\sigma}, \mathsf{0})]\!] = \mathbb{0}$. Thus, by (2.), $[\![F(\overline{\sigma}, \mathsf{0})]\!] \geq \frac{2}{3}$. So, by (1.), we conclude $f(\sigma)(\mathbb{0}) \geq \frac{2}{3}$. Similarly if $f_L(\sigma) = 1$. We conclude that $L \in \mathbf{BPP}$. □

Observe that, although this characterization of $\mathbf{BPP}$ is purely-logical and precise, this approach is still semantic: indeed, when dealing with condition (2.) the entropy check is translated into conditions which cannot be derived within a formal system, but are satisfied in the standard first-order model of arithmetic.

Chapter 14

Conclusion

To the best of our knowledge the project and approach developed in this dissertation are quite new. As a consequence, many problems and challenges are still open.

14.1 Main Contributions

The overall purpose of the present work is to bridge – possibly, in a uniform way – logic and probabilistic computation, so as to deepen our knowledge of both. Therefore, the main contribution of this dissertation is not the introduction of counting and measure-quantified logics *per se*; after all, the idea of considering logics with generalized quantifiers is not completely new. On the contrary, our primary goal consists in showing that, following a "quantitative approach" to logic, we managed to generalize standard achievements in TCS to the probabilistic setting, thus proving that "scalable" logical systems could somehow catch up (quite uniformly) with some old and recent results, for instance in computational complexity and PL theory.

On Counting Complexity. In Part I, we have introduced the counting logic **CPL** which, to the best of our knowledge, is the first logical system endowing **PL** with counting quantifiers. As seen, our main source of inspiration came from computational complexity and, in particular, from Wagner's class operator [227]. Our main contribution here is the investigation of the connection between our logic and counting classes. In particular, we have shown that formulae of **CPL** (in a special form) provide complete problems for each level of $\mathbf{CH}$, thus making **CPL** the "probabilistic counterpart" of **QPL**:

$$\text{Polynomial Hierarchy : } \mathbf{QPL} \iff \text{Counting Hierarchy : } \mathbf{CPL}.$$

Programming Language Theory. In Part II, we have defined the intuitionistic counting logic, **iCPL**, and shown that its computational fragment, $\mathbf{iCPL}_0$,

captures quantitative behavioral properties. Then, the main contribution here consists in the definition of a probabilistic CHC between this logic and a type system that expresses probability of termination. In particular, proofs in $\mathbf{iCPL}_0$ are shown to correspond, in the sense of Curry and Howard, to typing derivations for a randomized extension of the λ-calculus $\Lambda_{\mathbf{PE}}$, so that counting quantifiers "reveal" the probability of termination of the underlying probabilistic programs:

$$\text{simply typed } \lambda_{\to} : \text{intuitionistic } \mathbf{PL} \iff \text{randomized } \lambda\text{-calculus} : \mathbf{iCPL}_0.$$

Moreover, in analogy with what happens in the deterministic case, extending the type system with an intersection operator has led us to a full characterization of termination probability. Indeed, although intersection types do not have a clear logical counterpart, the existence of this extension convinces us that the introduced correspondence is actually meaningful. Then, the other new (and surprising) contribution of this Part is the proof that the peculiar features of probabilistic effects can be managed in an elegant way, using ideas coming from logic.

Measure-Quantified Arithmetic. In Part III, we have showed that promising results also extends to measure-quantified languages of arithmetic. In particular, in Chapter 12, we have presented a quantitative extension of the language of **PA**, able to formalize basic theorems from probability theory, which are not expressible in standard arithmetic, and have proved our randomized version of Gödel's arithmetization. These first achievements also seem to suggest that the language of **MQPA** can actually be the starting point to define an arithmetic theory relating with probabilistic computation as **PA** does in the deterministic case:

$$\text{deterministic computation} : \mathbf{PA} \iff \text{probabilistic computation} : \mathbf{MQPA}.$$

Finally, in Chapter 13, we have introduced a minimal extension of the language of arithmetic, such that bounded formulae provably total in a suitably-defined theory *á la Buss*, called $\mathbf{RS}_2^1$ and defined in this language, precisely capture poly-time random functions. Due to this fact (and together with the notion of measure quantifiers), we have obtained our main result here, namely a new *arithmetical* characterization of the semantic class $\mathbf{BPP}$, obtained by internalizing the error-bound check within the logical system:

$$\text{deterministic classes} : \mathbf{BA} \iff \text{probabilistic classes} : \mathbf{RS}_2^1.$$

Observe, indeed, that the logical characterization of probabilistic semantic classes appears as particularly hard and, so far, not many proposals have appeared in the literature. We think that our work could contribute to the understanding of this problem by showing not only how resource bounded randomized computation can be captured within the language of arithmetic, but also that the latter offers convenient tools to control error bounds, the essential ingredient in the definition of classes like $\mathbf{BPP}$ and $\mathbf{ZPP}$.

14.2 Future and Ongoing Work

As said, our logics are new and their study has just started. Indeed, several questions on their expressive power and on their relations with other systems are still open. On the one hand, connections with popular logics, like **QPL**, and probability and modal systems would deserve further attention. In this regard, a particularly promising direction concerns the comparison between our approach and the ones based on generalized quantifiers (and team semantics), as developed in the framework of finite model theory [135, 134, 136, 72]. On the other hand, the link between $\mathbf{CPL}_0$ and stochastic experiments – as sketched in Section 3.4 or [5] – should be analyzed in a formal way, and extended to more expressive languages, as **CPL** and **MQPA**, possibly able to "simulate" events corresponding, for example, to tossing (arbitrarily) biased coins.

Counting Propositional Logics. In this context, it appears valuable to consider the natural generalization of $\mathbf{CPL}_0$ obtained by switching from the semantics defined in Chapter 3 – based on the canonical cylinder space which associates any cylinder with measure $\frac{1}{2}$ – to *any* cylinder probability space relying on a *well-defined* measure $\nu_{\mathscr{C}}$, for example the one which associates measure $\frac{1}{3}$ to any cylinder $Cyl(i)$. Apparently, in this way both a semantics and a sound proof systems can be defined without substantial changes. Nevertheless, this study has been left for future investigation.[1]

Another promising research area concerns proof theory. Again, here many questions are open. For example, the proof theory of $\mathbf{CPL}_0$ and **CPL** have just been delineated, and their dynamics (namely, the underlying cut-elimination procedure) certainly deserves further investigations. Moreover, as seen, the calculi presented in Chapter 3 and 4 are not purely syntactical – this also making it difficult to study of the complexity of their proof difficult. At the same time, in Chapter 5, the validity of counting formulae is decided accessing an oracle for $\sharp\mathtt{SAT}$, i.e. counting the satisfying models of Boolean formulae. In this context, obtaining a calculus *without semantic conditions* is especially desirable. First ideas possibly guiding its design has been presented in [5], where an effective procedure to measure formulae of $\mathbf{CPL}_0$ is provided, *without appealing for an external source*, so making somehow explicit the task accomplished by the oracle in $\mathbf{LK}_{\mathbf{CPL}_0}$. This would pave the way to the design of calculi *without* semantical hypotheses, and could be the first step in shading new lights on the study of the complexity of deciding counting formulae. Nevertheless, also in this case, the effective study of the calculus has been left for future study.

Also the study of intuitionistic counting logics and probabilistic CHC, as developed in Part II, opens up several new avenues of research. For example, it would be interesting to extend the correspondence provided in Chapter 9 to polymorphic types or to control operators. Another intriguing direction concerns the possibility of studying the system of intersection types, as introduced

[1] First attempts in this direction are sketched in [5, 4].

in Chapter 10, to support program synthesis, again in analogy with what is done in the deterministic framework [117, 110].

Measure-Quantified Languages of Arithmetic. Concerning measure-quantified languages of arithmetic, one of the most compelling problems related to the **MQPA** is the definition of a corresponding sound and (sufficiently expressive) proof system. Furthermore, this language is somehow minimal "by design", in the sense that, as seen in Chapter 12, we considered predicate variables of the form $\texttt{Flip}(\cdot)$ only. Then, it would be somehow natural to generalize our study to more expressive languages, with countably many predicate variables $\texttt{Flip}_a(\cdot)$ and named quantifiers $\mathbf{C}_a^{t/s}$ and $\mathbf{D}_a^{t/s}$, following the path indicated by multivariate **CPL** in Chapter 4.

Concerning randomized bounded theories, we have seen in Chapter 13 that the logical characterization of probabilistic classes – in particular those having a semantic nature – is a great challenge. As part of our ongoing research, we are studying alternative characterizations of $\mathbf{BPP}$ obtained, for example, by internalizing the error-bound check within a logical system which captures our measure-sensitive quantifiers by standard first-order ones. This has also led us to introduce a family of effectively enumerable subclasses of $\mathbf{BPP}$, called $\mathbf{BPP}_{\mathsf{T}}$, consisting of languages captured by PTMs whose underlying error can be proved bounded in the corresponding arithmetical theory **T**. As a paradigmatic consequence of this approach, it seems possible to establish that the polynomial identity testing is in $\mathbf{BPP}_{\mathsf{PA}}$, this providing a first example of *reverse* computational complexity for probabilistic algorithms. Finally, given the tight connections linking bounded arithmetic and proof complexity, another natural direction of this study would concern the applications of randomized bounded theories to probabilistic approaches in this field, think for example of recent investigations on *random resolution refutations* [37, 118, 174].

Bibliography

[1] N. Alechina, M. Mendler, V. de Paiva, and E. Ritter. Categorical and Kripke Sematics for Constructive S5 Modal Logic. In Springer, editor, *Proc. Computer Science Logic (CSL)*, pages 292–307, 2021.

[2] E.W. Allender and K.W. Wagner. Counting Hierarchies: Polynomial Time and Constant Depth Circuits. In *Current Trends in Theoretical Computer Science*, pages 469–483, 1993.

[3] A. Andersson. On Second-Order Generalized Quantifiers and Finite Structures. *Annals of Pure and Applied Logic*, 115(1-3):1–32, 2002.

[4] M. Antonelli. Some Remarks on Counnting Propositional Logic. Available at: `https://arxiv.org/abs/2210.16160`, 2022.

[5] M. Antonelli. Two Remarks on Counting Propositional Logic. In *Proc. BEWARE*, AIxIA Conference, pages 20–32, 2023.

[6] M. Antonelli, U. Dal Lago, D. Davoli, I. Oitavem, and P. Pistone. An Arithmetic Theory to Characterize Poly-Time Random Functions. Available at: `https://arxiv.org/abs/2301.12028`, 2023.

[7] M. Antonelli, U. Dal Lago, and P. Pistone. On Counting Propositional Logic. Available at: `https://arxiv.org/abs/2103.12862`, 2021.

[8] M. Antonelli, U. Dal Lago, and P. Pistone. On Counting Propositional Logic and Wagner's Hierarchy. In CEUR Workshop Proceedings, editor, *Proc. Italian Conference of Theoretical Computer Science (ICTCS)*, volume 3072, pages 107–121, 2021.

[9] M. Antonelli, U. Dal Lago, and P. Pistone. On Measure Quantifiers in First-Order Arithmetic. In L. De Mol, Manea F. Weiermann, A., and D. Fernández-Duque, editors, *Proc. Computability in Europe Conference (CiE)*, pages 12–24, 2021.

[10] M. Antonelli, U. Dal Lago, and P. Pistone. On Measure Quantifiers in First-Order Arithmetic (Long Version). Available at: `https://arxiv.org/abs/2104.12124`, 2021.

[11] M. Antonelli, U. Dal Lago, and P. Pistone. Curry and Howard Meet Borel. *Proc. Symposium on Logic in Computer Science (LICS)*, (45):1–13, 2022.

[12] M. Antonelli, U. Dal Lago, and P. Pistone. Curry and Howard Meet Borel. Available at: `https://arxiv.org/abs/2203.11265`, 2022.

[13] M. Antonelli, U. Dal Lago, and P. Pistone. On Counting Propositional Logic and Wagner's Hierarchy. *Theoretical Computer Science*, forthcoming.

[14] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[15] M. Avanzini, U. Dal Lago, and A. Ghyselen. Type-Based Complexity Analysis of Probabilistic Functional Programs. In *Proc. Symposium on Logic in Computer Science (LICS)*, pages 1–13, Vancouver, BC, Canada, Canada, 2019. IEEE.

[16] J. Avigad and S. Feferman. Gödel's Functional (Dialectica) Interpretation. In S.R. Buss, editor, *Handbook of Proof Theory*, chapter VI, pages 337–405. Elsevier, 1995.

[17] F. Bacchus. Lp, a Logic for Representing and Reasoning with Statistical Knowledge. *Computational Intelligence*, 6(4):209–231, 1990.

[18] F. Bacchus. On Probability Distributions over Possible Worlds. *Machine Intelligence and Pattern Recognition*, 9:217–226, 1990.

[19] F. Bacchus. Probabilistic Belief Logics. In *Proc. 9th European Conference on Artificial Intelligence*, pages 59–64, 1990.

[20] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, 1990.

[21] T. Baker, J. Gill, and Solovay R. Relativizations of the P =? NP Question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[22] S. Bellantoni and S. Cook. A New Recursion-Theoretic Characterization of the Polytime Functions. *Computational Complexity*, 2:97–110, 1992.

[23] N.P. Benton, M. Bierman, and V. de Paiva. Computational Types from a Logical Perspective. *Journal of Functional Programming*, 8(2):177–193, 1998.

[24] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2009.

[25] P. Billingsley. *Probability and Measure*. Wiley, 1995.

[26] M. Boraćić. Probability Sequent Calculi and Entropy Based Nonclassical Logic Classification. *Bulletin of Symbolic Logic*, 25(4):446–447, 2019.

[27] M. Boraćić. Probabilized Sequent Calculus and Natural Deduction System for Classical Logic. In *Probabilistic Extensions of Various Logical Systems*, pages 197–213. Springer International Publishing, 2020.

[28] J. Borgström, U. Dal Lago, A.D. Gordon, and M. Szymczak. A Lambda-Calculus Foundation for Universal Probabilistic Programming. In *Proc. International Conference on Functioal Programming (ICFP)*, pages 33–46, 2016.

[29] O. Bournez and G. Florent. Proving Positive Almost-Sure Termination. In *Proc. International Conference on Rewriting Techinques and Applications (RTA)*, pages 323–337, 2005.

[30] O. Bournez and C. Kirchner. Probabilistic Rewrite Strategies. Applications to ELAN. In *Proc. International Conference on Rewriting Techinques and Applications (RTA)*, pages 252–266, 2002.

[31] F. Breuvart and U. Dal Lago. On Intersection Types and Probabilisitic Lambda Calculi. In *Proc. International Symposium on Principles and Practice of Declarative Programming*, number 8, pages 1–13, 2018.

[32] A. Brunel, M. Gaboardi, D. Mazza, and S. Zdancewic. A Core Quantitative Coefficient Calculus. In Springer, editor, *Proc. ESOP Conference*, pages 351–423, 2014.

[33] H.K. Büning and U. Bubeck. Theory of Quantified Boolean Formulas. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*. IOS Press, 2009.

[34] S.R. Buss. *Bounded Arithmetic*. PhD thesis, Princeton University, 1986.

[35] S.R. Buss. First-Order Proof Theory of Arithmetic. In Elsavier, editor, *Handbook of Proof Theory*. Buss, S.R., 1998.

[36] S.R. Buss and A. Ignjatović. Unprovability of Consistency Statements in Fragments of Bounded Arithmetic. *Annals of Pure and Applied Logic*, 74(3):221–244, 1995.

[37] S.R. Buss, A.L. Kolodziejczyk, and N. Thapen. Fragments of Approximate Computing. *Journal of Symbolic Logic*, 79(2):496–525, 2014.

[38] L. Caires, F. Pfenning, and B. Toninho. Linear Logic Propositions as Session Types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2014.

[39] J.M. Carlyle. Reduced Forms for Stochastic Sequential Machines. *Journal of Mathematical Analysis and Applications*, 7:167–174, 1963.

[40] A.K. Chandra, Kozen D.C., and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[41] A. Church and S. Kleene. Formal Definitions in the Theory of Ordinal Numbers. *Fundamenta Mathematicae*, 28:11–21, 1936.

[42] A. Cobham. The Intrinsic Computational Difficulty of Functions. In North-Holland, editor, *Logic, Methodology and Philosophy of Science II*. Bar-Hillel, Y., 1964.

[43] E.F. Codd. Relational Completeness of Data Base Sublanguages. In *Proc. 6th Courant Computer Science Symposium*, pages 65–98, 1972.

[44] S.A. Cook. The Complexity of Theorem-Proving Procedures. In *Proc. Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

[45] S.A. Cook and R.A. Reckhow. Efficiency of Propositional Proof Systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.

[46] S.A. Cook and A. Urquhart. Functional Interpretations of Feasibly Constructive Arithmetic. *Annals of Pure and Applied Logic*, 63:103–200, 1993.

[47] T. Coquand and M. Hofmann. A New Method for Establishing Conservativity of Classical Systems over Their Intuitionistic Version. *Mathematical Structures in Computer Science*, 9(4):323–333, 1999.

[48] R. Crubillé and U. Dal Lago. On Probabilistic Applicative Bisimullation and Call-by-Value λ-Calculi. In *Proc. of European Symposium on Programming (ESOP)*, pages 209–228, 2014.

[49] P.-L. Curien, M. Fiore, and G. Munch-Maccagnoni. A Theory of Effects and Resources: Adjunction Models and Polarised Calculi. In *Proc. Symposium on Principles of Programming Languages (POPL)*, pages 44–56, 2016.

[50] H. Curry and R. Feys. *Combinatory Logic*. North-Holland, 1958.

[51] H.B. Curry. The Combinatory Foundations of Mathematical Logic. *The Journal of Symbolic Logic*, 7:49–64, 1942.

[52] H.B. Curry. *Foundations of Mathematical Logic*. Mcgraw Hill, 1963.

[53] U. Dal Lago. On Probabilistic λ-Calculi. In G. Barthe, J. Katoen, and A. Silva, editors, *Foundations of Probabilistic Programming*, pages 121–144. Cambridge University Press, 2020.

[54] U. Dal Lago, C. Faggian, B. Valiron, and A. Yoshimizu. The Geometry of Parallelism: Cllassical, Probabilistic, and Quantum Effects. In *Proc. Symposium on Principles of Programming Languages (POPL)*, pages 833–845, 2017.

[55] U. Dal Lago, M. Gabbrielli, and S. Zuppiroli. Probabilistic Recursion Theory and Implicit Computational Complexity. *Scientific Annals of Computer Science*, 24(2):177–216, 2014.

[56] U. Dal Lago and U. Grellois. Probabilistic Termination by Monadic Affine Sized Typing. *ACM Transactions of Programming Languages and Systems*, 41(2):10–65, 2019.

[57] U. Dal Lago, G. Guerrieri, and W. Heijltjes. Decomposing Probabilistic Lambda-Calculi. In *Proc. Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 136–156, 2020.

[58] U. Dal Lago and M. Hofmann. Bounded Linear Logic, Revisited. In Springer, editor, *Proc. International Conference on Typed Lambda calculus and Applications (TLCA)*, pages 80–94, 2009.

[59] U. Dal Lago, R. Kahle, and I. Oitavem. A Recursion-Theoretic Characterization of the Probabilistic Class PP. In *Proc. of International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 1–12, 2021.

[60] U. Dal Lago, R. Kahle, and I. Oitavem. Implicit Recursion-Theoretic Characterization of Counting Classes. *Archive for Mathematical Logic*, 2022.

[61] U. Dal Lago and P. Parisen Toldin. A Higher-Order Characterization of Probabilistic Polynomial Time. *Information and Computation*, 241:114–141, 2015.

[62] U. Dal Lago, D. Sangiorgi, and M. Gabbrielli. On Coinductive Equivalences for Higher-Order Probabilistic Functional Programs. In *Proc. Symposium on Principles of Programming Languages (POPL)*, pages 297–308, 2014.

[63] U. Dal Lago and M. Zorzi. Probabilistic Operational Semantics for the Lambda Calculus. *RAIRO - Theoretical Informatics and Information*, 46(3):413–450, 2012.

[64] U. Dal Lago and S. Zuppiroli. Probabilistic Recursion Theory and Implicit Computational Complexity. *Proc. International Colloquium on Theoretical Aspects of Computing (ICTAC)*, 8687:97–114, 2014.

[65] R. Davies and F. Pfenning. A Modal Analysis of Staged Computation. *Journal of ACM*, 48(3):555–604, 2001.

[66] A.S. Davis. Markov Chains as Random Input Automata. *The American Mathematical Montly*, 68(3):264–267, 1961.

[67] M. Davis. *The Universal Computer: The Road from Leibniz to Turing*. CRC Press, 2011.

[68] D. Davoli. Bounded Arithmetic and Randomized Computation. Master Thesis, `http://amslaurea.unibo.it/26234/`, 2022.

[69] K de Leeuw, E.F. Moore, C.E. Shannon, and N. Shapiro. Computability by Probabilistic Machines. In Princeton University Press, editor, *Automata Studies*, number 34, pages 183–212. Shannon, C.E. and McCarthy, J., 1956.

[70] U. de'Liguoro and A. Piperno. Nondeterministic Extensions of Untyped Lambda-Calculus. *Information and Computation*, 122(2):249–177, 1995.

[71] A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic λ-Calculus and Quantitative Program Analysis. *Journal of Logic and Computation*, 15(2):159–179, 2005.

[72] A. Durnand, A. Haak, J. Kontinen, and H. Vollmer. Descriptive Complexity of #P Functions: A New Perspective. *Journal of Computer and System Sciences*, 116:40–54, 2021.

[73] T. Ehrhard and Tasson C. Probabilistic Call-by-Push Value. *Logical Methods in Computer Science*, 15(1):555–604, 2018.

[74] T. Ehrhard, M. Pagani, and C. Tasson. Full Abstraction for Probabilistic PCF. *Journal of ACM*, 65(4):1–44, 2018.

[75] T. Ehrhard, C. Tasson, and M. Pagani. Probabilistic Coherence Spaces are Fully Abstract for Probabilistic PCF. In *Proc. Symposium on Principles of Programming Languages (POPL)*, pages 309–320, 2014.

[76] K. Eickmeyer and M. Grohe. Randomisation and Derandomisation in Descriptive Complexity Theory. In *Computer Science Logic*. Springer, 2010.

[77] C. Faggian and S. Ronchi della Rocca. Lambda Calculus and Probabilistic Computation. In *Proc. Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2019.

[78] R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In *Proc. SIAM-AMS*, pages 43–73, 1974.

[79] R. Fagin and J.Y. Halpern. Uncertainty, Belief, and Probability. *Computational Intelligence*, 7(3):160–173, 1991.

[80] R. Fagin and J.Y. Halpern. Reasoning about Knowledge and Probability. *Journal of ACM*, 41(2):340–367, 1994.

[81] R. Fagin, J.Y. Halpern, and N. Megiddo. A Logic for Reasoning about Probabilities. *Information and Computation*, 87(1/2):78–128, 1990.

[82] F. Ferreira. Polynomial Time Computable Arithmetic and Conservative Extesions. Ph.D. Dissertation, December 1988.

[83] F. Ferreira. Polynomial-time computable arithmetic. In W. Sieg, editor, *Logic and Computation*, volume 106 of *Contemporary Mathematics*, pages 137–156. AMS, 1990.

[84] G. Ferreira and I. Oitavem. An Interpretation of S^1_2 in Σ^b_1-NIA. *Portugaliae Mathematica*, 63:427–450, 2006.

[85] M. Finger, G. De Bona, and F. G. Cozman. Towards Classifying Propositional Probabilistic Logics. *Journal of Applied Logic*, 12(3):349–368, 2014.

[86] L. Fortnow. Beyond NP: The work and legacy of Larry Stockmeyer. In *Proc. Symposium on Theory of Computing (STOC)*, 2005.

[87] R. Furber, R. Mardare, and M. Mio. Probabilistic Logics Based on Riesz Spaces. *Logical Methods in Computer Science*, 16(1), 2020.

[88] D.R. Ghica and A.I. Smith. Bounded Linear Types in a Resource Semiring. In Springer, editor, *Proc. European Symposium on Programming Languages and Systems (ESOP)*, pages 331–350, 2014.

[89] J.T. Gill. *Probabilistic Turing Machines and Complexity of Computation*. PhD thesis, University of California, Berkeley, 1972.

[90] J.T. Gill. Computational Complexity of Probabilistic Turing Machines. In *Proc. Symposium on Theory of Computing (STOC)*, pages 91–95, 1974.

[91] J.T. Gill. Computational Complexity of Probabilistic Turing Machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[92] J.-Y. Girard. *Interprétation Functionelle et Élimination des coupures dans l'arithmétique d'ordre supérieure*. PhD thesis, Université Paris VII, 1972.

[93] J.-Y. Girard. Between Logic and Quantic: a Tract. In *London Mathemmatical Society Lecture Note Series*, volume 316, pages 346–381. Cambridge University Press, 2004.

[94] J.-Y. Girard, A. Scedrov, and P.J. Scott. Bounded Linear Logic: A Modular Approach to Polynomial-Time Computability. *Theoretical Computer Science*, 97(1):1–66, 1992.

[95] Jean-Yves Girard. *Proof and Types*. Cambridge University Press, 1989.

[96] M. Girlando, S. Negri, and G. Sbardolini. Uniform Labelled Calculi for Conditional and Counterfactual Logics. In *Proc. Workshop on Logic, Language, Information and Computation (WoLLIC)*, pages 248–263, 2019.

[97] K. Gödel. Über Formal Unentscheidbare Sätze der *Principia Mathematica* und Verwandter Systeme. *Monatsch. Math. Phys.*, 38:173–178, 1931.

[98] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standdpunktes. *Dialectica*, 12:280–287, 1958.

[99] K. Gödel. On Undecidable Porpositions of Formal Mathematical Systems. In M. Davis, editor, *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvvable Problems and Computable Functions*. Dover Publications, 1965.

[100] K. Gödel. *On Formally Undecidable Porpositions of* Principia Mathematica *and Related Systems*. Dover Publications, 1992.

[101] O. Goldreich and D. Zuckerman. Another Proof that BPP $\subseteq$ PH (and more). In *Proc. ECCC*, 1997.

[102] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):279–299, 1984.

[103] N.D. Goodman, V. Mansinghka, D.M. Roy, K. Bonawitz, and J.B. Tenenbaum. Church: A Language for Generative Models. In *Proc. Conference on Uncertainty in Artificial Intelligence*, pages 220–229, 2008.

[104] N.D. Goodman and J.B. Tenenbaum. Probabilistic Models of Cognition.

[105] J. Goublult-Larrecq. A Probabilistic and Non-Deterministic Call-By-Push-Value Language. In *Proc. Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2019.

[106] J.Y. Halpern. An Analysis of First-Order Logics for Probability. *Artificial Intelligence*, 46(3):311–350, 1990.

[107] J.Y. Halpern. *Reasoning About Uncertainty*. MIT Press, 2003.

[108] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computation*, 6(5):512–535, 1994.

[109] J. Hartmainis and R.E. Stearns. On the Computational Complexity of Algorithms. *Transactions of the Americal Mathematical Society*, 117:285–306, 1965.

[110] F. Henglein and J. Rehof. Modal Intersection Types, Two-Level Languages, and Staged Synthesis. In *Semantics, Logics, and Calculi: Essays Dedicated to Hanne Riis ielson and Flemming Nielson on the Occasion of Their 60th Birthday*, pages 289–312. Springer.

[111] D. Hilbert. Mathematical Problems. *Bulletin of the American Mathematical Society*, 8(10):437–439, 1902.

[112] R. Hindley and J. Seldin. *Lambda Calculus and Combinators. An Introduction*. Cambridge University Press, 2008.

[113] M. Hofmann. Programming Languages Capturing Commplexity Classes. *SIGACT News*, 31(1):31–42, 2000.

[114] R. Horne. The Sub-Additives: A Proof Theory for Probabilistic Choice Extending Linear Logic. In Leibniz International Proceedings in Informatics, editor, *Proc. Formal Structures for Computation and Deduction (FSCD)*, pages 1–17, 2019.

[115] W.A. Howard. The Formulae-as-Types Notion of Construction. In J. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[116] N. Immerman. *Descriptive Complexity*. Springer, 1999.

[117] B. Düdder Martens M. J. Bessai, A. Dudenhefner and J. Rehof. Combinatory Process Synthesis. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*, pages 266–281. Springer, 2016.

[118] E. Jerábek. Approximate Counting in Bounded Arithmetic. *Journal of Symbolic Logic*, 72(3):959–993, 2007.

[119] C. Jones and G. Plotkin. A Probabilistic Powerdomain for Evaluations. In *Proc. Symposium on Logic in Computer Science (LICS)*, pages 186–195, 1989.

[120] A. Jung and R. Tix. The Troublesome Probabilistic Powerdomain. *Electronic Notes in Theoretical Computer Science*, 13:70–91, 1998.

[121] R.M. Karp. Reducibility among Combinatorial Problems. In *Complexity fo Computer Computations*, pages 85–103. Springer, 1972.

[122] S. Katsumata. Parametric Effect Monads and Semantics of Effect Systems. In *Proc. Principles of Programming Languages (POPL)*, pages 633–645, 2014.

[123] S. Katsumata. A Double Category Theoretic Analysis of Graded Linear Exponential Comonads. In Springer, editor, *Proc. Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 110–127, 2018.

[124] A.S. Kechris. *Classical Descriptive Set Theory*, volume 156 of *Graduate Texts in Mathematics*. Spriger-Verlag, 1995.

[125] S. Kleene. A Note on Recursive Functions. *Bulletin of the American Mathematical Society*, 42:544–546, 1936.

[126] S. Kleene. General Recursive Functions of Natural Numbers. *Mathematische Annalen*, 112:727–742, 1936.

[127] S. Kleene. λ-Definability and Recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

[128] S.C. Kleene. On the Interpretation of Intuitionistic Number Theory. *Journal of Symbolic Logic*, 10(4):109–124, 1945.

[129] S.C. Kleene. The Theory of Recursive Functions, approaching its Centennial. *Bulletin of the American Mathematical Society*, 5(1):43–61, 1981.

[130] N. Kobayashi, U. Dal Lago, and C. Grellois. On the Termination Problem for Probabilistic Higher-Order Recursive Programs. In *Proc. Symposium on Logic in Computer Science (LICS)*, pages 1–14, 2019.

[131] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[132] D. Koller, D. McAllester, and A. Pfeffer. Effective Bayesian Inference for Stochastic Programs. In *Proc. AAAI Conference*, pages 740–747, 1997.

[133] A.N. Kolmogorov. Grundbegriffe der Wahrscheinllichkeitrechnung. In *Ergebnisse Der Mathematik*. 1933.

[134] J. Kontinen. A Logical Characterization of the Counting Hierarchy. *AMC Transactions on Computer Science*, 10(1):1–21, 2009.

[135] J. Kontinen. Definabillity of Second Order Generalized Quantifiers. *Archive for Mathematical Logic*, 49:379–398, 2010.

[136] J. Kontinen and H. Niemisto. Extensions of MSO and the Monadic Counting Hierarchy. *Information and Computation*, 209:1–19, 2011.

[137] D. Kozen. Semantics of Probabilistic Programs. *Journal of Computer and System Sciences*, 53(3):165–198, 1982.

[138] J. Krajicek and P. Pudlak. Propositional Proof Systems, the Consistency of First-Order Theories and the Complexity of Computations. *Journal of Symbolic Logic*, 54(3):1063–1079, 1989.

[139] G. Kreisel. Interpretation of Analysis by Means of Constructive Funcionals of Finite Types. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North-Holland, 1959.

[140] D. Lehmann and S. Shelah. Reasoning with Time and Chance. *Information and Computation*, 53(3):165 – 198, 1982.

[141] L.A. Levin. Universal Sequential Search Problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[142] P. Lindström. First Order Predicate Logic with Generalized Quantifiers. *Theoria*, 32:186–185, 1966.

[143] T. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[144] P. Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. In *Proc. Logic Colloquium*, pages 73–118, 1973.

[145] A. McIver and C. Morgan. Developing and Reasoning about Probabilistic Programs in pGCL. In *Proc. Pernambuco Summmer School on Software Enginneering (PSSE)*, pages 123–155, 2004.

[146] A.R. Meyer and L.J. Stockmeyer. The Equivalence Problem For Regular Expressions with Squaring Requires Exponential Space. In *Proc. Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 125–129, 1972.

[147] A.R. Meyer and L.J. Stockmeyer. Word Problems Requiring Exponential Time (Preliminary Report). In *Proc. Symposium on Theory of Computing (STOC)*, pages 1–9, 1973.

[148] H. Michalewski and M. Mio. Measure Quantifiers in Monadic Second Order Logic. In *Proc. Logical Foundations of Computer Science (LFCS)*, pages 267–282, 2016.

[149] M. Mio, M. Skrzypczak, and H. Michalewski. Monadic Second Order Logic with Measure and Category Quantifiers. *Logical Methods in Computer Science*, 8(2), 2012.

[150] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[151] C. Morgenstern. The Measure Quantifier. *Journal of Symbolic Logic*, 44(1), 1979.

[152] A. Mostowski. On a Generalization of Quantifiers. *Fundamenta Mathematicae*, 44:12–36, 1957.

[153] R. Motowani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[154] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[155] S. Negri and J. von Plato. *Structural Proof Theory*. Cambridge University Press, 2001.

[156] S. Negri and J. von Plato. *Proof Analysis: A Contribution to Hilbert's Last Problem*. Cambridge University Press, 2011.

[157] S. Negri and J. von Plato. From Mathematical Axioms to Mathematical Rules of Proof: Recent Developments in Proof Analysis. In *Proc. Royal Society*, 2019.

[158] N.J. Nilsson. Probabilistic Logic. *Artificial Intelligence*, 28(1):71–87, 1986.

[159] N.J. Nilsson. Probabilistic Logic Revisited. *Artificial Intelligence*, 59(1/2):39–42, 1993.

[160] P. O'Hearn. On Bounched Typing. *Journal of Functional Programming*, 13(4):747–796, 2003.

[161] C.H. Papadimitriou. Games against Nature. *Journal of Computer and System Sciences*, 31(2):288–301, 1985.

[162] C.H. Papadimitriou. *Computational Complexity*. Pearson Education, 1993.

[163] C.H. Papadimitriou and S.K. Zachos. Two Remarks on the Power of Counting. *Theoretical Computer Science*, 145:269–275, 1982.

[164] I. Parberry and G. Schnitger. Parallel computation with threshold functions. *Journal of Commputer and System Sciences*, 36:278–302, 1988.

[165] M. Parigot. $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction. In *Proc. Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 190–201, 1992.

[166] R.J. Parikh. Language Generating Devices. *Quarterly Progress Report*, 60:199–212, 1961.

[167] S. Park. A Calculus for Probabilistic Languages. *ACM SIGPLAN Notices*, 38(3):38–49, 2003.

[168] S. Park, F. Pfanning, and S. Thrun. A Probabilistic Language Based on Sampling Functions. *ACM Transactions of Programming Languages and Systems*, 31(4):1–46, 2008.

[169] G. Peano. *Arithmetices Principia: Nova Methodo*. Harvard University, 1889.

[170] J. Pearl. *Probabilistic Reasoning in Intelligent Systems. Networks of Plausible Inference*. Elsavier, 1988.

[171] A. Pfeffer. IBAL: A Probabilistic Relational Programminng Language. In *International Joint Conference on Artificial Intelligence*, 2001.

[172] G. Plotkin and J. Power. Algebraic Operations of Generic Effects. *Applied Categorical Structures*, 11:69–94, 2003.

[173] G.D. Plotkin. Call-by-Name, Call-by-Value and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975.

[174] P. Pudlak and N. Thapen. Random Resolution Refutations. *Computational Complexity*, 28:185–239, 2019.

[175] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[176] M. O. Rabin. Probabilistic Automata. *Information and Computation*, 6(3):230–245, 1963.

[177] N. Ramsey and A. Pfeffer. Stochastic Lambda Calculus and Monads of Probability Distributions. In *Proc. Symposium on Principles of Programming Languages (POPL)*, pages 154–165, 2002.

[178] M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 62(1):107–136, 2006.

[179] R.M. Robinson. An Essentially Undecidable Axiom System. In *Proc. International Congress of Mathematics*, pages 729–730, 1950.

[180] N. Saheb-Djaromi. Probabilistic LCF. In ACM Press, editor, *Proc. International Symposium on Mathematical Foundations of Computer Science*, pages 154–165, 1978.

[181] E.S. Santos. Maximin Automata. *Information and Control*, 13:363–377, 1968.

[182] E.S. Santos. Maximin Sequential-Like Machines and Chains. *Mathematical Systems Theory*, 3(4):300–309, 1969.

[183] E.S. Santos. Probabilistic Turing Machines and Computability. *Proc. American Mathematical Society*, 22(3):704–710, 1969.

[184] E.S. Santos. Computability by probabilistic Turing Machines. *Transactions of the American Mathematical Society*, pages 159–165, 1971.

[185] E.S. Santos and W.G. Wee. General Formulation of Sequential Machines. *Information and Control*, 12(1):5–10, 1968.

[186] T. Sato and Y. Kameya. PRISM: A Symbolic-Statistical Modeling Language. In *Proc. International Joint Conference on Artificial Intelligence*, volume 2, 1997.

[187] M. Schönfinkel. Über die Bausteine der Mathematischen Logik. *Mathematische Annalen*, 92(3–4):305–316, 1924.

[188] R. Segala. A Compositional Trace-Based Semantics for Probabilistic Automata. In *Proc. CONCUR*, pages 234–248, 1995.

[189] W. Shakespeare (or the typist monkey). *The Tragedy of Macbeth*. 1605/1608.

[190] J.H. Siekman. Computational Logic. In J.H. Siekmann, editor, *Handbook of the History of Logic: Computational Logic*, volume 9, pages 15–30. Elsavier, 2014.

[191] J. Simon. *On Some Central Problems in Computational Complexity*. PhD thesis, Cornell University, 1975.

[192] J. Simon. On the Difference Between One and Many. In *Proc. International Colloquium on Automata, Languages and Programming*, pages 480–491, 1977.

[193] J. Simon. On Tape-Bounded Probabilistic Turing Machine Acceptors. *Theoretical Computer Science*, 16:75–91, 1981.

[194] S. Simpson. *Subsystems of Second Order Arithmetic*. Cambridge University Press-Press, 2009.

[195] P. Smith. *An Introduction to Gödel's Theorems*. Cambridge University Press, 2013.

[196] R.I. Soare. Computability and Recursion. *Bulletin of Symbolic Logic*, 2:284–321, 1996.

[197] R.I. Soare. *Turing Computability: Theory and Applications*. Springer, 2016.

[198] M.H. Sorensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149. Elsevier, 2006.

[199] C.I. Steinhorn. Borel Structures and Measure and Category Logics. volume 8, pages 579–596. Springer-Verlag, 1985.

[200] L. Stockmeyer. On Approximation Algorithms for ♯P. *SIAM Journal on Computing*, 14(4):849–861, 1985.

[201] L.J. Stockmeyer. The Polynomial-Time Hiearchy. *Theoretical Computer Science*, 3:1–22, 1977.

[202] Ehrhard T., J.-Y. Girard, P. Ruet, and P. Scott, editors. *Linear Logic in Computer Science*, volume 316 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2004.

[203] W.W. Tait. Intensional Interpretation of Functionals of Finite Type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

[204] A. Tarski, A. Mostowski, and R.M. Robinson. *Undecidable Theories*, volume 30. North-Holland, 1953.

[205] M. Tedre. *The Science of Computing: Shaping a Discipline*. CHR, 2014.

[206] R. Thiele. Hilbert's Twenty-Fourth Problem. *The American Mathematical Montly*, 110(1):1–24, 2003.

[207] M.A. Thornton, R. Drechsler, and D.M. Miller. *Logic Verification*, pages 201–230. Springer, 2001.

[208] S. Thrun. *Exploring Artificial Intelligence in the New Millennium*, chapter Robotic Mapping: A Survey, pages 1–35. Morgan Kaufmann Publishers Inc., 2003.

[209] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2006.

[210] S. Toda. On the Computational Power of PP and ♯P. In *30th Annual Symposium on Foundations of Computer Science*, pages 514–519, 1989.

[211] S. Toda. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[212] J. Torán. An Oracle Characterization of the Counting Hierarchy. In *Proceedings. Structure in Complexity Theory Third Annual Conference*, pages 213–223, 1988.

[213] J. Torán. Counting the Number of Solutions. In Springer, editor, *Proc. Mathematical foundations of Computer Science*, pages 121–134, 1990.

[214] J. Torán. Complexity classes defined by counting quantifiers. *Journal of the ACM*, 38(3):753–774, 1991.

[215] A. Turing. On Computable Numbers, with an Application to the *Entscheidungsproblem*. In *Proc. London Mathematical Society*, volume 42, pages 230–265, 1936.

[216] A. Turing. Computability and λ-Definability. *Journal of Symbolic Logic*, 2:153–163, 1937.

[217] L.G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[218] J. van Heijenoort. *From Frege to Gödel: a Source Book in Mathematical Logic, 1879-1931*. Harvard University Press, 1967.

[219] D. van Melkebeek. A Survey on Lower Bounds for Satisfiability and Related Problems. *FnT-TCS*, 2:197–303, 2007.

[220] D. van Melkebeek and T. Watson. A Quantum Time-Space Lower Bound for the Counting Hierarchy. available at: `https://minds.wisconsin.edu/handle/1793/60568`, 2007.

[221] J. von Plato. Gentzen's Proof System: Byproducts in a Work of Genius. *Bulletin of Symbolic Logic*, 18(3):313–367, 2012.

[222] J. von Plato. *The Great Formal Machinery Works*. Princeton, 2017.

[223] J. von Plato. What Are the Axioms for Numbers and Who Invented Them? In De Gruyter, editor, *Philosophy of Logic and Mathematics*. Mras, G.M. and Weingartner, P. and Ritter, B., 2019.

[224] P. Wadler. Propositions as Sessions. *AMC SIGPLAN Notices*, 47(9):273–286, 2012.

[225] K.W. Wagner. Compact Descriptions and the Counting Polynomial-Time Hierarchy. In *Frege Conference 1984: Proc. International Conference held at Schwerin*, pages 383–392, 1984.

[226] K.W. Wagner. Some Observations on the Connection Between Counting and Recursion. *Theor. Comput. Sci.*, 47:131–147, 1986.

[227] K.W. Wagner. The Complexity of Combinatorial Problems with Succinct Input Representation. *Acta Informatica*, 23:325–356, 1986.

[228] D. Wang, D.M. Kahn, and J. Hoffmann. Raising Expectations: Automating Expected Cost Analysis with Types. In *Proc. ICFP*, 2020.

[229] J. Warrell and M.B. Gerstein. Dependent Type Networks: A Probabilistic Logic via the Curry-Howard Correspondence in a System of Porbabilistic Dependent Types. unpublished manuscript, http://papers.gersteilab.org/papers/UDL-19/index-all.html, 2018.

[230] G. Winskel. *The Formal Semantics of Programming Languages*. The MIT Press, 1993.

[231] C. Wrathall. Complete Sets and the Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

[232] S.K. Zachos. Probabilistic Quantifiers and Games. *Journal of Computer and System Sciences*, 36(3):433–451, 1988.

[233] S.K. Zachos and H. Heller. A Decisive Characterization of BPP. *Information and Control*, pages 125–135, 1986.

[234] N. Zyuzin and A. Nanevski. Contextual Modal Types for Algebraic Effects and Handlers. In *Proc. ICFP*, pages 1–29, 2021.