Université Sorbonne Paris Cité

Thèse de Doctorat

de l'Université Sorbonne Paris Cité

Préparée à l'Université Paris-Diderot

École Doctorale de Sciences Mathématiques de Paris Centre ED386

Institut pour la Recherche en Informatique Fondamentale (IRIF)

BEHAVIORAL DISTANCES FOR PROBABILISTIC HIGHER-ORDER PROGRAMS

Par Raphaëlle Crubillé

Dirigée par Thomas Ehrhard et Ugo Dal Lago

Présentée et soutenue publiquement à Paris le 20 juin 2019

Examinateur :	Catuscia Palamidessi	Directrice de recherche, INRIA
Examinateur :	Jean Goubault-Larrecq	Professeur des Universités, ENS Cachan
Examinateur :	Christine Tasson	Maître de conférences, Université Paris Diderot
Rapporteur :	Paul Blain Levy	Reader, University of Birmingham
Rapporteur :	Alex Simpson	Redni professor, Univerza v Ljubljani
Directeur de thèse :	Thomas Ehrhard	Directeur de recherche, CNRS
Co-directeur de thèse :	Ugo Dal Lago	Professore associato, Università di Bologna

Résumé

Titre : Distances comportementales pour des programmes probabilistes d'ordre supérieur.

Mots-clés : *Computations probabilistes d'ordre supérieur, sémantique des langages de programmation, bisimulation applicative, espace cohérents probabilistes, distances entre programmes*

Cette thèse est consacrée à l'étude d'équivalences et de distances comportementales destinées à comparer des programmes probabilistes d'ordre supérieur. Le manuscrit est divisé en trois parties. La première partie consiste en une présentation des langages probabilistes d'ordre supérieur, et des notions d'équivalence et de distance contextuelles pour de tels langages.

Dans une deuxième partie, on suit une approche *opérationnelle* pour construire des notions d'équivalences et de métriques plus simples à manipuler que les notions contextuelles : on prend comme point de départ les deux équivalences comportementales pour le lambda-calcul probabiliste équipé d'une stratégie d'évaluation basée sur *l'appel par nom* introduites par Dal Lago, Sangiorgi and Alberti : ces derniers définissent deux équivalences– la trace équivalence, et la bisimulation probabiliste, et montrent que pour ce langage, la trace équivalence permet de complètement caractériser–i.e. est *pleinement abstraite*– l'équivalence contextuelle, tandis que la bisimulation probabiliste est une approximation correcte de l'équivalence contextuelle, mais n'est pas pleinement abstraite. Dans la partie opérationnelle de cette thèse, on montre que la bisimulation probabiliste redevient pleinement abstraite quand on remplace la stratégie d'évaluation par nom par une stratégie d'évaluation *par valeur*. Le reste de cette partie est consacrée à une généralisation *quantitative* de la trace équivalence, i.e. une *trace distance* sur les programmes. On introduit d'abord une trace distance pour un λ-calcul probabiliste affine, i.e. où le contexte peut utiliser son argument au plus une fois, et ensuite pour un λ-calcul probabiliste où les contextes ont la capacité de copier leur argument; dans ces deux cas, on montre que les distances traces obtenues sont pleinement abstraites.

La troisième partie considère deux modèles dénotationnels de langages probabilistes d'ordre supérieur : le modèle des espaces cohérents probabiliste, dû à Danos et Ehrhard, qui interprète le langage obtenu en équipant PCF avec des probabilités discrètes, et le modèle des cônes mesurables et des fonctions stables et mesurables, développé plus récemment par Ehrhard, Pagani and Tasson pour le langage obtenu en enrichissant PCF avec des probabilités continues. Cette thèse établit deux résultats sur la structure de ces modèles. On montre d'abord que l'exponentielle de la catégorie des espaces cohérents peut être exprimée en utilisant le comonoide commutatif libre : il s'agit d'un résultat de généricité de cette catégorie vue comme un modèle de la logique linéaire. Le deuxième résultat éclaire les liens entre ces deux modèles : on montre que la catégorie des cônes mesurables et des fonctions stables et mesurable est une extension conservatrice de la catégorie de co-Kleisli des espaces cohérents probabilistes. Cela signifie que le modèle récemment introduit par Ehrhard, Pagani et Tasson peut être vu comme la généralisation au cas continu du modèle de PCF équipé avec des probabilités discrètes dans les espaces cohérents probabilistes.

Abstract

Title: Behavioral distances for probabilistic higher-order programs.

Keywords: *probabilistic higher-order computations, semantics of programming languages, applicative bisimulation, probabilistic coherence spaces, distance for programs.*

The present thesis is devoted to the study of behavioral equivalences and distances for higher-order probabilistic programs. The manuscript is divided into three parts. In the first one, higher-order probabilistic languages are presented, as well as how to compare such programs with context equivalence and context distance.

The second part follows an operational approach in the aim of building equivalences and metrics easier to handle as their contextual counterparts. We take as starting point here the two behavioral equivalences introduced by Dal Lago, Sangiorgi and Alberti for the probabilistic lambda-calculus equipped with a call-by-name evaluation strategy: the trace equivalence and the bisimulation equivalence. These authors showed that for their language, trace equivalence completely characterizes context equivalence–i.e. is fully abstract, while probabilistic bisimulation is a sound approximation of context equivalence, but is not fully abstract. In the operational part of the present thesis, we show that probabilistic bisimulation becomes fully abstract when we replace the call-by-name paradigm by the call-by-value one. The remainder of this part is devoted to a quantitative generalization of trace equivalence, i.e. a *trace distance* on programs. We introduce first e trace distance for an affine probabilistic λ-calculus–i.e. where a function can use its argument at most once, and then for a more general probabilistic λ-calculus where functions have the ability to duplicate their arguments. In these two cases, we show that these trace distances are fully abstract.

In the third part, two denotational models of higher-order probabilistic languages are considered: the Danos and Ehrhard's model based on probabilistic coherence spaces that interprets the language PCF enriched with discrete probabilities, and the Ehrhard, Pagani and Tasson's one based on measurable cones and measurable stable functions that interpret PCF equipped with continuous probabilities. The present thesis establishes two results on these models structure. We first show that the exponential comonad of the category of probabilistic coherent spaces can be expressed using the free commutative comonoid: it consists in a genericity result for this category seen as a model of Linear Logic. The second result clarify the connection between these two models: we show that the category of measurable cones and measurable stable functions is a conservative extension of the co-Kleisli category of probabilistic coherent spaces. It means that the recently introduced model of Ehrhard, Pagani and Tasson can be seen as the generalization to the continuous case of the model for PCF with discrete probabilities in probabilistic coherent spaces.

Introduction

Context

Randomized algorithms—i.e. algorithms which are allowed to perform probabilistic choices, formalized for instance as throwing a dice—are present from the beginning in computer science [36, 96]: they are for instance central to most algorithms in computational cryptography (where, e.g., secure public key encryption schemes are bound to be probabilistic [57]). Probabilistic algorithms may also allow to solve a problem more efficiently—in less time—than deterministic algorithms, with the payoff that the output coincides with the correct answer with high probability, but not necessarily always. This idea is reflected for instance in computational complexity, e.g. the `BPP`complexity class consists of all problems that are solved by polynomial-time algorithms with probability at least $\frac{2}{3}$. For instance, we know that the polynomial identity testing problem—i.e. the problem of deciding whether some given polynomial with several variables is the 0-polynomial—is known to be in `BPP`, but it is still not known whether it is in P. Recently, there was a renewed—and widened—interest in the role of probability theory in computing, in particular in the *machine-learning* community. At the core of many machine-learning algorithms, there is indeed *Bayesian reasoning*, that can be roughly summarized as the following paradigm: we start from statistical models—i.e. a network of probability distributions over the set of possible events—that express our *beliefs* about the world, and then we update those models using *Bayesian inference*—this way developing the ability to condition values of variables via observations. This kind of reasoning usually focuses on statistical models expressed using *continuous* probability distributions, for instance probability distributions on reals.

In order to formally write and execute probabilistic algorithms, both abstract and concrete programming languages have been developed. Our focus here is on functional programming languages that are also *higher-order*, i.e. where subroutines can take functions in input and produce functions in output. Higher-order languages enjoy a strong connection with logic, formalized by the Curry-Howard correspondence, that originally connects typed λ-calculus—a minimal deterministic higher-order language without ground types or side-effects—with intuitionistic logic. This connection with logic makes higher-order languages a natural place to develop mathematical tools guaranteeing programs to be well-behaved—where what well-behaved means may depend on which properties we are interested in checking, for instance that the execution terminates, or that there is no deadlock. Randomized algorithms can be written in language like Ocaml or Haskell, that comes with a `random` primitive—that returns a real number chosen randomly between 0 and 1. Several concrete probabilistic higher-order languages handling Bayesian reasoning have also been introduced such as Church [58], or more recently Anglican [117].

In the present thesis, we focus on a problem central to program semantics: *comparing* two programs implementing distinct algorithms, in order to decide whether they are

equivalent. This problem occurs for instance when we want to know whether a program implements a given specification, or when we do code optimization, and we want to be sure that we do not trigger any major change in the program behavior. Observe that, even in the case of deterministic programs, it is not enough to simply *execute* two programs once to decide if they always behave the same way, since the behavior of the program can depend on the external environment: we need to check that *no* environment can force the two programs to behave differently. In the case of higher-order programming languages, this can be illustrated by taking programs distinguishers to be program themselves. This idea was formalized in *Morris's context equivalence* [87], where the external environment is modeled as a *context*, i.e. a program that contains a hole; the interaction of a program with its environment is then modeled as filling the context's hole with the program. Two programs are *context equivalent* when their observable behavior—i.e. what an external observer can see during the execution—is the same in *any context*. Morris's context equivalence is an almost universally accepted criterion for program equivalence, but is complicated to use in practice, because *all* possible contexts need to be considered. From there, the goal becomes to find handier ways to compare programs, that are *sound with respect to Morris's context equivalence*—i.e. they give us stronger or equal guarantees than context equivalence—or ideally even *fully abstract*—i.e. fully characterizes context equivalence.

Program analysis—including the study of Morris's context equivalence—asks for a formalization of the behavior of programs. In the present thesis, we will focus on two approaches: the *operational view* where we mostly see programs as *processes* that can be executed, and the *denotational view*, designed to abstract away from the implementation details and the way the program is actually executed, in order to focus on its *mathematical meaning*. Denotational semantics was introduced by the pioneering work of Scott and Strachey [109, 111] on domain theory, with the aim of building *denotational models*, i.e. mathematical universes in which programs can be *interpreted*. In those universes, there are often objects that are not the interpretation of any program, and this gap can be used to inform the development of new programming constructs. Program interpretation is usually built in a *compositional* way—i.e. the interpretation of a program can be recovered from the interpretation of its sub-programs. Two programs are considered equivalent in a denotational model when their denotational interpretations coincide. Operational and denotational approaches should be seen as complementary: while operational semantics allows to formalize—hence to reason about—how a program is executed, denotational semantics aim to *understand* what a program actually do. When introducing a denotational model for some language, this connection need to be made formal by proving an *adequacy theorem*, that states that the program interpretation in the denotational model is invariant under one step of execution on the operational side. Beyond the adequacy result, we can look for stronger connections between operational and denotational worlds, by looking at the ability of a denotational model to talk about the operational context equivalence. A denotational model is *sound* when, in order to know whether two program of our language are context equivalent, we can compare their interpretations in our model: if the two interpretations coincide, then we know that the two programs are equivalent. The model is *fully-abstract* when we obtain this way a characterization of context equivalence, meaning that it is *enough* to look at program interpretations in the model to decide context equivalence.

Among the operational methods for studying Morris context equivalence, we will focus in the present thesis on *coinductive* methodologies [103], that were first used in computer science for concurrency. Those techniques have been imported into the study of sequen-

tial higher-order languages by Abramsky's applicative bisimulation [2] in the setting of λ-calculus. Applicative bisimulation has been generalized to a wide range of higher-order languages: for instance languages with ground types [92], non-deterministic languages [76], stateful languages [106, 69]... Abramsky's applicative bisimulation consists in building a transition system that models the operational semantics of the language *interactively*: this way we obtain a labeled graph that represents all the possible interactions between a program and the external environment. Then, we use a built-in—usually coinductive—notion of equivalence for the corresponding class of transition systems, this way obtaining an equivalence on programs. In λ-calculus, the unique way for the environment to interact with a program is to apply it to some argument, hence the name *applicative* bisimulation. This approach enables us to use coinductive techniques—as for instance up-to reasoning [103]—developed as enhancement of the bisimulation proof method, and thus gives us new tools to handle context equivalence between programs. While denotational semantics aims to *understand* the *structure* of those functions that can be implemented by programs—thus enabling the use of adapted mathematical tools—in the contrary coinductive methodologies transform higher-order programs into first-order processes—i.e. forgetting about functions in order of being able to reason on a transition graph. By the way, the quest for soundness and full abstraction also holds for operationally inspired techniques for program equivalence.

State of the Art.

The operational semantics of higher-order probabilistic languages have been studied in depth in recent years, both for discrete and continuous probabilistic programming languages, with two main approaches. The first one handles explicitly probabilistic distributions, using a *monadic* type system: if σ is a type, then $P\sigma$ is the type for probability distributions over type σ. Since probability distributions have a *monadic structure*, this approach fits into Moggi's computational λ-calculus [86]: a probabilistic program taking objects of type σ in input and outputing objects of type τ can be typed as $\sigma \to P\tau$, meaning that it can be seen as a process taking non-probabilistic data of type σ, and outputing a probability distribution over data of type τ. It also gives a roadmap to then find a denotational semantics, by building a *categorical model of computational λ-calculus*, as defined by Moggi in [86]. This approach has been for instance used for the discrete language used by Jones and Plotkin [68], or the continuous languages introduced by Park, Pfenning and Thrun [91]. In the present thesis, we will focus on another approach, that consists in seeing *all* programs as possibly probabilistic, hence treating the same way non-probabilistic and probabilistic data. It is the case of the untyped discrete probabilistic λ-calculus $\Lambda_{\oplus}$ introduced by Dal Lago and Zorzi [33]—and that will be our base language in this thesis. This approach is especially adapted to probabilistic extensions of PCF—the λ-calculus with integers as ground types introduced by Scott—since the monadic approach enforces a call-by-value evaluation strategy, while the PCF evaluation strategy is call-by-name: it is the case of the discrete probabilistic variants of PCF used by Ehrhard, Pagani and Tasson [46], or Goubault-Larrecq [60] . This approach has also been used for higher-order languages with *continuous* probabilities as for instance the variant of PCF with continuous probabilities used by Ehrhard, Pagani and Tasson in [45]—that have continuous distributions, but does not allow bayesian reasoning—or the λ-calculus with continuous probabilities and primitives for Bayesian reasoning introduced by Borgström et al [16] as a foundational calculus for Church.

A generalization of Abramsky's applicative bisimulation to a probabilistic higher-order

language was introduced by Dal Lago, Sangiorgi and Alberti in [32]: the authors introduced *two* equivalence relations on programs there—both of them being probabilistic adaptations of Abramsky's applicative bisimulation: probabilistic trace equivalence and probabilistic applicative bisimilarity. They showed that for their probabilistic λ-calculus equipped with a *call-by-name* evaluation strategy, probabilistic trace equivalence is fully abstract with respect to context equivalence, while probabilistic applicative bisimilarity is sound, but not fully abstract. These results were generalized by Deng, Feng, and Dal Lago [38] to a λ-calculus designed for quantum computation. More recently, Sangiorgi and Vignudelli [107] studied *environmental bisimilarity* for a λ-calculus with state—i.e. the possibility for the program to write and read from memory.

Several denotational models have been introduced for higher-order languages enriched with either discrete or continuous probabilistic primitives, starting with Jones and Plotkin's seminal work [68, 67]. From there, different approaches to build models have been enriched to model discrete probabilities: the domain-theoretic approach with the line of work initiated by Plotkin and Keimel using *Kegelspitzen* [95], the game-semantics one starting from the denotational model of probabilistic Idealized Algol of Danos and Harmer [35], and the Linear Logic one with the introduction by Danos and Ehrhard of *probabilistic coherence spaces* [34]. There have been several full-abstraction results for higher-order languages enriched with discrete probabilities: for instance the Danos and Harmer's model of probabilistic Idealized Algol, or the model introduced by Danos and Ehrhard for probabilistic PCF. It is also the case for a domain-theoretic model introduced by Goubault-Larrecq for probabilistic PCF extended with *statistical convergence testers*, i.e. the ability to talk *inside* the language about the probability that a program terminates [60]. More recently, several models have been introduced for higher-order languages with *continuous* probabilities: for instance the model developed by Staton et al [63] using quasi-Borel spaces with a focus on interpreting Bayesian reasoning, or models that extend the game-semantics approach to handle continuous probability [89, 90]. Here, we will mostly focus on the model introduced by Ehrhard, Pagani and Tasson [45] for PCF with continuous probability, that is based to a probabilistic generalization of the domain-theoretic notion of *stability*—used for instance by Berry for building dI-domains to handle *sequentiality*.

Challenges

- The results obtained by Dal Lago, Sangiorgi and Alberti [32] strongly rely on the probabilistic language they chose: for instance, two programs can be context equivalent in some language, and *not* context equivalent in an extension of the same language that gives more discriminating power to contexts. Even when fixing the *syntax* of the language to be the one of the probabilistic λ-calculus [32], the results are still only valid for the considered *evaluation strategy*—i.e. which part of the program should be executed first. The evaluation strategy chosen in [32] is call-by-name, meaning that when the program consists of a function applied to an argument, the argument is passed unevaluated to the function. While the λ-calculus is *confluent*—i.e. the evaluation strategy does not matter—it is not the case anymore when we add probabilistic primitives to the language, which limit the scope of the soundness and full-abstraction results [32].

- The study of Morris context equivalence for higher-order language enriched with *continuous* probabilities is still in its infancy, even though machine-learning algorithms have focused the attention on programming languages able to directly handle

probability distributions on reals. There is for instance no definition of applicative bisimulation for an higher-order continuous probabilistic language. As mentioned earlier, sound denotational models have been developed in this setting in recent years, but there is no full abstraction result yet. On the operational side, Culpepper et al [30, 123] have full-abstraction results for equivalences build in the tradition of Logical Relations, for a language with continuous random variables and scoring, but there is as for now no work on coinductive methodologies for a higher-order language with continuous probabilities.

- In a probabilistic setting, the observable behavior of any program is inherently *quantitative*: for instance, two programs can behave the same way on almost all probabilistic paths, but have a totally different behavior on the few remaining paths. Sometimes, we are not really interested to know if two programs have *exactly* the same behavior, but rather investigate how *close* they are. This is the case for instance in *computational indistinguishability*—a security notion coming from cryptography, where two algorithms are *indistinguishable* when the difference between their observable behavior when interacting with any possible adversary must be *negligible* with respect to the size of the security parameter. Accordingly, we would like to be able to study a quantitative counterpart to Morris's context equivalence: context distance, namely a measure of how much a context may *distinguish* two programs. Notions of distance have been studied in depth for probabilistic processes—for instance [42, 40, 119]. Gebler and Tini [49] introduced and studied a context distance for a *first-order* probabilistic language. We can also mention *differential privacy* [43], which does not use Morris context distance, but look at certifying that algorithms do not leak *too much* information. However, when this thesis started, there was—at the best of my knowledge—no study of context distance for higher-order languages. More recently, Gaboardi [6] et al looked at a denotational model for a probabilistic language based on *metric spaces*, thus going towards a denotational handling of context distance.

Our Approach

The present thesis aims at adapting tools that come from program semantics for non-probabilistic—either deterministic or non-deterministic—programs, to the setting of programming languages in which some kind of probabilistic primitive is available. In order to do this, a major trend in the literature consists in transforming a *qualitative* structure already used in language-programming theory into a *quantitative* one. We can interpret this way the probabilistic generalizations of Abramsky's applicative bisimulation by Dal Lago, Sangiorgi and Alberti: indeed, they can be seen as transforming the underlying LTS used for applicative bisimulation on the λ-calculus into a quantitative transition system. For probabilistic trace equivalence, this transformation is *distribution-based*, hence we obtain a *Observable Labeled Transition System (WLTS)* [50], i.e. a LTS where each state is equipped with a weight. In the case of probabilistic applicative bisimulation, this transformation is *state-based*, meaning that the transitions themselves become quantitative, and the transition system obtained this way is a *Labeled Markov Chain (LMC)* [39] —a structure developed to represent the evolution of probabilistic processes. In game-semantics [35], the approach taken is usually to transform strategies into *probabilistic* strategies. Likewise, probabilistic coherence spaces—the objects at the core of the denotational model developed by Danos and Ehrhard—are built in the tradition of *web-based*

Linear Logic models [54], but the *cliques* becomes *quantitative*.

These quantitative structures often come with new tools to analyze them, that are not present in their qualitative counterparts. For instance, programs are interpreted in Danos and Ehrhard model as *power series*, thus allowing us to use the rich class of results on analytic functions from real analysis [71]. Similarly, LMCs comes not only with a built-in notion of equivalence —the coinductive notion of bisimulation given by Larsen and Skou [75]—but also with notions of distances, based on the Kantorowitch-Wasserstein lifting [42]. On the other hand, *quantitative* problems arise when studying probabilistic languages: for instance, as highlighted above, program equivalence may sometimes be seen as too much discriminating, and we can instead ask for *approximated equivalences*—designed to guarantee that two programs have almost always the same behavior—or for *distances* —i.e. determining how much their behavior is different.

Beyond the usual *qualitative* measures to evaluate models relevance—i.e. soundness and full abstraction—we would like to have a better understanding of the connection between the *quantitative* structure of the models, and the quantitative behavior of programs. This work was more specifically designed as an attempt to use existing models of higher-order languages to talk about context distance. This approach led us first to take a closer look at context distance directly at the level of programs, and more precisely to explore the *trivialization problem*: depending on the expressive power of contexts, when do there exist programs at context distance *strictly* between 0 and 1? We then chose three models of probabilistic higher-order languages, with the aim to explore built-in distances in those models, that could allow us to gather information about context distance: we considered two operational—the state-based and distribution-based probabilistic generalizations of Abramsky's applicative bisimulation—and a denotational one—the model based on probabilistic coherence spaces introduced by Danos and Ehrhard. On the operational side, this approach yields encouraging results: we present here results on how to retrieve the context distance from the distribution-based model. We also obtained partial results on the state-based models, that we do not present in this thesis, but that can be found elsewhere [29]. On the denotational side, we ended up to explore more widely the structure of the probabilistic coherent spaces model, with results designed to go towards a better understanding of the kind of quantitative reasoning enabled by the study of probabilistic higher-order languages.

Contributions

The contributions of the present thesis can be split into two parts: the first ones are results on behavioral equivalences and metrics, which arise from an analysis of higher-order probabilistic programming languages using transition systems. Their common starting point is the seminal work by Dal Lago, Sangiorgi and Alberti on probabilistic generalizations of Abramsky's applicative bisimulation, where they consider a call-by-name pure probabilistic λ-calculus.

- **Full abstraction of applicative probabilistic bisimulation for a call-by-value pure probabilistic λ-calculus.** Dal Lago, Sangiorgi and Alberti [32] showed that for their call-by-name probabilistic λ-calculus, applicative probabilistic bisimulation is sound, but not fully-abstract. By contrast, we show here that applicative probabilistic bisimulation is fully abstract for the call-by-value counterpart of their calculus. This result shows a gap between non-deterministic and probabilistic choice, since in a λ-calculus with non-deterministic choice, applicative bisimilarity is not fully abstract neither in call-by-name nor in call-by-value [77]. We also showed that

the asymmetrical counterpart of probabilistic applicative bisimulation—probabilistic applicative simulation—isn't fully abstract in call-by-value, thus highlighting a gap between simulation and bisimulation. This result is a joint work with Dal Lago, and published in [24].

- **Full abstraction of applicative probabilistic simulation for call-by-value probabilistic λ-calculus with Abramsky's parallel convergence tester [4].** In order to bridge the gap between the context preorder and applicative probabilistic bisimulation, we looked at how to enhance the expressive power of contexts, thus making them able to distinguish more programs. Introducing a programming construct with some sort of *non-sequential* behavior has often proved successful in the literature to transform a sound model into a fully abstract one—for instance, Plotkin showed that the Scott model of continuous functions is fully abstract for PCF extended with parallel or [93]. The parallel convergence tester works as follows: it takes two programs as arguments, then evaluate them *in parallel*, and stops as soon as one of those evaluation stops. This result is a joint work with Dal Lago, Sangiorgi and Vignudelli, and published in [27].

- **Two trivialization results for a Morris-style context distance**—a quantitative generalization of Morris's context equivalence introduced as a way to talk about how far apart two programs are. The context distance *trivializes* when the context distance between any pair of non-equivalent programs is 1, hence the context distance does not give more information as context equivalence. We show that trivialization occurs as soon as the considered language has copying abilities and moreover at least one of the two following assertions holds: all programs terminates, or Abramsky's parallel convergence tester is present in the language. The proofs are based on the construction of *amplification contexts*, i.e. contexts powerful enough to transform any small difference in programs behavior into a much bigger one. These results are a joint work with Dal Lago, and published in [26].

- **Definition of fully abstract trace distances for both an affine probabilistic λ-calculus, and a probabilistic λ-calculus with copying abilities that are fully abstract with respect to the context distance on their respective languages.** These trace distances are a quantitative generalization of the notion of probabilistic trace equivalence. On the *affine* λ-calculus, our trace distance is a direct generalizing to the quantitative case of the probabilistic trace equivalence from Dal Lago, Sangiorgi and Alberti. However, the direct way of generalizing trace equivalence lead to a distance which is *unsound* for the probabilistic λ-calculus where functions have an unrestricted access to their arguments. To overcome this problem, we introduced a different Labeled Transition System to model interactively the operational semantics of the language, with the aim to keep trace of the way different copies of the same program could be used by an external environment. To build this LTS, we started from a probabilistic higher-order language with explicit copying operator, that embeds the more generic probabilistic λ-calculus from Dal Lago, Sangiorgi and Alberti. These results are joint works with Dal Lago and published in [29, 26].

The present thesis also contains contributions about the denotational semantics of probabilistic higher-order languages. Our main focus here is the denotational model of probabilistic coherence spaces, introduced by Danos and Ehrhard [34], and later shown to be

fully abstract for probabilistic PCF by Ehrhard, Pagani and Tasson [46]. We present here two results that shed light on the structure of this model:

- **The exponential comonad of the category of probabilistic coherence spaces (PCoh)—seen as a model of Linear Logic—is the free exponential comonoid**, thus making **PCoh** a Lafont model of linear logic. The exponential comonad ! in Linear Logic transform an object A into an object $!A$ that interpret an unlimited source of objects of type A. In a model of Linear Logic, several *distinct* exponential comonads may coexist: it is for instance the case in the category of coherence spaces, introduced by Girard [51]. When the free exponential comonoid exists, however, it defines *uniquely*— i.e. by a universal property from category theory, that specifies it is *more generic* that any other exponential structure—an exponential comonad. This result gives a category-theoretic justification to the choice of exponential comonad of probabilistic coherence spaces done by Danos and Ehrhard. Our proof use the construction developed by Melliès, Tabareau and Tasson [82] to build a free exponential comonoid in a symmetric monoidal category equipped with finite products. This result is a joint work with Ehrhard, Pagani and Tasson, and was published in [28].

- **The model of PCF equipped with continuous probability, based on measurable cones and stable, measurable functions—introduced recently by Ehrhard, Pagani, Tasson [45]—is a conservative extension of the model of probabilistic coherent spaces.** In particular, we showed that stable functions between *discrete cones*—i.e. cones that can be expressed as PCSs—are in fact morphisms in the co-Kleisli category of **PCoh**, that is power series with non-negative coefficients. This result is significant, because it allows to see stable functions between cones as a generalization of analytic functions, and so we can hope to replicate the full-abstraction proof for the probabilistic language $\mathsf{PCF}_{\oplus}$ in the model of probabilistic coherence spaces, with the aim to obtain a full-abstraction proof of PCF extended with continuous probabilities in the model of measurable cones and stable, measurable functions. Our proof use a generalization done by McMillan [80] of a theorem from real analysis due to S.Bernstein—presented for instance in [113]—, that we recall at the beginning of this Chapter. The result was published in [23].

Outline.

The manuscript is divided in three parts:

- In a prelude, we mostly precise our approach, and the operational context which we want to study : in Chapter 1, we present the probabilistic higher-order languages, in Chapter 2 we introduce Morris style context-equivalence, and in Chapter 3 we formalize its generalization into a context distance for probabilistic languages.

- In a second part, we use tools from quantitative transition systems to study higher-order probabilistic programming languages: Chapter 4 is a background chapter, where we present both probabilistic applicative bisimulation and probabilistic trace equivalence as they were introduced by Dal Lago, Sangiorgi and Alberti, and we state the soundness and full-abstraction results already established in the literature. In Chapter 5, we complete this picture by showing various new soundness and full-abstraction results: the more interesting one—as explained there—being full abstraction for CBV probabilistic applicative bisimulation. In Chapter 6, we show

our trivialization results for the context distance. In the next Chapter 7, we define probabilistic λ-calculi that allows for a finer understanding of copying: an affine language, where no copying can occur, and a language with an explicit operator for copying. In Chapter 8, we define fully-abstract trace distances for both the affine calculus and the calculus with explicit copying, and we show that in these case the trivialization phenomenon does not occur.

- The third part focus on denotational models of probabilistic higher-order languages. In Chapter 9, we first give a brief overview of the denotational models developed for probabilistic higher-order languages; then we present in depth two of these models: the model introduced by Ehrhard and Danos for PCF with discrete probabilities based on probabilistic coherent spaces, and the model developed by Ehrhard, Pagani and Tasson for PCF with continuous probabilities using measurable cones and stable measurable functions. While doing so, we will highlight the categorical structure behind those models, i.e. that the first one is build as a model of Linear Logic, and the second one is directly given as a cartesian closed category. Before doing that, we will briefly recall what a model of Linear Logic is, and we will give two standard ways to build one: as a Seely category and as a Lafont model. In Chapter 10, we show that the model of probabilistic coherence spaces is a Lafont model of Linear Logic. In Chapter 11, we show that the model based on measurable cones and measurable, stable functions is a conservative extension of the one build of probabilistic coherent spaces.

We sum up graphically the structure of the thesis in Figure 1; the arrows indicate the dependencies between chapters.

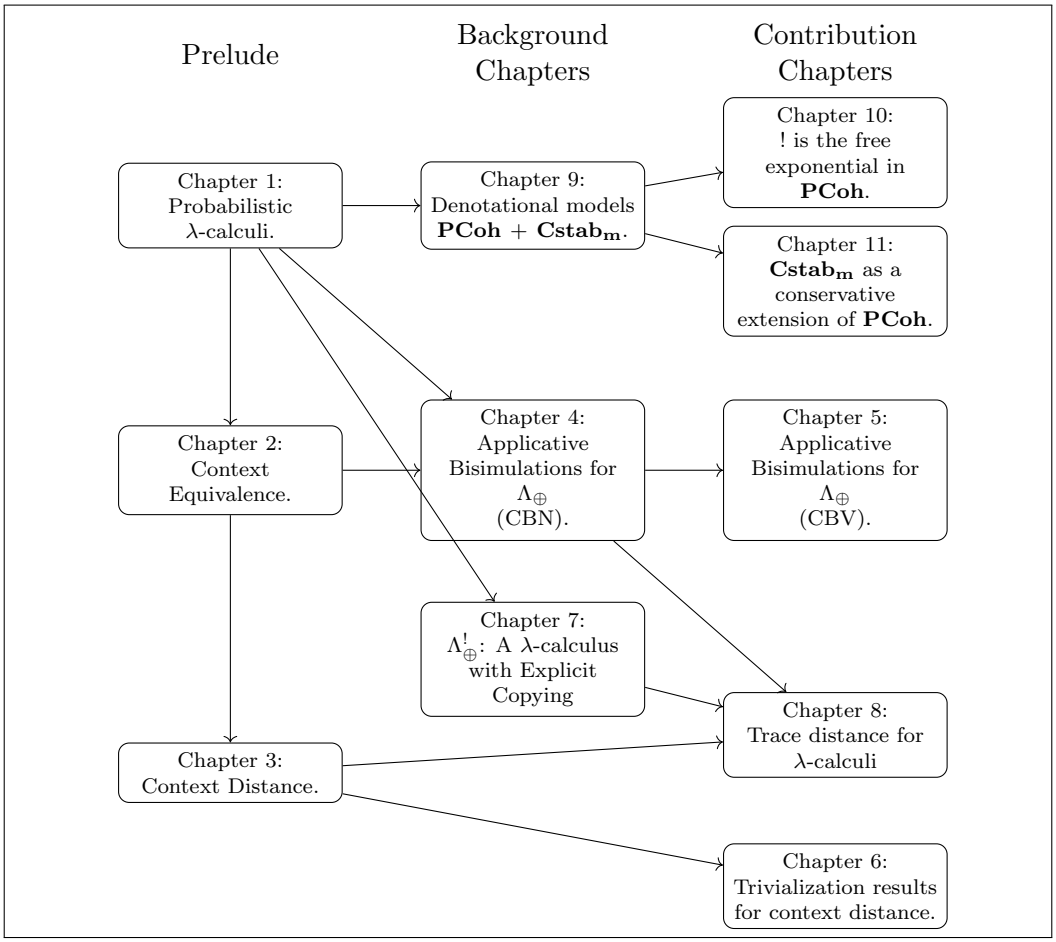


Figure 1: Organization of the Thesis

Contents

Part I

Prelude.

Chapter 1

Probabilistic λ-calculi

In the present chapter, we present several abstract languages that have been introduced in order to model and study higher-order probabilistic computation. They are all based on λ-calculus, a functional higher-order language introduced by Church. In the literature, λ-calculus comes in many flavors: typed or untyped, pure or effectful, with different evaluation strategies... In the first part of this chapter, we recall briefly some basic information about the pure untyped λ-calculus. Through the present thesis, we will be interested only in *head-first*, *weak* reduction—i.e. the left-most term is executed first, and no reduction occurs under a λ-abstraction; this paradigm will be precised in Section 1.1 —for the probabilistic λ-calculi we will consider; the reader interested in the study or probabilistic λ-calculi with a *strong* reduction strategy can look for instance at Thomas Leventis's work [78]. Accordingly, we focus on two evaluation strategies in our presentation of the pure λ-calculus: the head-first, weak call-by-value and call-by-name strategies. In a second part, we do a detailed presentation of $\Lambda_{\oplus}$, the λ-calculus extended with a probabilistic binary choice operator introduced by Dal Lago and Zorzi [33]. We will do this presentation in such a way as to highlight the *genericity* of such a probabilistic extension for a deterministic higher-order language; we will use the same framework later in this thesis to equip with probabilistic primitives various other λ-calculi. In the third part, we present two *typed* discrete probabilistic λ-calculi with ground types, that have been studied in the literature. The first one is a probabilistic variant of Gödel's System $\mathbb{T}$, that has been studied by Breuvart, Dal Lago and Herrou [19]: we will use it in the present thesis when we will need a probabilistic programming language where all programs terminate with probability 1. The second one is the probabilistic variant of Plotkin's PCF that has been given a fully abstract denotational semantics using probabilistic coherent spaces by Ehrhard, Pagani and Tasson [46]. Finally, we illustrate how to equip an higher-order language with *continuous* probabilities by presenting the extension of PCF with *continuous* probabilistic primitives studied by Ehrhard, Pagani and Tasson in [46].

1.1 The Pure λ-Calculus Λ

λ calculus was introduced by Church in the 1930s, first as a logical formalism, and then as a programming language. It has then been used to design concrete programming language, as for instance Caml or Haskell. It is a functional higher-order language, in the sense that programs represent functions, and that both their input and their output are also functions. As a programming language, λ-calculus is Turing-complete—i.e. every function computable by a Türing machine can be expressed in λ-calculus. It moreover has the benefit of having a very clean mathematical presentation, with very strong connections with

the field of logic. We give here a brief overview of the syntax and operational semantics of pure untyped λ-calculus—i.e. without computational effects or ground types—and while doing so we introduce formalism that we will reuse when presenting *probabilistic* λ-calculi. A more detailed introduction to the λ-calculus can be found for instance in [8].

1.1.1 Syntax of Pure λ-Calculus

The pure λ-calculus is characterized as a set of *terms*, designed to represent programs, and a family of *reduction rules*, that specifies how to execute the programs represented by the terms. We first define what *terms* are: let us suppose that we dispose of a countable set of variables $\mathbf{V}$. We define the set of λ-terms Λ by giving a grammar in BNF (Backus Normal Form, see [85]) that generates it:

$$M, N, L \ldots \in \Lambda ::= x \;\Big|\; \lambda x.M \;\Big|\; MN \qquad \text{when} \qquad x \in \mathbf{V}. \tag{1.1}$$

Observe that there are two constructs in the grammar above: the *λ-abstraction construct*, and the *application construct*. A term of the shape $\lambda x.M$ represents a function as follows: M specify the behavior of the function on an argument represented by the variable x. The application construct models passing an argument N to a function M.

If a term is of the shape $\lambda x.M$, we say that the variable x is *bound*, meaning it is under the scope of a λ-abstraction. In the following, we will always consider terms up to α-equivalence, i.e. up to renaming of the bound variables. If M is a term, and x is a variable that occurs in a non-bounded position—i.e. not under the scope of a λ-abstraction—in M, we will say that x is *free* in M. We will denote $FV(M)$ the set of variables free in M. We say that a term is *closed*, when it does not have free variables. It is only those closed terms which have a *computational meaning*, in the sense that they can be executed. For this reason, we will call them *programs* in the following, and we will note $\mathbf{P}_\Lambda$ the set of programs.

1.1.2 Operational Semantics for Λ

We have now to specify how these programs should be executed, i.e. their *operational semantics*. Several distinct choices of reduction rules are *a priori* admissible: the ones we need to choose depend on the *evaluation strategy* we want to model, i.e. which part of the program is going to be evaluated first. Observe that we could also choose to give a *non-deterministic* reduction relation, in such a way that every reduction strategy would then be valid. However, we would have then to consider the question of *confluence*: does a program has the same computational behavior under *all* evaluation strategies? In the case of pure λ-calculus, it is well-known that it is actually the case. However, as soon as we make *computational effects* available, as for instance the ability for the program to read and write, or to do probabilistic choices—as we will do in the next section—the confluence property is lost.

For this reason, we look here only at *deterministic* reduction relations: at each step of the evaluation of the program, there is only one possible thing to do to go on with the evaluation. More precisely, we will consider through the present thesis two evaluation paradigms, called *call-by-name (CBN)*, and *call-by-value (CBV)*. Both specify head-first, weak reduction—meaning respectively that we first reduce the left-sided term in the application case, and that we never reduce under a λ-abstraction. The difference between them lies in the moment where arguments are passed to functions: in a CBN strategy, we pass terms as argument to functions *without* evaluating them first, while in a CBV strategy we do the evaluation first. In the literature, *strong* CBV and CBN strategies—i.e.

where reduction may occur also under λ-abstraction—have also been considered. Weak paradigms present however benefits for concrete programming languages, since it allows for instance to not reduce the body of a function when it is never going to be used. For instance the evaluation strategy in Haskell is weak call-by-name, while in Caml it is weak call-by-value.

We present here CBN and CBV operational semantics for programs in Λ. We do this using the notion of *redexes* and *evaluation contexts*, that give us a generic framework to define a reduction strategy that we will be able to reuse when looking at probabilistic λ-calculi. This style of operational semantics was first advocated by Felleisen and Flatt in [47].

CBN and CBV One-Step Reduction Relations for Λ.

The basic reduction rule for Λ-programs is the β-rule, which corresponds to passing an argument to a function. We denote by $M\{N/x\}$ the term obtained by replacing in M every free occurrence of the variable x by N. To define the β-rule for the call-by-value paradigm, we need first to define formally when a program is a *value*: for the pure λ-calculus, it only happens when it is a λ-abstraction, i.e. it can be written as $\lambda x.M$. We note $\mathcal{V}_\Lambda$ the set of all values: it corresponds to those programs that are already in normal form.

Definition 1.1.1 (The CBN and CBV β-rule) *We define the CBN and CBV* β-rules *as respectively the following relations* $\to_\beta^{CBN}, \to_\beta^{CBV} \subseteq \boldsymbol{P}_\Lambda \times \boldsymbol{P}_\Lambda$*:*

$$(\lambda x.M)N \to_\beta^{CBN} M\{N/x\}; \tag{1.2}$$

$$(\lambda x.M)V \to_\beta^{CBV} M\{V/x\}, \qquad V \in \mathcal{V}_\Lambda. \tag{1.3}$$

Looking at Definition 1.1.1 above, we see that when we follow the CBN reduction strategy, the β-rule can be used on an application as soon as the left-sided program is a λ-abstraction, while in the CBV paradigm it is allowed only when *both* the left-sided and the right-sided programs are values: it models the fact that the argument is evaluated before being passed to the function in CBV, but not in CBN.

Redexes are those programs that can be reduced using directly the β-rule—or more generally by other basic reduction rules when we will consider extensions of λ-calculus. When a program is not a redex, it can possibly contain sub-programs that are redexes in a position of being executed—i.e. not under the scope of a λ. The aim of the evaluation consists in reducing all redexes inside a program until it contains no more executable redexes, and then the program is *in normal form*. Observe that in a program, there can possibly be *more than one* redex, hence the evaluation strategy must specify which redex is going to be reduced first. To formalize the evaluation strategy, we use the notion of *evaluation contexts*.

Definition 1.1.2 (Λ-Evaluation Contexts for CBN and CBV.) *We define the* Λ-evaluations contexts for CBN and CBV *as respectively the sets* $\mathbf{E}_\Lambda^{CBN}$ *and* $\mathbf{E}_\Lambda^{CBV}$ *generated by the grammars below:*

$$\mathcal{E} \in \mathbf{E}_\Lambda^{CBN} ::= [\cdot] \mid \mathcal{E}M, \qquad M \in \boldsymbol{P}_\Lambda; \tag{1.4}$$

$$\mathcal{E} \in \mathbf{E}_\Lambda^{CBV} ::= [\cdot] \mid \mathcal{E}M \mid V\mathcal{E} \qquad M \in \boldsymbol{P}_\Lambda, V \in \mathcal{V}_\Lambda. \tag{1.5}$$

Observe that in an evaluation context $\mathcal{C}$, there is *exactly* one occurrence of the hole $[\cdot]$. We denote by $\mathcal{C}[M]$ the term obtain when we replace the hole $[\cdot]$ with M in $\mathcal{C}$. A choice

of evaluation contexts for some language $\mathcal{L}$ is *valid* when evaluation contexts uniquely specify where the reduction should take place, i.e. any program M that is not in normal form can be written *in a unique way* as $\mathcal{E}[R]$, with R a redex, and $\mathcal{E}$ an evaluation context. We can see that it is indeed the case for the evaluation contexts for the language Λ that we presented in Definition 1.1.2 above.

We now use the β-rule, and the set of evaluation contexts to define the *one-step reduction relation*, in a generic way. Observe that our choice of evaluation strategy is entirely encoded into the way we define the β-rules, and the evaluation contexts we take.

Definition 1.1.3 *Let $S \in \{CBN, CBV\}$ denoting one of the two evaluation strategies we consider. Then the* reduction relation with respect to the strategy S, *that we denote $\to^S$ is the binary relation in $\boldsymbol{P}_\Lambda \times \boldsymbol{P}_\Lambda$ defined by the following rule:*

$$\frac{M \to_\beta^S N \qquad \mathcal{E} \in \mathbf{E}_\Lambda^S}{\mathcal{E}[M] \to^S \mathcal{E}[N]}$$

We denote by $\to^\star$ the reflexive and transitive closure of $\to$: it means that a step of $\to^\star$ corresponds to an arbitrary number of execution steps.

Example 1.1.1 *We illustrate Definition 1.1.3 on two prototypical Λ-programs: the program I representing the identity function, and the program Ω well-known to be non-terminating—i.e. its execution loops forever. The* identity term *is defined as $I := \lambda x.x$. Observe that it is a normal form, and moreover for any program M, $IM \to M$, meaning that the function I simply output the argument it has been given as input. The term Ω that we define is the textbook example of a term whose reduction never stops: $\Omega := (\lambda x.xx)(\lambda x.xx)$. We can see that $\Omega \to {}^S\Omega$ for $S \in \{CBV, CBN\}$, hence we can indeed see that the reduction of Ω is never going to reach a normal form.*

We illustrate now on an example the difference between the call-by-name and the call-by-value strategy.

Example 1.1.2 *We define a program $M := (\lambda x.I)\Omega$. If we consider the CBN evaluation strategy, we have $M \to^{CBN} I$, and since I is a normal form it means that the execution stops there. By contrast, when we look at the CBV evaluation strategy, we see that we need to first reduce Ω before doing the β-reduction. As a consequence $M \to^{CBV} M$, hence the execution of M never stops.*

Big-Step and Small-Step Operational Semantics

The reduction relations we have defined above talk about *exactly one* step of execution. We want now to look more globally at program execution by associating to every program the normal form reached at the end of its execution—when it exists. We present here two equivalent ways of doing so: *small-step* operational semantics and *big-step* operational semantics. The small-step semantics explicitly uses the one-step execution relation, and iterates here until reaching a normal form. The big-step semantics is more self-contained, and takes a more compositional view of computation: the execution of a program is deduced from the result of the execution of dependent programs.

Definition 1.1.4 (Small-step operational Semantics on Λ) *Given a reduction relation $\to$ on Λ, the* operational semantics on Λ with respect to $\to$ *is the relation $\downarrow \subseteq$ Programs $\times$ Normal Forms defined inductively using the following rules:*

$$\frac{V \text{ is a normal form.}}{V \downarrow V} \qquad \frac{M \to N \qquad N \downarrow V}{M \downarrow V}$$

Observe that the fact that we have fixed a particular reduction strategy ensures that for every program M, there is at most one normal form V such that $M \downarrow V$. If a program M is such that there is no normal form V with $M \downarrow V$, we will write $M \uparrow$: it means that the program M never terminates. On the contrary, we will sometimes write $M \downarrow$ to mean that there exists a normal form V such that $M \downarrow V$.

Proposition 1.1.1 (Big-step semantics for CBN Λ) *The operational semantics with respect to the* call-by-name evaluation strategy *can be characterized as the smallest relation* $\downarrow$ *verifying the following rules:*

$$\frac{}{\lambda x.M \downarrow \lambda x.M} \qquad \frac{M \downarrow \lambda x.L \qquad L\{N/x\} \downarrow V}{MN \downarrow V}$$

The first rule says that every abstraction is already a normal form. The second one says that, if we have already done the evaluation of M and we have obtained the normal form $\lambda x.L$, then to evaluate the program MN, we have only to do the β-rule: $(\lambda x.L)N \to L\{N/x\}$, and then evaluate the resulting program.

We can do the same as Proposition 1.1.1 for the *call-by-value* operational semantics: in Proposition 1.1.2 we give a big-step characterization of the call-by-value operational semantics: the λ-abstraction rule is the same as for CBN, but the application rule changes, since now we need to evaluate both the left-sided and the right-sided programs before doing the substitution.

Proposition 1.1.2 (Big-step semantics for CBV Λ) *The operational semantics with respect with the* call-by-value evaluation strategy *can be characterized as the smallest relation* $\downarrow$ *verifying the following rules:*

$$\frac{}{\lambda x.M \downarrow \lambda x.M} \qquad \frac{M \downarrow \lambda x.L \qquad N \downarrow W \qquad L\{W/x\} \downarrow V}{MN \downarrow V}$$

This way of looking at operational semantics is called *big-step*, since we do not consider in the rules just one execution step, but all of them at the same time. In the present thesis, we will consider interchangeably the small-step or the big-step view of operational semantics, depending on which will be more convenient at the time.

Encoding of Basic Data Types in Λ

We present here examples designed both to illustrate the expressive power of λ-calculus, and to introduce programs that will be useful in the examples later in the present thesis. Specifically, we give here an encoding in Λ of booleans and their if-then-else procedure, then we will present an efficient scheme due to Scott for encoding the natural numbers in weak λ-calculus, and finally we will present briefly the fixpoint construction. Unless otherwise specified, the evaluation strategy considered here is the call-by-name ones, but the situation would be similar with the call-by-value strategy.

Definition 1.1.5 (Encodding of booleans) *We define λ-terms* ***true****,* ***false*** *and ITE as follows:*

$$\begin{aligned} \boldsymbol{true} &:= \lambda x.(\lambda y.x); \\ \boldsymbol{false} &:= \lambda x.(\lambda y.y); \\ ITE &:= \lambda x.\lambda y.\lambda z.xyz. \end{aligned}$$

We can see that for every programs M, N:

$$\text{ITE } \mathbf{true} MN \to^\star M$$
$$\text{and } \text{ITE } \mathbf{false} MN \to^\star N.$$

It means that ITE represents a function that takes three arguments, and behave as follows: when it has **true** as first argument, it outputs its second argument, while when it has **false** as first argument it outputs its third argument. In other words, **true** simulate the boolean *true*, **false** simulate the boolean *false*, and ITE simulates the *if-then-else* construct.

We now present the Scott encoding of integers in λ-calculus. While the Church encoding is more often used, the Scott encoding is —as highlighted in [31]—more efficient when dealing with *weak* operational semantics.

Definition 1.1.6 *For every $n \in \mathbb{N}$, we define a program $\underline{n}$ in Λ by the following inductive definition on n:*

$$\underline{0} := \lambda x.\lambda y.x; \qquad \underline{n+1} := \lambda x.\lambda y.y\,\underline{n}\,.$$

We then define Λ-programs encoding the successor function, and a case construct.

$$\underline{S} := \lambda n.\lambda x.\lambda y.(yn); \qquad \underline{case} := \lambda n.\lambda a.\lambda f.(naf).$$

Observe that for every $n \in \mathbb{N}$, $\underline{n}$ is a value, and as a consequence $\underline{n} \downarrow \underline{n}$. The programs $\underline{S}$, and $\underline{\text{case}}$ allow to encode basic operations on integers. We need now to show that those programs behave as expected with respects to integers, i.e. to look at their operational semantics.

Example 1.1.3 *Let be $n \in \mathbb{N}$. Then looking at the reduction relation, we can see that:*

- $\underline{S}\ \underline{n} \downarrow \underline{n+1}$;
- *for any programs M, N, it holds that $\underline{case}\ \underline{0}\ MN \to^\star M$;*
- *for any programs M, N, it holds that $\underline{case}\ \underline{n+1}\ MN \to^\star N\,\underline{n}$.*

While there is no recursion operator available in the *syntax* of pure λ-calculus, we are able to encode it. We present here Curry's fixpoint combinator Y:

$$Y := \lambda x.\,((\lambda y.xyy)(\lambda y.xyy))\,.$$

Example 1.1.4 *If we consider any term M, we see that:*

$$YM \to (\lambda y.Myy)(\lambda y.Myy) \to M(\lambda y.Myy)(\lambda y.Myy).$$

1.2 $\Lambda_\oplus$: A Probabilistic Extension of Λ.

We are now going to present $\Lambda_\oplus$, the probabilistic extension of pure λ-calculus designed by Dal Lago and Zorzi in [33]. The idea is to add to the pure λ-calculus a binary probabilistic choice operator $\oplus$. Intuitively, we want the program $M \oplus N$ behaves with probability $\frac{1}{2}$ as M, and with probability $\frac{1}{2}$ as N. Formally, it means that the set of $\Lambda_\oplus$-terms is generated by the following grammar:

$$M, N, L \ldots \in \Lambda_\oplus ::= x \;\big|\; \lambda x.M \;\big|\; MN \;\big|\; M \oplus N \qquad \text{when} \qquad x \in \mathbf{V}.$$

Here we have simply taken the grammar generating the pure λ-calculus (1.1), and added the probabilistic choice operator to it. Observe that, as a consequence all closed λ-terms—as for instance those defined in Example 1.1.1 and Example 1.1.5—can also be seen as $\Lambda_\oplus$ programs.

We need now to endow $\Lambda_\oplus$ with an operational semantics, to specify how such programs are *executed*. As previously, we are going to specify a CBN and a CBV evaluation strategies. We are going to follow the same path as we did for the pure λ-calculus: first we are going to define one-step reduction relations, and then we will use them to give the operational semantics.

1.2.1 One Step Reduction Relation.

As before, the one-step reduction relation is meant to talk about *one step* of execution of the program. However, observe that we need now to take into account the fact that program behavior is now probabilistic. One solution would be to use a *probabilistic reduction relation*: it is for instance what is done by Ehrhard, Pagani and Tasson [46] when defining a probabilistic variant of PCF. Here however the approach we take is different: we replace the Λ one-step reduction relation $\to \subseteq \mathbf{P}_\Lambda \times \mathbf{P}_\Lambda$ between programs by a *deterministic* relation $\to \subseteq \mathbf{P}_{\Lambda_\oplus} \times \cup_{i\in\{1,2\}} \mathbf{P}^i_{\Lambda_\oplus}$ between programs and *sequences of programs*. It means that when M is not a normal form, we will write:

$$M \to N_1, \dots, N_n,$$

which is designed to express that M is going to be reduced in each of the N_i with probability $\frac{1}{n}$.

The Redex Rules

As before, we need first to specify what are the redexes, and the rules to reduce them. Recall that in Λ, we had only one kind of redexes: the ones of shape $(\lambda x.M)N$. We now have to deal also with a second kind of redexes: the ones corresponding to the evaluation of a probabilistic choice $M \oplus N$. As before, we have to make a decision here regarding the evaluation paradigm: do we want to first do the probabilistic choice and then evaluate the resulting program, or first evaluate both M and N and only then choose fairly between the resulting normal forms ? By analogy with β-reduction, we can see these options respectively as a *choice-by-name* and *choice-by-value*. In their original paper on $\Lambda_\oplus$, Dal Lago and Zorzi look at a choice-by-value paradigm—with the aim to be coherent with their overall call-by-value reduction strategy. In the present thesis, however, we will only consider choice-by-name—regardless if we consider a call-by-name or call-by-value reduction strategy—because we consider it to be more significant computationally: we want a program $M \oplus N$ to encode a probabilistic choices between two *execution branches* represented by M and N, instead of the juxtaposition of these two execution branches followed by a probabilistic choice on values. To illustrate this point, we can look at the program $I \oplus \Omega$: in choice-by-name, it behaves half the time as I, and never terminates half the time, while in choice-by-value it never terminates.

In Definition 1.2.1 below, we define the redex rules for $\Lambda_\oplus$. We keep unchanged β-reduction rules from Definition 1.1.1, and we add a redex rule corresponding to a choice-by-name paradigm.

Definition 1.2.1 (The CBN and CBV Redex Rules for $\Lambda_\oplus$) *We define the CBN and CBV* redex-rules *as respectively the following relations* $\to_R^{CBN}, \to_R^{CBV} \subseteq \boldsymbol{P}_{\Lambda_\oplus} \times \boldsymbol{P}_{\Lambda_\oplus}$*:*

$$M \oplus N \to_R M, N \quad \text{for } S \in \{CBN, CBV\}; \tag{1.6}$$

$$(\lambda x.M)N \to_\beta^{CBN} M\{N/x\}; \tag{1.7}$$

$$(\lambda x.M)V \to_\beta^{CBV} M\{V/x\} \quad \text{when } V \in \mathcal{V}_\Lambda. \tag{1.8}$$

The One-Step Reduction Relation for $\Lambda_\oplus$.

The evaluation contexts are built exactly as those defined for the pure λ-calculus—recalled in Definition (1.1.2). We define $\mathcal{V}_{\Lambda_\oplus}$—the set of $\Lambda_\oplus$-values—as the set of all λ-abstractions in $\Lambda_\oplus$, and the evaluation contexts for $\Lambda_\oplus$ in respectively the CBN and the CBV evaluations strategies as follows:

$$\mathcal{E} \in \mathbf{E}_{\Lambda_\oplus}^{CBN} ::= [\cdot] \mid \mathcal{E}M, \qquad M \in \mathbf{P}_{\Lambda_\oplus}; \tag{1.9}$$

$$\mathcal{E} \in \mathbf{E}_{\Lambda_\oplus}^{CBV} ::= [\cdot] \mid \mathcal{E}M \mid V\mathcal{E} \qquad M \in \mathbf{P}_{\Lambda_\oplus}, V \in \mathcal{V}_{\Lambda_\oplus}. \tag{1.10}$$

The following definition is simply the adaptation of Definition 1.1.3 to the probabilistic case: we have now to take sequences of programs into account.

Definition 1.2.2 *Let $S \in \{CBN, CBN\}$. The reduction relation on $\Lambda_\oplus$ with respect to S is defined as:*

$$\frac{M \to_R^S N_1, \dots N_n \qquad \mathcal{E} \in \mathbf{E}_{\Lambda_\oplus}^S}{\mathcal{E}[M] \to^S \mathcal{E}[N_1], \dots, \mathcal{E}[N_n]}$$

We are going to illustrate on an example how we can write reduction for programs that are probabilistic in a *intrinsic way*, i.e which cannot be written in pure λ-calculus.

Example 1.2.1 *We consider the following $\Lambda_\oplus$ closed term:*

$$M = \mathit{ITE}\ (\boldsymbol{true} \oplus \boldsymbol{false})\, I\, \Omega,$$

*where ITE , **true**, **false** are as defined in Example 1.1.5. We look at what happens when we reduce M by the CBN one-step reduction relation:*

$$M \to (\boldsymbol{true} \oplus \boldsymbol{false})\, I\, \Omega \to \boldsymbol{true}\, I\, \Omega\,,\ \boldsymbol{false}\, I\, \Omega.$$

1.2.2 Operational Semantics

Recall that in the deterministic case, the operational semantics of a program M is given by a relation $M \downarrow V$, meaning that when we execute the program M, we obtain the value V. However, in a probabilistic setting, when we execute a program, we may obtain *several* different values with different probabilities. It means that the operational semantics of a probabilistic program cannot consist in a value, but in *a sub-distribution of values*, designed to indicate for each value, with how much probability we may obtain it when executing the program. We precise in Definition 1.2.3 what we mean formally when we talk about sub-distributions.

Definition 1.2.3 *We call* sub-distribution over a set S *any function $\mathscr{D} : S \to [0,1]$, such that $\sum_{s \in S} \mathscr{D}(s) \leq 1$. We call* weight of a sub-distribution $\mathscr{D}$*, and we denote $|\mathscr{D}|$ the quantity $\sum_{s \in S} \mathscr{D}(s)$. We say that a sub-distribution is a* proper distribution *when its weight is equal to* 1*.*

In the following, we will often say simply *distributions* when talking about sub-distributions.

Observe that a sub-distribution can possibly contain no value at all: we call *empty sub-distribution over* S, and note $\emptyset$ the zero function $S \to [0,1]$. If s is an element of S, we can construct a distribution giving probability 1 to s: we call s-Dirac distribution, and we denote $\{s^1\}$ the function defined by $\{s^1\}(s') = 1$ if s' is equal to s, and $\{s^1\}(s') = 0$ otherwise. Moreover, if $\mathscr{D}$ is a sub-distribution over S, we will call *support of* $\mathscr{D}$, and we will note $\mathsf{S}(\mathscr{D})$ the set of all the $s \in S$ with $\mathscr{D}(s) > 0$.

Approximate Operational Semantics

We are now going to explain how we use a one-step reduction relation for $\Lambda_\oplus$, for instance the ones we defined before for CBN and CBV reduction strategies, in order to define an operational semantics for $\Lambda_\oplus$. The idea is to do it in a similar way to what we have done for the pure λ-calculus in Definition 1.1.4. Recall that the idea was to say: if the program M is already a normal form, then $M \downarrow M$. Otherwise, we do one step of computation, and if the resulting term is evaluated into a normal form V, then M is evaluated in the same normal form V. We are going to adapt this definition to the probabilistic case.

Actually, we are going to give the definition of approximate operational semantics in a generic way, that is for any probabilistic language $\mathcal{L}_\oplus$ endowed with a one-step reduction relation $\to_{\mathcal{L}_\oplus}$ between terms and finite sequences of terms. Definition 1.2.4 can obviously be applied when we take as language $\Lambda_\oplus$, with the CBV or the CBN one-step reduction relation.

Definition 1.2.4 *Let $\mathcal{L}_\oplus$ be a probabilistic language, and $\to \subseteq \boldsymbol{P}_{\mathcal{L}_\oplus} \times \cup_{i\in\{1,2\}} \boldsymbol{P}_{\mathcal{L}_\oplus}$ its one-step reduction relation. We define the* approximation semantics for $\mathcal{L}_\oplus$ with respect to $\to$ *as the smallest relation $\Rightarrow$ between programs and sub-distributions over normal forms for $\to$ verifying:*

$$\mathit{Empty}\ \frac{}{M \Rightarrow \emptyset.} \qquad \mathit{NF}\ \frac{V \text{ is a normal form for } \to.}{V \Rightarrow \{V^1\}}$$

$$\mathit{Step}\ \frac{M \to N_1, \ldots, N_n \qquad (N_i \Rightarrow \mathscr{E}_i)_{1\le i\le n}}{M \Rightarrow \sum_{1\le i\le n} \frac{1}{n} \cdot \mathscr{E}_i}$$

Observe the second and third rules are similar to the one in Definition 1.1.4 for pure λ-calculus. Accordingly, our notion of CBN (CBV) operational approximation semantics for $\Lambda_\oplus$ can be seen as a probabilistic extension of the CBN (CBV) operational semantics for Λ, in the following sense: if M is a Λ-program such that $M \downarrow V$, then it also holds that $M \Rightarrow \{V^1\}$ when M is seen as a $\Lambda_\oplus$-program. We illustrate on an example the role played by the first rule.

Example 1.2.2 *We use here the CBN one-step reduction relation, and we consider the program $M := (\boldsymbol{true} \oplus \boldsymbol{false})\, I\, \Omega$. We know already, as seen in Example 1.2.1, that:*

$$M \to \boldsymbol{true}\, I\, \Omega\,,\ \boldsymbol{false}\, I\, \Omega. \tag{1.11}$$

Moreover, we know that when we consider Λ with a CBN semantics, it holds that $\boldsymbol{true}\, I\, \Omega \downarrow I$, hence:

$$\boldsymbol{true}\, I\, \Omega \Rightarrow \{I^1\}. \tag{1.12}$$

We now use (1.11) *and* (1.12) *to deduce a non-trivial approximation semantics for* M*. Indeed, we can construct the following valid proof derivation for* $\Rightarrow$*:*

$$\frac{M \to \textbf{\textit{true}}\, I\, \Omega\,,\, \textbf{\textit{false}}\, I\, \Omega. \quad (1.11) \qquad \textbf{\textit{true}}\, I\, \Omega \Rightarrow \{I^1\} \quad (1.12) \qquad \dfrac{}{\textbf{\textit{false}}\, I\, \Omega \Rightarrow \emptyset}}{M \Rightarrow \frac{1}{2} \cdot \{I^1\}}$$

Observe that in Example 1.2.2, the first rule of Definition 1.2.4 is used to *give up* on a branch of the probabilistic execution tree. We see this rule is necessary, because not all branches of this probabilistic tree are necessarily going to reach a value. However, adding this rule forces us to accept that we may give up at any point in the execution tree. As a consequence, for a program $\mathscr{D}$, there may be different sub-distribution $\mathscr{D}$ such that $M \Rightarrow \mathscr{D}$. In particular, we have always $M \Rightarrow \emptyset$, which says that if we decide to stop the execution before the beginning, no value at all is collected. Observe moreover that we cannot be sure that there exists even *one* of the approximations semantics of a program M that describes *completely* the possible behavior of M. Indeed, the fact that $\Rightarrow$ is inductively defined implies that, whenever $M \Rightarrow \mathscr{D}$, then $\mathscr{D}$ has *finite support*, that is that it always collects only a *finite* number of values. But it is possible to write $\Lambda_\oplus$ program whose execution terminates with non-zero probability on an *infinite* number of different values. We give below an example of such program:

Example 1.2.3 *Here,* Y *corresponds to the fixpoint combinator defined in Example 1.1.4,* $\underline{S}$ *and* $\underline{n}$ *are the* Λ*-programs used respectively to encode the successor function and the integer* n *using Scott's scheme, as we presented in Example 1.1.3. We consider the following program—that we have extracted from [33].*

$$M := (Y \lambda x.\lambda y.(y \oplus x(\underline{S}\, y)))\, \underline{0}$$

We are able to show, that for every $N \in \mathbb{N}$, $M \Rightarrow \mathscr{D}_N$ *with* $\mathscr{D}_N := \sum_{1 \leq n \leq N} \frac{1}{2^n} \{\underline{n}^1\}$*. The proof can be done by induction on* N*.*

Observe that when we execute the program M of Example 1.2.3, we are going to obtain each Scott's numeral n with probability $\frac{1}{2^n}$. Accordingly, we would like to obtain as semantics of M the distribution over values $\mathscr{D} = \sum_{1 \leq n}^{\infty} \frac{1}{2^n} \{\underline{n}^1\}$. However, $\mathscr{D}$ cannot be an approximation semantics for M since every approximation semantics has finite support. To overcome this problem, we consider a partial order on the set of sub-distribution over normal-form, in such a way that $\mathscr{D}$ will be the *least upper-bound* of the approximation semantics for M—that are the $(\mathscr{D}_N)_{N \in \mathbb{N}}$ as illustrated in Example 1.2.3.

Operational Semantics for $\Lambda_\oplus$

We define a partial order on sub-distribution over a set, that consists simply in taking the pointwise order.

Definition 1.2.5 *Let be* $\mathscr{D}$*,* $\mathscr{E}$ *two sub-distributions over a set* S*. We say that* $\mathscr{D} \leq \mathscr{E}$*, if for any* $s \in S$*,* $\mathscr{D}(s) \leq \mathscr{E}(s)$*.*

Looking at the set of sub-distributions over a set S endowed with this partial order, we see that it is a complete meet semi-lattice. Moreover, as shown in [33], the set of all approximation semantics for some fixed program M is *directed*.

Lemma 1.2.1 *For every program* M*, the set* $\{\mathscr{D} \mid M \Rightarrow \mathscr{D}\}$ *is a directed subset of the set of sub-distributions over normal forms.*

Lemma 1.2.1 allows us to talk about the supremum of all approximation semantics of M. Accordingly, we take the *operational of a program* M as the supremum of all its approximation semantics, as stated in Definition 1.2.6 below for a generic probabilistic programming language.

Definition 1.2.6 *Let $\mathcal{L}_\oplus$ be a probabilistic language, and $\rightarrow \subseteq \boldsymbol{P}_{\mathcal{L}_\oplus} \times \cup_{i\in\{1,2\}} \boldsymbol{P}_{\mathcal{L}_\oplus}$ its one-step reduction relation. We define the* semantics of the program M with respect to $\rightarrow$ *as:*

$$[\![M]\!] = \sup\{\mathscr{D} \mid M \Rightarrow \mathscr{D}\},$$

where $\Rightarrow$ denotes the approximation semantics relation as defined in Definition 1.2.4.

Example 1.2.4 *We look again at the program M defined in Example 1.2.3. We can see that the operational semantics of M is as expected, i.e. $[\![M]\!] = \sum_{1\leq n}^{\infty} \frac{1}{2^n}\{\underline{n}^1\}$.*

1.2.3 Big-step operational semantics for $\Lambda_\oplus$.

For pure λ-calculus, we have seen that we could characterize alternatively the operational semantics by way of a *big-step* definition. We are actually able to do the same for $\Lambda_\oplus$. As for the small-step semantics, the idea is to define for every program a family of approximation semantics, and then to take the supremum. It is a non-trivial result, shown in [33], that for both the CBN and the CBV strategy, it characterizes the operational semantics for $\Lambda_\oplus$ as defined in Definition 1.2.6.

CBN Big-step Approximation Semantics

Definition 1.2.7 *We define the* big step approximation semantics relation for call-by-name *as the smallest relation $\Downarrow$ between program and sub-distribution over normal forms verifying:*

$$\textit{stop}\ \frac{}{M \Downarrow \emptyset} \qquad \textit{nf}\ \frac{}{\lambda x.M \Downarrow \{\lambda x.M^1\}} \qquad \textit{prob}\ \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2}\mathscr{D} + \frac{1}{2}\mathscr{E}}$$

$$\textit{appl-cbn}\ \frac{M \Downarrow \mathscr{D} \qquad (L\{N/x\} \Downarrow \mathscr{E}_L)_{(L \text{ with } \lambda x.L \in \mathsf{S}(\mathscr{D}))}}{MN \Downarrow \sum_L \mathscr{D}(\lambda x.L) \cdot \mathscr{E}_L}$$

Observe that, similarly to what happened in the small-step approximation semantics, we need a rule *(stop)* allowing us to give up on some part of the evaluation of a term. The rule *(prob)* says that the semantics of a program $M \oplus N$ is the probabilistic sum of the semantics of M and the semantics of N. The other two rules *(appl-cbn)* and *(nf)* are the probabilistic counterparts of the rules given for pure λ-calculus in Proposition 1.1.1. Big-step approximation semantics and small-step approximation semantics *do not* coincide: indeed it is not the case that the same sub-distributions $\mathscr{D}$ verify $M \Rightarrow \mathscr{D}$ and $M \Downarrow \mathscr{D}$. However, the supremum of these two sets coincide, as shown in [33]. That's the sense of Proposition 1.2.2 below.

Proposition 1.2.2 (From [33]) *Let be M a program in $\Lambda_\oplus$. Then $[\![M]\!]^{cbn} = \sup\{\mathscr{D} \mid M \Downarrow \mathscr{D}\}$.*

CBV Big-step Approximation Semantics

We are also able to do something similar to characterize the CBV operational semantics. More precisely, we define $\Downarrow_{cbv}$—the big-step approximation semantics relation for the CBV strategy—by keeping the rules *stop*, *nf*, and *prob* given in Definition 1.2.7, and replacing the rule *appl-cbn* by the following *appl-cbv* rule:

$$\text{appl-cbv}\ \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E} \qquad \{L\{V/x\} \Downarrow \mathscr{F}_{L,V}\}_{\lambda x.L \in \mathsf{S}(\mathscr{D}), V \in \mathsf{S}(\mathscr{E})}}{MN \Downarrow \sum \mathscr{D}(\lambda x.L) \cdot \mathscr{E}(V) \cdot \mathscr{F}_{L,V}}$$

Similarly to Proposition 1.2.2 for the CBN case, it holds that small-step and big-step operational semantics for CBV $\Lambda_\oplus$ coincide, as shown by Dal Lago and Zorzi [33], and stated in Proposition 1.2.3 below:

Proposition 1.2.3 (From [33]) *Let be M a program in $\Lambda_\oplus$. Then*

$$[\![M]\!]^{cbv} = \sup\{\mathscr{D} \mid M \Downarrow_{cbv} \mathscr{D}\}.$$

1.3 Probabilistic Typed Higher-Order Languages

As seen before, we can express some wild programs in Λ and $\Lambda_\oplus$, in particular non-terminating programs as Ω. A standard way to tame them, is by adding additional constraints by way of a type system. It may also be convenient to have primitives to handle ground data—for instance natural numbers or booleans—instead of dealing with encodings of those types in untyped λ-calculus. In the present thesis, we will follow this approach in order to obtain probabilistic higher-order languages where all programs terminate with probability 1, or when we will want to have more control on which argument of functions can be copied. We present below two probabilistic higher-order languages with ground types: the first one is a probabilistic variant of Gödel's system T that has been studied by Breuvart, Dal Lago and Herrou in [19], while the second one is a probabilistic variant of the more expressive Scott's PCF, that has been used as base language to define a probabilistic denotational semantics by Ehrhard, Pagani and Tasson [46].

1.3.1 $\mathsf{PCF}_\oplus$: A Probabilistic Variant of PCF

The language PCF was first introduced by Scott in [110] as a formal system, and then an operational semantics was given by Plotkin [94]: it is an higher-order typed language with natural numbers as ground type and fixpoint operator. A probabilistic extension of PCF has been first considered by Saheb-Djahromi [101]. In the present thesis, we will use the variant considered by Ehrhard, Pagani and Tasson [46], that is obtained from PCF by adding to it a *fair coin primitive*—i.e. a program that returns 0 with probability $\frac{1}{2}$, and 1 with probability $\frac{1}{2}$. The overall evaluation strategy of $\mathsf{PCF}_\oplus$ is call-by-name, but ground types are used in a call-by-value fashion in the if-then-else construct. We present here only the syntax and the type system for this language; a detail led presentation of the operational semantics can be found in [46]. There is only one ground type N—that models natural numbers—and an arrow construct, hence the grammar of types is given by:

$$\sigma, \tau \in \mathscr{A}_{\mathsf{PCF}_\oplus} ::= \mathrm{N} \mid \sigma \to \tau.$$

We look now at the syntax of $\mathsf{PCF}_\oplus$ —we again consider a fixed countable set of variables $\mathbf{V}$.

Definition 1.3.1 *The $\mathsf{PCF}_\oplus$ terms are generated as follows:*

$$M ::= x \mid \underline{n} \mid \mathtt{succ}(M) \mid \mathtt{if}(M, N, z \cdot L) \mid \mathtt{coin} \mid \lambda x^\sigma.M \mid MN \mid \mathit{fix}\, M \qquad \text{where } \sigma \in \mathscr{A}_{\mathsf{PCF}_\oplus},\ x \in \boldsymbol{V},\ \underline{n} \in \mathbb{N}.$$

The $\mathrm{fix}\, M$ construct is an explicit function construct: it is needed because the fixpoint operators from λ-calculus—for instance the Curry fixpoint that we presented in Example 1.1.4—cannot be typed in the PCF type system. As said before, the if-then-else construct is not the standard one in PCF, since in a probabilistic setting we need to handle natural numbers in a call-by-value way, as explained in depth in [46]. The program $\mathtt{if}(M, N, z \cdot L)$ first executes M, then executes N—as expected—when the natural number obtained is $\underline{0}$. The call-by-value behavior occurs when the natural number obtained is of the form $\underline{n}$ with $n > 0$: the program then executes $L\{\underline{n}/z\}$. This construction was chosen because we want to be able to use the output value of the conditional test when executing the branch modeled by L, and since the execution of M is probabilistic, there is no way to be sure to recompute the exact same value if it is not stored in he variable z.

We now give the *typing rules* for $\mathsf{PCF}_\oplus$. A *typing context* is a partial function Γ from variables to $\mathscr{A}_{\mathsf{PCF}_\oplus}$, with finite domain; it represents a way to associate a type to each free variable that occurs in an open term. We note $\mathrm{Dom}(\Gamma)$ the domain of the function Γ. If $x \notin \mathrm{Dom}(\Gamma)$, $(x : \sigma, \Gamma)$ represents the function which extends Γ to $\mathrm{Dom}(\Gamma) \cup \{x\}$, by associating σ to x.

Definition 1.3.2 *A* typing judgment *is an assertion of the form $\Gamma \vdash M : \sigma$, where Γ is a context, M is a term, and σ is a type. A typing judgment is* valid *if it can be inductively derived by the rules of the formal system given in Figure 1.1.*

$$\frac{(x,\sigma) \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{n \in \mathbb{N}}{\Gamma \vdash \underline{n} : \mathrm{N}} \qquad \frac{\Gamma \vdash M : \mathrm{N} \qquad \Gamma \vdash N : \sigma \qquad \Gamma, z : \mathrm{N} \vdash L : \sigma}{\Gamma \vdash \mathtt{if}(M, N, z \cdot L)\sigma}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \qquad \frac{}{\Gamma \vdash \mathtt{coin} : \mathrm{N}} \qquad \frac{\Gamma \vdash M : \mathrm{N}}{\Gamma \vdash \mathtt{succ}(M) : \mathrm{N}}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \qquad \frac{\Gamma, x : \sigma \to \tau \vdash M : \sigma \to \tau}{\Gamma \vdash \mathrm{fix}\, M : \sigma \to \tau}$$

Figure 1.1: Type Assignment in $\mathsf{PCF}_\oplus$

It is important to notice that for a term $\mathrm{fix}\, M$ to be typable, we need M to be a function—i.e. typable by an arrow type $\sigma \to \tau$. Moreover, observe that the type system do not take probabilities into account; it is obtained from the type system for deterministic PCF simply by adding the rule for the fair coin primitive. It is because we consider probabilistic and non-probabilistic data in the same way. As mentioned in introduction, an alternative typing discipline for probabilistic languages views probability as a *monad*, this way reflecting the behavior of programs in types: if σ is a type, $\Box\sigma$ would be the type of probabilistic distributions over σ, and the fair coin operator would always produces elements of type $\Box\mathrm{N}$.

Example 1.3.1 *The program Ω from λ-calculus—see Example 1.1.1—is not typable in PCF. However, we can use the fixpoint primitive to build never-terminating programs. We define $\Omega := (\mathit{fix}\, x)\ \underline{0}$. We can also easily simulate a fair choice operator $\oplus$ similar to the one used in $\Lambda_\oplus$:*

$$\underline{\oplus} := \lambda x.\lambda y.\mathtt{if}(\mathtt{coin}, x, z \cdot y).$$

We call *$\mathsf{PCF}_\oplus$-programs* those terms M such that there exists σ with $\vdash M : \sigma$. In [46], Ehrhard, Pagani and Tasson give an operational semantics for $\mathsf{PCF}_\oplus$, that associates to every $\mathsf{PCF}_\oplus$-program M of type σ a sub-distribution $[\![M]\!]$ over the normal-forms of type σ.

Example 1.3.2 *The following type assignments are valid:*
- *For every type σ, and typing context Γ, $\Gamma \vdash \Omega : \sigma$;*
- $\emptyset \vdash \mathit{fix}\ ((\lambda z.\underline{0}) \oplus \lambda z.((x\,\underline{0}) + \underline{1})) : \mathrm{N} \to \mathrm{N}$.

1.3.2 $\mathbb{T}_\oplus$: A Probabilistic Variant of Gödel's System $\mathbb{T}$.

Similarly to PCF, Gödel's $\mathbb{T}$ is a λ-calculus with natural numbers as ground type. Contrary to PCF, there is no fixpoint primitive in $\mathbb{T}$, and as a consequence the execution of a program typable in $\mathbb{T}$ always stop. Breuvart, Dal Lago and Herrou [19] studied a probabilistic extension of $\mathbb{T}$, such that this termination property still holds. Here, we only present the syntax of $\mathbb{T}_\oplus$, and we state its termination property. The type system and the operational semantics can be found in [19]. The type system of $\mathbb{T}_\oplus$ is *à la Curry*, i.e. without explicit typing annotations in terms. The grammar for types in $\mathbb{T}_\oplus$ is as follows: the only ground type is the type for natural numbers, and we can build product types and functional types.

$$\sigma, \tau \in \mathscr{A}_{\mathbb{T}_\oplus} ::= \mathrm{N} \mid \sigma \to \tau \mid \sigma \times \tau.$$

The syntax of $\mathbb{T}_\oplus$ is the one of $\mathbb{T}$ extended with a binary fair choice operator $\oplus$.

Definition 1.3.3 *The $\mathbb{T}_\oplus$ terms are generated as follows:*

$$M ::= x \ \Big|\ \underline{0} \ \Big|\ \mathtt{succ}(M) \ \Big|\ \langle M, N\rangle \ \Big|\ \pi_1 \ \Big|\ \pi_2 \ \Big|\ \mathtt{rec} \ \Big|\ M \oplus N \ \Big|\ \lambda x.M \ \Big|\ MN, \qquad \text{with } x \in \boldsymbol{V},\ \underline{n} \in \mathbb{N}.$$

The type system for this language is the type system for $\mathbb{T}$ extended with a rule for the binary choice operator that can be summed up as follows: $M \oplus N$ is of type σ if and only if both M and N are of type σ.

$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M \oplus N : \sigma}.$$

The operational semantics given by Breuvart, Dal Lago and Herrou is call-by-value. The behavior of λ-abstraction and application is as usual in λ-calculi, and the binary fair choice operator $\oplus$ behaves as the one in $\Lambda_\oplus$. The construct $\langle M, N\rangle$ represent a pair, and π_1, π_2 are respectively the left and right projections: if V, W are two values, the one step reduction relation is such that $\pi_1\langle V, W\rangle \to V$, and $\pi_2\langle V, W\rangle \to W$. The operator $\mathtt{rec}$ is a recursion operator on natural numbers, whose operational behavior is given by the following rules:

$$\mathrm{rec}\langle\langle V, W\rangle, \underline{0}\,\rangle \to V; \quad \text{and} \quad \mathrm{rec}\langle\langle V, W\rangle, \mathtt{succ}(\,\underline{n}\,)\rangle \to V\,\underline{n}\,\mathrm{rec}\langle\langle V, W\rangle, \underline{n}\,\rangle.$$

Observe that contrary to $\mathsf{PCF}_\oplus$, there is no primitive if-then-else construct in $\mathbb{T}_\oplus$. However, it can be easily encoded in $\mathbb{T}_\oplus$ using the recursion operator. Accordingly we will consider to have such a primitive available when using the language $\mathbb{T}_\oplus$ in Chapter 6.

Definition 1.3.4 *Let $\mathcal{L}_\oplus$ be a probabilistic programming language, equipped with an operational semantics $[\![\cdot]\!]$. We say that $\mathcal{L}_\oplus$ is* almost surely terminating (AST) *when for every program M, it holds that $|[\![M]\!]| = 1$.*

Proposition 1.3.1 (From [19]) *The language $\mathbb{T}_\oplus$ is AST.*

The authors of [19] actually showed a much stronger result—that all the branches in the execution tree have *finite length*—but we will need only the almost-sure termination property in the present thesis.

1.4 Extending λ-calculus with Continuous Probabilities.

As highlighted in Introduction, in some cases we want to model systems where the underlying space of events has inherent *continuous* aspects: for instance in hybrid control systems [5], as used e.g. in flight management. In this Section, we look at how to extend higher-order language extended with primitives designed to model *continuous probabilities*. First, we will recall basic concepts of measure theory—needed to handle continuous probability distributions in a rigorous way—and then we will present the language $\mathsf{PCF}_{\text{sample}}$ introduced by Ehrhard, Pagani and Tasson [45] as a continuous probabilistic extension of PCF.

1.4.1 Overview of Measure Theory

Before now, we were considering *discrete probabilities*, and consequently we were able to handle distributions over a countable set X in rather informal ways: they are those functions $\mathscr{D} : X \to [0,1]$ such that $\sum_{x\in X} \mathscr{D}(x) = 1$. Suppose now that f is a function $X \to \mathbb{R}$, and that we are interested in the average value of the following procedure: first we sample from the distribution $\mathscr{D}$, and then we apply f to the obtained value. Then this average value—the expected value of f with respect to $\mathscr{D}$—can be computed as $\sum_{x\in X} f(x) \cdot \mathscr{D}(x)$. That's because defining a distribution over a countable set means assigning a *probability* to every element of this set. However, when X is not countable, it is not always possible to do that: indeed when we consider for instance the *uniform distribution* $\mathscr{U}$ over $[0,1]$, we see that for every element $a \in [0,1]$, the probability of obtaining exactly a when sampling from $\mathscr{U}$ is 0. Therefore, what we want to do in that case to describe this distribution is to assign probability not to *single points* anymore, but to *intervals*: that way, we can say that for every a, b with $a \leq b$, the probability of obtaining an element in $[a,b]$ when sampling from $\mathscr{U}$ is $b - a$. Now, to compute the expected value of a function $f : [0,1] \to \mathbb{R}$, we need to use an *integral* instead of a discrete sum: indeed this expected value may be computed as $\int_{x\in[0,1]} f(x) \cdot dx$. Measure theory allows to generalize the ideas behind continuous probability beyond the real case. It provides a general framework to talk about probabilities and integration on a large class of spaces, called *measurable spaces*. We review the basics ideas here, but the reader should refer for instance to [108] for a more complete presentation of this field.

In the setting of measure theory, specifying a probability distribution means to assign some non-negative real to some subsets—seen as *event sets*, and called the *measurable* subsets. There are however constraints on the collection of measurable subsets we can choose, namely that they have to form a *σ-algebra*.

Definition 1.4.1 (Measureable Spaces) *A* σ-algebra—*also called σ-field—over a set S is a collection $\Sigma \subseteq Parts(X)$, such that:*

- $\emptyset \in \Sigma$;
- *For every* countable *family* $(A_n)_{n\in\mathbb{N}}$, *with* $A_i \in \Sigma$, $\bigcup_{1\leq i\leq n} A_i \in \Sigma$;
- *For every* $A \in \Sigma$, $X \setminus A \in \Sigma$.

We call measurable space *a pair* (X, Σ), *where* Σ *is a* σ*-algebra over* X. *If* $\Omega = (X, \Sigma)$ *is a measurable space, we will use* $x \in \Omega$ *to means* $x \in S$, *and* A *a measurable subset of* Ω *to means that* $A \in \Sigma$. *We call measurable those function* $f : X \to Y$, *such that for every* $A \in \Sigma_X$, *it holds that* $f^{-1}(A) \in \Sigma_Y$.

A useful fact is that from any collection of subsets, we can form a σ-algebra by considering the closure by countable unions, countable intersections and relative complement. It allows us to construct the smallest σ-algebra containing this collection.

Proposition 1.4.1 *Let* X *be a set, and* $\Theta \subseteq Parts(X)$ *a collection of subsets of* X. *Then there exists a* smallest σ*-algebra containing* Θ, *that we call the* σ*-algebra* generated by Θ.

Example 1.4.1 (Borel σ-algebra) *For any topological space* X, *the* Borel σ-algebra *is the smallest* σ*-algebra over* X *containing all* open *sets. In the particular case where we consider* X *to be* $\mathbb{R}$ *with the usual topology, the Borel* σ*-algebra coincides with the smallest* σ*-algebra that contains intervals.*

We call measurable functions $\mathbb{R}^n \to \mathbb{R}^k$ the functions measurable when both $\mathbb{R}^n$ and $\mathbb{R}^k$ are endowed with the Borel Σ-algebra associated with the standard topology of $\mathbb{R}$. The relevant properties of the class of measurable functions $\mathbb{R}^n \to \mathbb{R}^k$ is that they are closed under arithmetic operations, composition, and pointwise limit, see for example Chapter 21 of [108].

Definition 1.4.2 (Measure over a Measurable Space) *Let* (X, Σ) *be a measurable space. A* measure *over* (X, Σ) *is a function* $\nu : \Sigma \to [0, \infty]$ *such that:*

- $\nu(\emptyset) = 0$
- *for every* countable *collection* $(A_i)_{i\in\mathbb{N}}$ *of pairwise disjoint* $A_i \in \Sigma$: $\nu(\bigcup_{i\in\mathbb{N}} A_i) = \sum_{i\in\mathbb{N}} \nu(A_i)$.

When $\nu(X) < \infty$, *we say that* ν *is a* finite measure. *When moreover* $\nu(X) \leq 1$, *we say that* ν *is a* sub-distribution over X, *and we say that* ν *is a* (proper) distribution *when* $\nu(X) = 1$. *We note Meas*$((X, \Sigma))$ *the set of all measures over the measurable space* (X, Σ).

As said before, one of the main purposes of measure theory is to generalize the Riemann integral well-known for real functions. We fix here a measurable space(X, Σ_X), and ν is a measure over X, and we equip $\mathbb{R}$ with the Borel algebra. To every measurable function $f : X \to \mathbb{R}_+$, a quantity $\int_X f \cdot d\nu \in \mathbb{R}_+ \cup \{+\infty\}$ is defined, call the *Lebesgue integral of* f *with respect to* ν. We do not recall here the construction or properties of the Lebesgue integral, but all results needed for our purposes may be found in [108]. If $f : X \to \mathbb{R}$ is a measurable function, f is *Lebesgue-integrable with respect to* ν when $\int_X |f| \cdot d\nu < +\infty$, and we note $L^1_\nu(X)$ the set of those functions.

Measure theory also allows us to recover the intuitive view of discrete probability theory, as we highlight now.

Example 1.4.2 *Let* X *be a countable space. Then* $Parts(X)$ *is the smallest* σ*-algebra that contains the* atoms $\{a\}$ *for every* $a \in X$. *We call it the* generic σ-algebra over X, *and we call measurable spaces of the form* $(X, Parts(X))$ discrete measurable spaces.

Proposition 1.4.2 *Let S be any countable set, equipped with the generic σ-algebra over S. Then it holds that:*

- *if X is a measurable space, every function $f : S \to X$ is measurable;*
- *for every measure μ on S, and $f : S \to \mathbb{R}$, $\int f \cdot d\mu = \sum_{s \in S} f(s) \cdot \mu(s)$.*

A measure on a discrete measurable space is entirely characterized by its values on atoms: indeed, for any $A \in \mathrm{Parts}(S)$, we can write A as the countable disjoint union $\bigcup_{a \in A} \{a\}$, and consequently $\mu(A) = \sum_{s \in A} \mu(\{a\})$. It means that on a discrete space, a measure can be seen simply as a function $\mu : S \to \mathbb{R}_+ \cup \{\infty\}$.

Lemma 1.4.3 *Let S be a countable space, and $f : S \to \mathbb{R}_+ \cup \{\infty\}$. Then $\mu_f : Parts(S) \to \mathbb{R}_+ \cup \infty$ defined as $\mu_f(A) = \sum_{s \in A} \mu(\{a\})$ is a measure over $(S, Parts(S))$, and moreover:*

- *μ_f is a finite measure, if $\sum_{s \in \S} f(s) < \infty$;*
- *μ_f is a distribution if $\sum_{s \in \S} f(s) = 1$;*
- *μ_f is a sub-distribution if $\sum_{s \in \S} f(s) = 1$.*

Proof. We verify easily that μ_f is a measure: indeed $\mu_f(\emptyset) = 0$, and for every countable collection of pairwise disjoints $A_n \in \mathrm{Parts}(S)$, it holds that $\mu_f(\bigcup_{n \in \mathbb{N}} A_n) = \sum_{n \in \mathbb{N}} \mu_f(A_n)$. Moreover,$\mu_f(S) = \sum_{s \in S} f(s)$, which concludes the proof. □

1.4.2 The Language $\mathsf{PCF}_{\mathbf{sample}}$

An example of higher-order language that handles continuous probability distributions is $\mathsf{PCF}_{\mathtt{sample}}$, defined by Ehrhard, Pagani and Tasson in [45]. $\mathsf{PCF}_{\mathrm{sample}}$ can be seen as a continuous counterpart to $\mathsf{PCF}_{\oplus}$, where the ability to flip a fair coin is replaced by the possibility to sample from the uniform distribution on $[0, 1]$. As such, it does not offer the full range of continuous probabilistic primitives that other languages [16, 115] often consider, for instance the score primitive used for Bayesian conditioning. An investigation on the expressiveness of $\mathsf{PCF}_{\mathrm{sample}}$ with respect to these probabilistic primitives can be found in [45]. The base type of $\mathsf{PCF}_{\mathtt{sample}}$ is the real type R, and types are generated by:

$$\sigma, \tau \in \mathscr{A}_{\mathsf{PCF}_{\mathrm{sample}}} ::= R \;\Big|\; \sigma \to \tau.$$

The terms of the language are generated by the grammar below:

$$M, N \in \mathsf{PCF}_{\mathrm{sample}} ::= x \;\Big|\; \lambda x^{\sigma}.M \;\Big|\; MN \;\Big|\; \mathrm{fix}\, M$$
$$\Big|\; \mathtt{let}(x, M, N) \;\Big|\; \mathtt{if}(M, N, L) \;\Big|\; \underline{r} \;\Big|\; \mathtt{sample} \;\Big|\; \underline{f}(M_1, \ldots, M_n),$$

where r is any real number, and f is in a fixed countable set of measurable functions $\mathbb{R}^n \to \mathbb{R}$. The constant `sample` stands for the uniform distribution over $[0, 1]$. Observe that since we can choose arbitrarily the countable set of measurable functions we take as primitives, we can encode classical probability distributions as soon as they can be obtained in a measurable way from the uniform distribution, as it is for instance the case for Gaussian or normal distributions. Moreover, we can argue that this language is also *more general* than $\mathsf{PCF}_{\oplus}$: first it allows to encode integers (since $\mathbb{N} \subseteq \mathbb{R}$) and basic arithmetic operations over them. Secondly, since the order operator $\geq : \mathbb{R} \times \mathbb{R} \to \{0, 1\} \subseteq \mathbb{R}$ is measurable, we can construct in $\mathsf{PCF}_{\mathrm{sample}}$ a program that simulates the fair coin as follows: $\underline{\geq}(\mathtt{sample}, \frac{1}{2})$.

In [45] Ehrhard,Pagani and Tasson formalize the operational semantics for $\mathsf{PCF}_{\text{sample}}$. They give a *call-by-name* semantics, but the let construct allows to simulate for instance the non-standard if-then-else construct of $\mathsf{PCF}_{\oplus}$. Recall that for any $\mathsf{PCF}_{\oplus}$-term M of type τ, $[\![M]\!]$ is a discrete distribution over normal forms of type τ. Accordingly, if M is now a $\mathsf{PCF}_{\text{sample}}$-term of type τ, $[\![M]\!]$ should be a *continuous* probability distribution of the values of type τ. However, as we highlighted when presenting the basic principles of continuous probabilistic reasoning, to talk about a continuous probability distribution on some set X, we have first to specify which σ-*algebra* on X we consider. For this reason, the first step if the construction of an operational semantics for $\mathsf{PCF}_{\text{sample}}$ [45] is to define, for each type τ, a σ-algebra over each set of normal forms of type τ, and then the operational semantics is built as a *Markov process* on this probabilistic space.

Chapter 2

Observational Equivalence

Having a good—and usable in practice—notion of program equivalence is a problem that appears very often in computer science: for example, when we want to check that a program implements a given specification, or that a *compiled program* is *equivalent* to the original program. When the two programs considered output data of ground type—e.g. booleans or integers—it is sufficient, while not necessarily easy, to look at whether they output the same value—or, in the probabilistic case, the same distribution over values. However, we consider here *higher-order* languages, hence the result obtained after evaluating a program can itself be a *function*. To illustrate this point, let us consider for instance the two Λ-programs $\lambda x.x$ and $\lambda x.(\lambda y.y)x$: since we are in a weak evaluation paradigm—see our presentation of Λ in Chapter 1—those two programs are already in normal form. These normal forms are distinct, but whenever we pass them an argument M, the resulting terms $(\lambda x.x)M$ and $(\lambda x.(\lambda y.y)x)M$ have the same operational semantics: it is an example of two programs that reduce to distinct values but that we nonetheless would like to see as equivalent.

This idea leads to the notion of *context equivalence*, that was formalized by Morris [87]: we say that two programs are equivalent if they have the same *observable behavior* in any *context*. In this chapter, our goal is to formalize *Morris context equivalence* in the setting of a generic probabilistic programming language. Observe first that in order to instantiate this definition on a particular language, we need to decide what is an observable behavior, and what is a context.

2.1 Observables

We first have to specify what is the *observable behavior* of a program. It depends on the language we consider, as well as the distinguishing abilities we want to give to the observer. For example, if we allow the observer to count the number of execution steps, then two programs with different running times would never be equivalent. Such a notion may be of interest—e.g. in computational cryptography, where *side-channel attacks* can be executed by adversaries with this ability—but in the present thesis, we will never allow observers to distinguish between programs that output the exact same data, no matter how much time they take to produce it.

2.1.1 Deterministic Case:

Let us first consider the pure λ-calculus Λ, with the weak reduction paradigm that we describe in Chapter 1. Recall that here our programs are actually closed λ-terms. The

reasonable thing to observe in this setting is program termination: consequently, we formalize our notion of observables by a function $\mathrm{Obs}() : \mathbf{P}_\Lambda \to \{0,1\}$, such that for any closed λ-terms M, we define:

$$\mathrm{Obs}(M) = \begin{cases} 0 \text{ if } M \uparrow \\ 1 \text{ if } M \downarrow \end{cases} \in \{0,1\}.$$

Observe that $\{0,1\}$—which can be seen as the set of *observations*—is naturally equipped with an order structure. It allows us to talk not only about observational equivalence, but also about the *observational preorder*. If we compare for instance the *observable behavior* of the non-terminating program Ω and the identity function I—both defined in Example 1.1.1 of Chapter 1—we see that $0 = \mathrm{Obs}(\Omega) < \mathrm{Obs}(I) = 1$: it means that $\Omega \leq^{obs} I$, and that Ω and I are not context equivalent—where $\leq^{obs}$ is the observational preorder that we will formally define in the next section.

The above definition is suitable for the *untyped* λ-calculus. If we consider a typed higher-order language $\mathcal{L}$ with ground types, for example PCF, we can also wish to allow observations *only at ground type*—e.g. at type Bool. Formally, it means that the observation becomes two functions that express whether a program of type Bool stops on respectively **true** or **false**:

$$\mathrm{Obs}^T(), \mathrm{Obs}^F() : \{M \in \mathbf{P}_\mathcal{L} \mid\vdash M : \mathbf{Bool}\} \to \{0,1\}.$$

2.1.2 Non-deterministic Case:

We now look briefly at how the above notion of observation should be adapted for non-deterministic higher-order languages, for instance the pure λ-calculus extended with a binary non-deterministic operator $\oplus$. Observe that we cannot say anymore that a program *either* terminates *or* has a non-terminating run. Actually, we have to choose between observing the *may-convergence*, that is the fact that *at least* one run of the program terminates, or the *must-convergence*, that is the fact that *every* possible run of the program terminates. Observational equivalence for both *may*-convergence and *must*-convergence have been investigated in depth by Lassen [77], as well as coinductive methods for studying equivalence in a non-deterministic setting.

2.1.3 Probabilistic Case:

When we pass from the non-deterministic case to the probabilistic case, we see a change in the nature of the observables: they are not binary anymore, but quantitative. Indeed, a probabilistic program terminates *with a given probability*, that can be strictly between 0 and 1. Formally, we define a function $\mathrm{Obs}() : \mathbf{P}_{\Lambda_\oplus} \to [0,1]$ as:

$$\mathrm{Obs}(M) = \mathrm{Prob}(M \downarrow) \in [0,1].$$

It is the fact that observables are now quantitative that will allow us later to extend the notion of observational equivalence to metrics, because now we can compare the observables of two programs, and talk about *how far* they are, instead of just look at if they coincide or not.

Observe, that as in the probabilistic case, if we are considering a probabilistic λ-calculus with base type (for example, a probabilistic variant of PCF), we can also restrict the observed program to consider only the type **Bool**, and then we take as observables the sub-distribution over the booleans obtained when evaluating the program.

2.2 Contexts

Contexts are meant to represent every way the environment can interact with the program. Here, we consider that the environment itself consists in agents that use the *same* programming language. Since languages that we consider are compositional, we can express that by saying that the environment is a term of a language with a hole inside, that is replaced by the program at the time of the interaction. This idea give us a generic way to construct, for a compositional language $\mathcal{L}$ (where $\mathcal{L}$ is one of the variation around the λ-calculus we are considering here), a set $\mathbf{C}^{\mathcal{L}}$ of contexts for $\mathcal{L}$.

We give here formally the grammar generating contexts in $\mathbf{C}^{\Lambda_\oplus}$ when we take as language the probabilistic λ-calculus:

$$\mathcal{C} \in \mathbf{C}^{\Lambda_\oplus} ::= [\cdot] \;\Big|\; \lambda x.\mathcal{C} \;\Big|\; \mathcal{C}M \;\Big|\; M\mathcal{C} \;\Big|\; \mathcal{C} \oplus M \;\Big|\; M \oplus \mathcal{C}.$$

We will denote as $\mathcal{C}[M]$ the term obtained by replacing the hole $[\cdot]$ by M. We would like the execution of $\mathcal{C}[M]$ to correspond to the interaction between the execution environment $\mathcal{C}$ and the program M. Observe, however, that we have to be careful here: for this to be the case, we need to be able to certify that $\mathcal{C}[M]$ is indeed an executable program, that is a *closed* term: it means that every free variable in M has to be *bounded* by the context.

To do that, we introduce *closeness judgment on contexts*, which are of the forms: $x_1, \dots x_n \dashv \mathcal{C}$. It is designed to mean: for any term with free variables among the $x_1, \dots, x_n$, then $\mathcal{C}[M]$ is a closed term. To design such ordered family of distinct variables $x_1, \dots, x_n$, we will use the notation $\overline{x}$, and we will denote by $\overline{\mathbf{V}}$ the set of all such families. We will denote by $\overline{x}, x$ the appending of x to the list $\overline{x}$, where x is distinct of all elements in $\overline{x}$. We define validity for judgments of this form by inductive rules on the structure of $\mathcal{C}$, that can be found in Figure 2.1. Observe that, if we consider a typed

$$\frac{}{\dashv [\cdot]} \qquad \frac{\overline{x} \dashv \mathcal{C} \quad x \notin \overline{x}}{\overline{x}, x \dashv \lambda x.\mathcal{C}}$$

$$\frac{\overline{x} \dashv \mathcal{C}}{\overline{x} \dashv M\mathcal{C}} \qquad \frac{\overline{x} \dashv \mathcal{C}}{\overline{x} \dashv \mathcal{C}M} \qquad \frac{\overline{x} \dashv \mathcal{C} \quad M \in \mathbf{P}}{\overline{x} \dashv \mathcal{C} \oplus M} \qquad \frac{\overline{x} \dashv \mathcal{C} \quad M \in \mathbf{P}}{\overline{x} \dashv M \oplus \mathcal{C}}$$

Figure 2.1: Closeness Judgment for $\Lambda_\oplus$ contexts.

language, we will have to certify that the resulting program is well-typed too, and so we have to replace our closeness judgment on contexts by *well-typed judgments on contexts.*

2.3 Observational Equivalence

We are now ready to give the definition of *observational equivalence.* We first formalize the minimal requirements that must hold for a programming language so that our definition of observational equivalence will make sense: we need a notion of programs, a notion of contexts and a notion of observables,

Definition 2.3.1 *An* untyped observable programming language *consists of:*

- *a set of terms Terms, built from the variables* $\mathbf{V}$ *and syntactical constructs, and a function* $FV() : Terms \to \mathbf{V}$ *that denotes the free variables of a term;*

- *a notion of observation for programs: i.e. Obs*() : $\boldsymbol{P} \to \mathcal{O}$*, where* $\boldsymbol{P}$ *is the set of closed terms, and* $\mathcal{O}$ *is a set of possible outcomes for the observation function;*
- *a notion of contexts: i.e. for each family of variables* $\overline{x}$*, a set* $\mathbf{C}_{\overline{x}}$ *of contexts—where a context is formally a function Terms* $\to$ *Terms—such that for every term* M *with* $FV(M) \subseteq \overline{x}$*,* $\mathcal{C}[M] \in \boldsymbol{P}$*.*

When we consider a typed language, we have to take into account the fact that, as mentioned in Section 2.1, the observation is not necessarily done at *all* types.

Definition 2.3.2 *A* typed observable programming language *consists of:*

- *a set of terms Terms, built from the variables* $\boldsymbol{V}$ *and syntactical constructs, a set of types* $\mathscr{A}$ *and a set of* valid typing judgments *of the form* $\Gamma \vdash M : \sigma$*, where* σ *is a type in* $\mathscr{A}$*, and* Γ *a typing context of the form* $x_1 : \sigma_1, \ldots, x_n : \sigma_n$*;*
- *a set of observables types* $\mathscr{A}_{obs}$*, a set of possible outcomes for the observation* $\mathcal{O}$*, and a notion of observation for programs: i.e. for every* $\sigma \in \mathscr{A}_{obs}$*:* $Obs_\sigma() : \{M \in \boldsymbol{P}_\sigma\} \to \mathcal{O}$*, where* $\boldsymbol{P}_\sigma = \{M \mid\vdash M : \sigma\}$*.*
- *a notion of contexts: i.e. for each typing context* Γ*, types* σ*,* τ*, a set* $\mathbf{C}_{(\Gamma,\sigma)\to\tau}$ *of contexts* $\mathcal{C}$ *such that for every term* M *with* $\Gamma \vdash M : \sigma$*,* $\mathcal{C}[M] \in \boldsymbol{P}_\tau$*. We will also note* $\Gamma, [\cdot] : \sigma \vdash \mathcal{C} : \tau$ *to mean that* $\mathcal{C} \in \mathbf{C}_{(\Gamma,\sigma)\to\tau}$*.*

Two terms M and N are observationally equivalent if, no matter which contexts we consider, we obtain exactly the same observables. We have however to be careful: the set of contexts we consider depends on the free variables of the terms we want to compare. For these reasons, we work with generalizations of relations, that we call *open relations with respect to the language* $\mathcal{L}$: those are families of relations $R = (R_{\overline{x}})_{\overline{x}\in\overline{\mathbf{V}}}$ over terms of $\mathcal{L}$, indexed by families $\overline{x}$ of variables. Moreover, we will also note $\overline{x} \vdash MRN$ for denoting $M(R_{\overline{x}})N$.

We extend now the usual notions of reflexivity, symmetry and transitivity to open relations.

Definition 2.3.3 *Let be* R *an open relation with respect to the language* $\mathcal{L}$*. We say that* R *is:*

- reflexive*, if for any* M*, for any* $\overline{x}$ *such that the free variables of* M *are among the* $\overline{x}$*, it holds that* $\overline{x} \vdash MRM$*;*
- transitive*, if for any* $\overline{x} \in \overline{\boldsymbol{V}}$*,for any* M, N, L *with* $\overline{x} \vdash MRN$ *and* $\overline{x} \vdash NRL$*, it holds that* $\overline{x} \vdash MRL$*;*
- symmetric*, if for any* $\overline{x} \in \overline{\boldsymbol{V}}$*,for any* M, N *with* $\overline{x} \vdash MRN$*, it holds that* $\overline{x} \vdash NRM$*.*

We say that a transitive and reflexive open relation is an open preorder, while a transitive, reflexive and symmetric open relation is an open equivalence relation.

We give below the generic definition of *observational equivalence* for a language $\mathcal{L}$ with an observables function Obs(), and a set of contexts $\mathbf{C}$.

Definition 2.3.4 (Observational Equivalence) *Let* $\mathcal{L}$ *be an untyped observable programming language, with* $(\mathbf{C}_{\overline{x}})_{\overline{x}\in\overline{\boldsymbol{V}}}$ *its contexts. The* observational equivalence $\equiv_{\mathcal{L}}$ *is the open relation with respect to* $\mathcal{L}$ *defined by: for* $\overline{x} = x_1, \ldots, x_n$ *a family of variables of* $\mathcal{L}$*, and* M*,* N *two terms of* $\mathcal{L}$ *with free variables among the* $x_1, \ldots, x_n$ *, it holds that* $\overline{x} \vdash M \equiv^{obs}_{\mathcal{L}} N$*, when:*

$$\forall \mathcal{C} \in \mathbf{C}_{\overline{x}}, \quad Obs(\mathcal{C}[M]) = Obs(\mathcal{C}[N]).$$

We can see that $\equiv_{\mathcal{L},\mathbf{C}}$ is indeed an open equivalence relation. We will say that M and N are *observationally equivalent*—or *context equivalent*—if there exists a family $\overline{x}$ of variables with $\overline{x} \vdash M \equiv_{\mathcal{L}} N$.

We are now going to give some examples. We take here $\mathcal{L}$ as the probabilistic language $\Lambda_{\oplus}$. In the case of the (deterministic) λ-calculus, the evaluation strategy we choose has no effect on the equivalence between terms (it comes from the fact that the calculus is confluent). However, it is not the case anymore in the probabilistic case, as we are going to illustrate here by two examples.

Example 2.3.1 (CBV Evaluation Strategy) *Let us consider the two following terms M and N:*

$$M := (\lambda x.I) \oplus (\lambda x.\Omega) \qquad\qquad N := \lambda x.\,(I \oplus \Omega)\,.$$

Observe that the difference between these two terms is the place where we do the probabilistic choice: in the left term, the probabilistic choice is done when the term is evaluated, while in the right term, it is done when we pass an argument to the program and evaluate the result.

Let us first place ourselves in a call-by-value strategy. We consider the context:

$$\mathcal{C} = (\lambda y.(yI)(yI))[\cdot].$$

Using the definition of operational (CBV) semantics for $\Lambda_{\oplus}$ in Chapter 1, we can see that: $[\![\mathcal{C}[M]]\!]^{CBV} = \frac{1}{2}\{I^1\}$, and $[\![\mathcal{C}[N]]\!]^{CBV} = \frac{1}{4}\{I^1\}$. Observe that, when we know the semantics of a program L, then we know $Obs(L)$: indeed, $Obs(L)$ is the probability of termination of L, which is exactly the weight of the sub-distribution $[\![L]\!]$. As a consequence, we can see that: $Obs(\mathcal{C}[M]) = \frac{1}{2}$, and similarly $Obs(\mathcal{C}[N]) = \frac{1}{4}$: it shows that M and N are not observationally equivalent for a CBV evaluation strategy.

In Example 2.3.2 below, we are going to consider the same terms as in Example 2.3.1 above, but in the setting of a CBN evaluation strategy. We are going to see that, contrary to the CBV case, these two terms are equivalent.

Example 2.3.2 (CBN Evaluation Strategy) *Observe that the context $\mathcal{C}$ used in Example 2.3.1 can not anymore be used to distinguish M and N: indeed, we can see that $[\![\mathcal{C}[M]]\!]^{CBN} = [\![\mathcal{C}[N]]\!]^{CBN} = \frac{1}{4}\{I^1\}$. Actually, no context is going to work, because M and N are observationally equivalent for a CBN semantics. To show that, we need to show that* every *context $\mathcal{C}$ is unable to distinguish them, that is $|[\![\mathcal{C}[M]]\!]| = |[\![\mathcal{C}[N]]\!]|$. Recall that the definition of $[\![\cdot]\!]$ is based on the definition of an approximation semantics $\Downarrow$. It means that, to do the proof of equivalence, we need to use the approximation semantics. More precisely, we have to show that for any approximation semantics $\mathscr{D}$ of $\mathcal{C}[M]$, we can find an approximation semantics $\mathscr{E}$ of $\mathcal{C}[N]$ of at least as much weight as $\mathscr{D}$, and reversely. Formally, we can see that for every terms L such that $FV(L) \subseteq x$:*

- *for every sub-distribution $\mathscr{D}$ such that $L\{M/x\} \Downarrow \mathscr{D}$, there exist a sub-distribution $\mathscr{E}$ with $|\mathscr{D}| \leq |\mathscr{E}|$, and moreover $L\{N/x\} \Downarrow \mathscr{E}$;*
- *and the symmetric condition: $\forall \mathscr{E}$ such that $L\{N/x\} \Downarrow \mathscr{E}$, $\exists \mathscr{D}$ with $|\mathscr{D}| \geq |\mathscr{E}|$, and moreover $L\{M/x\} \Downarrow \mathscr{D}$.*

Example 2.3.2 show us that, even for terms which have a simple shape as M and N, the proof of equivalence become complicated. In Chapters 4 and 5, we will see that we can use characterization of the equivalence both in CBN and in CBV, that makes this kind of proofs much simpler.

We now give the definition of observational equivalence for a typed language. We call *open typed relations with respect to the language* $\mathcal{L}$ the families of relations $R = (R_{(\Gamma,\sigma)})_{\Gamma \text{ a typing context}, \sigma \in \mathscr{A}}$, where $R_{\Gamma,\sigma}$ is a relation over the set of those terms M such that $\Gamma \vdash M : \sigma$. Moreover, we will also note $\Gamma \vdash MRN : \sigma$ for denoting $M(R_{(\Gamma,\sigma)})N$.

Definition 2.3.5 (Observational Equivalence for typed Languages) *Let $\mathcal{L}$ be an typed observable programming language, with $(\mathbf{C}_{(\Gamma,\sigma)\to\tau})_{\overline{x}\in\overline{V}}$ its contexts. The* observational equivalence $\equiv_{\mathcal{L}}$ *is the open typed relation with respect to $\mathcal{L}$ defined by: for Γ a typing context, and σ a type, the relation $\equiv_{(\Gamma,\sigma)} \subseteq \{M \mid \Gamma \vdash M : \sigma\}^2$ is defined as:*

$$\overline{x} \vdash M \equiv^{obs}_{\mathcal{L}} N \quad \textit{when:} \quad \forall \tau \in \mathscr{A}_{obs}, \mathcal{C} \in \mathbf{C}_{(\Gamma,\sigma)\to\tau},\ Obs_\tau(\mathcal{C}[M]) = Obs_\tau(\mathcal{C}[N]).$$

2.3.1 On the Observational Equivalence on Open Terms

In Definition 2.3.4, we were considering all terms in the language, even those which are not closed. However, in this section, we are going to argue that it is actually sufficient to look at the equivalence between programs, i.e. closed terms: the idea is that, if we know everything on the equivalence between programs, then we are able to deduce if any two terms are equivalent or not. Let us suppose that we consider a language $\mathcal{L}$ such that: for any open terms M, and any family of variables $x_1, \ldots, x_n$ containing the free variables of M, it holds that M is observationally equivalent to $(\lambda x_1 \ldots \lambda x_n.M)x_1, \ldots, x_n$. It is a reasonable thing to ask, since these two terms are β-equivalent. In particular, we can see that it is the case for $\Lambda_\oplus$ with the CBV or CBN semantics.

With this hypothesis, it can be seen that the following holds:

$$x_1, \ldots, x_n \vdash M \equiv^{\text{obs}}_{\mathcal{L}} N \Leftrightarrow \vdash (\lambda x_1 \ldots \lambda x_n.M) \equiv^{\text{obs}}_{\mathcal{L}} (\lambda x_1 \ldots \lambda x_n.N)$$

These considerations will allow us, in the following, to focus on equivalence relation on *programs*. In particular, the coinductive definition of *applicative bisimularity*, that we will give in Chapter 4 for $\Lambda_\oplus$, doesn't take open terms into account.

2.3.2 Context Preorder

We would like also to express the fact that, no matter its execution environment, a program is going to terminate more often than another. That is the sense of the *observational context preorder*, which consists in an asymmetrical version of the observational equivalence.

Definition 2.3.6 *Let $\mathcal{L}$ be an untyped observable programming language. Let $x_1, \ldots, x_n$ be a family of variables of $\mathcal{L}$, and M, N two terms of $\mathcal{L}$ with free variables among the $x_1, \ldots, x_n$. We write $x_1, \ldots, x_n \vdash M \leq^{obs}_{\mathcal{L}} N$, and we say that M is observationally smaller than N, when for every $\mathcal{C} \in \mathbf{C}$ with $x_1, \ldots, x_n \vdash \mathcal{C}$ it holds that:*

$$Obs(\mathcal{C}[M]) \leq Obs(\mathcal{C}[N]).$$

Example 2.3.3 *We place ourselves in the CBV operational strategy, and we look again at the programs $M = (\lambda x.I) \oplus (\lambda x.\Omega)$ and $N = \lambda x.\,(I \oplus \Omega)$ of Example 2.3.1. Recall that we have shown in Example 2.3.1 that these two programs are not observationally equivalent, by building a context $\mathcal{C}$ such that $Obs(\mathcal{C}[N]) < Obs(\mathcal{C}[M])$. We will show in Section 5.2.4 of Chapter 5 that however they are comparable, in the sense that $N \leq_{\Lambda_\oplus} M$.*

Chapter 3

Go Beyond Observational Equivalence

As explained in the previous chapter, two programs are observationally equivalent if they have *exactly* the same behavior in any context. However, when we consider *quantitative* languages, for instance probabilistic languages, we could wish to be able to talk about not only equivalent programs, but also *close* programs, i.e. programs that behave *almost* the same way. We are going to illustrate this idea in Example 3.0.1 below.

Example 3.0.1 *For every $\epsilon \in]0,1]$, we define:*

$$M := \Omega \qquad \text{and} \qquad N_\epsilon := \Omega \oplus^\epsilon I.$$

Observe that, when ϵ become very small, M and N_ϵ have almost the same behavior: the program M never terminates, when N_ϵ terminates with the very small probability ϵ. However, it is immediate that these two terms are not equivalent whenever $\epsilon > 0$, since it is sufficient to consider the evaluation context $[\cdot]$ to distinguish them.

We would like to be able to use the *quantitative* difference between the observables to define a *metric* about terms: when we consider two programs, we want to know *how far* they are from each other. In that end, we are going to introduce a metric generalization of the observational equivalence in Section 3.1 below.

3.1 Observational Metric

We consider from now on a probabilistic programming language, hence a language with observation outcomes the set $\mathcal{O} = [0,1]$. The idea is to say that the context $\mathcal{C}$ is able to *distinguish* two terms M and N as far as $|\mathrm{Obs}(\mathcal{C}[M]) - \mathrm{Obs}(\mathcal{C}[N])|$, and then, similarly to the observational equivalence, we say that the *observational distance between M and N* is the supremum of how far any context can distinguish them. We need first to transform the *open relations* that we considered in Chapter 2 into *open quantitative valuations.*

Definition 3.1.1 *A quantitative valuation on a set X is simply a function $q : X \times X \to [0,1]$. An* open quantitative valuation with respect to a language $\mathcal{L}_\oplus$ *is a family $(q_{\overline{x}})_{\overline{x} \in \overline{\boldsymbol{V}}}$, where for each $\overline{x}$, $q_{\overline{x}}$ is a quantitative valuation on the terms M with $FV(M) \subseteq \overline{x}$.*

Definition 3.1.2 *Let $\mathcal{L}_\oplus$ be an untyped observable programming language, with $[0,1]$ as set of possible observation outcomes. We define $\delta^{ctx}_{\mathcal{L}_\oplus}$ as the open quantitative valuation*

with respect to $\mathcal{L}_\oplus$ defined as: for every set of variables $\overline{x}$, and M, N two terms of $\mathcal{L}_\oplus$ with free variables among the $\overline{x}$,

$$\delta^{ctx}_{\mathcal{L}_\oplus,\overline{x}}(M,N) := \sup\{|Obs(\mathcal{C}[M]) - Obs(\mathcal{C}[N])| \text{ with } \mathcal{C} \in \mathbf{C}_{\overline{x}}\}.$$

Observe that the observational distance behaves indeed as a distance, in the sense that is symmetric, reflexive, and verify the triangular inequality. Formally, it is exactly what is required to be a *pseudo-metric*, in the mathematical sense.

Definition 3.1.3 *Let S be any set. We say that a function $q : S \times S \to \mathbb{R}_+$ is a* pseudo-metric *if:*

- *q is reflexive, i.e. for every $x \in S$, $q(x,x) = 0$;*
- *q is symmetric, i.e. for every $x, y \in S$, $q(x,y) = q(y,x)$;*
- *the triangular inequality holds for q, i.e. for every $x, y, z \in S$, $q(x,z) \leq q(x,y) + q(y,z)$.*

Moreover, we can recover the observational equivalence from the observational distance, since the observational equivalence is exactly the *kernel* of the observational distance—i.e. two programs M and N are observationally equivalent if and only the observational distance between them is 0. It means that the observational distance encodes *at least as much* information as the observational equivalence.

We are going to illustrate Definition 3.1.2 here on a simple example: we are going to consider the two terms I and Ω. We know already that they are not equivalent, but we now look at *how much* different they are.

Example 3.1.1 *If we take $\mathcal{C} = [\cdot]$ as context, we see that $Obs(\mathcal{C}[I])$—the probability of termination of $\mathcal{C}[I] = I$—is equal to 1, and in the other hand, $Obs(\mathcal{C}[\Omega]) = 0$. It means that $\mathcal{C}$ pulls apart I and 1 from the maximal separation possible, which is 1. As a consequence, we see that $\delta^{ctx}_{\Lambda_\oplus}(\Omega, I) = 1$.*

We have seen that I and Ω are *as far as possible* with our definition of distance. Now, we are going to look at what happens when we consider programs that are intuitively *between* I and Ω: we consider the terms $N_\epsilon = \Omega \oplus^\epsilon I$, as in Example 3.0.1. Recall that we showed in Example 3.0.1 that the N_ϵ and Ω were not equivalent by considering the context $\mathcal{C}$. Using the same context, we see in Example 3.1.2 below that it gives us also quantitative information, i.e. that the distance between Ω and N_ϵ is greater or equal to ϵ.

Example 3.1.2 *For every $\epsilon \in]0,1]$, we define $N_\epsilon = \Omega \oplus^\epsilon I$. By considering again the context $\mathcal{C} = [\cdot]$, we see that $Obs(\mathcal{C}[N_\epsilon]) = \epsilon$. We know already, as seen in Example 3.1.1, that $Obs(\mathcal{C}[I]) = 1$, and $Obs(\mathcal{C}[\Omega]) = 0$. Looking at the definition of context distance, we see that it implies:*

$$\delta^{ctx}_{\Lambda_\oplus}(\Omega, N_\epsilon) \geq \epsilon \qquad \text{and} \qquad \delta^{ctx}_{\Lambda_\oplus}(I, N_\epsilon) \geq (1 - \epsilon).$$

Observe that, in Example 3.1.2, we show a lower bound on the distance between the programs we were considering. It is much more complicated to show an upper bound, since to do that, we have to consider the separation induced by *every possible* context. We can notice that, the more expressive power we allow to contexts, the more the distance between two given terms may decrease, since then the contexts are able to do more tests to separate the terms.

We look now at the observational distance for a typed probabilistic language $\mathcal{L}_\oplus$. We work here with *typed open quantitative valuations*, i.e. family of quantitative valuations $(q_{(\Gamma,\sigma)})_{\Gamma \text{ a typing context},\sigma\in\mathscr{A}}$ where each $q_{(\Gamma,\sigma)}$ is a quantitative valuation over $\{M \mid \Gamma \vdash M : \sigma\}$.

Definition 3.1.4 *Let $\mathcal{L}_\oplus$ be a typed observable programming language, with $[0,1]$ as set of possible observation outcomes. We define $\delta^{ctx}_{\mathcal{L}_\oplus}$ as the open quantitative valuation with respect to $\mathcal{L}_\oplus$ given by:*

$$\forall M, N \text{ with } \Gamma \vdash M :, N\sigma, \quad \delta^{ctx}_{\mathcal{L}_\oplus,(\Gamma,\sigma)}(M,N) := \sup_{\tau\in\mathscr{A}_{obs}} \{|\mathit{Obs}_\tau(\mathcal{C}[M]) - \mathit{Obs}_\tau(\mathcal{C}[N])| \text{ with } \mathcal{C} \in \mathbf{C}_{(\Gamma,\sigma)\to\tau}\}.$$

3.2 The Trivialization Problem

We argued, as a motivation to look at the distance, that it would allow us to express quantitative information, and not only whether two terms are equivalent or not. However, observe that it is not immediate that the observational distance is something else that an equivalence: depending of the expressive power of the contexts, they can indeed achieve to amplify every small difference to separate completely two terms.

3.2.1 Amplifying the Observed Distance by Copying

We are going to illustrate here, on an example, how, from a small difference in the semantics of two terms, it is sometimes possible to construct contexts that amplify it. Recall that we have shown in the previous section, that the distance between I and $N_\epsilon = \Omega \oplus^\epsilon I$ is at least $(1-\epsilon)$. We are going to show that, if the class of $\mathbf{C}$ we consider is *expressive enough*, then the distance between I and N_ϵ is actually 1 when $\epsilon > 0$.

Example 3.2.1 *Suppose that we consider a class of contexts $\mathbf{C}$, such that it contains the family of contexts:*

$$\mathcal{C}_n = (\lambda x.\underbrace{(xI)\dots(xI)}_{n \text{ times}})(\lambda y.[\cdot]).$$

We see that $\mathit{Obs}(\mathcal{C}_n[I]) = 1$: the program $\mathcal{C}_n[I]$ terminates with probability 1. On the other hand, it holds that $\mathit{Obs}(\mathcal{C}_n[N_\epsilon]) = \epsilon^n$: indeed the program $\mathcal{C}_n[N_\epsilon]$ terminates if every one of the n copies of N_ϵ terminates, and each copy terminates with probability ϵ.

As a consequence, the separation induced by the context $\mathcal{C}_n$ between I and N_ϵ is $(1-\epsilon^n)$. By taking the supremum over all these context $\mathcal{C}_n$, we see that the observational distance between I and N_ϵ is 1.

We can see that if we take $\mathbf{C}^{\Lambda_\oplus}$ (defined in 2.2) as our class of contexts, it will verify conditions of Example 3.2.1: indeed it has as much expressive power as the language $\Lambda_\oplus$ itself. More generally, observe that the important point here is the *ability of copying* of contexts: the idea is, if we are able to build a context $\mathcal{C}$ separating *a little* two programs, then we can build a context that has the same behavior as several sequential copies of $\mathcal{C}$.

3.2.2 Amplification Contexts

We can see that the behavior illustrated in Example 3.2.1 is not specific to the terms I and $\Omega \oplus^\epsilon I$. More generally, if we have two closed terms M, N, and that we have been able to construct a context $\mathcal{D}$, such that $\mathrm{Obs}(\mathcal{D}[M]) = 1$, and $\mathrm{Obs}(\mathcal{D}[N]) = (1-\epsilon)$, with $\epsilon < 1$, we can use the family of contexts $\mathcal{C}_n$ to *amplify* this distance ϵ. Indeed,it holds that

the separation induced between M and N by the context $\mathcal{C}_n[\mathcal{D}]$ is always $(1-(1-\epsilon)^n)$. It means that the family of contexts $\mathcal{C}_n$ gives us a *generic way* to amplify a difference between two terms that we have already been able to observe. For this reason, we say that the family of contexts $\mathcal{C}_n$ defined in Example 3.2.1 above are *amplification contexts*.

However, observe that the family $\mathcal{C}_n$ can amplify an observed difference only if one of the program $\mathcal{D}[M]$ or $\mathcal{C}[N]$ terminates with probability 1, where $\mathcal{D}$ is the original context separating M and N of a distance ϵ. As an illustration of that, we look in Example 3.2.2 below what happens if we take the programs Ω and $\Omega \oplus^{\epsilon} I$.

Example 3.2.2 *We have seen in Example 3.1.2 that the terms Ω and $\Omega \oplus^{\epsilon} I$ were at distance at least ϵ, and we have shown that this separation was done by the context $\mathcal{D} = [\cdot]$.*

We see that $Obs(\mathcal{C}_n[\Omega]) = 0$: the program $\mathcal{C}_n[I]$ never terminates. On the other hand, it holds (as before) that $Obs(\mathcal{C}_n[\Omega \oplus^{\epsilon} I]) = \epsilon^n$. But contrary to the previous case, we see here that using the context $\mathcal{C}_n$ does not amplify the difference: actually, the context $\mathcal{C}_n$ separates less *these two programs than the original context $[\cdot]$.*

3.2.3 Trivialization

By looking at Example 3.2.1 and 3.2.2, we see that in some circumstances, even a small difference in the behavior of two programs can be amplified until the maximal distance, but that building such amplification contexts can be complicated depending on the expressive power of the language, and on the programs we consider.

Actually, observe that it is not direct (and actually not always true, depending on the language we consider), that there *exist* terms such that the distance between them is not 0 or 1.

Definition 3.2.1 *Let be $\mathcal{L}$ a language, and $\mathbf{C}$ a class of contexts. We say that the observational distance on $\mathcal{L}$ with respect to the class* $\mathbf{C}$ trivializes*, if for every comparable programs M, N of the language, $\delta^{ctx}(M,N) \in \{0,1\}$.*

Observe that, if we know that the observational distance *trivializes*, it means that it is exactly the same thing as the observational equivalence. It means that trivialization of the observational distance can be seen as some kind of *negative result*, since it proves that it is not possible to talk about *quantitative* information about the similarity in terms, for a given language and class of context. It may also be seen as an indication that the class of context we consider is very expressive, and possibly too much to obtain the quantitative information we want.

3.2.4 Link with Cryptography

Here, we highlight a motivation for our developments on metrics, coming from security, in order to give intuition of what kind of *quantitative information* we may want. A central notion in security is the notion of *computational indistinguishability*: two programs are computationally indistinguishable, if for any adversary *running in polynomial time*, the *advantage of the adversary* is a negligible function of the security parameter. The advantage of the adversary is then defined as the probability for the adversary to be able to recognize if it interacts with the program M or with the program N. We can relate out notion of observational distance with computational indistinguishability by seeing the adversary as a context $\mathcal{C}$, and expressing its advantage using observables, i.e. as: $|\mathrm{Obs}(\mathcal{C}[M]) - \mathrm{Obs}(\mathcal{C}[N])|$.

Part II

Operational Reasoning on Discrete Probabilistic λ-calculi.

Chapter 4

Background: Applicative Bisimilarities for Higher-Order Probabilistic Calculi

As illustrated in the examples in Chapter 2, it is relatively easy to show that two given programs are *not* equivalent: this boils down to exhibiting *one* context able to distinguish them. However, it is much harder to show that these programs *are* equivalent, since we have to show that their observables are the same under *any* context, as highlighted in Example 2.3.2. For this reason, an important part of the work on equivalences for higher-order languages has been focused on developing notions or proof techniques for program equivalence. To be useful, such an alternative equivalence relation R should fulfill two objectives: proving that two programs are related by R has to be easier than proving that they are contextually equivalent, and moreover whenever two programs are R-related, they should also be context equivalent. The second requirement is the so-called *soundness property*, and is formalized in Definition 4.0.1 below.

Definition 4.0.1 *An equivalence relation R on programs is said to be* sound with respect to observational equivalence *if for any programs M and N, MRN implies $\vdash M \equiv^{obs} N$.*

A weaker requirement—but easier to enforce—that we can ask an equivalence notion to verify, is *observational correctness*: if two programs are equivalent, then their observables are the same. For most equivalence notions that we will consider in the following, observational correctness will be a immediate consequence of the definition.

Definition 4.0.2 *An equivalence relation R on programs is said to be an* observationally correct equivalence*, if for every programs such that $M\,R\,N$, it holds that $Obs(M) = Obs(N)$. A pre-order R on programs is said to be an* observationally correct pre-order*, if $M\,R\,N$ implies that $Obs(M) \leq Obs(N)$.*

Ideally, we would also like the relation R to *coincide* with context equivalence: when this is the case, we say that R is *fully abstract.*

Several different lines of work have been followed in order to define sound equivalence relations for higher-order languages. One possibility is to define a *denotational semantics*, i.e. a mathematical model of the language, and to say that two programs are equivalent when they have the same interpretation in the denotational model. We will review the results of this approach for the probabilistic case in Chapter 9, when we will present a fully abstract denotational model of $\mathsf{PCF}_{\oplus}$ introduced by Ehrhard, Pagani and Tasson

in [46]. In this section, we will focus on the approach initiated by Abramsky in [2] when he defined so-called *applicative bisimulation* for call-by-name λ-calculus.

4.1 Abramsky's Applicative Bisimulation

In concurrency theory, a widely used notion to define equivalences between *processes* is the one based on *bisimulations relations* (see [105] for an overview, and an historical take on the parallel origins of bisimulation notions in computer science, philosophy and set theory). Bisimulation is a generic coinductive way of defining equivalence relations for non-deterministic processes, which comes equipped with a proof technique to show that two states s, t of a system are indeed equivalent: it is sufficient to construct a relation containing (s, t) which moreover is a pre-fixpoint for some operator on relations. For programming languages, bisimulation-based equivalence have been first developed for the concurrent language CCS by Milner in [83]. In its pioneering work [2], Abramsky adapted it in order to define *applicative bisimulation* for an higher-order language, the deterministic λ-calculus with a weak call-by-name semantics. Programs of Λ are seen as processes, to which the environment may ask to perform some elementary *applicative* action, consisting in passing another program as argument to the current process. From there, Abramsky defined his equivalence relation for Λ based on ideas from bisimulations for such processes. Abramsky showed that his notion of applicative bisimulation is fully abstract for weak call-by-name Λ. From there, notions of applicative bisimulations have been shown to be sound for various λ-calculi, for instance for λ-calculus endowed with a non-deterministic choice operator [77], as well as with a probabilistic one [32].

In this chapter, we do a brief presentation of Abramsky's applicative bisimulation, and we formalize the link with bisimulation for processes, by giving the underlying Labeled Transition System used to define applicative bisimulation. Then, we present two sound applicative equivalence notions developed by Dal Lago, Sangiorgi and Alberti [32] for $\Lambda_{\oplus}$: trace equivalence, and probabilistic applicative bisimulation. We do this presentation in some details, since a major contribution of the present thesis is to generalize this applicative approach to a quantitative setting, with the aim of obtaining *metrics* sound with respect to *context distance*.

Abramsky's applicative bisimilarity comes actually in two flavors, depending on the underlying operating semantics of the weak λ-calculus, which can be call-by-name or as call-by-value: in a CBN setting, the environment may pass any argument to the program, while it can only pass *values* in a CBV setting. We present below Abramsky's applicative similarity for *call-by-name* Λ.

Definition 4.1.1 (From [2]) *A relation R on the closed terms of Λ is a* simulation *if for any programs M and N, MRN implies:*

- *if $M \downarrow V$, then there exists a normal form W such that $N \downarrow W$, and moreover VRW.*
- *if $(\lambda x.M)R(\lambda x.N)$, then for all programs L, $M\{L/x\}RN\{L/x\}$.*

A relation R is a bisimulation *if both R and R^{-1} are simulations (where R^{-1} stands for the R-opposite relation: $MR^{-1}N$ when NRM.).*

Example 4.1.1 *We give here an example of simulation to illustrate Definition 4.1.1. Let us consider: $M = \lambda y.\Omega$, and $N = \lambda y.((\lambda z.\Omega)I)$. We define $R = \{(M, N)\} \cup \{(\Omega, (\lambda z.\Omega)I)\}$. Observe that both R and R^{-1} are simulations, so R is a bisimulation.*

We consider simulations and bisimulations to be pre-ordered by the inclusion preorder. As shown in [2], the relation $\preceq^{\text{appl}}$ defined as $(\cup_{R|\text{a simulation}} R) \subseteq \mathbf{P} \times \mathbf{P}$—the union of all the simulations—is also a simulation, and in the same way the relation $\equiv^{\text{appl}}$ defined as $(\cup_{R|\text{a bisimulation}} R) \subseteq \mathbf{P} \times \mathbf{P}$ is also a bisimulation. It means that $\equiv^{\text{appl}}$ is the *greatest bisimulation*, and $\preceq^{\text{appl}}$ the greatest simulation.

Definition 4.1.2 (Abramsky's Applicative Bisimilarity) *We will call* applicative similarity *the relation $\preceq^{appl}$, and* applicative bisimilarity *the relation $\equiv^{appl}$. If $M \equiv^{appl} N$, we say that M and N are* bisimilar*.*

Moreover, $\preceq^{\text{appl}}$ is a pre-order, and $\equiv^{\text{appl}}$ an equivalence relation. We can also show that $(\equiv^{\text{appl}}) = (\preceq^{\text{appl}} \cap \preceq^{\text{appl}^{-1}})$[1]. If two programs are related by $\equiv^{\text{appl}}$, we say that they are *bisimilar*. We illustrate in Example 4.1.2 below the *bisimulation proof technique*: to show that two terms are bisimilar, it is sufficient to show that there exists a bisimulation that relates them.

Example 4.1.2 *Recall the programs M and N of Example 4.1.1. We have shown in Example 4.1.1 that there exists a bisimulation R, such that MRN. Since $\equiv^{appl}$ is the greatest bisimulation, it holds that $R \subseteq (\equiv^{appl})$, and consequently $M \equiv^{appl} N$.*

In [2], Abramsky showed that applicative bisimulation is *fully abstract* for call-by-name Λ, hence can be used as a complete proof technique for context equivalence, i.e that it is sound with respect to context equivalence. We illustrate on the example below how soundness of applicative bisimulation allows to turn the bisimulation proof technique into a proof technique for context equivalence.

Example 4.1.3 *Consider again the programs M and N defined in Example 4.1.1. In Example 4.1.2, we have shown that they are bisimilar. As a consequence, the soundness property of applicative bisimulation tells us that M and N are context equivalent.*

4.1.1 Modeling Λ Operational Semantics by a Labeled Transition System (LTS)

Abramsky's applicative bisimulation is actually an instance of bisimulation on Labeled Transition Systems (LTSs), a generic way of representing non-deterministic processes that evolve by interacting with their environment, by performing actions. Here, we make explicit the underlying LTS for applicative bisimulation. A LTS is a labeled directed graph, where the nodes are the different possible system states, the labels are all the possible actions the environment can do, and the edges specify how the system state changes when the environment does some action on the system.

Definition 4.1.3 *A Labeled Transition System (LTS) is a triple $\mathscr{L} = (\mathcal{S}, \mathcal{L}, \rightarrow)$, where $\mathcal{S}$ is a countable set of* states*, $\mathcal{L}$ is a countable set of* labels*, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ is the* transition relation *of $\mathscr{L}$.*

We say that a LTS is *deterministic* if it does not have internal non-determinism, i.e. that for any pair (s, a), there exists *at most* one state t such that $s \xrightarrow{a} t$. We say that an action a is *admissible from a state s* if there exists a state t with $(s, a, t) \in \rightarrow$.

Bisimulation techniques are a well-accepted way of defining equivalence for structures describing non-deterministic processes, for instance on relational structures (unlabeled

[1] It will still be true in the probabilistic case, but it doesn't hold for instance in the non-deterministic case.

directed graphs) or on Kripke structures (see [105]). We give below the formal definition of bisimulation in the LTS case, since it is the one we use for describing Abramsky's applicative bisimulation. Intuitively, we ask to a *simulation* R on the states of a LTS to verify the following property: if two states s and t are related by R,whenever s may reach a state s' by doing an action a, then t is able to *simulate* this move, i.e. to perform the same action a and to end up in a state t' such that $s'Rt'$.

Definition 4.1.4 *Let $\mathscr{L} = (\mathcal{S}, \mathcal{L}, \rightarrow)$ be a LTS.*

- *A* simulation *is a relation R on $\mathcal{S}$, such that $\forall s, t \in \mathcal{S}$ with sRt, it holds that whenever $s \xrightarrow{a} u$, there exists v such that $t \xrightarrow{a} v$, and moreover uRv.*
- *A bisimulation is a relation R such that both R and R^{-1} are simulations.*

The greatest simulation—called similarity and denoted $\preceq_{\mathscr{L}}$ —and the greatest bisimulation—called bisimilarity and denoted $\equiv_{\mathscr{L}}$—exist (see [32]), and they are respectively the union of all simulations and the union of all bisimulations; we call them respectively *similarity* and *bisimilarity* . We explicit now the link with Abramsky's applicative bisimulation, by building a deterministic LTS $\mathscr{L}_\Lambda$ modeling the operational semantics of Λ. Then, we will see that bisimilarity on $\mathscr{L}_\Lambda$ coincides with Abramsky's applicative bisimulation.

Definition 4.1.5 (The Applicative LTS for Λ) *We define $\mathscr{L}_\Lambda$ as the LTS $(\mathcal{S}_\Lambda, \mathcal{L}_\Lambda, \rightarrow_\Lambda)$ where we take:*

- *the set of states $\mathcal{S}_\Lambda = \boldsymbol{P}_\Lambda \uplus \widehat{\mathcal{V}}_\Lambda$, where closed terms and normal forms are taken as usual modulo α-equivalence and $\widehat{\mathcal{V}} = \{\widehat{V} | V \in \mathcal{V}\}$ is a set containing copies of the normal forms in Λ decorated with $\widehat{\ }$. We call these normal forms* distinguished normal forms.
- *the set of labels $\mathcal{L}_\Lambda = \boldsymbol{P}_\Lambda \uplus \{eval\}$, where, again, closed terms are taken modulo α-equivalence.*
- *the transition relation $\rightarrow$ defined by:*
  - *for every $M \in \boldsymbol{P}_\Lambda$, $M \xrightarrow{eval} \widehat{\lambda x.N}$ when $M \downarrow \lambda x.N$.*
  - *for every $\widehat{\lambda x.M} \in \widehat{\mathcal{V}}$ and for every $N \in \boldsymbol{P}_\Lambda$, $\widehat{\lambda x.M} \xrightarrow{N} M\{N/x\}$.*

We can see the LTS $\mathscr{L}_\Lambda$ as an interactive way of modeling the operational semantics of Λ: the environment is able to make a program evolves by asking it to do some action: either to be evaluated, or, if it is already a value, to be applied to some argument. We illustrate this idea by representing a fragment of $\mathscr{L}_\Lambda$ in Figure 4.1. We can see that the LTS $\mathscr{L}_\Lambda$ is indeed the underlying structure for applicative bisimulation, in the sense that $\preceq^{\text{appl}}$ and $\sim_{\mathscr{L}_\Lambda}$ coincide on programs, as well as $\equiv^{\text{appl}}$ and $\equiv_{\mathscr{L}_\Lambda}$.

4.1.2 The Non-Deterministic Case

Notions of applicative bisimulations have also been developed for Λ endowed with a non-deterministic operator $\oplus$. Recall from Chapter 2 that there are two different ways of defining context equivalence for non-deterministic higher-order languages: one where we take *may-convergence* as observable, and the other one where we take *must-convergence*. These two paradigms lead to two distinct ways of adapting Abramsky's applicative bisimilarity, which are sound respectively for may and must context equivalence (see the extensive study carried out by Lassen in his Phd Thesis [77] on equivalences for non-deterministic Λ). In particular, it was shown [77] that applicative bisimilarity in this setting is not *fully abstract*, neither for call-by-name nor for call-by-value operational semantics.

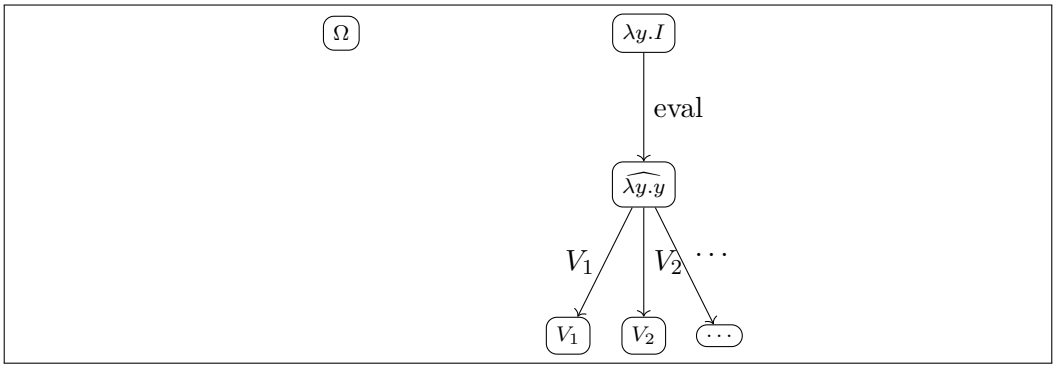


Figure 4.1: Fragment of the Applicative CBV LTS $\mathscr{L}_\Lambda$.

4.2 Sound Equivalences for $\Lambda_\oplus$.

We are going in this section to present two different generalizations of Abramsky's applicative bisimulation, developed by Dal Lago, Sangiorgi and Alberti in [74] for the weak probabilistic λ-calculus $\Lambda_\oplus$ endowed with a call-by-name semantics: *trace equivalence*, and *probabilistic applicative bisimilarity*. We will do this presentation in such a way as to highlight how both equivalences can be seen as arising from a probabilistic generalization of the Λ LTS.

4.2.1 Trace Equivalence for CBN $\Lambda_\oplus$.

The difficult part while proving that two terms are context equivalent is usually to handle the *universal quantification* on contexts. One way to make it easier may be to decide to consider only a restricted set $\mathscr{C}$ of contexts: to obtain a sound equivalence, we need to choose $\mathscr{C}$ such that, if no context in $\mathscr{C}$ is able to distinguish two programs, then there is no arbitrary context either that can distinguish them. Here, we present *trace equivalence*, also known as CIU equivalence, which is the equivalence notion obtained by restricting the contexts to *applicative contexts* , i.e contexts that are in the shape: $[\cdot]M_1 \dots M_n$, and so that correspond to the environment passing successively the terms $M_1, \dots, M_n$ as arguments to the program. In the deterministic λ-calculus, trace equivalence coincide with applicative bisimilarity. It is not the case anymore for non-deterministic λ-calculus (see [77]).

For CBN $\Lambda_\oplus$, trace equivalence was defined and studied in [74]. To highlight the link with bisimulation for LTSs, we give here the definition using *applicative tests*, that we'll also call traces, and that can be seen both as an applicative context, or as a sequence of actions done on the program by the environment.

Definition 4.2.1 *A* trace α *is a sequence in the form* $M_1 \cdot \cdots M_n$*, where* $M_1, \cdots M_n$ *are programs. In other words, traces are generated by the following grammar:*

$$\alpha ::= \epsilon \mid M \cdot \alpha$$

We note $\mathcal{T}r$ *the set of traces, and we associate to any trace* α *a context* $\mathcal{C}_\alpha$*, that we define inductively as:*

$$\mathcal{C}_\epsilon = [\cdot] \qquad \textit{and} \qquad \mathcal{C}_{M\cdot\alpha} = \mathcal{C}_\alpha[[\cdot]M].$$

Observe that the contexts associated to traces are exactly the applicative contexts, and as a corollary, they are also *evaluation* contexts. We may now *test a program M under a trace α* by evaluating $\mathcal{C}_\alpha[M]$, and this succeed whenever the execution of $\mathcal{C}_\alpha[M]$ terminates.

Definition 4.2.2 *We define the probability that the program M performs successfully under the trace α as $Prob(M\downarrow)(\alpha) = Obs(\mathcal{C}_\alpha[M])$.*

Two programs are *trace equivalent*, if their probability of success is the same for all traces, i.e. if no applicative context is able to distinguish them.

Definition 4.2.3 *We say that M and N are trace equivalent, and we note $M \equiv^{tr} N$, if for every trace α, it holds that $Prob(M\downarrow)(\alpha) = Prob(N\downarrow)(\alpha)$. We note $M \leq^{tr} N$ if for every tract α, $Prob(M\downarrow)(\alpha) \leq Prob(N\downarrow)(\alpha)$*

Example 4.2.1 *We consider the programs $M_1 = (\lambda x.I) \oplus (\lambda x.\Omega)$ and $M_2 = \lambda x.(I \oplus \Omega)$. We are going to show that they are trace equivalent, i.e. that for every trace α, $Prob(M_1\downarrow)(\alpha) = Prob(M_2\downarrow)(\alpha)$. The proof is done by case analysis on the trace α:*

- *$Prob(M_1\downarrow)(\epsilon) = Prob(M_2\downarrow)(\epsilon) = 1$.*
- *We look now to the case where $\alpha = L \cdot \beta$. Unfolding the definition that a program performs successfully under a trace α, we can see that for every term M, $Prob(M\downarrow)(\alpha) = Obs(\mathcal{C}_\beta[ML])$. Moreover, since $\mathcal{C}_\beta$ is an evaluation context, it holds that:*

$$Obs(\mathcal{C}_\beta[ML]) = \sum_{V\in\mathcal{V}} [\![ML]\!](V) \cdot Obs(\mathcal{C}_\beta[V]).$$

Since $[\![M_1L]\!] = [\![M_2L]\!] = \frac{1}{2} \cdot [\![L]\!]$, we can conclude that indeed $Prob(M_1\downarrow)(\alpha) = Prob(N_2\downarrow)(\alpha)$.

Proposition 4.2.1 (From [74]) *Trace equivalence is fully abstract for CBN $\Lambda_\oplus$.*

The definition of trace equivalence we have given in Definition 4.2.3, following [74], is inductive in nature. But trace equivalence may also be defined coinductively, using a LTS obtained by doing the probabilistic lifting of $\mathscr{L}_\Lambda$. We present here this alternative view, that allows us to see trace equivalence on $\Lambda_\oplus$ as a probabilistic generalization of Abramsky's Applicative bisimulation. To do that, we construct a LTS $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$, which we obtain from $\mathscr{L}_\Lambda$ by adding probabilities as follows: states are not programs anymore, but distributions over programs, while the labels of $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$ are defined as in $\mathscr{L}_\Lambda$. The transition function of $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$ is obtained from the one of $\mathscr{L}_\Lambda$ by lifting it to distributions.

Definition 4.2.4 *We define the* probabilistic trace LTS $\mathscr{L}({\Lambda_\oplus}^{cbn}) = (\mathcal{S}_{\mathscr{L}({\Lambda_\oplus}^{cbn})}, \mathcal{L}_{\mathscr{L}({\Lambda_\oplus}^{cbn})}, \rightarrow)$ *where:*

- *the set of states is $\mathcal{S}_{\mathscr{L}({\Lambda_\oplus}^{cbn})} = \Delta^{=}(\boldsymbol{P}_{\Lambda_\oplus}) \uplus \Delta^{=}(\widehat{\mathcal{V}}_{\Lambda_\oplus})$;*
- *the set of labels is $\mathcal{L}_{\mathscr{L}({\Lambda_\oplus}^{cbn})} = \boldsymbol{P}_{\Lambda_\oplus} \uplus \{eval\}$;*
- *the transition relation $\rightarrow$ is defined by:*
  - *if $\mathscr{D} \in \Delta^{=}(\boldsymbol{P}_{\Lambda_\oplus})$, $\mathscr{D} \xrightarrow{eval} \sum_{M\in\boldsymbol{P}_{\Lambda_\oplus}} \mathscr{D}(M) \cdot [\![M]\!]$.*

– *if* $\mathscr{D} \in \Delta^{=}(\widehat{\mathcal{V}}_{\Lambda_\oplus})$*, and* $N \in \boldsymbol{P}_{\Lambda_\oplus}$*:*

$$\mathscr{D} \xrightarrow{N} \sum_{\lambda x.M \in \widehat{\mathcal{V}}_{\Lambda_\oplus}} \mathscr{D}(\lambda x.M) \cdot \{M\{N/x\}^1\}.$$

We would like now to use bisimulation one $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$ to define an equivalence relation for $\Lambda_\oplus$ programs: two programs M and N would be equivalent whenever $\{M^1\}$ and $\{N^1\}$ are bisimilar as states of $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$. However, the notion of bisimulation for LTSs we have presented in Definition 4.1.3 is not coarse enough: indeed, we can see that for instance the states $\{I \oplus \Omega^1\}$ and $\{I^1\}$ are bisimilar in the LTS $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$. To overcome this problem, we need to specify that whenever two distributions have different weight, they cannot be bisimilar. To formalize this idea, we introduced an enriched notion of LTSs —*weighted LTSs*—, where we add to the LTS a function associating a weight to every state, and we ask for an bisimulation on weighted LTS, that every bisimilar states should have the same weight.

Definition 4.2.5 *A* weighted LTS (WLTS) *is a pair* $(\mathscr{L}, w)$*, where* $\mathscr{L} = (\mathcal{S}, \mathcal{L}, \to)$ *is a LTS, and* w *is a function* $\mathcal{S} \to [0,1]$*. A* simulation *on the WLTS* $(\mathscr{L}, w)$ *is a simulation* R *on the LTS* $\mathscr{L}$*, such that moreover* sRt *implies that* $w(s) \leq w(t)$*. A* bisimulation *on* $(\mathscr{L}, w)$ *is a simulation* R *on* $(\mathscr{L}, w)$ *such that* R^{-1} *also is a simulation on* $(\mathscr{L}, w)$*.*

Lemma 4.2.2 *The greatest simulation and bisimulation on a WLTS exist, and moreover they are respectively a pre-order and an equivalence. We call them respectively similarity and bisimilarity.*

Proof. Let be $\mathcal{S}$ the set of states of the WLTS. We first show that the greatest *simulation* exists. We define:

$$\widetilde{R} = (\cup_{R \text{ is a WLTS-simulation}} R) \subseteq \mathcal{S} \times \mathcal{S}.$$

We want to show that $\widetilde{R}$ is the greatest simulation; to do that it is enough to show that it is indeed a simulation. Let s, t be such that $s\widetilde{R}t$. Then there exists R a WLTS-bisimulation, such that sRt. We see immediately that it implies that $w(s) \leq w(t)$. Moreover, suppose that $s \xrightarrow{a} s'$. Then, since sRt, it holds that there exists t', such that $t \xrightarrow{a} t'$, and $s'Rt'$. Since $R \subseteq \widetilde{R}$, it means that $s'\widetilde{R}t'$, which allows us to conclude that $\widetilde{R}$ is a simulation.

We still have to show that $\widetilde{R}$ is a preorder. First, we see easily that it is reflexive: indeed $R = \{(s,s) \mid s \in \mathcal{S}\}$ is a simulation, and as a consequence $R \subseteq \widetilde{R}$. We want now to show that it is transitive: let s, t, u be such that $s\widetilde{R}t$, and $t\widetilde{R}u$. It means that there exists R_1, R_2 two simulations, such that sR_1t and tR_2u. We define $R_1 \circ R_2 = \{(s,u) \mid \exists t \text{ s.t. } sR_1t \wedge tR_2u\}$. We can see that $R_1 \circ R_2$ is a WLTS simulation, which implies that $\widetilde{R}$ is transitive.

The proof for bisimilarity is similar to the one for similarity. □

We can now enriched the LTS $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$ into a WLTS, by taking as weight function $w : \mathcal{S}_{\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})} \to [0,1]$ defined by: $w(\mathscr{D}) = \sum_{s \in \mathsf{S}(\mathscr{D})} \mathscr{D}(s)$. In the following, we will denote (by abuse of notation) also $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$ the WLTS $(\mathscr{L}({\Lambda_\oplus}^{\text{cbn}}), w)$.

Observe that, as in $\mathscr{L}_\Lambda$, the transition relation of the WLTS $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$ is deterministic: it comes from the fact that we have embedded probabilistic aspects on the semantics of $\Lambda_\oplus$ purely in the states. In the next section, we will present another way of adding probabilities to the LTS $\mathscr{L}_\Lambda$, by transforming it into a *labeled Markov Chain* where the transition function itself becomes probabilistic.

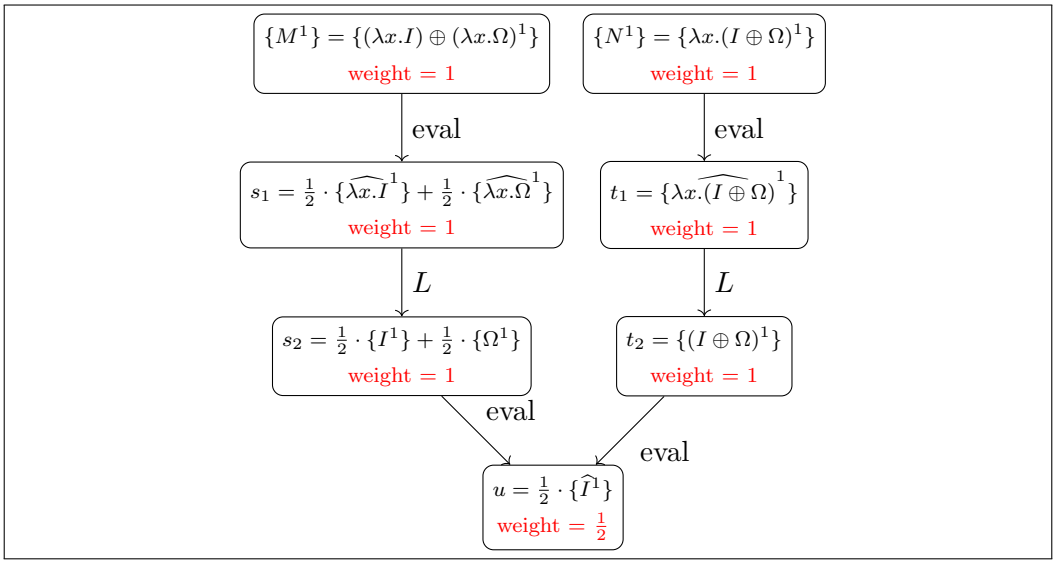


Figure 4.2: A Fragment of the weighted LTS $\mathscr{L}({\Lambda_{\oplus}}^{\text{cbn}})$

Example 4.2.2 *In Figure 4.2, we give as example a fragment of the weighted LTS that talks about programs of Example 4.2.1.*

Example 4.2.3 *We consider M and N as in Example 4.2.1:*

$$M = (\lambda x.I) \oplus (\lambda x.\Omega) \qquad \text{and } N = \lambda x.(I \oplus \Omega).$$

We define, using the notations s_i, t_j, u to denote the states of $\mathscr{L}({\Lambda_{\oplus}}^{cbn})$ as represented in Figure 4.2:

$$R = (\{M^1\}, \{N^1\}) \cup (s_1, t_1) \cup (s_2, t_2) \cup (u, u).$$

Looking at Figure 4.2, we can see that both R and R^{-1} are simulation, hence $\{M^1\}$ and $\{N^1\}$ are bisimilar as states of the WLTS $\mathscr{L}({\Lambda_{\oplus}}^{cbn})$.

On *deterministic* WLTS, bisimulation can be characterized by linear tests, i.e by looking at the behavior of states when we apply to them finite sequences of action.

Definition 4.2.6 (Traces for a LTS) *Let be $\mathscr{L} = (\mathcal{S}, \mathcal{L}, \to)$ a LTS. A* trace in $\mathscr{L}$ *is a (possibly empty) finite sequence of actions $\alpha = a_1, \ldots, a_n$ such that for every i, $a_i \in \mathcal{L}$. Formally, the set of traces is generated by the following grammar:*

$$\alpha \in \mathcal{T}_{\mathscr{L}} ::= \epsilon \mid a \cdot \alpha \qquad \text{where } a \in \mathcal{L}.$$

If $s \in \mathcal{S}$, and $\alpha \in \mathcal{T}_{\mathscr{L}}$, we denote $s \xrightarrow{\alpha} t$ when there exist $u_0, \ldots, u_n \in \mathcal{S}$, such that $u_0 = s$, $u_n = t$, and $\forall i$, $u_{i-1} \xrightarrow{a_i} u_i$. A trace α is said to be *admissible* for s if there exists t such that $s \xrightarrow{\alpha} t$. The set of admissible traces for s is indicated as $\mathcal{T}(s)$. We denote $\text{Dom}(\alpha)$ the set of those states s such that $\exists t$, $s \xrightarrow{\alpha} t$. The following is a well-posed definition since by definition the underlying LTS of a deterministic WLTS is deterministic.

Definition 4.2.7 (Success Probability of a Trace) *Let be $\mathscr{L} = (w, (\mathcal{S}, \mathcal{L}, \to))$ a deterministic WLTS. Let be $s \in \mathcal{S}$, and $\alpha \in \mathcal{T}(s)$. Then there is a unique u such that $s \xrightarrow{\alpha} u$, and we define the* success probability of α starting from s, *that we note $Prob(\alpha \downarrow)s$, as $w(u)$.*

We can now use the traces to give an inductive characterization of WLTS bisimilarity.

Definition 4.2.8 (Trace Equivalence for a Deterministic WLTS) *Let be $\mathscr{L}^w = (w, (\mathcal{S}, \mathcal{L}, \rightarrow))$ a deterministic WLTS. We define the* trace preorder *$\preceq^{tr}_{\mathscr{L}^w}$ on $\mathcal{L}$ as: $s \preceq^{tr}_{\mathscr{L}^w} t$ when for every trace t, if there exists s' such that $s \xrightarrow{\alpha} s'$, then there exists also t' such that $t \xrightarrow{\alpha} t'$, and moreover $w(s') \leq w(t')$.*

We say that two states $s, t \in \mathcal{S}$ are trace equivalent*, and we denote $s \equiv^{tr}_{\mathscr{L}^w} t$ when $s \preceq^{tr}_{\mathscr{L}^w} t$ and $t \preceq^{tr}_{\mathscr{L}^w} s$.*

Proposition 4.2.3 *Let be $\mathscr{L}$ a deterministic WLTS. Then two states s, t of $\mathscr{L}^w$ are WLTS-similar if and only if $s \preceq^{tr}_{\mathscr{L}^w} t$, and they are WLTS-bisimilar if and only if they are trace equivalent.*

Proof. We consider the deterministic WLTS $\mathscr{L} = (w, (\mathcal{S}, \mathcal{L}, \rightarrow))$

- Let be α a finite sequence over $\mathcal{L}$: we are going to show by induction on the length of α that if $s \preceq_{\mathscr{L}} t$, it holds that whenever $s \xrightarrow{\alpha} s'$, there exists t' with $t \xrightarrow{\alpha} t'$ and $w(s') \leq w(t')$.
  - If α is the empty sequence, then the result holds: indeed since s and t are WLTS-similar, it holds that $w(s) \leq w(t)$.
  - If $\alpha = a \cdot \beta$. We suppose that there exists s', such that $s \xrightarrow{a \cdot \beta} s'$. It means that there exists s'', such that $s \xrightarrow{a} s''$, and $s'' \xrightarrow{\beta} s'$. Since s and t are WLTS-similar, $s \xrightarrow{a} s''$ implies that there exists t'', such that $t \xrightarrow{a} t''$, and $s'' \preceq_{\mathscr{L}} t''$. We can now apply the induction hypothesis to β, and it tells us that there exists t' such that $t'' \xrightarrow{\beta} t'$, and moreover $w(s'') \leq w(t'')$. From there, we can deduce that $t \xrightarrow{\alpha} t'$, and we see that the result holds.
- Now, we suppose that $s \preceq^{\mathrm{tr}}_{\mathscr{L}} t$. In order to show that s and t are WLTS-similar, we construct a simulation relating them. We take:

  $$R = \{(s', t') \mid \exists \alpha,\ s \xrightarrow{\alpha} s' \wedge t \xrightarrow{\alpha} t'\}.$$

  We have now to show that it is a WLTS simulation. First, since $s \preceq^{\mathrm{tr}}_{\mathscr{L}} t$, we see that whenever $s' R t'$, it holds that $w(s') \leq w(t')$. Moreover, for any action a such that $s \xrightarrow{a} s'$, we can deduce from $s \preceq^{\mathrm{tr}}_{\mathscr{L}} t$ that there exists t' such that $t \xrightarrow{a} t'$, and moreover we see by looking at the definition of R that it also holds that $s' R t'$. So R is a WLTS-simulation, and the result holds.

□

Proposition 4.2.4 *Let be $\mathscr{L}$ a deterministic WLTS. For every $M, N \in \boldsymbol{P}_{\Lambda_{\oplus}}$, M and N are trace equivalent if and only if $\{M^1\}$ and $\{N^1\}$ are bisimilar in the WLTS $\mathscr{L}({\Lambda_{\oplus}}^{cbn})$.*

Proof. To do the proof, we are going to use Proposition 4.2.3: we see that it is enough to show that $\{M^1\} \equiv_{\mathscr{L}({\Lambda_{\oplus}}^{\mathrm{cbn}})} \{N^1\}$ whenever M and N are trace equivalent as $\Lambda_{\oplus}$-programs. We are going to use the fact that the traces applied to programs can be converted in finite sequence of actions on the WLTS $\mathscr{L}({\Lambda_{\oplus}}^{\mathrm{cbn}})$: for every $\alpha = N_1 \dots N_n \in \mathcal{T}r$, we denote $\hat{\alpha} = \mathrm{eval} \cdot N_1 \cdot \mathrm{eval} \dots N_n \cdot \mathrm{eval}$. We see that for every program M, it holds that $\mathrm{Prob}(M\downarrow)(\alpha) = w(\mathscr{D})$, where $\mathscr{D}$ is such that $\{M^1\} \xrightarrow{\hat{\alpha}} \mathscr{D}$.

- We suppose that $\{M^1\} \equiv^{\mathrm{tr}}_{\mathscr{L}({\Lambda_\oplus}^{\mathrm{cbn}})} \{N^1\}$. Then as we have observed above, for every trace $\alpha \in \mathcal{T}r$, there exists $\mathscr{D}, \mathscr{E}$ such that $\mathrm{Prob}(M\downarrow)(\alpha) = w(\mathscr{D})$, $\mathrm{Prob}(N\downarrow)(\alpha) = w(\mathscr{E})$ and $\{M^1\} \xrightarrow{\hat{\alpha}} \mathscr{D}$, $\{N^1\} \xrightarrow{\hat{\alpha}} \mathscr{E}$. Since $\{M^1\} \equiv^{\mathrm{tr}}_{\mathscr{L}({\Lambda_\oplus}^{\mathrm{cbn}})} \{N^1\}$, it holds that $w(\mathscr{D}) = w(\mathscr{E})$. It means that $\mathrm{Prob}(M\downarrow)(\alpha) = \mathrm{Prob}(N\downarrow)(\alpha)$, and since it holds for every $\alpha \in \mathcal{T}r$, we see that $M \equiv^{\mathrm{tr}} N$.

- Now, we suppose that $M \equiv^{\mathrm{tr}} N$, and we want to show that $\{M^1\} \equiv^{\mathrm{tr}}_{\mathscr{L}({\Lambda_\oplus}^{\mathrm{cbn}})} \{N^1\}$. Let α be a finite sequence of actions in $\mathscr{L}({\Lambda_\oplus}^{\mathrm{cbn}})$, and $\mathscr{D}$ such that $\{M^1\} \xrightarrow{\alpha} \mathscr{D}$. Our goal is to show that the exists $\mathscr{E}$ with $\{N^1\} \xrightarrow{\alpha} \mathscr{E}$, and $w(\mathscr{D}) = w(\mathscr{E})$. We do the proof by case analysis on α: first, we see that α has to be an alternated sequence of the shape $\mathrm{eval} \cdot L_1 \cdot \mathrm{eval} \ldots$, otherwise we would not have $\{M^1\} \xrightarrow{\alpha} \mathscr{D}$. There is actually only two possible cases:

  - or $\alpha = \mathrm{eval} \cdot L_1 \cdot \mathrm{eval} \ldots L_n \cdot \mathrm{eval}$, and then we see that there exists indeed $\mathscr{E} = \sum_V [\![N L_1 \ldots L_n]\!](V) \cdot \{\widehat{V}^1\}$ with $\{N^1\} \xrightarrow{\alpha} \mathscr{E}$, and moreover $w(\mathscr{D}) = \mathrm{Prob}(M\downarrow)(L_1 \ldots L_n)$, and $w(\mathscr{E}) = \mathrm{Prob}(N\downarrow)(L_1 \ldots L_n)$. Since M and N are trace equivalent, the result holds.
  - either $\alpha = \mathrm{eval} \cdot L_1 \cdot \mathrm{eval} \ldots L_n$, and then we see that there exists indeed $\mathscr{E} = \sum_K [\![N L_1 \ldots L_{n-1}]\!](\lambda x.K) \cdot \{K\{L_n/x\}^1\}$ with $\{N^1\} \xrightarrow{\alpha} \mathscr{E}$. Moreover $w(\mathscr{D}) = \mathrm{Prob}(M\downarrow)(L_1 \ldots L_{n-1})$, and $w(\mathscr{E}) = \mathrm{Prob}(N\downarrow)(L_1 \ldots L_{n-1})$, so as previously the result holds since M and N are trace equivalent.

□

The characterization of bisimilarity on deterministic WLTSs using traces makes it easier to show that two states are *not* bisimilar, as we highlight in Example 4.2.4 below.

Example 4.2.4 *We consider the two programs $M = \lambda x.I$, and $N = (\lambda x.I \oplus \lambda x.\Omega)$, in the setting of the CBN reduction theory. We associate to M and N the $\mathscr{L}({\Lambda_\oplus}^{cbn})$ states respectively $\{M^1\}$ and $\{N^1\}$. In Figure 4.3 we represent the relevant fragment of $\mathscr{L}({\Lambda_\oplus}^{cbn})$. We consider now the trace $\alpha = eval \cdot I \cdot eval$. We can see that $\alpha \in \mathcal{T}(\{M^1\}) \cap \mathcal{T}(\{N^1\})$; it means that we can look at the success probability of α starting from respectively $\{M^1\}$ and $\{N^1\}$. Looking at Figure 4.3, we see that $Prob(\alpha\downarrow)(\{M^1\}) = 1$, while $Prob(\alpha\downarrow)(\{N^1\}) = \frac{1}{2}$. Using the trace characterization of bisimilarity on $\mathscr{L}({\Lambda_\oplus}^{cbn})$, we see that it means that $\{M^1\}$ and $\{N^1\}$ are not bisimilar.*

4.2.2 Larsen Skou applicative bisimulation for CBN $\Lambda_\oplus$.

In [74], Dal Lago, Sangiorgi and Alberti introduced also another sound notion of equivalence for CBN $\Lambda_\oplus$, by adapting the Λ LTS into a Labeled Markov Chain (LMC)—aka reactive probabilistic labeled transition system—to model the operational semantics of $\Lambda_\oplus$, and then use the bisimulation notion introduced by Larsen and Skou for LMCs to define an equivalence on $\Lambda_\oplus$ programs, that they call probabilistic applicative bisimulation.

We first introduce the notion of Labeled Markov Chain, that also appears in the literature as *reactive probabilistic labeled transition systems*. It is a probabilistic generalization of LTSs, in the sense that when we do some action starting from some system state, we do not reach a single system state anymore, but a *distribution*—or sub-distribution—over states. LMCs allow for *external* non-determinism (several different actions may be performed from a given state), but not for *internal* non-determinism: given a state, and a

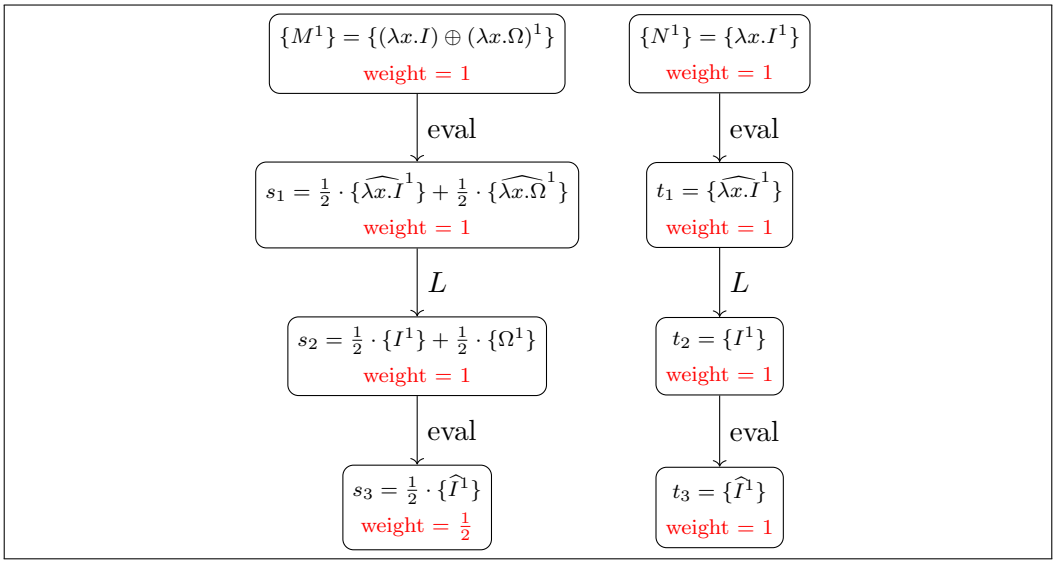


Figure 4.3: A Fragment of the weighted LTS $\mathscr{L}({\Lambda_{\oplus}}^{\mathrm{cbn}})$

label, the system can reach only *one* distribution over states. The LMCs we consider take divergence into account, since an action can lead to a strict sub-distribution. For this reason, it has been also called in the literature *partial LMC*, or *LMC with divergence*.

Definition 4.2.9 *A* labeled Markov chain *(LMC) is a triple $\mathscr{M} = (\mathcal{S}, \mathcal{L}, \mathcal{P})$, where $\mathcal{S}$ is a countable set of* states*, $\mathcal{L}$ is a countable set of* labels*, and $\mathcal{P}$ is a* transition probability matrix*, i.e., a function $\mathcal{P} : \mathcal{S} \times \mathcal{L} \times \mathcal{S} \to \mathbb{R}$ such that for every state $s \in \mathcal{S}$ and for every label $l \in \mathcal{L}$, $\sum_{t \in \mathcal{S}} \mathcal{P}(s, l, t) \leq 1$.*

Larsen and Skou gave in [75] a definition of bisimulation for Labeled Markov Chain.

Notation 4.2.5 *For any $R \subseteq X \times Y$, when A is a subset of X, we denote $R(A) = \{z \mid \exists x \in A,\ xRz\} \subseteq Y$.*

We can observe that this notation is well-behaved with respect to composition of relation, in the sense where for every relation R, S, it holds that $(R; S)(A) = S(R(A))$.

Definition 4.2.10 *Let X, Y be two countable sets, and $R \subseteq X \times Y$ a binary relation. We define the* asymmetric lifting of R to sub-distributions*, as the relation $\overrightarrow{(R)} \subseteq \Delta(X) \times \Delta(Y)$ given by:*

$$\mathscr{D}\ \overrightarrow{(R)}\ \mathscr{E} \quad \text{when} \quad \forall A \subseteq X,\ \mathscr{D}(A) \leq \mathscr{E}(R(A)).$$

We first state some easily checked fact about the asymmetric lifting of a relation.

Remark 4.2.1 *Let X, Y, Z be countable sets. Then the following properties hold:*

- *If $R \subseteq S$, then $\overrightarrow{(R)} \subseteq \overrightarrow{(S)}$.*
- *If $R \subseteq S \times T, S \subseteq T \times U$ binary relation, then $\mathscr{D}\ \overrightarrow{(R)}\ \mathscr{E}$ and $\mathscr{E}\ \overrightarrow{(S)}\ \mathscr{F}$ implies $\mathscr{D}\ \overrightarrow{(R; S)}\ \mathscr{F}$.*

- *If $(\mathscr{D}_i)_{i\in\mathbb{N}}$ and $(\mathscr{E}_i)_{i\in\mathbb{N}}$ are two countable family of sub-distributions over respectively X and Y such that for every i, it holds that $\mathscr{D}_i \ \overrightarrow{(S)}\ \mathscr{E}_i$, and $(\alpha_i)_{i\in\mathbb{N}}$ a family of non-negative coefficients with $\sum_{i\in\mathbb{N}}\alpha_i \leq 1$, then $\sum_{i\in\mathbb{N}}\alpha_i\cdot\mathscr{D}_i \ \overrightarrow{(S)}\ \sum_{i\in\mathbb{N}}\alpha_i\cdot\mathscr{E}_i$.*
- *If $\mathscr{E}\leq\mathscr{F}$, and $\mathscr{D}\ \overrightarrow{(R)}\ \mathscr{E}$, then also $\mathscr{D}\ \overrightarrow{(R)}\ \mathscr{F}$.*

Definition 4.2.11 *Let $\mathscr{M}=(\mathcal{S},\mathcal{L},\mathcal{P})$ be a labeled Markov chain. A* probabilistic simulation *is a relation R on $\mathcal{S}$ such that $(s,t)\in R$ implies that for every $l\in\mathcal{L}$, $(\mathcal{P}(s,l,\cdot))\ \overrightarrow{(R)}\ (\mathcal{P}(t,l,\cdot))$. A* probabilistic bisimulation *is a relation R on $\mathcal{S}$ such that both R and R^{-1} are probabilistic simulation relations. We say that s is simulated by t ($s\preceq_{\mathscr{M}}t$) if there exists a probabilistic simulation R such that sRt. States s,t are bisimilar ($s\equiv_{\mathscr{M}}t$) if there exists a probabilistic bisimulation R such that sRt.*

Lemma 4.2.6 *For any labeled Markov chain $\mathscr{M}=(\mathcal{S},\mathcal{L},\mathcal{P})$:*

- *relations $\preceq_{\mathscr{M}}$ and $\equiv_{\mathscr{M}}$ are the largest simulation and the largest bisimulation on $\mathcal{S}$, respectively;*
- *relation $\preceq_{\mathscr{M}}$ is a preorder and relation $\equiv_{\mathscr{M}}$ is an equivalence.*

We can adapt the Λ LTS to obtain a LMC $\mathcal{M}^{\text{cbn}}_{\Lambda_\oplus}$ for $\Lambda_\oplus$. The states and labels are the same as in $\mathscr{L}_\lambda$, while the transition relation $\to$ of $\mathscr{L}_\lambda$ is replaced by a probability matrix.

Definition 4.2.12 (from [74]) *The labeled Markov chain $\mathcal{M}^{cbn}_{\Lambda_\oplus}=(\mathcal{S}^{cbn}_{\Lambda_\oplus},\mathcal{L}^{cbn}_{\Lambda_\oplus},\mathcal{P}^{cbn}_{\Lambda_\oplus})$ is given by:*

- *the set of states is $\mathcal{S}^{cbn}_{\Lambda_\oplus}=\boldsymbol{P}_{\Lambda_\oplus}\uplus\widehat{\mathcal{V}}_{\Lambda_\oplus}$;*
- *the set of labels is $\mathcal{L}^{cbn}_{\Lambda_\oplus}=\boldsymbol{P}_{\Lambda_\oplus}\uplus\{eval\}$;*
- *the transition probability matrix $\mathcal{P}^{cbn}_{\Lambda_\oplus}$ is defined as:*
  - *for every $M\in\boldsymbol{P}_{\Lambda_\oplus}$ and for every $\hat{V}\in\mathcal{V}_{\Lambda_\oplus}$:*

$$\mathcal{P}^{cbn}_{\Lambda_\oplus}(M,eval,\hat{V})=[\![M]\!](V)$$
$$\mathcal{P}^{cbn}_{\Lambda_\oplus}(M,eval,M')=0\quad\forall M'\in\boldsymbol{P}_{\Lambda_\oplus};$$

  - *for every $\widehat{\lambda x.M}\in\widehat{\mathcal{V}_{\Lambda_\oplus}}$ and for every $N\in\boldsymbol{P}_{\Lambda_\oplus}$:*

$$\mathcal{P}^{cbn}_{\Lambda_\oplus}(\widehat{\lambda x.M},N,M\{x/N\})=1;$$
$$\mathcal{P}^{cbn}_{\Lambda_\oplus}(\widehat{\lambda x.M},N,M')=0\quad\forall M'\in\boldsymbol{P}_{\Lambda_\oplus}\ s.t.\ M'\neq M\{x/N\}.$$

Example 4.2.5 *We fix the CBN semantics, and we consider the terms $M=\lambda x.(I\oplus\Omega)$ and $N=(\lambda x.I)\oplus(\lambda x.\Omega)$. Recall from Example 4.2.3 that they are trace equivalent. However, we are going to show that they are not in $\equiv_{\mathcal{M}^{cbn}_{\Lambda_\oplus}}$. In Figure 4.4, we have represented the relevant fragment of the LMC $\mathcal{M}^{cbn}_{\Lambda_\oplus}$. We do the proof by contradiction: suppose that there exists a simulation R, such that MRN. We take $X=\{\widehat{\lambda y.(\Omega\oplus I)}\}$: it holds that $\mathcal{P}^{cbn}_{\Lambda_\oplus}(M)(eval)(X)=1$. From the definition of simulation for LMC, we can deduce that $\mathcal{P}^{cbn}_{\Lambda_\oplus}(N)(eval)(R(X))=1$. As a consequence (and by looking at Figure 4.4), it holds that $\widehat{\lambda y.(\Omega)}$ and $\widehat{\lambda y.I}$ are in $R(X)$, which means that both $\widehat{\lambda y.(\Omega\oplus I)}R\widehat{\lambda y.(\Omega)}$ and $\widehat{\lambda y.(\Omega\oplus I)}R\widehat{\lambda y.(I)}$. But from the latter, doing again one step in the bisimulation game, we can deduce: $(\Omega\oplus I)RI$. But then we reach a contradiction, since I can do the action eval with probability 1, while $\Omega\oplus I$ can do it with probability only $\frac{1}{2}<1$.*

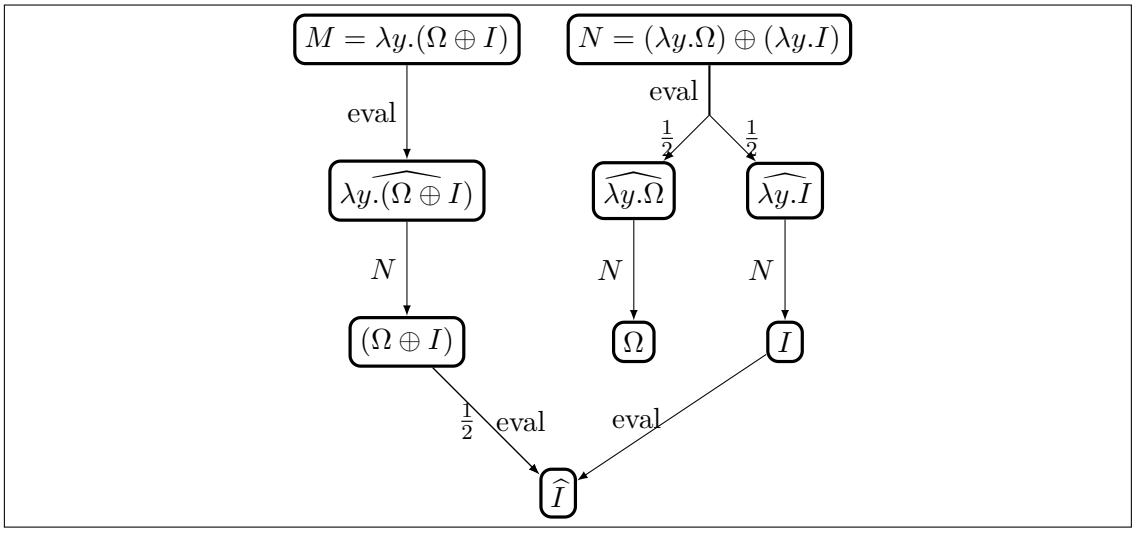


Figure 4.4: A Fragment of the LMC $\mathcal{M}^{\text{cbn}}_{\Lambda_\oplus}$.

As we have done for Λ with $\mathscr{L}_\Lambda$, we are now going to use bisimulation on the LMC $\mathcal{M}^{\text{cbn}}_{\Lambda_\oplus}$ to define an equivalence relation on $\Lambda_\oplus$ programs.

Definition 4.2.13 *We define $\equiv_{\Lambda_\oplus}$ the equivalence relation on the programs of $\Lambda_\oplus$ defined by: $M \equiv_{\Lambda_\oplus} N$ when M and N are bisimilar seen as states of $\mathcal{L}^{cbn}_{\Lambda_\oplus}$.*

Observe that Example 4.2.5 tells us that probabilistic applicative bisimulation is *coarser* than trace equivalence, since there exist terms that are trace equivalent, but that probabilistic applicative bisimulation is able to distinguish. As a corollary, since trace equivalence is fully abstract, it means that probabilistic applicative bisimulation cannot be complete.

Proposition 4.2.7 (from [74]) *Probabilistic applicative bisimulation for CBN $\Lambda_\oplus$ is sound, but not fully abstract.*

Chapter 5

Full Abstraction of Applicative Larsen-Skou's bisimilarity in CBV.

We have presented in Chapter 4 two distinct equivalence notions developed by Dal Lago, Sangiorgi and Alberti [74] for the probabilistic higher-order language $\Lambda_\oplus$ endowed with a *CBN semantics*: trace equivalence and probabilistic applicative bisimilarity. In the same work, they also investigate the link of these equivalences with context equivalence: trace equivalence is fully abstract, while probabilistic applicative bisimulation is sound, but not complete. One of the contributions of this thesis is a study of these equivalences in the setting of *call-by-value* $\Lambda_\oplus$: in this chapter, we show that CBV trace equivalence is not even sound, while CBV probabilistic applicative bisimulation is fully abstract.

Most of the results we present here have been published in a joint work with Ugo Dal Lago [24], in the setting of the typed language $\mathsf{PCFL}_\oplus$, which is a CBV probabilistic variant of PCFL—an extension of PCF with pairs and infinite lists introduced by Pitts in [92]. Furthermore, we also studied probabilistic applicative bisimulation for CBV $\Lambda_\oplus$ extended with Abramsky and Ong's *parallel convergence tester* operator, in a joint work with Dal Lago, Sangiorgi and Vignudelli [27]. To keep the proofs simple, we chose to present our results here in the setting of pure CBV $\Lambda_\oplus$, but it is worth noting that the techniques presented here are robust enough to be extended without much complications to higher-order languages with more programming features than $\Lambda_\oplus$. Throughout this chapter, we will denote by $\to$, $\Downarrow$, and $[\![\cdot]\!]$ respectively the one-step reduction relation, big-step evaluation relation and the operational semantics with respect to the weak CBV strategy on $\Lambda_\oplus$, as they were defined in Chapter 1.

5.1 Probabilistic Applicative Bisimulation for CBV $\Lambda_\oplus$.

We see in Chapter 4 that probabilistic applicative bisimulation was defined by Abramsky for weak CBN Λ in [2]. Ong introduced in [88] a variant of Abramsky's applicative bisimulation for *call-by-value* Λ, and showed it to be fully abstract with respect to context equivalence. At the end of their work on equivalences for CBN $\Lambda_\oplus$, Dal Lago, Sangiorgi and Alberti [74] consider briefly the CBV case; they give a definition of probabilistic applicative bisimulation in this setting, and claim that this equivalence relation is sound with respect to context equivalence. In this section, we present probabilistic applicative bisimulation for CBV $\Lambda_\oplus$, and we spell out the soundness proof, that uses Howe's method similarly to what is done in [74] for the CBN case.

5.1.1 The Labelled Markov Chain $\mathcal{M}^{\mathbf{cbv}}_{\Lambda_\oplus}$.

In order to adapt the definition of probabilistic applicative bisimulation—given in Definition 4.2.12 for CBN $\Lambda_\oplus$—to the CBV setting, we have first to slightly change the Markov Chain $\mathcal{M}^{\mathrm{cbn}}_{\Lambda_\oplus}$, in order to model CBV interactions instead of CBN ones: accordingly the labels of $\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}$ are *values* instead of being the set of all possible programs.

Definition 5.1.1 (from [74]) *The CBV labeled Markov chain $\mathcal{M}^{cbv}_{\Lambda_\oplus}$ is defined as $\mathcal{M}^{cbv}_{\Lambda_\oplus} = (\mathcal{S}^{cbv}_{\Lambda_\oplus}, \mathcal{L}^{cbv}_{\Lambda_\oplus}, \mathcal{P}^{cbv}_{\Lambda_\oplus})$ where:*

- *the set of states is $\mathcal{S}^{cbv}_{\Lambda_\oplus} ::= \mathcal{S}^{cbn}_{\Lambda_\oplus} = \boldsymbol{P}_{\Lambda_\oplus} \uplus \widehat{\mathcal{V}}_{\Lambda_\oplus}$.*
- *the set of labels is $\mathcal{L}^{cbv}_{\Lambda_\oplus} ::= \mathcal{V}\Lambda_\oplus \uplus \{eval\}$;*
- *the transition probability matrix $\mathcal{P}^{cbv}_{\Lambda_\oplus}$ is defined as:*
  - *for every $M \in \boldsymbol{P}_{\Lambda_\oplus}$ and for every $\hat{V} \in \mathcal{V}_{\Lambda_\oplus}$: $\mathcal{P}^{cbv}_{\Lambda_\oplus}(M, eval, \hat{V}) = [\![M]\!](V)$; and $\mathcal{P}^{cbv}_{\Lambda_\oplus}(M, eval, M') = 0 \quad \forall M' \in \boldsymbol{P}_{\Lambda_\oplus}$.*
  - *for every $\widehat{\lambda x.M} \in \widehat{\mathcal{V}_{\Lambda_\oplus}}$ and for every $V \in \mathcal{V}_{\Lambda_\oplus}$: $\mathcal{P}^{cbv}_{\Lambda_\oplus}(\widehat{\lambda x.M}, V, M\{x/V\}) = 1$; and $\mathcal{P}^{cbv}_{\Lambda_\oplus}(\widehat{\lambda x.M}, V, M') = 0$ for all $M' \in \boldsymbol{P}_{\Lambda_\oplus}$ such that $M' \neq M\{x/V\}$.*

Similarly to the CBN case—see Section 4.2.2 from Chapter 4—we define a preorder $\preceq^{\mathrm{cbv}}_{\Lambda_\oplus}$ and an equivalence relation $\equiv^{\mathrm{cbv}}_{\Lambda_\oplus}$ on $\Lambda_\oplus$ programs, by taking respectively the similarity and bisimilarity relations on the LMC $\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}$. The question now, is whether the bisimilarity $\equiv^{\mathrm{cbv}}_{\Lambda_\oplus}$ may be used to obtain some information on CBV $\Lambda_\oplus$ context equivalence. In Section 5.1.2 below, we show that $\equiv^{\mathrm{cbv}}_{\Lambda_\oplus}$ is sound with respect to context equivalence, and in Section 5.2 we show that it even *coincides* with context equivalence.

In order to keep the notations readable, in the remainder of this chapter, we will denote the CBV probabilistic applicative similarity $\preceq^{\mathrm{cbv}}_{\Lambda_\oplus}$ simply as $\preceq$, and the bisimilarity $\equiv^{\mathrm{cbv}}_{\Lambda_\oplus}$ simply as $\equiv$.

5.1.2 $\equiv^{\mathbf{cbv}}_{\Lambda_\oplus}$ is Sound wrt CBV Context Equivalence

Few years after Abramsky's introduction of applicative bisimulation, Howe [64] gave an alternative—more direct[1]—soundness proof for this equivalence relation. His proof technique was then adapted by Ong [88] to show soundness of applicative bisimulation for CBV Λ. Later, in [65], Howe gave a more general framework for his method, that encompassed both call-by-name and call-by-value reduction strategies, as well as non-deterministic variants of Λ obtained by adding some non-deterministic operator. From there, this proof method has been adjusted to higher-order languages with various programming features, for instance to higher-order languages with types [59], or with states [99]. When proving soundness for probabilistic applicative bisimulation in CBN $\Lambda_\oplus$, Dal Lago, Sangiorgi and Alberti adapted Howe's method to a *probabilistic* setting. Here, we are going to adapt their proof technique in order to show soundness of probabilistic applicative bisimilarity for CBV $\Lambda_\oplus$.

More precisely, Howe's method is designed to show a stronger result than *soundness*. Recall that—as defined in Section 2.3—we call *open relation* a family $R = (R_{\overline{x}})_{\overline{x} \subseteq \mathbf{V}}$, such that each $R_{\overline{x}}$ is a relation over the set of $\Lambda_\oplus$- terms that have their free variables contained in $\overline{x}$. Howe's method provides a path to prove that some fixed open relation R is *compositional*, in the following sense:

[1] Abramsky's original proof is done by way of a denotational semantics, while Howe's proof uses only operational semantics.

Definition 5.1.2 *Let R be an open relation for an untyped observable language $\mathcal{L}_\oplus$. We say that R is* compositional *if whenever two terms M and N are such that $M\, R_{\overline{x}}\, N$, then for every context $\mathcal{C}$ that captures all variables in $\overline{x}$, it holds that $\mathcal{C}[M]\, R_\emptyset\, \mathcal{C}[N]$.*

So the first step we have to take is to *extend* $\equiv$—a relation on *programs*—into an open relation on *all* terms: we do that below by defining the *open extension* of the simulation preorder.

Open Extension of Simulation Preorder.

To transform a relation R on programs into an open relation $(R)^\circ$ on all terms, there is a generic way formalized by instance in [92]: two terms are related by $(R)^\circ$, if for every possible choice of values[2] to replace their free variables, the programs obtained after doing the substitution are related by R.

Definition 5.1.3 *Let R be a relation on $\Lambda_\oplus$ program. We define the* call-by-value open extension *of R, that we denote $(R)^\circ$ by considering all closing substitutions, i.e., for all M, N with $FV(M), FV(N) \subseteq \overline{x} = \{x_1, \ldots, x_n\}$, we have $\overline{x} \vdash M (R)^\circ N$ if: for all $V_1, \ldots, V_n \in \mathcal{V}\Lambda_\oplus$,*

$$M\{x_1, \ldots, x_n / V_1, \ldots, V_n\} R N\{x_1, \ldots, x_n / V_1, \ldots, V_n\}.$$

Observe that, since $\preceq$ is a preorder on programs, its open extension $\preceq^\circ$ is also a reflexive and transitive open relation—in the sense of Definition 2.3.3 in Chapter 2. Our goal now is to show that $\preceq^\circ$ is compositional; we will see that among other things, compositionality for $\preceq^\circ$ induces soundness for the original relation $\preceq$ with respect to the context preorder.

Compatibility for Open Relations.

Instead of considering directly compositionality, we first introduce *compatibility*, an inductively defined property on open relations that guarantee compositionality. Intuitively, an open relation R is *compatible*, if when we start from pairs of programs related by R, and we apply any possible sequence of rules in the grammar of $\Lambda_\oplus$, we end up with a pair of program still in R.

Definition 5.1.4 *An open relation R is compatible if and only if the following conditions hold:*
(Com1) $\forall \overline{x} \in \overline{\boldsymbol{V}}, \forall x \in \overline{x},\ \overline{x} \vdash x R x$;
(Com2) $\forall \overline{x} \in \overline{\boldsymbol{V}}, \forall y \notin \overline{x}, \forall M, N,\ \overline{x}, y \vdash MRN \Rightarrow \overline{x} \vdash \lambda y.MR\lambda y.N$;
(Com3) $\forall \overline{x}, \forall M, N, P, Q,\ (\overline{x} \vdash MRN \wedge \overline{x} \vdash PRQ) \Rightarrow \overline{x} \vdash MPRNQ$;
(Com4) $\forall \overline{x}, \forall M, N, P, Q,\ \overline{x} \vdash (MRN) \wedge \overline{x} \vdash (PRQ) \Rightarrow \overline{x} \vdash (M \oplus P) R (N \oplus Q)$.

Observe that this definition is purely syntactic: in particular, it would be identical if we were in a call-by-name setting. As a consequence, our definition of compatibility is exactly the same as the one from Dal Lago, Sangiorgi and Alberti for CBN $\Lambda_\oplus$ [32].

We see now that compatibility is a *stronger* requirement than compositionality: first, observe that we cannot derive (Com3) by compositionality, since the contexts we consider have only *one* hole. However, compatibility *implies* compositionality, as stated in Lemma 5.1.1 below.

[2]Our definition of open extension depends on our reduction strategy; indeed in CBV, only values may be substituted during the execution of a program, while in CBN all terms may be.

Lemma 5.1.1 *Let R be a compatible open relation with respect to $\Lambda_\oplus$. Then the following hold:*

1. *R is a reflexive open relation;*
2. *R is* compositional*: if $\mathcal{C}$ is any context such that $\overline{x} \dashv \mathcal{C}$, and $\overline{y}, M, N$ are such that $\overline{x}, \overline{y} \vdash M\, R\, N$, then it holds that $\overline{y} \vdash \mathcal{C}[M]\, R\, \mathcal{C}[N]$.*

Proof.

- We first show that R is reflexive. Let M be such that $FV(M) \subseteq \overline{x}$. We want to show that $\overline{x} \vdash M\, R\, M$. The proof is done by induction on the structure of M.
  - We first suppose that $M = x$. Since $x \in FV(M)$, it holds that $x \in \overline{x}$. As a consequence, we see using (Com1) that $\overline{x} \vdash xRx$.
  - We consider now the case where $M = \lambda x.N$. Since terms are considered modulo α-equivalence, we can suppose that $x \notin \overline{x}$. Moreover, we see that $FV(N) \subseteq \overline{x}, x$. As a consequence, we have by induction hypothesis that $\overline{x}, x \vdash NRN$. Applying (Com2), we can conclude that $\overline{x} \vdash MRM$.
  - If $M = NL$, we have by induction hypothesis that $\overline{x} \vdash NRN$ and $\overline{x} \vdash LRL$. We can then conclude by applying (Com3).
  - If $M = N \oplus L$, we obtain the result by applying (Com4).
- We now prove that R is compositional, by induction on the validity proof tree of $\kappa : \overline{x} \dashv \mathcal{C}$. Recall that validity for closeness judgments on contexts is defined inductively in Figure 2.1 of Chapter 2.
  - If κ is $\dashv [\cdot]$, then $\mathcal{C}[M] = M$, $\mathcal{C}[N] = N$, and the result holds.
  - If κ is $\overline{x}, x \dashv \lambda x.\mathcal{D}$, then the last rule in the proof tree of κ is of the shape: $\dfrac{\overline{x} \dashv \mathcal{D} \qquad x \notin \overline{x}}{\overline{x}, x \dashv \lambda x.\mathcal{D}}$. Then $\mathcal{C}[M] = \lambda x.\mathcal{D}[M]$, while $\mathcal{C}[N] = \lambda x.\mathcal{D}[N]$. By induction hypothesis, it holds that $x, \overline{y} \vdash (\mathcal{D}[M])\, R\, (\mathcal{D}[N])$. It allows us to apply (Com2), and to conclude that $\overline{y} \vdash (\lambda x.\mathcal{D}[M])\, R\, (\lambda x.\mathcal{D}[N])$.
  - If κ is $\overline{x} \dashv \mathcal{D} \oplus L$. Looking at the rules in Figure 2.1, it means that $\overline{x} \dashv \mathcal{D}$, and $L \in \mathbf{P}$. By induction hypothesis, we obtain that $\vdash \mathcal{D}[M]\, R\, \mathcal{D}[N]$. The first statement of Lemma 5.1.1 tells us that R is reflexive: as a consequence, since L is a closed term, $FV(L) = \emptyset$, and consequently $\vdash L\, R\, L$. We can conclude by applying (Com4).

□

We explain now the relevance of the compatibility property for our purposes, namely proving soundness of probabilistic applicative bisimulation. If a binary relation R on programs is an *observationally correct equivalence*—in the sense of Definition 4.0.2 in Chapter 4, i.e an equivalence relation such that whenever two programs are connected, they have the same termination probability—then compatibility of $(R)^\circ$ implies soundness with respect to context equivalence. A similar result holds in the asymmetric case: if R is a *observationally correct preorder*—i.e. a preorder such that whenever N is related to M, the termination property of M is greater or equal to that of N—then as soon as $(R)^\circ$ is compatible, it is also sound with respect to the context preorder.

Proposition 5.1.2 *Let be R a relation on $\Lambda_\oplus$ programs, such that $(R)^\circ$ is compositional. Then:*

1. *if R is an observationally correct preorder, then $(R)^\circ$ is sound with respect to the context preorder;*
2. *if R is an observationally correct equivalence, then $(R)^\circ$ is sound with respect to context equivalence.*

Proof. We suppose that R is a relation on programs verifying item 1, and such that moreover $(R)^\circ$ is compatible. Let be M, N such that $\overline{x} \vdash M (R)^\circ N$; our objective is to show that $\overline{x} \vdash M \leq^{\text{obs}} N$. Let be $\mathcal{C}$ any context in the set $\mathbf{C}^{\Lambda_\oplus}$ such that $\overline{x} \dashv \mathcal{C}$: it guarantees that both $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are closed terms. Since $(R)^\circ$ is compositional, Lemma 5.1.1 tells us that $\overline{x} \vdash M (R)^\circ N$ implies $\vdash \mathcal{C}[M] (R)^\circ \mathcal{C}[N]$. Looking at the definition of the open extension, we see that since $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are closed terms, it means: $\vdash \mathcal{C}[M] R \mathcal{C}[N]$. Now, we apply (1), and we can deduce from there that $\text{Obs}(\mathcal{C}[M]) \leq \text{Obs}(\mathcal{C}[N])$. Since it is the case for any context $\mathcal{C}$ with $\overline{x} \dashv \mathcal{C}$, it means that $\overline{x} \vdash M \leq^{\text{obs}} N$.

The proof of (2) is done similarly □

From there, we may see that in our path towards proving soundness of $\preceq$ and $\equiv$ with respect to respectively context preorder and context equivalence, the key step consists in showing that $\preceq^\circ$ is compatible. Indeed, since $\preceq$ is observationally correct, Proposition 5.1.2 tells us that compatibility of $\preceq^\circ$ implies soundness of $\preceq$ with respect to context equivalence. Moreover, soundness of $\equiv$ with respect to context equivalence is implied as soon as we know that $\preceq$ is sound with respect to the context preorder—because $\equiv$ coincides with $\preceq \cap (\preceq)^{-1}$ (see Chapter 4), and also $\equiv^{\text{ctx}}$ coincides with $\leq^{\text{ctx}} \cap (\leq^{\text{ctx}})^{-1}$.

Howe's Method for CBV $\Lambda_\oplus$.

This paragraph is devoted to show that $\preceq^\circ$ is indeed compatible, by adapting Howe's method to our setting. The main idea behind Howe's method is to define an auxiliary relation $(\preceq^\circ)^H$, which is compatible *by design*, and thereafter to show that in fact $\preceq^\circ$ and $(\preceq^\circ)^H$ coincides. This auxiliary relation $(\preceq^\circ)^H$ is build as the *Howe's lifting of* $\preceq^\circ$, where Howe's lifting is a generic way to construct syntactically a bigger open relation R^H starting from any reflexive open relation R.

Definition 5.1.5 *Let R be an open relation. The* Howe's lifting of R *is the open relation R^H defined inductively by the rules in Figure 5.1.*

$$\frac{\overline{x} \cup \{x\} \vdash x \, R \, M}{\overline{x} \cup \{x\} \vdash x \, R^H \, M} \qquad \frac{\overline{x} \cup \{x\} \vdash M \, R^H \, N \quad \overline{x} \vdash \lambda x.N \, R \, L}{\overline{x} \vdash \lambda x.M \, R^H \, L}$$

$$\frac{\overline{x} \vdash M \, R^H \, N \quad \overline{x} \vdash L \, R^H \, K \quad \overline{x} \vdash NK \, R \, T}{\overline{x} \vdash ML \, R^H \, T}$$

$$\frac{\overline{x} \vdash M \, R^H \, N \quad \overline{x} \vdash L \, R^H \, K \quad \overline{x} \vdash (N \oplus K) \, R \, T}{\overline{x} \vdash (M \oplus L) \, R^H \, T}$$

Figure 5.1: Howe's Construction

Observe that Howe's construction is asymmetric in nature; it is the reason why Howe's method has to focus on the *similarity preorder*, instead of considering directly bisimilarity. Since the definition of Howe's lifting depends only on the language, and not on the

reduction strategy, our definition of R^H coincides with the one presented in [74]. It means that the structural properties of Howe's lifting shown in [74] still hold: we state them in Lemma 5.1.3 below.

Lemma 5.1.3 (from [74]) *Let R be an open relation with respect to $\Lambda_\oplus$.*

- *Compatibility: If R is reflexive, then R^H is compatible.*
- *Pseudo-Transitivity: If R is transitive, then: $\left(\overline{x} \vdash M\, R^H\, N\right) \wedge (\overline{x} \vdash N\, R\, L) \Rightarrow \left(\overline{x} \vdash M\, R^H\, L\right)$.*
- *Extended Reflexivity: If R is reflexive, then $\overline{x} \vdash M\, R\, N$ implies $\overline{x} \vdash M\, R^H\, N$.*

Observe that these properties do not imply *a priori* that R^H is a *preorder*, since pseudo-transitivity is weaker that transitivity. When moreover it holds that $\overline{x} \vdash M\, R^H\, N$ implies $\overline{x} \vdash M\, R\, N$ —i.e. that the Howe's lifting is contained in the original relation—we say that the Howe's lifting of R is *conservative*.

When building $\preceq^\circ$ as the open extension of $\preceq$, we have noted that $\preceq^\circ$ is a reflexive and transitive open relation. As a consequence, we can deduce from Lemma 5.1.3 that $(\preceq^\circ)^H$ is compatible, and contains $\preceq^\circ$. So to conclude that $\preceq^\circ$ is compatible, we just need to know that the Howe's lifting of $(\preceq^\circ)^H$ is conservative, i.e that $(\preceq^\circ)^H \subseteq \preceq^\circ$. Recall that to show that some relation R on programs in contained in $\preceq$, there is the powerful *coinductive proof technique*, i.e. it is enough to show that R is a simulation. Here, we need to adapt this tool for the purpose of showing that an open relation is contained in $\preceq^\circ$. For an open relation R, we call the binary relation $R_\emptyset$ over $\mathbf{P}$ the *restriction of R to programs*, and we denote it $R_{|\mathbf{P}}$. We would like to know that hypothesizing $(\preceq^\circ)^H_{|\mathbf{P}} \subseteq \preceq$,implies that $(\preceq^\circ)^H \subseteq \preceq^\circ$. It comes as a consequence from the fact that $(\preceq^\circ)^H$ is *value-substitutive*, in the sense of the definition below.

Definition 5.1.6 *Let be R an open relation. Then we say that R is* value-substitutive *if for all terms M, N and values V, W such that $\overline{x}, x \vdash M (R)^{\circ H} N$ and $\emptyset \vdash V (R)^{\circ H} W$, it holds that $\overline{x} \vdash M\{x/V\}\, (R)^{\circ H}\, N\{x/W\}$*

Looking at the structure of Howe's lifting, we can see that $(\preceq^\circ)^H$ is value-substitutive. That's because Howe's lifting is designed in such a way that for any relation R on programs $(R)^{\circ H}$ is value-substitutive, as expressed in Lemma 5.1.4 below.

Lemma 5.1.4 *Let be R a relation on $\Lambda_\oplus$ programs. Then $(R)^{\circ H}$ is value-substitutive.*

Proof. We have to show that $\forall M, N \in$ Terms, and $V, W \in \mathcal{V}$ such that $\overline{x}, x \vdash M (R)^{\circ H} N$ and $\emptyset \vdash V (R)^{\circ H} W$, it holds that $\overline{x} \vdash M\{x/V\}\, (R)^{\circ H}\, N\{x/W\}$. The proof is by induction on the proof derivation for $\overline{x}, x \vdash M((R)^\circ)^H N$.

- If $M = x$ and $\overline{x}, x \vdash x(R)^\circ N$—with $x \notin x$—then the result holds by definition of call-by-value open extension: for all $V \in \mathcal{V}\Lambda_\oplus$, $\overline{x} \vdash V(R)^\circ N\{x/V\}$.

- In the case where the derivation is of the form:

$$\frac{\overline{x}, x, y \vdash P\, (R)^{\circ H}\, Q \qquad \overline{x}, x \vdash \lambda y.Q\, (R)^\circ\, N}{\overline{x}, x \vdash \lambda y.P\, (R)^{\circ H}\, N}$$

  Then it holds by induction hypothesis that $\overline{x}, y \vdash P\{V/x\}\, (R)^{\circ H}\, Q\{V/x\}$, and by definition of open extension, it holds that $\overline{x} \vdash \lambda y.Q\{V/x\}\, (R)^\circ\, N\{V/x\}$. From there, by applying the abstraction rule of Howe's construction, we obtain the result.

The proof is similar in the remaining cases. □

We now see that value-substitutivity allows us to extend the coinductive proof technique to *open* relations: to show that a value-substitutive open relation R is contained in $\preceq^{\circ}$, it is enough to show that its restriction to program is a simulation, hence contained in $\preceq$. The validity of this proof technique comes as a direct consequence of Lemma 5.1.5 below.

Lemma 5.1.5 *Let R be a reflexive, value-substitutive open relation, and S a relation on programs, such that $R_{|\boldsymbol{P}} \subseteq S$. Then it holds that $R \subseteq (S)^{\circ}$.*

Proof. We suppose that $R_{|\mathbf{P}} \subseteq S$. Let $\overline{x} = \{x_1, \ldots, x_n\} \in \overline{\mathbf{V}}$, and M, N be such that $\overline{x} \vdash M\, R\, N$. We want to show that $\overline{x} \vdash M\, (S)^{\circ}\, N$. Recall the definition of open extension given in Definition 5.1.3: we have to show that for all $V_1, \ldots, V_n$ values, it holds that $M\{x_1, \ldots, x_n/V_1, \ldots, V_n\}\, S\, N\{x_1, \ldots, x_n/V_1, \ldots, V_n\}$. Since R is value-substitutive, we can deduce from $\overline{x} \vdash M\, R\, N$ that:

$$\emptyset \vdash M\{x_1, \ldots, x_n/V_1, \ldots, V_n\}\, R\, N\{x_1, \ldots, x_n/V_1, \ldots, V_n\},$$

which allows us to conclude the proof since $R_{\emptyset} = R_{|\mathbf{P}}$ is contained in S. □

Since $(\preceq^{\circ})^H$ is value-substitutive—from Lemma 5.1.4—Lemma 5.1.5 tells us that conservativity of the Howe's lifting of $\preceq^{\circ}$ will be implied as soon as we know that $(\preceq^{\circ})^H{}_{|\mathbf{P}}$ is contained in $\preceq$. For this purpose, we use the *coinductive proof technique*, and show that $(\preceq^{\circ})^H{}_{|\mathbf{P}}$ is a simulation[3] on the LMC $\mathcal{M}^{\mathrm{cbv}}_{\Lambda_{\oplus}}$. To be precise, we have to consider the binary relation on $\mathcal{S}^{\mathrm{cbv}}_{\Lambda_{\oplus}}$ naturally induced by $(\preceq^{\circ})^H_{|\mathbf{P}}$: M and N are connected as soon as $M\, (\preceq^{\circ})^H_{|\mathbf{P}}\, N$, and $\widehat{V}$, $\widehat{W}$ are connected as soon as $V\, (\preceq^{\circ})^H_{|\mathbf{P}}\, W$. We will denote $(\preceq^{\circ})^H_{|\mathcal{S}^{\mathrm{cbv}}_{\Lambda_{\oplus}}}$ this relation on $\mathcal{S}^{\mathrm{cbv}}_{\Lambda_{\oplus}}$.

We first recall the *Disentangling Lemma*, shown by Dal Lago, Sangiorgi and Alberti for the purpose of the soundness proof for probabilistic applicative bisimilarity in CBN. Our formulation here is actually a corollary of their Disentangling Lemma. The intuition is that when $\mathscr{D}$ and $\mathscr{E}$ are related by the asymmetric lifting of R, then it is possible to go from $\mathscr{D}$ to $\mathscr{E}$ by moving weight on paths that are *valid* for R, that is the path is either in R, or go from the divergent state $\bot$ to a state of T.

Lemma 5.1.6 (From [32]) *Let S, T be two countable sets, and $R \subseteq S \times T$ a binary relation. Let $\mathscr{D}$ be a* finitely supported *distribution over S, and $\mathscr{E}$ any sub-distribution over T. Then $\mathscr{D}\ \overrightarrow{(R)}\ \mathscr{E}$ if and only if there exists a sub-distribution π over R such that :*

- *for every $t \in S$, $\sum_{s \in \mathsf{S}(\mathscr{D})} \pi(s,t) \leq \mathscr{E}(t)$;*
- *for every $t \in T$, $\sum_{t \in \mathsf{S}(\mathscr{E})} \pi(s,t) = \mathscr{D}(s)$.*

We will say that such a π is an asymmetric coupling from $\mathscr{D}$ to $\mathscr{E}$ that respects R, *and we will write $\pi \blacktriangleleft \langle \mathscr{D} \& \mathscr{E} \rangle$.*

Lemma 5.1.7 (Key Lemma) *$(\preceq^{\circ})^H_{|\mathcal{S}^{cbv}_{\Lambda_{\oplus}}}$ is a simulation on the LMC $\mathcal{M}^{cbv}_{\Lambda_{\oplus}}$.*

[3] In the proof of congruence for the probabilistic call-by-value λ-calculus in [74], we had to consider the transitive closure of $(\preceq^{\circ})^H$, since the definition of simulation required the relation to be preorder, which implies that the transitivity of $(\preceq^{\circ})^H$ is needed. Here, however, following the presentation in [27], we have relaxed the definition of simulation, so this is not anymore necessary.

Proof. Looking at the definition of simulation, and at the structure of the LMC $\mathcal{M}_{\Lambda_{\oplus}}^{\text{cbv}}$, we see that we have to show:

1. For every $V \in \mathcal{V}$, and $M, N \in$ Terms such that $\emptyset \vdash \lambda x.M \, (\preceq^{\circ})^H \, \lambda x.N$, it holds that $\emptyset \vdash M\{V/x\} \, (\preceq^{\circ})^H \, N\{V/x\}$.

2. For all programs M, N such that $\emptyset \vdash M \, (\preceq^{\circ})^H \, N$, it holds that $[\![M]\!] \;\; (\overrightarrow{(\preceq^{\circ})^H}) \;\; [\![N]\!]$—where $(\overrightarrow{(\preceq^{\circ})^H})$ is the asymmetric lifting to sub-distributions, as defined in Definition 4.2.10 of Chapter 4 i.e.:$\forall A \subseteq \mathcal{V}, [\![M]\!](A) \leq [\![N]\!]((\preceq^{\circ})^H(A))$.

Proof of item 1. If $\emptyset \vdash \lambda x.M \, (\preceq^{\circ})^H \, \lambda x.N$, and that V is a value. Then looking at Howe's construct, we see that it means that there exists L with $x \vdash M \, (\preceq^{\circ})^H \, L$ and $\emptyset \vdash \lambda x.L \;\; \preceq^{\circ} \;\; \lambda x.N$. Since $(\preceq^{\circ})^H$ is reflexive and value substitutive, we obtain that $\emptyset \vdash M\{V/x\} \, (\preceq^{\circ})^H \, L\{V/x\}$, and since $\preceq$ is a simulation, we obtain that $\emptyset \vdash L\{V/x\} \, (\preceq^{\circ} \,)^H \, N\{V/x\}$. We can now conclude by the pseudo-transitivity property from Lemma 5.1.3, and we see that $\emptyset \vdash M\{V/x\} \, (\preceq^{\circ})^H \, N\{V/x\}$.

Proof of item 2. We first use the following property of the asymmetric lifting—that can be easily checked from its definition: for any binary relation on $S \times T$, if D is a directed set of sub-distributions over S, and $\mathscr{E}$ any sub-distribution over T, it holds that:

$$\forall \mathscr{D} \in D, \mathscr{D} \;\; \overrightarrow{(R)} \;\; \mathscr{E} \Rightarrow \sup D \;\; \overrightarrow{(R)} \;\; \mathscr{E}.$$

This remark allows us to use the approximations semantics instead of the exact semantics of M, so we we can now reformulate our goal as:

$$\emptyset \vdash M \, (\preceq^{\circ})^H \, N \qquad \Rightarrow \qquad \forall \mathscr{D} \text{ with } M \Downarrow \mathscr{D}, \quad \mathscr{D} \;\; (\overrightarrow{(\preceq^{\circ})^H}) \;\; [\![N]\!]. \tag{5.1}$$

It is easier to show (5.1) that directly the statement in Lemma 5.1.7, since now we can reason by induction on the proof derivation of $M \Downarrow \mathscr{D}$.

We will also use the fact that given a set of set of open terms X, let $\lambda x.X = \{\lambda x.M \mid M \in X\}$. Since all values are λ-abstractions, a set $A \subseteq \mathcal{V}$ may be represented as $\lambda x.X$, where X is a set of open terms that no other free variables that x. Now, by looking at the definition of $(\preceq^{\circ})^H$, we see that $(\preceq^{\circ})^H(A) = \preceq^{\circ} (\lambda x.(\preceq^{\circ})^H(X))$. To sum up:

$$(\preceq^{\circ})^H \cap \{(L, K) \mid L \in \mathcal{V}\} = \{(\lambda x.L, K) \mid \exists T \text{ with } L \, (\preceq^{\circ})^H \, T \wedge \lambda x.T \;\; \preceq^{\circ} \;\; K\}, \tag{5.2}$$

We now show (5.1) by induction on the proof derivation of $M \Downarrow \mathscr{D}$. We consider separately each possible rule which can be applied at the bottom of the derivation:

- If the rule used corresponds to the fact that, for every term, the empty distribution is an approximate semantics, the derivation is: $\dfrac{}{M \Downarrow \emptyset} \, bv$ then $\mathscr{D} = \emptyset$, and indeed $\emptyset \;\; (\overrightarrow{(\preceq^{\circ})^H}) \;\; [\![N]\!]$.
- If M is a value $V = \lambda x.L$ and the last rule of the derivation is $\dfrac{}{V \Downarrow \{V^1\}} \, b_v$ then $\mathscr{D} = \{V^1\}$ is the Dirac distribution for V and, by the definition of Howe's lifting, $\big(\emptyset \vdash \lambda x.L \, (\preceq^{\circ})^H \, N\big)$ was derived by the following rule:

$$\frac{x \vdash L (\preceq^{\circ})^H K \qquad \emptyset \vdash \lambda x.K \preceq^{\circ} N}{\emptyset \vdash \lambda x.L (\preceq^{\circ})^H N}$$

Recall that ($\emptyset \vdash \lambda x.K \preceq^\circ N$) and that $\preceq$ is a simulation on our LMC. Since $\preceq$ and $\preceq^\circ$ coincide on closed terms, it tells us that $\{\lambda x.K^1\}$ $(\overrightarrow{\preceq^\circ})$ $[\![N]\!]$. Our goal now is to show that also $\lambda x.K$ $((\overrightarrow{\preceq^\circ})^H)$ $[\![N]\!]$. Let $\lambda x.X \subseteq \mathcal{V}$. If $\lambda x.L \notin \lambda x.X$ then $\mathscr{D}(\lambda x.X) = 0$. Otherwise, $\mathscr{D}(\lambda x.X) = \mathscr{D}(\lambda x.L) = 1 = [\![N]\!](\preceq^\circ \{\lambda x.K\})$. It follows from $L(\preceq^\circ)^H K$ and from $\lambda x.L \in \lambda x.X$ that $\lambda x.K \in \lambda x.((\preceq^\circ)^H X)$; hence, $[\![N]\!](\preceq^\circ \{\lambda x.K\}) \leq [\![N]\!](\preceq^\circ \lambda x.((\preceq^\circ)^H X))$. So using (5.2), we can conclude that $\{V^1\}$ $((\overrightarrow{\preceq^\circ})^H)$ $[\![N]\!]$.

- If the derivation of $M \Downarrow \mathscr{D}$ is of the following form:

$$\frac{M_1 \Downarrow \mathscr{K} \qquad M_2 \Downarrow \mathscr{F} \qquad \{P\{x/V\} \Downarrow \mathscr{E}_{\mathscr{P},\mathscr{V}}\}_{\substack{\lambda x.P \in \mathsf{S}(\mathscr{K}),\\ V \in \mathsf{S}(\mathscr{F})}}}{M_1 M_2 \Downarrow \sum_{V \in \mathsf{S}(\mathscr{F})} \mathscr{F}(V) \left(\sum_{\lambda x.P \in \mathsf{S}(\mathscr{K})} \mathscr{K}(\lambda x.P).\mathscr{E}_{P,V} \right)}$$

Then $M = M_1 M_2$ and we have that the last rule used in the derivation of $\emptyset \vdash M\, (\preceq^\circ)^H N$ is:

$$\frac{\emptyset \vdash M_1 (\preceq^\circ)^H M_1' \qquad \emptyset \vdash M_2 (\preceq^\circ)^H M_2' \qquad \emptyset \vdash M_1' M_2' \preceq^\circ N}{\emptyset \vdash M_1 M_2 (\preceq^\circ)^H N}$$

We are first going to apply the induction hypothesis to the derivation of $M_1 \Downarrow \mathscr{K}$ and $M_2 \Downarrow \mathscr{F}$: by using the Disentangling Lemma—that we recalled in Lemma 5.1.6—we see that there are two couplings π_1, π_2 that respect $((\overrightarrow{\preceq^\circ})^H)$ such that $\pi_1 \blacktriangleleft \langle \mathscr{K} \& [\![M_1']\!] \rangle$ and $\pi_2 \blacktriangleleft \langle \mathscr{F} \& [\![M_2']\!] \rangle$.

Recall that our goal is to show that $\mathscr{D}$ $((\overrightarrow{\preceq^\circ})^H)$ $[\![N]\!]$. First, we use these couplings to express the sub-distribution $\mathscr{D}$:

$$\begin{aligned}
\mathscr{D} &= \sum_V \mathscr{F}(V) \sum_{\lambda x.P} \mathscr{K}(\lambda x.P).\mathscr{E}_{P,V} \\
&= \sum_{V,P} \left(\sum_W \pi_1(V,W) \right) \left(\sum_Q \pi_2(\lambda x.P, \lambda x.Q) \right) \mathscr{E}_{P,V} \\
&= \sum_{W,Q} \sum_{P,V} \pi_1(V,W) \cdot \pi_2(\lambda x.P, \lambda x.Q) \cdot \mathscr{E}_{P,V}
\end{aligned}$$

We fix now W and Q, and we consider V, P such that $\pi_1(V,W) > 0$ and $\pi_2(\lambda x.P, \lambda x.Q) > 0$. Observe that since π_1 and π_2 respect $(\preceq^\circ)^H$, it means that $\emptyset \vdash V\, (\preceq^\circ)^H W$ and $\emptyset \vdash \lambda x.P\, (\preceq^\circ)^H \lambda x.Q$. It implies—by construction of Howe's lifting—that there exists S with $x \vdash P\, (\preceq^\circ)^H S$ and $\emptyset \vdash \lambda x.S \preceq^\circ \lambda x.Q$.

- We first utilize $x \vdash P\, (\preceq^\circ)^H S$. Since $(\preceq^\circ)^H$ is value-substitutive, it holds that $\emptyset \vdash P\{V/x\}\, (\preceq^\circ)^H S\{W/x\}$; we may apply the induction hypothesis, and we obtain that
$$\mathscr{E}_{P,V} \ ((\overrightarrow{\preceq^\circ})^H) \ [\![S\{W/x\}]\!]. \tag{5.3}$$
- We now utilize the fact that $\emptyset \vdash \lambda x.S \preceq^\circ \lambda x.Q$. Since $\preceq$ is a simulation—and $\preceq$ and $\preceq^\circ$ coincide on closed terms—and that in the LMC $\mathcal{L}^{\mathrm{cbv}}_{\Lambda_\oplus}$, for $U \in \{S, Q\}$, $\widehat{\lambda x.U} \xrightarrow{W} \{U\{W/x\}^1\}$, it holds that $\emptyset \vdash S\{W/x\} \preceq^\circ Q\{W/x\}$. By doing again one step forward in this LMC, we see that for $U \in \{S, Q\}$, $U\{W/x\} \xrightarrow{\mathrm{eval}} [\![\widehat{U\{W/x\}}]\!]$. Since $\preceq$ is a simulation, it tells us that:
$$[\![S\{W/x\}]\!] \ (\overrightarrow{\preceq^\circ}) \ [\![Q\{W/x\}]\!]. \tag{5.4}$$

Using Remark 4.2.1 of Chapter 4 on composition for the relational lifting, we see that we can combine (5.3) and (5.4) to obtain $\mathscr{E}_{P,V}\ (\preceq^\circ \circ \overrightarrow{(\preceq^\circ)^H})\ [\![Q\{W/x\}]\!]$. Recall that we know also from Lemma 5.1.3 that $\preceq^\circ \circ (\preceq^\circ)^H \subseteq (\preceq^\circ)^H$. Moreover, we know also—from Remark 4.2.1 of Chapter 4 —that whenever $R \subseteq S$, then $\overrightarrow{(R)} \subseteq \overrightarrow{(S)}$. As a consequence, we deduce that $\mathscr{E}_{P,V}\ (\overrightarrow{(\preceq^\circ)^H})\ [\![Q\{W/x\}]\!]$.

Summing up, and using that $\overrightarrow{(\preceq^\circ)^H})$ is stable by linear combinations —from Remark 4.2.1 of Chapter 4 —we obtain that:

$$\mathscr{D}\ (\overrightarrow{(\preceq^\circ)^H})\ \sum_{W,Q}\sum_{P,V} \pi_1(V,W)\cdot\pi_2(\lambda x.P,\lambda x.Q)\cdot[\![Q\{W/x\}]\!]. \tag{5.5}$$

Now, we use the fact that $\pi_1 \blacktriangleleft \langle \mathscr{K}\&[\![M_1']\!]\rangle$ and $\pi_2 \blacktriangleleft \langle \mathscr{F}\&[\![M_2']\!]\rangle$ tell us: $\sum_P \pi_1(\lambda x.P,\lambda x.Q) \leq [\![M_1']\!](\lambda x.Q)$, and similarly $\sum_V \pi_2(V,W) \leq [\![M_2']\!](W)$. From there, we can use the fact that $\overrightarrow{(R)}$ is *right-stable* under pointwise order—see once again Remark 4.2.1—we can deduce from (5.5) that:

$$\mathscr{D}\ (\overrightarrow{(\preceq^\circ)^H})\ \sum_{W,Q}[\![M_1']\!](\lambda x.Q)\cdot[\![M_2']\!](W)\cdot[\![Q\{W/x\}]\!]. \tag{5.6}$$

Observe now that the right part in (5.6) is exactly $[\![M_1'M_2']\!]$. As a consequence (5.6) implies $\mathscr{D}\ (\overrightarrow{(\preceq^\circ)^H})\ [\![M_1'M_2']\!]$.

We are now almost ready to conclude: indeed we know by hypothesis that $\emptyset \vdash M_1'M_2' \preceq^\circ N$, and that $\preceq^\circ$ is a bisimulation, it holds that also $[\![M_1'M_2']\!]\ (\overrightarrow{\preceq^\circ})\ [\![N]\!]$, and we can conclude by using again that $\preceq^\circ \circ (\preceq^\circ)^H \subseteq (\preceq^\circ)^H$ and the composition Lemma for relational lifting that $\mathscr{D}\ (\overrightarrow{(\preceq^\circ)^H})\ [\![N]\!]$.

- If $M \Downarrow \mathscr{D}$ is derived by:

$$\frac{M_1 \Downarrow \mathscr{D}_1 \qquad M_2 \Downarrow \mathscr{D}_2}{\Downarrow M_1 \oplus M_2\, \frac{1}{2}\mathscr{D}_1 + \frac{1}{2}\mathscr{D}_2}$$

then $\emptyset \vdash M\ (\preceq^\circ)^H\ N$ is derived by:

$$\frac{\emptyset \vdash M_1\ (\preceq^\circ)^H\ N_1 \qquad \emptyset \vdash M_2\ (\preceq^\circ)^H\ N_2 \qquad \emptyset \vdash N_1 \oplus N_2\ \preceq^\circ\ N}{\emptyset \vdash M_1 \oplus M_2\ (\preceq^\circ)^H\ N}$$

By the inductive hypothesis, for $i \in \{1,2\}$ we have that for any $\lambda x.X \subseteq \mathcal{V}$,

$$\mathscr{D}_i(\lambda x.X) \leq [\![N_i]\!](\preceq^\circ\ \lambda x.((\preceq^\circ)^H X))$$

Hence, the result follows from:

$$\begin{aligned}&\frac{1}{2}\cdot\mathscr{D}_1(\lambda x.X) + \frac{1}{2}\cdot\mathscr{D}_2(\lambda x.X)\\ &\leq \frac{1}{2}\cdot[\![N_1]\!](\preceq^\circ\ \lambda x.((\preceq^\circ)^H X)) + \frac{1}{2}\cdot[\![N_2]\!](\preceq^\circ\ \lambda x.((\preceq^\circ)^H X))\end{aligned}$$

□

We have now all ingredients necessary to state compatibility of $\preceq^\circ$.

Theorem 5.1.8 *$\preceq^\circ$ is compatible.*

Proof. Recall that we know already that $(\preceq^\circ)^H$ is compatible, and moreover $\preceq^\circ \subseteq (\preceq^\circ)^H$: as a consequence it is enough to show that the Howe's lifting of $\preceq^\circ$ is conservative. Using the key Lemma 5.1.7, we see that $(\preceq^\circ)^H_{|\mathcal{S}^{\mathrm{cbn}}_{\Lambda_\oplus}}$ is a simulation. Since $\preceq$ is the greatest simulation, it holds that $(\preceq^\circ)^H_{|\mathbf{P}} \subseteq \preceq$. We can conclude by applying Lemma 5.1.5: it tells us that $(\preceq^\circ)^H \subseteq \preceq^\circ$. □

We may now apply Proposition 5.1.2 since $\preceq$ is a *observationally correct preorder*, and $\preceq^\circ$ a compatible open relation, and we obtain that $\preceq$ is sound with respect for the context preorder, and by symmetry we also obtain soundness of $\equiv$.

Corollary 5.1.1 *$\preceq^{cbv}_{\Lambda_\oplus}$ is sound with respect to context preorder, and $\equiv^{cbv}_{\Lambda_\oplus}$ is sound with respect to context equivalence.*

5.2 Full Abstraction for Probabilistic Applicative Bisimulation.

In the setting of CBN $\Lambda_\oplus$, Dal Lago, Sangiorgi and Alberti [74] showed that neither probabilistic applicative bisimilarity nor probabilistic applicative similarity are complete—i.e bisimilarity does not coincide with context equivalence, and similarity does not coincide with context preorder. In this section, we show that by contrast, for CBV $\Lambda_\oplus$, probabilistic applicative bisimilarity and context equivalence actually coincide, while probabilistic applicative similarity do not coincide with the context preorder. It highlights a previously unobserved gap both between the CBN and the CBV semantics, and between the symmetric and the asymmetric case. At the time where we published it—in a joint work with Ugo Dal Lago [24]—, it was an unexpected result, because these phenomenon appear neither in the deterministic case—where both applicative similarity and applicative bisimilarity are fully abstract, irrespective of whether the reduction strategy is call-by-name or call-by-value—nor in Λ enriched with a non-deterministic operator—where neither applicative similarity nor applicative bisimilarity are complete, irrespective of whether the reduction strategy is call-by-name or call-by-value, and of whether we take a may-converge or must-converge flavor [77].

To show full-abstraction of probabilistic applicative bisimilarity, we are going to use the characterization of LMC bisimilarity established by Breugel et. al., that is based on a notion of *testing*: two states are bisimilar if they have the same *success probability* under some family of tests. Our proof method consists in showing that when we consider the LMC $\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}$, every relevant test α may be emulated by a $\Lambda_\oplus$-context $\mathcal{C}$—i.e. for any program M, the termination probability of $\mathcal{C}[M]$ is the same as the probability that the test α succeeds when executed on M.

5.2.1 Labeled Markov Processes (LMP)

Breugel et al actually work with more general objects than LMCs: they consider *Labeled Markov Processes*, that are a generalization of LMCs designed to deal with *continuous* probabilities. The concept of probabilistic bisimulation has been generalized to LMPs by Desharnais et al, more than ten years ago [39, 41]. Later, Breugel et al showed that similarity and bisimilarity as defined in the aforementioned paper exactly correspond to appropriate, and relatively simple, notions of *testing* [118]. We are going to summarize here those of their results that are relevant for our purposes. To this end, we will need the elementary notions of measure theory that have been presented in Chapter 1.

The first step consists in giving a the definition of Labelled Markov Processes: they are a generalization of LMCs in which the set of states is not restricted to be countable:

Definition 5.2.1 *A* Labelled Markov Process *(LMP in the following) is a triple* $\mathscr{M}^{process} = (\Omega, Act, \mu)$*, consisting of a measurable set of states* $\Omega = (\mathcal{S}, \Sigma)$*—i.e.* $\mathcal{S}$ *is a set, and* Σ *a* σ*-field over* $\mathcal{S}$*—, a* countable *set of labels* Act*, and a transition probability function* $\mu : \mathcal{S} \times Act \times \Sigma \to [0,1]$*, which has to be a* Markov Kernel *on* (Ω, Act)*, i.e. such that:*

- *for all* $x \in \mathcal{S}$*, and* $a \in Act$*, the naturally defined function* $\mu_{x,a}(\cdot) : \Sigma \to [0,1]$ *is a sub-probability measure on* Ω*;*
- *for all* $a \in Act$*, and* $A \in \Sigma$*, the naturally defined function* $\mu_{(\cdot),a}(A) : \mathcal{S} \to [0,1]$ *is measurable.*

We note $Ker(\Omega, Act)$ *the set of all Markov Kernels on* (Ω, Act)*.*

The notion of (bi)simulation can be smoothly generalized to the continuous case. We give here the definition following [41]. The one given in [118] is slightly different, but as noted there it leads to the same similarity and bisimilarity notions.

Definition 5.2.2 (From [118]) *Let* (Ω, Act, μ) *be a LMP, with* $\Omega = (\mathcal{S}, \Sigma)$ *and let* R *be a relation on* $\mathcal{S}$*.*

- R *is a simulation, if it is a preorder and if whenever* $s\,R\,t$*, then for every* $a \in Act$ *and* A *measurable* R*-closed subset of* Ω*, it holds that* $\mu_{s,a}(A) \leq \mu_{t,a}(A)$*.*
- R *is a bisimulation if it is an equivalence relation, and if whenever* $s\,R\,t$*, then for every* $a \in Act$ *and* A *measurable and* R*-closed subset of* Ω*, it holds that* $\mu_{s,a}(A) = \mu_{t,a}(A)$*.*

For a LMP $\mathscr{M}^{\text{process}}$, both the greatest simulation and the greatest bisimulation exist [41]: we call them respectively similarity—that we denote $\preceq_{\mathscr{M}^{\text{process}}}$—and bisimilarity—denoted $\equiv_{\mathscr{M}^{\text{process}}}$. Labelled Markov Chains can be seen as a particular kind of LMPs—the ones on *discrete* measurable spaces, where the states space $\mathcal{S}$ is countable, and the σ field Σ is taken as $\text{Parts}(\mathcal{S})$ —as highlighted in Example 1.4.2 from Chapter 1 it is indeed a σ-field over $\mathcal{S}$. We express in Lemma 5.2.1 below that Markov Kernels on those kind of *discrete* LMPs are in one-to-one correspondence with probability matrices, meaning that discrete LMPs are exactly LMCs.

Lemma 5.2.1 *Let* $\mathcal{S}$ *be a countable set, and* Act *a countable set of labels. Let* $\Omega_{\mathcal{S}}$ *be the measurable set* $(\mathcal{S}, Parts(\mathcal{S}))$*. Then we can build a bijection* ϕ *from* $Ker(\Omega_s, Act)$ *into the sets of all probability matrices over* $\mathcal{S}$*, by taking:*

$$\phi : \mu \in Ker(\Omega_s, Act) \mapsto \mathcal{P}_\mu \text{ with } \mathcal{P}_\mu(s,a,t) = \mu_{s,a}(\{t\}).$$

Proof. First, we have to show that for every Markov Kernel $\mu \in \text{Ker}(\Omega_{\mathcal{S}}, Act)$, $\phi(\mu)$ is indeed a probability matrix i.e that for every state $s \in \mathcal{S}$ and for every label $l \in \mathcal{L}$, $\sum_{t \in \mathcal{S}} \phi(\mu)(s,l,t) \leq 1$. It is a direct consequence of the fact that $\mu_{s,a}(\cdot)$ is a sub-probability measure on $\Omega_{\mathcal{S}}$: indeed $1 \geq \mu_{s,a}(\mathcal{S}) = \sum_{s \in \mathcal{S}} \mu_{s,a}(\{t\})$ since $\mathcal{S}$ is countable.

Then, we define a function ψ from the set of probability matrices over $\mathcal{S}$ into the set of Markov kernels over $(\Omega_{\mathcal{S}})$, and we show that ψ is the inverse of ϕ. Whenever $\mathcal{P}$ is a probability matrix, we define $\psi(\mathcal{P}) : \mathcal{S} \times Act \times \text{Parts}(\mathcal{S}) \to [0,1]$ as $\psi(\mathcal{P})(s, Act, S) = \sum_{x \in \mathcal{S}} \mathscr{M}(s,a,x)$. We see that $\psi(\mathcal{P})$ is indeed a sub-distribution over $\Omega_{\mathcal{S}}$. Moreover since $\mathcal{S}$ is countable, any function $\mathcal{S} \to [0,1]$ is a measurable function: it means that $\psi(\mathcal{P})$ is indeed a Markov kernel. Moreover, we see easily that both $\phi \circ \psi$ and $\psi \circ \phi$ coincide with the identity, that concludes the proof.

□

In the following, if $\mathscr{M} = (\mathcal{S}, Act, \mathcal{P})$ is a LMC, we will use $\mathscr{M}^\sigma$ to denote the LMP $((\mathcal{S}, \text{Parts}(\mathcal{S})), Act, \phi^{-1}(\mathcal{P}))$. We show now that bisimilarity and similarity are preserved by ϕ.

Lemma 5.2.2 *Let $\mathscr{M} = (\mathcal{S}, Act, \mathcal{P})$ be a LMC, and s, t two states in $\mathcal{S}$. Then it holds that:*

1. *$s \preceq_{\mathscr{M}} t$, if and only if $s \preceq_{\mathscr{M}^\sigma} t$;*
2. *$s \equiv_{\mathscr{M}} t$, if and only if $s \equiv_{\mathscr{M}^\sigma} t$;*

Proof. We first show item 1. We are going to show that any pre-order R on $\mathcal{S}$ is a simulation for the LMC $\mathscr{M}$ if and only if it is a simulation for the LMP $\mathscr{M}^\sigma$.

- Let R be a preorder on $\mathcal{S}$ which is a simulation for the LMC $\mathscr{M}$. Let $s, t \in \mathcal{S}$ be such that $s\,R\,t$. Let $a \in Act$ and A be a measurable R-closed subset of $\Omega_{\mathcal{S}}$. Since R is a simulation on the LMC $\mathscr{M}$, it holds that $\mathcal{P}(s, a, A) \leq \mathcal{P}(t, a, R(A)))$. Since A is R-closed, $\mathcal{P}(t, a, R(A)) = \mathcal{P}(t, a, A)$, hence $\mathcal{P}(s, a, A) \leq \mathcal{P}(t, a, (A))$. We can rewrite this inequality using the bijection ϕ as $\mu_{\phi^{-1}(\mathscr{M})}(s, a, A) \leq \mu_{\phi^{-1}(\mathscr{M})}(t, a, A)$. As a consequence, R is a simulation on the LMP $\phi^{-1}(\mathscr{M}) = \mathscr{M}^\sigma$.
- Let R be a preorder on $\mathcal{S}$ which is a simulation for the LMP $\mathscr{M}^\sigma$. Let $s, t \in \mathcal{S}$ be such that $s\,R\,t$. Let $a \in Act$ and $X \subseteq \mathcal{S}$. First, observe that $R(A)$ is a measurable set with respect to the σ-field $\Omega_{\mathcal{S}}$—since in the discrete σ-algebra, all subsets are measurable. Since R is a preorder, it holds that $R(R(X)) = R(X)$, hence $R(A)$ is a R-closed measurable subset of $\Omega_{\mathcal{S}}$. As a consequence, we can use the fact that R is a simulation for the LMP $\mathscr{M}^\sigma$, and we obtain that: $\mu_{\mathscr{M}^\sigma}(s, a, R(A)) \leq \mu_{\mathscr{M}^\sigma}(t, a, R(A))$, which we can rewrite by looking at the way we define $\mathscr{M}^\sigma$ from $\mathscr{M}$ as: $\mathcal{P}(s, a, R(A)) \leq \mathcal{P}(t, a, R(A))$. Since moreover $A \subseteq R(A)$—since R is reflexive—we can deduce that also $\mathcal{P}(s, a, A) \leq \mathcal{P}(s, a, R(A))$. By combining these two inequalities, we deduce that $\mathcal{P}(s, a, A) \leq \mathcal{P}(t, a, R(A))$ and from there we obtain that R is a bisimulation for the LMC $\mathscr{M}$.

Since we know that both the greatest simulation for the LMC $\mathscr{M}$ and the greatest simulation for the LMP $\mathscr{M}^\sigma$ are pre-order, we can conclude that they coincide. The proof of item 2 is similar. □

It means that we are able, using the function ϕ defined in Lemma 5.2.1, to transfer all results that hold for bisimilarity or similarity on LMPs to the analogous notions of LMCs. It is in particular the case of Breugel et. al.'s testing characterization for LMP bisimilarity, that we present in the next paragraph.

5.2.2 Bisimilarity and Similarity are Characterized by Tests

In 1980, Milner and Hennessy [62, 61] introduced the so-called *Hennessy-Milner* modal logic (HML), and showed it fully characterize bisimilarity on LTSs that are *image-finite*—i.e. such that, from any state s, and for every action a there is a finite number of state t such that $s \xrightarrow{a} t$—, i.e. that two states s and t are bisimilar if and only if they satisfy the same formulas. Using this logical characterization, Abramsky developed in [1] a *testing characterization* for those LTSs that are both *image-finite* and *finitely branching* —for every s, there is a finite number of actions a that may be done. Intuitively, a test specifies an interaction between an *experimenter* and a *process*, and this interaction may succeed or

fail. Since LTSs are *non-deterministic*, several runs of a test on a given process may lead to different outcomes; for this reason, the behavioral semantics of a test α is a function $[\![\alpha]\!] : \mathcal{S} \to \text{Parts}(\{\bot, \top\})$. Abramsky's showed that his testing language characterizes bisimilarity, in the sense that two states s, t are bisimilar if and only if, for every test α, $[\![\alpha]\!](s) = [\![\alpha]\!](t)$.

When introducing bisimilarity for LMCs [75], Larsen and Skou also introduced a test language designed to characterize it. As in the LTS case, different runs of a test may lead to different outcomes; however since we are now in a probabilistic setting, we are not interested anymore into the set of all possible outcomes, but in the probability of obtaining each outcome. It means that the semantics of a test is now a function $[\![\alpha]\!] : \mathcal{S} \to \Delta^{=}(\{\bot, \top\})$. Observe that since we consider *proper* distributions, the probability of obtaining $\bot$ can actually be computed from the probability of obtaining $\top$, and so we see that it is enough to specify for each state s the *success probability of* α *on* s. Larsen and Skou showed that when a LMC verifies the so-called *minimal deviation assumption* [75]—intuitively, it requires that whenever two transition probabilities are different, they must not be arbitrarily close—two states s and t are bisimilar if and only if for each test α expressible in the test language, the success probability of α is the same irrespective of whether it is applied to s or t. Unfortunately, the minimal deviation assumption is not verified by the LMC $\mathcal{L}^{\text{cbv}}_{\Lambda_\oplus}$ we are interested in. Breugel et al extended in [118] Larsen Skou's testing characterization to the more general class of LMPs, which encompass *all* LMCs, and so allows us to get rid of the minimal deviation requirement. They actually consider *two* test languages: a language $\mathcal{T}_0$ similar to the one of Larsen Skou, that characterizes bisimilarity, and a larger one $\mathcal{T}_1$ that characterizes similarity: two states s, t of a LMP are bisimilar if and only if they have the same probability of passing successfully each test of $\mathcal{T}_0$, while they are connected by the similarity relation if for every test in $\mathcal{T}_1$, t has a greater or equal probability of passing it successfully than s. We present here the test language $\mathcal{T}_1$, $\mathcal{T}_0$ and how to compute the success probability of tests when executed on some LMP state.

We first define tests as purely syntactic objects, generated by a BNF grammar, and then we construct a family of measurable functions $\mathcal{S} \to [0, 1]$, that specify the success probability of a given test starting from any state in $\mathcal{S}$.

Definition 5.2.3 *Let $\mathscr{M}^{process} = (\mathcal{S}, Act, \mu)$ be a LMP. We define the* test language for similarity $\mathcal{T}_1(\mathscr{M}^{process})$*, and the* test language for bisimilarity $\mathcal{T}_0(\mathscr{M}^{process})$ *respectively as:*

$$\alpha \in \mathcal{T}_1(\mathscr{M}^{process}) ::= \omega \;\mid\; a \cdot \alpha \;\mid\; \langle \alpha, \alpha \rangle \;\mid\; \alpha \vee \alpha;$$

$$\alpha \in \mathcal{T}_0(\mathscr{M}^{process}) ::= \omega \;\mid\; a \cdot \alpha \;\mid\; \langle \alpha, \alpha \rangle \qquad \textit{where } a \in Act.$$

Please observe that tests are *finite* objects. Each test in $\mathcal{T}_1$ specifies an interaction between an experimenter and a system with *buttons*: for each action in a, the experimenter may push the corresponding button, and then the system may fail or accept to carry on the interaction. We describe now for each test which experiment it specifies. ω is the test that always succeeds. The test $a \cdot \alpha$ represents the experimenter pushing the button a to ask the system to perform this action, and in case of success proceeds with performing the test α. The test $\langle \alpha, \beta \rangle$ is a *conjunctive test*, that makes two copies of the current underlying state of the system, tests them independently according to α and β and succeeds iff *both* tests succeed. The test $\alpha \vee \beta$ is a *disjunctive test*, that also makes two copies of the underlying state and tests them independently according to α and β, but succeeds as soon as*at least one of* these experiments succeed. We now give a *behavioral semantics* to tests, in a way consistent with the interpretation of tests as experiments described above. It consists in associating a success probability to every tests, that depends on the LMP describing the

underlying system being tested, and also on which state of this LMP we take as initial state when starting the test run. It is formalized accordingly in Definition 5.2.4 below.

Definition 5.2.4 (From [118]) *Given a labeled Markov Process $\mathscr{M}^{process} = (\mathcal{S}, \Sigma, \mu)$, we define the* behavioral semantics for tests in $\mathcal{T}_1(\mathscr{M}^{\text{process}})$*, as the indexed family $\{Pr^{\mathscr{M}^{process}}(\cdot, \alpha)\}_{\alpha \in \mathcal{T}_1(\mathscr{M}^{process})}$ (such that $Pr^{\mathscr{M}^{process}}(\cdot, \alpha) : \mathcal{S} \to \mathbb{R}$ is a measurable function) by induction on the structure of α:*

$$
\begin{aligned}
Pr^{\mathscr{M}^{process}}(s, \omega) &= 1;\\
Pr^{\mathscr{M}^{process}}(s, a \cdot \alpha) &= \int Pr^{\mathscr{M}^{process}}(\cdot, \alpha) d\mu_{s,a};\\
Pr^{\mathscr{M}^{process}}(s, \langle \alpha, \beta \rangle) &= Pr^{\mathscr{M}^{process}}(s, \alpha) \cdot Pr^{\mathscr{M}^{process}}(s, \beta);\\
Pr^{\mathscr{M}^{process}}(s, \alpha \vee \beta) &= Pr^{\mathscr{M}^{process}}(s, \alpha) + Pr^{\mathscr{M}^{process}}(s, \beta) - Pr^{\mathscr{M}^{process}}(s, \alpha) \cdot Pr^{\mathscr{M}^{process}}(s, \beta)
\end{aligned}
$$

Observe that both $\mathcal{T}_1$ and $\mathcal{T}_0$ encompass *traces*—i.e. the linear tests presented in Section 4.2.1 from Chapter 4—but also offer more possibilities to the experimenter who, *at each step of the experiment*, may also decide to *copy* the process he interacts with.

We consider now the particular case where LMPs are *discrete*, i.e of the form $\mathscr{M}^{\sigma}$ where $\mathscr{M}$ is a LMC, we denote $\Pr^{\mathscr{M}}(s, \alpha)$ to mean $\Pr^{\mathscr{M}^{\sigma}}(s, \alpha)$. Looking at the way success probability for tests on $\mathscr{M}^{\sigma}$ are defined, we see that we can give a more direct formulation of $\Pr^{\mathscr{M}}(s, \alpha)$ as expressed in Proposition 5.2.3 below, that takes advantage of the discrete setting to replace Lebesgue integration with an infinite sum.

Proposition 5.2.3 *Let $\mathscr{M} = (\mathcal{S}, \mathcal{L}, \mathcal{P})$ be a LMC.*

$$
\begin{aligned}
Pr^{\mathscr{M}}(s, \omega) &= 1;\\
Pr^{\mathscr{M}}(s, a \cdot \alpha) &= \sum_{t \in \mathcal{S}} Pr^{\mathscr{M}}(t, \alpha) \cdot \mathcal{P}(s, a, t);\\
Pr^{\mathscr{M}}(s, \langle \alpha, \beta \rangle) &= Pr^{\mathscr{M}}(s, \alpha) \cdot Pr^{\mathscr{M}}(s, \beta);\\
Pr^{\mathscr{M}}(s, \alpha \vee \beta) &= Pr^{\mathscr{M}}(s, \alpha) + Pr^{\mathscr{M}}(s, \beta) - Pr^{\mathscr{M}}(s, \alpha) \cdot Pr^{\mathscr{M}}(s, \beta)
\end{aligned}
$$

Proof. It is a direct consequence of the characterization of Lebesgue integrals for discrete spaces given in Chapter 1. □

Example 5.2.1 *We give here an example of how operates a test on the LMC $\mathcal{L}^{cbv}_{\Lambda_\oplus}$. We consider the test $\alpha = eval \cdot V \cdot \langle eval \cdot \omega, eval \cdot \omega \rangle \in \mathcal{T}_0(\mathcal{M}^{cbv}_{\Lambda_\oplus})$, where V is any value in $\mathcal{V}$. The interaction specified by this test is as follows: first the experimenter evaluates the program, and passes to it the value V as an argument, then he takes* two *copies of the resulting program, and evaluates each of them independently; the test succeeds if both evaluations terminate.*

We look now at what happens when we apply α to the programs $M = \lambda y.(\Omega \oplus I)$, and $N = (\lambda y.\Omega) \oplus (\lambda y.I)$. We represent in Figure 5.2 the main steps needed to compute $Pr^{\mathcal{L}^{cbv}_{\Lambda_\oplus}}(M, \alpha)$ and $Pr^{\mathcal{L}^{cbv}_{\Lambda_\oplus}}(N, \alpha)$. Due to the inductive nature of the definition of tests success probability, this computation has to be done starting from the states represented to the bottom of Figure 5.2, towards the states represented at the top: we start from the fact that $Pr(u, \omega) = 1$, and from there, we build step by step the success probability of α on M and N. At the end, we obtain that $Pr^{\mathcal{L}^{cbv}_{\Lambda_\oplus}}(M, \alpha) = \frac{1}{4}$, while $Pr^{\mathcal{L}^{cbv}_{\Lambda_\oplus}}(N, \alpha) = \frac{1}{2}$.

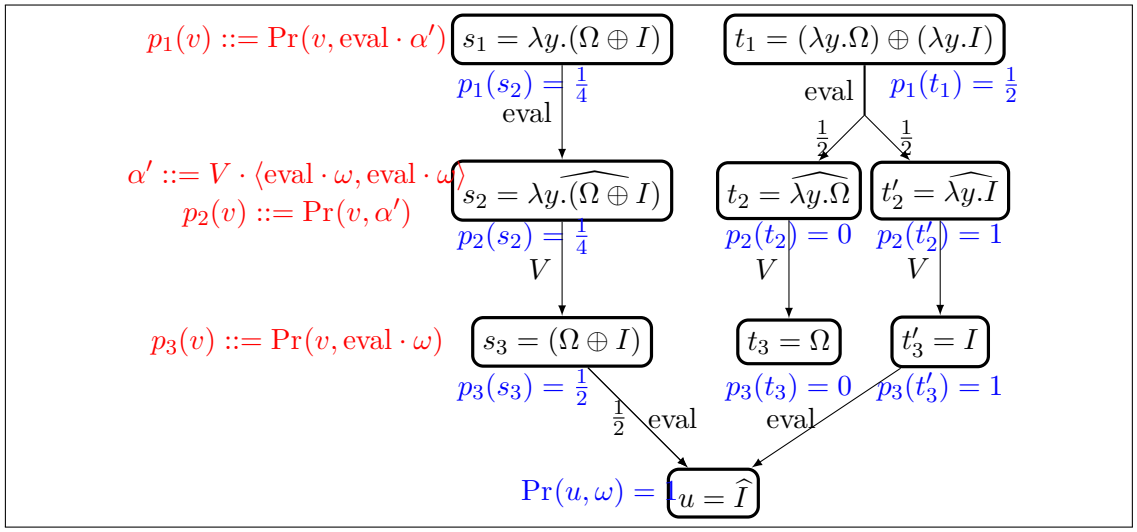


Figure 5.2: Examples of Tests's Success Probability in the LMC $\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}$.

We state now the *characterization of bisimilarity and similarity by testing*, as shown by Breugel et al. for LMPs: it is possible to decide which pair of states are similar and bisimilar by looking at the probability success of tests when applied to these states.

Theorem 5.2.4 ([118]) *Let $\mathscr{M}^{process} = (\mathcal{S}, \Sigma, \mu)$ be a LMP, and $s, t \in \mathcal{S}$. Then:*

- $s \equiv_{\mathscr{M}^{process}} t$ *iff* $Pr^{\mathscr{M}^{process}}(s, \alpha) = Pr^{\mathscr{M}^{process}}(t, \alpha)$ *for every test* $\alpha \in \mathcal{T}_0(\mathscr{M}^{process})$.
- $s \preceq_{\mathscr{M}^{process}} t$ *iff* $Pr^{\mathscr{M}^{process}}(s, \alpha) \leq Pr^{\mathscr{M}^{process}}(t, \alpha)$ *for every test* $\alpha \in \mathcal{T}_1(\mathscr{M}^{process})$.

Recall that bisimilarity on $\mathscr{M}$ and $\mathscr{M}^\sigma$ coincide, and it is also true for similarity. It means that for characterizing bisimilarity or similarity on $\mathscr{M}$, Theorem 5.2.4 tells us it is enough to consider the success probability of tests on $\mathscr{M}^\sigma$.

5.2.3 Every Test in $\mathcal{T}_0$ has an Equivalent Context

Our objective now is to apply the testing characterization given in Theorem 5.2.4 to the particular case where we consider the LMC $\mathcal{L}^{\text{cbv}}_{\Lambda_\oplus}$—introduced in Section 5.1.1 to define probabilistic applicative bisimilarity on CBV $\Lambda_\oplus$—in order to show completeness of $\equiv^{\text{cbv}}_{\Lambda_\oplus}$ with respect to context equivalence for CBV $\Lambda_\oplus$. Theorem 5.2.4 tells us indeed that proving $(\equiv^{\text{cbv}}_{\Lambda_\oplus}) \supseteq (\equiv^{\text{ctx}})$ boils down to show that whenever M and N have exactly the same convergence probability under any contexts, they have also exactly the same success probabilities for all tests. Or, more precisely, that for a given test α, there exists a context $\mathcal{C}$, such that for all programs M, the success probability of α on M is *exactly* the convergence probability of $\mathcal{C}[M]$:

$$\Pr^{\mathcal{L}^{\text{cbv}}_{\Lambda_\oplus}}(M, \alpha) = |[\![\mathcal{C}[M]]\!]|.$$

When we are able to build such context $\mathcal{C}$, we say that $\mathcal{C}$ *emulates* the test α. We have to be careful here, since we have in fact two different kinds of states in $\mathcal{S}^{\text{cbv}}_{\Lambda_\oplus}$: programs and distinguished values. In consequence, we define two encodings $\Theta^{\mathbf{P}}, \Theta^{\mathcal{V}} : \mathcal{T}_0 \to \mathbf{C}^{\Lambda_\oplus}$ that build contexts emulating the action of tests when they are executed respectively on programs and distinguished values. It means that for each test α, its success probability starting from the state M is the same as the termination probability of the context

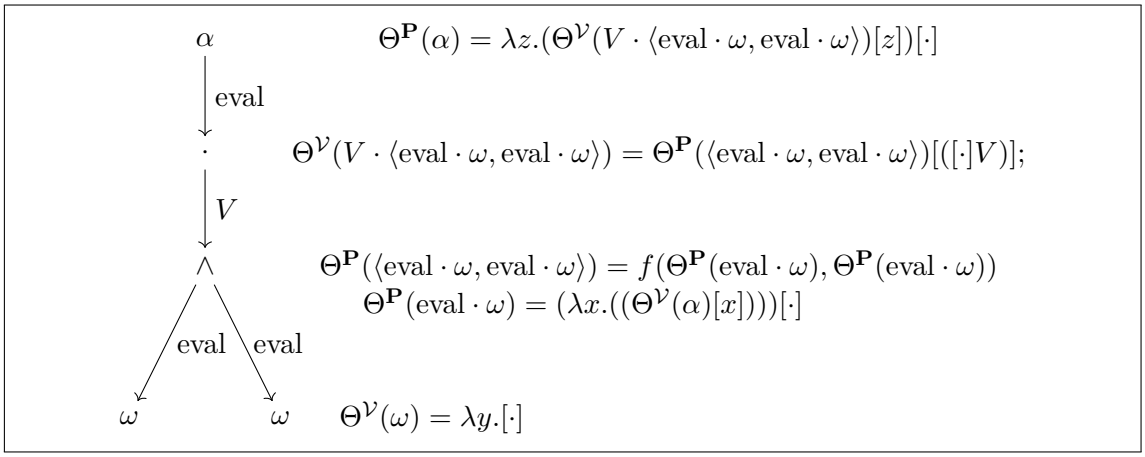


Figure 5.3: Context $\Theta^{\mathbf{P}}(\alpha)$ emulating the test $\alpha = \text{eval} \cdot V \cdot \langle \text{eval} \cdot \omega, \text{eval} \cdot \omega \rangle$.

$\Theta^{\mathbf{P}}(\alpha)[M]$, and similarly, its success probability starting from the state $\hat{V}$ is the same as the termination probability of $\Theta^{\mathcal{V}}(\alpha)[V]$.

Definition 5.2.5 *We define $\Theta^{\mathcal{V}}, \Theta^{\boldsymbol{P}} : \mathcal{T}_0(\mathcal{M}^{cbv}_{\Lambda_\oplus}) \to \mathbf{C}^{\Lambda_\oplus}$ by mutual induction on the structure of $\alpha \in \mathcal{T}_0(\mathcal{M}^{cbv}_{\Lambda_\oplus})$, as specified below:*

1. *Non-relevant actions: $\Theta^{\boldsymbol{P}}(V \cdot \alpha) = \Omega[\cdot]$ and $\Theta^{\mathcal{V}}(eval \cdot \alpha) = \Omega[\cdot]$;*
2. *Ever-Successful Test: $\Theta^{\boldsymbol{P}}(\omega) = \lambda x.[\cdot]$; $\Theta^{\mathcal{V}}(\omega) = \lambda x.[\cdot]$;*
3. *Action V: $\Theta^{\mathcal{V}}(V \cdot \alpha) = \Theta^{\boldsymbol{P}}(\alpha)[([\cdot]V)]$;*
4. *Action eval: $\Theta^{\boldsymbol{P}}(eval \cdot \alpha) = \lambda x.(\Theta^{\mathcal{V}}(\alpha)[x])[\cdot]$ with $x \notin \boldsymbol{V}\Theta^{\mathcal{V}}(\alpha)$;*
5. *Conjunction: $\Theta^{\boldsymbol{P}}(\langle \alpha, \beta \rangle) = f(\Theta^{\boldsymbol{P}}(\alpha), \Theta^{\boldsymbol{P}}(\beta))$ and $\Theta^{\mathcal{V}}(\langle \alpha, \beta \rangle) = f(\Theta^{\mathcal{V}}(\alpha), \Theta^{\mathcal{V}}(\beta))$; where $f : \mathbf{C}^{\Lambda_\oplus} \times \mathbf{C}^{\Lambda_\oplus} \to \mathbf{C}^{\Lambda_\oplus}$ is defined as[4]:*

$$f(\mathcal{C}, \mathcal{D}) = (\lambda x.(\lambda y, z.I)(\mathcal{C}[xI])(\mathcal{D}[xI]))(\lambda x.[\cdot]); \quad \text{where } x \notin \boldsymbol{V}\mathcal{C} \cup \boldsymbol{V}\mathcal{D}$$

Observe that Definition 5.2.5 is actually ambiguous, since we have to choose the variable x arbitrarily in item 4, item 5. However, the contexts it is possible to build are equivalent using α-conversion, which is enough since we fill them with *programs* only.

In the example below, we show how the ability of *CBV* contexts to *first* evaluate and *next* copy programs allows us to emulate tests with a conjunctive structure. By contrast, CBN contexts do no have this ability, and in consequence are not able to emulate conjunctive tests.

Example 5.2.2 *We look at the test $\alpha = eval \cdot V \cdot \langle eval \cdot \omega, eval \cdot \omega \rangle$ of Example 5.2.1: recall that we used this test to distinguish the programs $\lambda x.(\Omega \oplus I)$ and $(\lambda x.\Omega) \oplus (\lambda x.I)$, that cannot be distinguished by linear traces alone. We illustrate Definition 5.2.5 by building the context emulating the test α: we want to compute $\Theta^{\boldsymbol{P}}(\alpha)$. The inductive step-by-step computation from Definition 5.2.5 is represented in Figure 5.3, where the test α is represented by its syntax tree. We analyze here the behavior of the resulting context, looking at each block used to built it.*

[4]Actually, we could replace I in the definition of f by any other program that terminates with probability 1.

- *First, we see that* $\Theta^{\boldsymbol{P}}(eval{\cdot}\omega) = (\lambda x.(\lambda y.x))[\cdot]$. *We denote this context* $\mathcal{C}^{\downarrow}$*; intuitively, it evaluates its argument and then store the result. Observe that for every* N*, it holds that the convergence probability of* $\mathcal{C}^{\downarrow}[N]$ *is equal to that of* N*.*

- *We look now at the context* $\Theta^{\boldsymbol{P}}(\langle eval \cdot \omega, eval \cdot \omega\rangle)$*, that we denote* $\mathcal{C}^{2\times\downarrow}$*. Observe that as represented in Figure 5.3, we may compute* $\mathcal{C}^{2\times\downarrow}$ *by using* $\mathcal{C}^{\downarrow}$ *and the conjunctive function* f*. We obtain:*

$$\mathcal{C}^{2\times\downarrow} = (\lambda x.(\lambda y.\lambda z.I)\mathcal{C}^{\downarrow}[xI]\mathcal{C}^{\downarrow}[xI])(\lambda x.[\cdot]).$$

  We denote this context We see that for any program N*, it holds that:*

$$\mathcal{C}^{2\times\downarrow}[N] \to (\lambda y.\lambda z.I)(\mathcal{C}^{\downarrow}[(\lambda x.N)I])(\mathcal{C}^{\downarrow}[(\lambda x.N)I]).$$

  Recall from Chapter 2 that $(\lambda x.N)I$ *is contextually equivalent to* N*—since they reduce to exactly the same distribution over values. As a consequence, the termination probability of* $\mathcal{C}^{2\times\downarrow}[N]$ *is the same as that of the program below:*

$$(\lambda y.\lambda z.I)(\mathcal{C}^{\downarrow}[N])(\mathcal{C}^{\downarrow}[N]).$$

  On this form, it is easier to understand what $\mathcal{C}^{2\times\downarrow}$ *does: it takes two copies of its argument* N*, evaluates each of them independently using the context* $\mathcal{C}^{\downarrow}$*, and returns* I*—the identity term —whenever this computation terminates.*

- *Finally, we see that* $\Theta^{\boldsymbol{P}}(\alpha) = (\lambda x.\Theta^{\boldsymbol{P}}(\langle eval \cdot \omega, eval \cdot \omega\rangle)[xV])[\cdot]$*. It means that* $\Theta^{\boldsymbol{P}}(\alpha)$ *does the following: first it evaluates its argument, then it passes to them the value* V*, and finally it gives the result to the context* $\mathcal{C}^{2\times\downarrow}$*.*

If we sum up the considerations above, we see that the context $\Theta^{\boldsymbol{P}}(\alpha)$ *emulates indeed the interaction specified by the test* α*, that has been analyzed in details in Example 5.2.1.*

We now state the main result of this section: for each test $\alpha \in \mathcal{T}_0$, its behavior on programs is indeed *emulated* by the context $\Theta^{\mathbf{P}}(\alpha)$. This result is crucial for showing completeness of $\equiv^{\mathrm{cbv}}_{\Lambda_\oplus}$ with respect to context equivalence in CBV, since it amounts to show that the testing characterization of bisimilarity on $\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}$ can be *translated into* a characterization by a family of contexts.

Theorem 5.2.5 *Let* α *be a test in* $\mathcal{T}_0(\mathcal{M}^{cbv}_{\Lambda_\oplus})$*. Then for every* M *closed term and every* V *value it holds that:*

$$Pr^{\mathcal{M}^{cbv}_{\Lambda_\oplus}}(M,\alpha) = |[\![\Theta^{\boldsymbol{P}}(\alpha)[M]]\!]|; \qquad Pr^{\mathcal{M}^{cbv}_{\Lambda_\oplus}}(\hat{V},\alpha) = |[\![\Theta^{\mathcal{V}}(\alpha)[V]]\!]|.$$

Proof. We are going to show the thesis by induction on the structure of the test α.

- If $\alpha = \omega$, then for every closed term M, and every closed value V, both $\Pr^{\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}}(M,\omega)$ and $\Pr^{\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}}(\hat{V},\omega)$ are equal to 1, and we have defined accordingly $\Theta^{\mathbf{P}}(\omega) = \Theta^{\mathcal{V}}(\omega) = \lambda x.[\cdot]$. Since $\Theta^{\mathbf{P}}(\omega)[M]$ and $\Theta^{\mathcal{V}}(\omega)[V]$ are values, their termination probability is 1, and so the result holds.
- If $\alpha = \langle\beta_1,\beta_2\rangle$ we see by induction hypothesis, that for all $1 \le i \le 2$, $\forall M \in \mathbf{P}, \forall V \in \mathcal{V}$: $\Pr^{\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}}(M,\beta_i) = |[\![\Theta^{\mathbf{P}}(\beta_i)[M]]\!]|$ and $\Pr^{\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}}(\hat{V},\beta_i) = |[\![\Theta^{\mathcal{V}}(\beta_i)[V]]\!]|$. Looking at Definition 5.2.5, we see that :

$$\Theta^{\mathbf{P}}(\langle\beta_1,\beta_2\rangle) = f(\Theta^{\mathbf{P}}(\beta_1),\Theta^{\mathbf{P}}(\beta_2))$$
$$\Theta^{\mathcal{V}}(\langle\beta_1,\beta_2\rangle) = f(\Theta^{\mathcal{V}}(\beta_1),\Theta^{\mathcal{V}}(\beta_2))$$

The overall behavior of the context $f(\mathcal{C},\mathcal{D})$ is to copy the content of its hole into the holes of the two contexts $\mathcal{C}$ and $\mathcal{D}$. For any closed term M, and as soon as x does not already appear neither in $\mathcal{C}$ nor in $\mathcal{D}$, we can express the convergence probability of $f(\mathcal{C},\mathcal{D})[M]$ as a function of the convergence probability of $\mathcal{C}[M]$ and $\mathcal{D}[M]$:

$$\begin{aligned}|[\![f(\mathcal{C},\mathcal{D})[M]]\!]| &= (|[\![\mathcal{C}[(\lambda x.M)I]]\!]|)\cdot(|[\![\mathcal{D}[(\lambda x.M)I]]\!]|)\\ &= (|[\![\mathcal{C}[M]]\!]|)\cdot(|[\![\mathcal{D}[M]]\!]|) \qquad \text{since } \forall L\in\mathbf{P},\ L\equiv^{\text{ctx}}(\lambda x.L)I\end{aligned}$$

As a consequence, for any closed term M, and any value V:

$$|[\![\Theta^{\mathbf{P}}(\langle\beta_1,\beta_2\rangle)[M]]\!]| = \Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(M,\beta_1)\cdot\Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(M,\beta_2) = \Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(M,\langle\beta_1,\beta_2\rangle);$$
$$|[\![\Theta^{\mathcal{V}}(\langle\beta_1,\beta_2\rangle)[V]]\!]| = \Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(\hat{V},\beta_1)\cdot\Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(\hat{V},\beta_2) = \Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(\hat{V},\langle\beta_1,\beta_2\rangle);$$

and so the result holds.

- If $\alpha = a\cdot\beta$, then we consider separately the case where a is the action eval, and the case where it is a value:
  - First, let us suppose that $a=\text{eval}$. We first consider $\Theta^{\mathcal{V}}(\alpha)$: since the eval action is relevant only for states of $\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}$ which are terms (and not distinguished values), we want to show that $\Theta^{\mathcal{V}}(\alpha)[V]$ always diverges. Since $\Theta^{\mathcal{V}}(\alpha)=\Omega[\cdot]$ and since $[\![\Omega]\!]=\emptyset$, we have indeed that for any $V\in\mathcal{V}$, $[\![\Theta^{\mathcal{V}}(\alpha)[V]]\!]=\emptyset$. We consider now $\Theta^{\mathbf{P}}(\alpha)$: by the induction hypothesis, we know that $\Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}([\![V]\!],\beta) = |[\![\Theta^{\mathcal{V}}(\beta)[V]]\!]|$. Please recall that we have defined $\Theta^{\mathbf{P}}(a\cdot\beta)=\lambda x.(\Theta^{\mathcal{V}}(\beta)[x])[\cdot]$, where x is a fresh variable that in particular does not already appear in $\Theta^{\mathcal{V}}(\beta)$. We can see now that for every $M\in\mathbf{P}$, it holds that:

$$\begin{aligned}|[\![\Theta^{\mathbf{P}}(a\cdot\beta)[M]]\!]| &= \sum_V [\![M]\!](V)\cdot|[\![\Theta^{\mathcal{V}}(\beta)[V]]\!]|\\ &= \sum_V [\![M]\!](V)\cdot\Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}([\![V]\!],\beta)\\ &= \sum_{s\in\mathcal{S}^{\text{cbv}}_{\Lambda_\oplus}} \mathcal{P}^{\text{cbv}}_{\Lambda_\oplus}(M,\text{eval},s)\cdot\Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(s,\beta) = \Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(M,\beta)\end{aligned}$$

  - Now, we suppose that $a=V$, with $V\in\mathcal{V}$. First, recall that we have defined $\Theta^{\mathbf{P}}(V\cdot\beta)=\Omega[\cdot]$, and so $\Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(M,V\cdot\beta)=0=|[\![\Theta^{\mathbf{P}}(V\cdot\beta)[M]]\!]|$. We look now at $\Theta^{\mathcal{V}}(V\cdot\beta)$, that as been defined in Definition 5.2.5 as $\Theta^{\mathbf{P}}(\beta)[[\cdot]V]$. For any $W=\lambda x.M\in\mathcal{V}$, we can see that:

$$\begin{aligned}|[\![\Theta^{\mathcal{V}}(V\cdot\beta)[W]]\!]| &= |[\![\Theta^{\mathbf{P}}(\beta)[WV]]\!]|\\ &= \Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(WV,\beta) \quad \text{by induction hypothesis}\\ &= \Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(M\{V/x\},\beta) \quad \text{since } [\![WV]\!]=[\![M\{V/x\}]\!]\\ &= \Pr^{\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}}(\hat{W},V\cdot\beta).\end{aligned}$$

□

We can now use the contexts emulating $\mathcal{T}_0(\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus})$ to finally show that $\equiv^{\text{cbv}}_{\Lambda_\oplus}$ is fully abstract with respect to context equivalence in CBV $\Lambda_\oplus$.

Theorem 5.2.6 *$\equiv$ is fully abstract with respect to the contextual equivalence.*

Proof. We already know from corollary 5.1.1 that $\equiv$ is sound, that is $\equiv \subseteq \equiv^{\mathrm{ctx}}$. Hence, what is left to show is that $\equiv^{\mathrm{ctx}} \subseteq \equiv$. Let $M, N \in \mathbf{P}$ be such that $M \equiv^{\mathrm{ctx}} N$. We want to show that $M \equiv N$. The testing characterization of simulation tells us that it is sufficient to show that, for every test $\alpha \in \mathcal{T}_0(\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus})$, $\Pr^{\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}}(M, \alpha) = \Pr^{\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}}(N, \alpha)$, which in turn is a consequence of Theorem 5.2.5, since every test α of $\mathcal{T}_0(\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus})$ can be simulated by a context of CBV $\Lambda_\oplus$. □

5.2.4 The Asymmetric Case: $\preceq^{\mathbf{cbv}}_{\Lambda_\oplus}$ is not complete

Theorem 5.2.6 establishes a precise correspondence between bisimilarity and context equivalence. However, it cannot be extended into a correspondence between similarity and the context preorder; we are actually able to show that similarity and context preorder *do not* coincide, by giving a counterexample, namely a pair of terms which can be compared in the context preorder but which are not similar. This counter-example actually consists of the same programs as the counter example to completeness of probabilistic applicative bisimilarity for *CBN* $\Lambda_\oplus$. Let us fix the following terms: $M = \lambda y.(\Omega \oplus I)$ and $N = (\lambda y.\Omega) \oplus (\lambda y.I)$.

Recall that in CBN, M and N are contextually equivalent, but not bisimilar. By contrast, in CBV M and N are neither context equivalent, nor bisimilar: it is for instance witnessed by the context $\mathcal{C} = \Theta^{\mathbf{P}}(\alpha)$ build in Example 5.2.2 as the context emulating the test $\alpha = \mathrm{eval} \cdot V \cdot \langle \mathrm{eval} \cdot \omega, \mathrm{eval} \cdot \omega \rangle$ that distinguished M and N. Since $\mathcal{C}$ emulates α, we can in particular deduce from the success probability of α when applied to M and N—as computed in Example 5.2.1 —that $|[\![\mathcal{C}[M]]\!]| < |[\![\mathcal{C}[N]]\!]|$: it means that the context $\mathcal{C}$ and the test α actually witness respectively $\neg(N \leq^{\mathrm{ctx}} M)$ and $\neg(N \preceq^{\mathrm{cbv}}_{\Lambda_\oplus} M)$.

We look now at the reverse comparison: does $M \leq^{\mathrm{ctx}} N$ and $M \preceq^{\mathrm{cbv}}_{\Lambda_\oplus} N$ hold ? We first consider the situation with respect to the probabilistic applicative similarity.

Lemma 5.2.7 *$M \preceq^{cbv}_{\Lambda_\oplus} N$ does not hold.*

Proof. We are able to build a test $\beta \in \mathcal{T}_1(\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus})$, that witnesses $\neg(M \preceq^{\mathrm{cbv}}_{\Lambda_\oplus} N)$. We take $\beta = \mathrm{eval} \cdot V \cdot (\mathrm{eval} \cdot \omega \vee \mathrm{eval} \cdot \omega)$, and we compute its success probability when executed on M and N as represented in Figure 5.4: we see that that $\Pr^{\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}}(\beta, M) = \frac{3}{4}$, while $\Pr^{\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}}(\beta, N) = \frac{1}{2}$. □

Observe that the test β that witnesses $\neg(M \preceq^{\mathrm{cbv}}_{\Lambda_\oplus} N)$ contains a *disjunctive construct*, hence is not in $\mathcal{T}_0(\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus})$. It means that we cannot construct a context emulating it by using $\Theta^{\mathbf{P}}$. Actually, there *doesn't' exist* a context that emulates β, because it *does* hold that $M \leq^{\mathrm{ctx}} N$. Since we cannot use soundness of applicative similarity to show it, we need to deal with the *universal quantification* over all contexts. The proof has been detailed in [24], we recall here only the main steps. Let us first introduce some notation: we call $L_0 = \lambda y.\Omega$, and $L_1 = \lambda y.I$. If $b = b_1, \ldots, b_n \in \{0,1\}^n$, then L_b denotes the sequence of terms $L_{b_1} \cdots L_{b_n}$. If K is a term, $K \Rightarrow p$ means that there is a distribution $\mathscr{D}$ such that $K \Rightarrow \mathscr{D}$ and $|\mathscr{D}| = p$ (where $\Rightarrow$ is small-step approximation semantics; see Definition 1.2.4 from Chapter 1). The idea, now, is to prove that in any term K, if we replace an occurrence of M by an occurrence of N, we obtain a term T which converges with probability smaller than the one with which K converges. We first need an auxiliary lemma, which proves a similar result for L_0 and L_1.

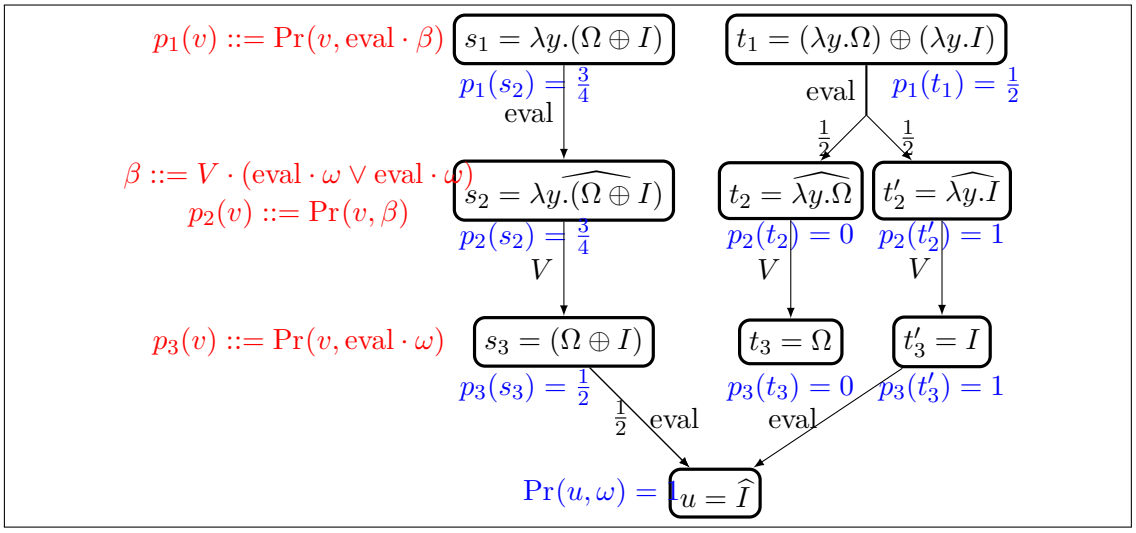


Figure 5.4: The test $\beta \in \mathcal{T}_1(\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus})$ witnessing $\neg(M \preceq^{\text{cbv}}_{\Lambda_\oplus} N)$

Lemma 5.2.8 *For every term K, if $(K\{x/L_0\}) \Rightarrow p$, then there is another real number $q \geq p$ such that $(K\{x/L_1\}) \Rightarrow q$.*

We are now ready to prove the central lemma of this section, which takes a rather complicated form just for the sake of its inductive proof:

Lemma 5.2.9 *Suppose that K is a term and suppose that $(K\{x, \overline{y}/M, L\}) \Rightarrow p$, where $\overline{y} = y_1, \ldots, y_n$. Then for every $b \in \{0,1\}^n$ there is p_b such that $(K\{x, \overline{y}/N, L_b\}) \Rightarrow p_b$ and $\sum_b \frac{p_b}{2^n} \geq p$.*

We may then conclude by using Lemma 5.2.9 in the case where $n = 0$, that indeed $M \leq^{\text{ctx}} N$. As a consequence, $\preceq^{\text{cbv}}_{\Lambda_\oplus}$ is not complete with respect to the context preorder, since there are programs connected by $\leq^{\text{ctx}}$ but not by $\preceq^{\text{cbv}}_{\Lambda_\oplus}$.

Proposition 5.2.10 *$\preceq^{cbv}_{\Lambda_\oplus}$ is not fully abstract with respect to the contextual preorder.*

5.2.5 Adding Parallel Convergence Testing to $\Lambda_\oplus$

The incompleteness result for the preorder $\preceq^{\text{cbv}}_{\Lambda_\oplus}$ may be seen as a consequence of the impossibility to write $\Lambda_\oplus$ contexts that emulate *disjunctive tests*: the crucial point here is that *failure* for *tests* means *non-termination* for the corresponding *contexts*. For this reason, a context is not actually able to emulate the *independent* execution of two tests, since once a program in $\Lambda_\oplus$ begin to evaluates a sub-term, it may only stop when the evaluation of this sub-term stops. However, when we consider a *conjunctive test* $\langle \alpha, \beta \rangle$, we are able to *sequentialise* the experiment: we can decide that the experimenter first executes the test α, and then if α *succeeds* he executes the test β. But it is no possible to sequentialise the disjunctive test $\alpha \vee \beta$, since the experiment should succeed as soon as *one of the two tests* succeed: it is not possible for the experimenter to decide to begin by executing α or β. To sum up, non-completeness of the similarity $\preceq^{\text{cbv}}_{\Lambda_\oplus}$ comes from the fact that the $\Lambda_\oplus$ contexts may only emulates *sequential tests*.

A way to address this issue is to add *parallelism* to the language $\Lambda_\oplus$, by enriching it with a *parallel convergence testing operator*, similar to the one introduced by Abramsky and Ong in [4] for lazy Λ-calculus. The addition to a higher-order language of some kind of

parallel operator has been used before to close the gap between soundness of a denotational semantics and its full abstraction, for instance in [4] for CBN Λ, or by Plotkin in [93], where he adds a *parallel or* operator to PCF [5]. In a joint work with Dal Lago, Vignudelli and Sangiorgi [27], we looked at applicative similarity for the higher-order language $\Lambda^{\parallel}_{\oplus}$, which is $\Lambda_{\oplus}$ extended with a parallel convergence testing operator. We sum up briefly the results presented there.

The syntax of $\Lambda^{\parallel}_{\oplus}$ is generated formally as follows:

$$M, N, L \ldots \in \Lambda^{\parallel}_{\oplus} ::= \; x \;\mid\; \lambda x.M \;\mid\; MN \;\mid\; M \oplus N \;\mid\; [M \parallel N]\,.$$

The behavior of the parallel convergence testing construct $([\cdot \parallel \cdot])$ is meant to be as specified in [4]: the program $[M \parallel N]$ returns I if *at least* one of the two programs M or N terminates, and otherwise does not terminate. We need to extend the operational semantics to implement this behavior; we define the operational semantics of $\Lambda^{\parallel}_{\oplus}$, by adding the two following redex-rules to the $\Lambda_{\oplus}$ reduction relation presented in Chapter 1:

$$[V \parallel N] \to_R I \qquad \text{and} \qquad [M \parallel V] \to_R I,$$

We define now the *evaluation contexts* of $\Lambda^{\parallel}_{\oplus}$ as follows:

$$\mathcal{E} ::= [\cdot] \;\mid\; \mathcal{E}M \;\mid\; V\mathcal{E} \;\mid\; [\mathcal{E} \parallel M] \;\mid\; [M \parallel \mathcal{E}]\,.$$

Observe that the reduction may now happens arbitrarily in the two branches of $[M \parallel N]$: it means that the one-step semantics becomes *non-deterministic*. However it is still true that $\{\mathscr{D} \mid M \Rightarrow \mathscr{D}\}$ is a directed set, which allows to extend the operational semantics $[\![\cdot]\!]$ to $\Lambda^{\parallel}_{\oplus}$, by taking $[\![M]\!] = \sup\{\mathscr{D} \mid M \Rightarrow \mathscr{D}\}$. Equivalently, we can also define a *big-step* approximations operational semantics:

$$\frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{[M \parallel N] \Downarrow (|\mathscr{D}| + |\mathscr{E}| - (|\mathscr{D}| \cdot |\mathscr{E}|)) \cdot \{I^1\}}\, b_{or},$$

and we can still show that $[\![M]\!] = \sup\{\mathscr{D} \mid M \Downarrow \mathscr{D}\}$.

We may easily define a LMC $\mathcal{M}^{\mathrm{cbv}}_{\Lambda^{\parallel}_{\oplus}}$ that is the counterpart of the LMC $\mathcal{M}^{\mathrm{cbv}}_{\Lambda_{\oplus}}$ for CBV $\Lambda^{\parallel}_{\oplus}$, that allows us to define probabilistic applicative similarity and bisimilarity for CBV $\Lambda^{\parallel}_{\oplus}$. Soundness of probabilistic applicative similarity and bisimilarity for CBV $\Lambda^{\parallel}_{\oplus}$ is shown in [27], using Howe's method—the proof is an immediate extension of the one for CBV $\Lambda_{\oplus}$. We look now at the full abstraction problem for $\preceq^{\mathrm{cbv}}_{\Lambda^{\parallel}_{\oplus}}$ with respect to the context preorder. Recall that in the completeness proof for *bisimilarity* for CBV $\Lambda_{\oplus}$, we build contexts designed to *emulate* all tests in $\mathcal{T}_0$. By adding the parallel convergence tester we gain the ability to emulate also the *disjunctive test*, hence the whole testing language $\mathcal{T}_1$. We illustrate this by showing that we can write a context that emulates the test $\beta = V \cdot (\mathrm{eval} \cdot \omega \vee \mathrm{eval} \cdot \omega)$. Recall that it is the precise test we have used in Section 5.2.4 when showing that it was not the case that $\lambda y.(\Omega \oplus I) \preceq^{\mathrm{cbv}}_{\Lambda_{\oplus}} (\lambda y.\Omega) \oplus (\lambda y.I)$.

Example 5.2.3 *We consider $\beta = eval \cdot V \cdot (eval \cdot \omega \vee eval \cdot \omega)$. We consider the $\Lambda^{\parallel}_{\oplus}$ context $\mathcal{C} = (\lambda x.\,[xV \parallel xV]\,)[\cdot]$. Observe that for every program M: $[\![\mathcal{C}[M]]\!] = (|[\![MV]\!]| - |[\![MV]\!]|) \cdot \{I^1\}$.*

[5] It should be noted that in a calculus with ground data, *parallel or* and *parallel convergence testing* are not equivalent: for instance parallel or is not expressible in PCF enriched with a parallel convergence tester, see for instance [4]

As we outlined in [27] we are able to *systematize* the construction we have done in Example 5.2.3, i.e. to extend the encodings $\Theta^{\mathcal{V}}, \Theta^{\mathbf{P}} : \mathcal{T}_0(\mathcal{M}^{\text{cbv}}_{\Lambda_\oplus}) \to \mathbf{C}^{\Lambda_\oplus}$ into $\Theta^{\mathcal{V}}_{\parallel}, \Theta^{\mathbf{P}}_{\parallel} : \mathcal{T}_1(\mathcal{M}^{\text{cbv}}_{\Lambda^{\parallel}_\oplus}) \to \mathbf{C}^{\Lambda^{\parallel}_\oplus}$, in such a way that the context $\Theta^{\mathbf{P}}_{\parallel}(\alpha)$ indeed emulates the test α when applied on states of the LMC that are programs. It means that all tests in $\mathcal{T}_1$ can be emulated by contexts of $\Lambda^{\parallel}_\oplus$, and consequently we have the following full-abstraction theorem for CBV $\Lambda^{\parallel}_\oplus$:

Theorem 5.2.11 *Both $\preceq^{cbv}_{\Lambda^{\parallel}_\oplus}$ and $\equiv^{cbv}_{\Lambda^{\parallel}_\oplus}$ are fully abstract with respect to respectively context preorder and context equivalence for CBV $\Lambda^{\parallel}_\oplus$.*

5.3 On Trace Equivalence For CBV $\Lambda_\oplus$.

Recall that for CBN $\Lambda_\oplus$, Dal Lago, Sangiorgi and Alberti studied *two* behavioral equivalences: probabilistic applicative bisimilarity and *trace equivalence*, that was *fully abstract* in this setting. We look here briefly to trace equivalence for CBV $\Lambda_\oplus$. We can actually follow *two* distinct approach when generalizing trace equivalence to CBV $\Lambda_\oplus$, depending how we interpret CBN trace equivalence. The first possibility consists in considering that *trace equivalence* is the behavioral equivalence witnessed by *traces*, i.e. linear tests, while the second one consists in considering that it is the behavioral equivalence corresponding to *evaluation contexts*. Recall that in CBN, these two notions coincide, because evaluation contexts are exactly applicative contexts, i.e. those contexts that emulate linear tests.

5.3.1 Applicative Contexts Equivalence for CBV

We first define the CBV counterpart of CBN applicative contexts that were defined in Section 4.2.1 of Chapter 4: *CBV applicative contexts* are those contexts of shape $[\cdot]V_1 \dots V_n$, where $V_1, \dots, V_n$ are values. Similarly to what was done for CBN, we say that two programs M, N are trace equivalent, if for every CBV applicative context $\mathcal{C}$, it holds that $\text{Obs}(\mathcal{C}[M]) = \text{Obs}(\mathcal{C}[N])$. However, the induced equivalence relation is *unsound*. Indeed, let us consider again the terms: $M = \lambda x.(I \oplus \Omega)$ and $N = (\lambda x.I) \oplus (\lambda x.\Omega)$. We have shown in example 4.2.1 that M and N were trace equivalent in CBN, and we can see similarly that they are also trace equivalent in CBV. However, they are not context equivalent in CBV, as we have shown in Example 2.3.1 of Chapter 2: it is because in CBV, we can write contexts that evaluate their argument *before* copying and using it, which is never done by an applicative context.

It means that trace equivalence is not really relevant when we consider $\Lambda_\oplus$ endowed with a CBV evaluation strategy.

5.3.2 On CBV CIU-Equivalence

We look now at what happens if we consider the equivalence notion induced by *CBV evaluations contexts*. By contrast with the CBN case, evaluations contexts in CBV are more expressive as linear tests. Indeed, recall from Chapter 1 that CBV evaluations contexts are generated by the grammar below:

$$\mathcal{E} ::= [\cdot] \mid \mathcal{E}M \mid V\mathcal{E}$$

We denote $\equiv_{\text{CIU}}$ the equivalence relation based on evaluation contexts. Observe that it is immediate that $\equiv^{\text{ctx}} \subseteq \equiv_{\text{CIU}}$.

We show now that the reverse inclusion also holds. First, observe that it is equivalent to show $\equiv_{\text{CIU}} \subseteq \equiv^{\text{cbv}}_{\Lambda_{\oplus}}$. Suppose that M, N are two programs that verify $\neg(M \equiv^{\text{cbv}}_{\Lambda_{\oplus}} N)$; we want to show that $\neg(M \equiv_{\text{CIU}} N)$. We are again going to use the testing characterization. Observe that whenever a test α is of the shape $\text{eval} \cdot \alpha$, $\Theta^{\mathbf{P}}(\alpha)$ is an evaluation context. Moreover, we can see that whenever two terms can be distinguished by some test in $\mathcal{T}_0$, the they may also be distinguished by a test of the shape $\text{eval} \cdot \alpha$. As a consequence, M and N may also be separated by an evaluation context, and so we obtain the result.

Proposition 5.3.1 *$\equiv_{CIU}$ is fully abstract with respect to context equivalence for CBV $\Lambda_{\oplus}$.*

Chapter 6

The Trivialization Problem for Observational Distance: A Case Study.

We argued in Chapter 3 that for the purpose of comparing probabilistic programs, *observational distance* is often a more adapted notion than the 2-valued observational equivalence, because it allows us to talk about programs which behave very similarly, although not being equivalent. This chapter and the next one are devoted to a more in-depth look at the structure of observational distance, when considering higher-order languages with discrete probabilistic primitives. In the present chapter, we focus on the *trivialization problem*, which we have informally described before in Section 3.2: more precisely, we identify two distinct cases where we can build enough *amplification contexts*—see the considerations on trivialization in Section 3.2—to enable trivialization of the observational distance: in the former all programs terminate with probability 1, while in the latter the expressive power of contexts is enhanced with parallel or. We develop these two cases on *concrete* languages: for the first one, we consider $\mathbb{T}_{\oplus}$ the probabilistic extension of Gödel's system T—introduced by Breuvart, Dal Lago and Herrou [19]—that we presented in Section 1.3.2 of Chapter 1. For the second one, we will consider the language $\Lambda_{\oplus}^{\parallel}$—i.e. the variant of $\Lambda_{\oplus}$ enriched with parallel convergence testing that we have presented in Section 5.2.5 of Chapter 5.

The work we present here was published jointly with Ugo Dal Lago [29].

6.1 On Amplification Contexts

As we have illustrated in Example 3.2.1 of Chapter 3, there can well be classes of terms such that the observational distance collapses to observational equivalence, due to the copying capabilities of the underlying language. The trivialization problem can in fact be seen as a question about the expressive power of contexts: given two duplicable terms, how much can any context *amplify* the observable differences between their behaviors? More precisely, we would like to identify *trivializing* fragments of $\Lambda_{\oplus}^{!,\parallel}$, that is to say fragments such that for any pair of duplicable terms, their context distance (with respect to the fragment) is either 0 or 1.

Our approach is as follows: as soon as two programs M, N are not observationally equivalent, there exists at least one context $\mathcal{C}$ that *separates* them. Our idea is to take this context as a starting point, and to see if it is then possible to amplify the difference

induced by $\mathcal{C}$, by building another context $\mathcal{D}$ with

$$|\mathrm{Obs}(\mathcal{D}[\mathcal{C}[M]]) - \mathrm{Obs}(\mathcal{D}[\mathcal{C}[N]])| > |\mathrm{Obs}(\mathcal{C}[M]) - \mathrm{Obs}(\mathcal{C}[N])|.$$

The context $\mathcal{D}$ that we are going to build for this purpose actually interacts with its argument only by copying it, and then evaluating all the copies: essentially, $\mathrm{Obs}(\mathcal{D}[L])$ depends only of the termination probability of L—it is because $\mathcal{D}$ is designed just to *amplify* existing differences in the observables.

We want to present this approach in the setting of a generic probabilistic programming language, no matter typed or untyped: this way, we will be able to apply it to both $\mathbb{T}_{\oplus}$—probabilistic variant of Gödel's system T—and $\Lambda^{\parallel}_{\oplus}$. We use here the formalism for a generic observable programming language that we have introduced in Chapter 2—in Definition 2.3.1 for an untyped language, and in Definition 2.3.2 for a typed language. In order to talk about *amplification contexts* in this setting, we first need to enforce that contexts can be *composed*.

Definition 6.1.1 (Compositionality for Contexts) *Let $\mathcal{L}_{\oplus}$ be an observable programming language. We say that the* compositionality requirement for contexts holds *when:*

- *if $\mathcal{L}_{\oplus}$ is an untyped language, we ask that for every $\mathcal{C}, \mathcal{D} \in \mathbf{C}_{\emptyset}$, $\mathcal{C}[\mathcal{D}]$ is also in $\mathbf{C}_{\emptyset}$;*
- *if $\mathcal{L}_{\oplus}$ is a typed language, we ask that for every type σ, ι, for every observable type $\tau \in \mathscr{A}_{obs}$, if $\mathcal{C} \in \mathbf{C}_{(\emptyset,\sigma)\to\tau}$, and $\mathcal{D} \in \mathbf{C}_{(\emptyset,\iota)\to\sigma}$, it holds that $\mathcal{C}[\mathcal{D}] \in \mathbf{C}_{(\emptyset,\iota)\to\tau}$.*

In the following, the compositionality requirement for contexts will hold for all the probabilistic programming languages that we will consider. Looking now at the amplification context we mentioned above, we see that we must require that it accepts *all observable programs* in argument. We formalize this by asking it to be an *universal context*, in the sense of the definition below.

Definition 6.1.2 (Universal context.) *Let $\mathcal{L}_{\oplus}$ be a probabilistic observable programming language. We say a* universal observable context *is:*

- *any context $\mathcal{C} \in \mathbf{C}_{\emptyset}$, if $\mathcal{L}_{\oplus}$ is an untyped language;*
- *if $\mathcal{L}_{\oplus}$ is a typed language, a context $\mathcal{C}$ such that: $\forall \sigma \in \mathscr{A}_{obs}, \exists \tau \in \mathscr{A}_{obs}$ with $\mathcal{C} \in \mathbf{C}_{(\emptyset,\sigma)\to\tau}$.*

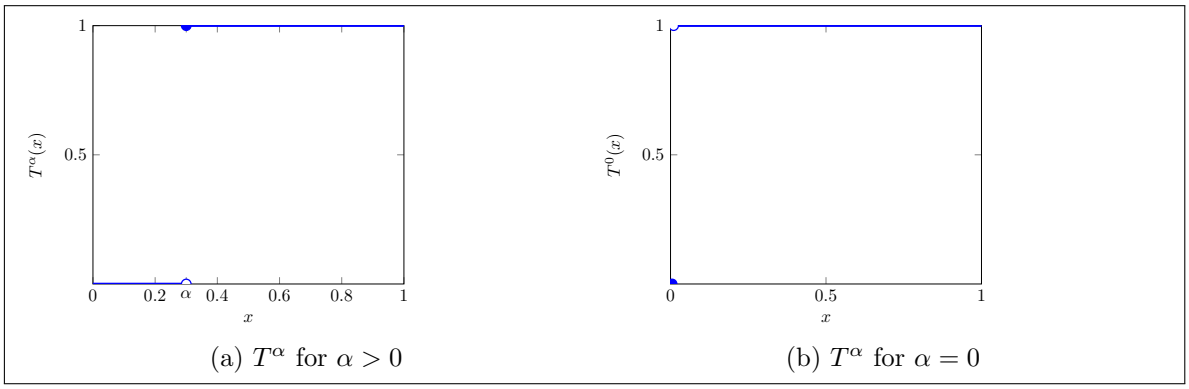
Ideally, we would like to be able to build for every α an universal amplification contexts $\mathcal{D}_{\alpha}$ that *simulate* the *threshold functions* T^{α}, that we define in Definition 6.1.4. We first precise in Definition 6.1.3 below what we mean by simulating a function by a context.

Definition 6.1.3 *We say that a universal observable context $\mathcal{C}$* simulates *a function $g : [0,1] \to [0,1]$ if:*

$$\forall M \text{ observable program }, \mathit{Obs}(\mathcal{C}[M]) = g(\mathit{Obs}(M)).$$

Indeed, if we have $\mathrm{Obs}(\mathcal{C}[M]) < \mathrm{Obs}(\mathcal{C}[N])$, then we can take any $\alpha \in [0,1]$ between them, and we see that $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are *completely separated* by T^{α}, in the following sense: $|T^{\alpha}(\mathrm{Obs}(\mathcal{C}[M])) - T^{\alpha}(\mathrm{Obs}(\mathcal{C}[N]))| = 1$.

Definition 6.1.4 (Threeshold Functions) *For α in [0,1], we define the function $T^{\alpha} : [0,1] \to \{0,1\}$ as follows:*



(a) T^α for $\alpha > 0$ (b) T^α for $\alpha = 0$

Figure 6.1: The threshold test functions T^α.

- $T^0(0) = 0$, *and* $T^0(x) = 1$ *for* $x > 1$.
- *if* $\alpha > 0$*:* $T^\alpha(x) = 0$ *if* $x < \alpha$, 1 *otherwise.*

The threshold functions are represented in Figure 6.1. However, requiring the existence of a context $\mathcal{D}$ that simulate T^α for every $\alpha \in [0,1]$ is too strong: we are now going to progressively weaken this condition. First, we observe considering every $\alpha \in [0,1]$ is not necessary: since the set $\mathbb{Q}$ is dense in $\mathbb{R}$, it is sufficient to look at the T^α for $\alpha \in \mathbb{Q} \cap [0,1]$. We then see that we don't really have to find a context that *exactly* simulates T^α, but that it is enough to find a *family of contexts* that approximates this function, in the sense of Definition 6.1.5 below.

Definition 6.1.5 *Let* $f : [0,1] \to [0,1]$ *be a real function, and* $I \subseteq [0,1]$*. An* approximation by contexts of f on I *is a family* $(\mathcal{C}_n)_{n\in\mathbb{N}}$ *of universal contexts such that for every program* M *with* $Obs(M) \in I$*,*

$$\lim_{n\to\infty} Obs(\mathcal{C}_n[M]) = f(Obs(M)).$$

We illustrate now the notion of approximation by contexts. Recall the $\Lambda_\oplus$-contexts $\mathcal{C}_n$ considered in Example 3.2.1 in Chapter 3, that we used to show that $\delta^{\mathrm{ctx}}_{\Lambda_\oplus}(I, I \oplus \Omega) = 1$. In Example 6.1.1 below, we reformulate them as a family of approximations for T^1.

Example 6.1.1 *We first build in* $\Lambda_\oplus$ *a* n-ary conjunction operator $\bigwedge^n([\cdot]_1, \dots, [\cdot]_n)$ *such that if* $M_1, \dots, M_n$ *are programs,* $\bigwedge^n(M_1, \dots, M_n)$ *terminates if and only if* all *the* M_i *terminate. Indeed, for every* $n \in \mathbb{N}$*, and* n *terms* $M_1, \dots, M_n$*, we take:*

$$\bigwedge^n(M_1 \dots M_n) = (\lambda z_1.\lambda z_2. \dots \lambda y.(y z_1 \dots z_n))M_1 \dots M_n.$$

Using this conjunction operator, we can now use the copying ability of $\Lambda_\oplus$ *to build for every* $n \in \mathbb{N}$*, a* $\Lambda_\oplus$*-context* $\mathcal{C}_n$ *as follows:*

$$\mathcal{C}_n = (\lambda x.\bigwedge^n(xI, \dots, xI))(\lambda x.[\cdot]).$$

The behavior of this context on a program M *is as follows: first it produces* n *copies of* M*, then it evaluates successively all those copies of* M*, and stops only if all of them terminate. As a consequence:*

$$\forall \text{ program } M, \quad Obs(\mathcal{C}_n[M]) = Obs(M)^n \to_{n\to\infty} T^1(Obs(M)).$$

It means that $(\mathcal{C}_n)_{n\in\mathbb{N}}$ *is an approximation family for the threshold test* T^1 *on the whole interval* $[0,1]$*. As a consequence, for every program* M, N*, with* $Obs(M) = 1$*,* $Obs(N) < 1$*, it holds that* $\delta^{ctx}_{\Lambda_\oplus}(M,N) = 1$*.*

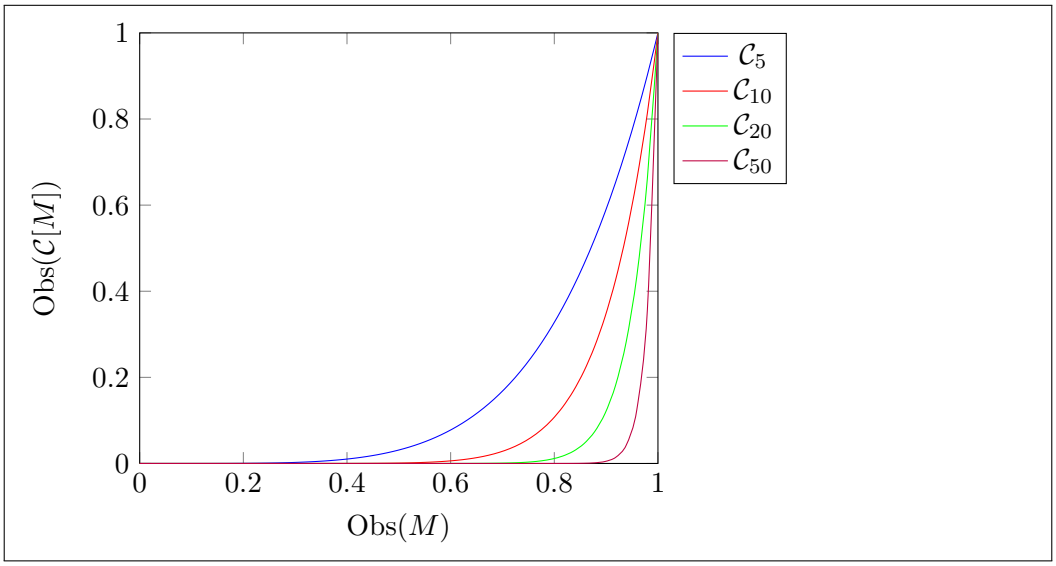


Figure 6.2: Approximation Contexts $(\mathcal{C}_n)_{n\in\mathbb{N}}$ in $\Lambda_\oplus$ for T^1.

As soon as we are able to approximate this way every threshold function by a family of contexts, we can show that the observational distance trivializes. We state this formally in the following lemma.

Lemma 6.1.1 *Let* $\mathcal{L}_\oplus$ *be any probabilistic observable programming language. If for every* $\alpha \in (0,1)$ *there exists an approximation of* T^α *by universal* $\mathcal{L}_\oplus$*-contexts on* $[0,1] \setminus \{\alpha\}$*, then the observational distance on* $\mathcal{L}_\oplus$ *trivializes, i.e.:*

$$\forall M, N \text{ comparable programs } \in \mathcal{L}_\oplus, \quad \delta^{ctx}_{\mathcal{L}_\oplus}(M,N) \in \{0,1\}.$$

Proof. We give the proof for the untyped case, but the proof when $\mathcal{L}_\oplus$ is a *typed* probabilistic language is similar. Let M, N be two programs in $\mathcal{L}_\oplus$. If M and N are observationally equivalent in $\mathcal{L}_\oplus$, then $\delta^{\text{ctx}}_{\mathcal{L}_\oplus}(M,N) = 0$, and so the result holds. We suppose now that M and N are not observationally equivalent: it means that there exists a $\mathcal{L}_\oplus$ context $\mathcal{C}$ such that $\text{Obs}(\mathcal{C}[M]) \neq \text{Obs}(\mathcal{C}[N])$. We now fix $\epsilon > 0$, and show that $1 - \delta^{\text{ctx}}_{\mathcal{L}_\oplus}(M,N) \leq \epsilon$. We denote $t_0 := \text{Obs}(\mathcal{C}[M])$, and $t_1 := \text{Obs}(\mathcal{C}[N])$. Suppose for instance that $t_0 < t_1$. Then we take $\alpha \in (t_0, t_1)$. By hypothesis, there exists a sequence of universal $\mathcal{L}_\oplus$ contexts $(\mathcal{D}_n)_{n\in\mathbb{N}}$ that approximates T^α. Looking at Definition 6.1.5, we see that it implies:

$$\lim_{n\to\infty} \text{Obs}(\mathcal{D}_n[\mathcal{C}[M]]) = 0 \quad \text{and} \quad \lim_{n\to\infty} \text{Obs}(\mathcal{D}_n[\mathcal{C}[N]]) = 1$$

As a consequence, there exists $n \in \mathbb{N}$ such that both:

$$\text{Obs}(\mathcal{D}_n[\mathcal{C}[M]]) \leq \frac{\epsilon}{2} \quad \text{and} \quad \text{Obs}(\mathcal{D}_n[\mathcal{C}[N]]) \geq 1 - \frac{\epsilon}{2},$$

and from there we deduce that indeed $1 - \delta^{\text{ctx}}_{\mathcal{L}_\oplus}(M,N) \leq \epsilon$. □

We now add a supplementary layer to the condition given in Lemma 6.1.1 that guarantees trivialization. Indeed, in some cases it is too complicated to explicitly write the family of contexts approximating the threshold function itself, but we are able to write a sequence of functions that tends towards it, and such that every function in it sequence can be approximated by contexts. We precise this idea in Lemma 6.1.2 below.

Lemma 6.1.2 *Let $\mathcal{L}_{\oplus}$ be a probabilistic observable programming language. If for every $\alpha \in (0,1)$ there exists a sequence of real function $f_n : [0,1] \to [0,1]$ such that:*

- *for every $t \in [0,1] \setminus \{\alpha\}$, $\lim_{n\to\infty} f_n(t) = T^{\alpha}(t)$;*
- *and for every $n \in \mathbb{N}$, there exists a family $(\mathcal{C}_m^n)_{m\in\mathbb{N}}$ of universal contexts that approximates f_n on $[0,1]$,*

then the observational distance on $\mathcal{L}_{\oplus}$ trivializes.

In the case where the language considered is $\Lambda_{\oplus}$—with the CBN or the CBV operational semantics—it is not possible to approximate all the T^{α} by $\Lambda_{\oplus}$ contexts: indeed the observational distance on $\Lambda_{\oplus}$ doesn't trivialize since for instance $\delta^{\mathrm{ctx}}_{\Lambda_{\oplus}}(\Omega, (I \oplus \Omega)) = \frac{1}{2}$—the formal proof can be done using the coinductive tools we will develop in Chapter 8. It also means that there is no possible ways to approximate T^0 in $\Lambda_{\oplus}$. We are now going to explore two paths that lead to trivialization: in the first one, we enhances the expressive power of contexts by adding a parallel convergence tester operator, while the second consists in enforcing termination for *all* programs. We will illustrate these two approaches respectively on the languages $\mathbb{T}_{\oplus}$ and $\Lambda_{\oplus}^{\parallel}$, by showing that in these two cases, we are able to approximate the threshold functions, and that consequently the observational distance trivializes.

6.2 Trivialization in $\Lambda_{\oplus}^{\parallel}$

We work here with the language $\Lambda_{\oplus}^{\parallel}$, which is strictly more expressive that $\Lambda_{\oplus}$, since it has a parallel convergence tester operator: $[M \parallel N]$ returns I if *at least* one of the two programs M and N terminates, and doesn't terminate otherwise. Recall that we have introduced and studied the language $\Lambda_{\oplus}^{\parallel}$ in Section 5.2.5 of Chapter 5. To be consistent with our development there, we keep the same CBV operational semantics—but our trivialization result would also hold for a CBN strategy. Our goal here is to show that all threshold functions T^{α} can be approximated by $\Lambda_{\oplus}^{\parallel}$ contexts, which will then allow us to obtain trivialization of the observational distance on $\Lambda_{\oplus}^{\parallel}$ by using Lemma 6.1.1.

Recall from Example 6.1.1 that we can approximate T^1 in $\Lambda_{\oplus}$; we can see that the same reasoning still holds in $\Lambda_{\oplus}^{\parallel}$. In contrast, we are not able to approximate T^0 in $\Lambda_{\oplus}$. Here, we first illustrate how the presence of the parallel convergence tester enhances the discriminating power of contexts by highlighting that T^0 can be approximated by a family of $\Lambda_{\oplus}^{\parallel}$ contexts.

Example 6.2.1 *In the same way as we defined n-ary conjunction operator in Example 6.1.1, we now define a* n-ary disjunction operator *in $\Lambda_{\oplus}^{\parallel}$.*

$$\bigvee^{n}(M_1, \dots M_n) = [M_1 \parallel [M_2 \parallel \dots]]\,;$$

The term $\bigvee^{n}(M_1, \dots, M_n)$ behaves as a n-ary disjunction: if $M_1, \dots, M_n$ are programs, it terminates if at least one *of the M_i terminates. We now build the family of $\Lambda_{\oplus}^{\parallel}$ universal*

contexts designed to approximate the threshold function T^0. *For every* $n \in \mathbb{N}$, *we build* $\mathcal{C}_n$ *as:*

$$\mathcal{C}_n = (\lambda x.\bigvee^n((xI),\dots,(xI)))(\lambda y.[\cdot]).$$

Essentially, $\mathcal{C}_n$ *makes* n *copies of its argument, and then converges towards* I *if* at least *one of these copies itself terminates. As a consequence,* $Obs(\mathcal{C}_n[M]) = 1-(1-Obs(M))^n$. *Thus the family* $(\mathcal{C}_n)_{n\in\mathbb{N}}$ *is an approximation by contexts of the threshold function* T^0 *on* $[0,1]$, *see Figure 6.3.*

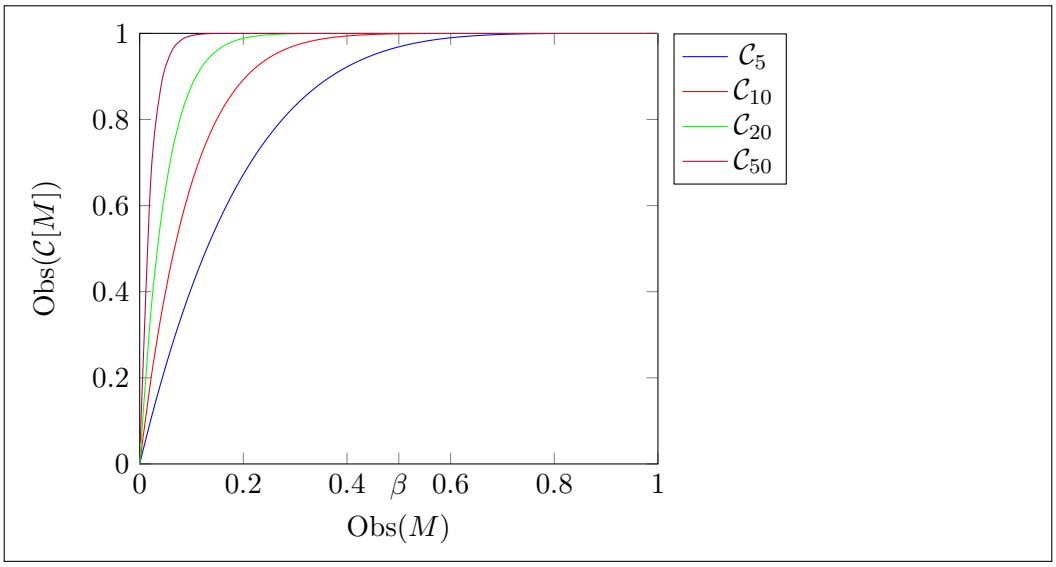


Figure 6.3: Approximation family $(\mathcal{C}_n)_{n\in\mathbb{N}}$ in $\Lambda^{\parallel}_{\oplus}$ for T^0 with $n \in \{5, 10, 20, 50\}$.

Observe that from Example 6.2.1, we can in particular derive that $\delta^{\mathrm{ctx}}_{\Lambda^{\parallel}_{\oplus}}(\Omega, \Omega \oplus I) = 1$. Our goal now is to show the existence, for every $\alpha \in (0,1)$, of an approximation by contexts of T^α on $[0,1] \setminus \{\alpha\}$. From there, we will be able to apply Lemma 6.1.1, and thus to show that $\delta^{\mathrm{ctx}}_{\Lambda^{\parallel}_{\oplus}}$ trivializes. To this purpose, we are going to *mix* the conjunction and the disjunction operators—defined respectively in Example 6.1.1 and 6.2.1—in order to obtain a new kind of contexts that we will use later to build threshold approximations.

Definition 6.2.1 *We call* mixed amplification context of conjunction order n and disjunction order m, *the universal context* $(\Diamond^m_n)_{n,m\in\mathbb{N}}$ *defined as follows:*

$$\Diamond^m_n = \lambda y.\left(\bigwedge^n(\bigvee^m((xI),\dots,(xI)),\dots,\bigvee^m((xI),\dots,(xI)))\right)(\lambda y.[\cdot])$$

Observe that the contexts $\Diamond^m_n$ compute m-ary conjunctions of n-ary disjunctions. If we look at the termination probability, we see that:

$$\mathrm{Obs}(\Diamond^m_n[M]) = (1-\mathrm{Obs}(M)^m)^n.$$

Using these mixed amplifications contexts, we are now able to approximate *threshold tests*. More precisely, for every $\alpha \in (0,1)$, we need to associate to every index $n \in \mathbb{N}$ a mixed amplification context of the form $\Diamond^{r(n,\alpha)}_n$. We do this construction in Definition 6.2.2 below, and we will then show that the family of contexts $(\mathcal{C}^\alpha_n)_{n\in\mathbb{N}}$ obtained this way is indeed an approximation by contexts of T^α on $[0,1] \setminus \{\alpha\}$.

Definition 6.2.2 *Let $\alpha \in (0,1)$, and $n \in \mathbb{N}$. We build the* n-th context approximation of T^α *as the $\Lambda^{\|}_{\oplus}$ context $\mathcal{C}^\alpha_n := \Diamond_n^{r(n,\alpha)}$, where we take:*

$$r(n,\alpha) = \left\lceil \left(\frac{1}{1-\alpha}\right)^n \right\rceil.$$

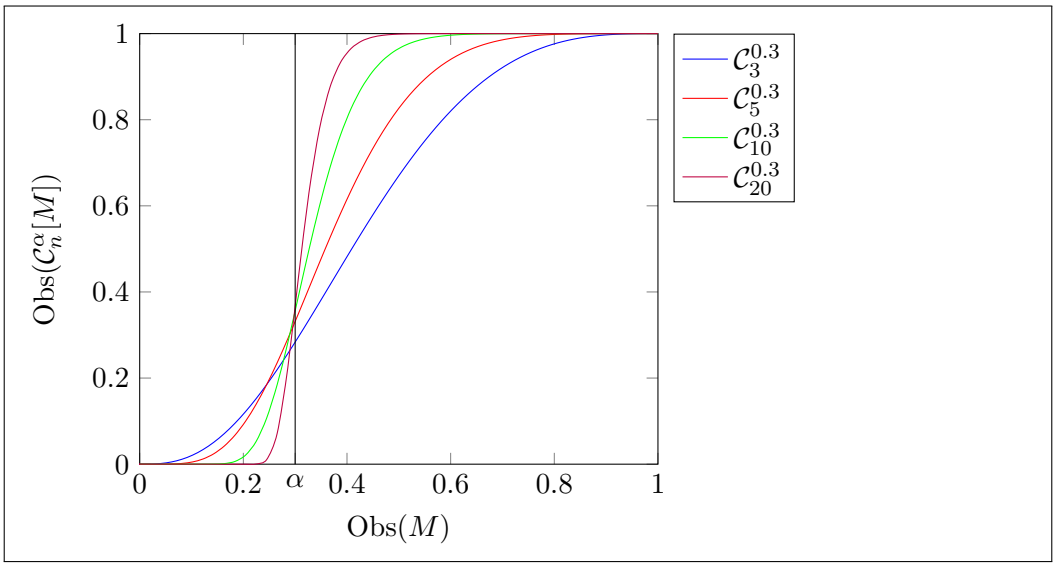


Figure 6.4: Behavior of the contexts $\mathcal{C}^\alpha_n$ for $n \in \{3,5,10,20\}$.

We illustrate in Figure 6.4 the action of $\mathcal{C}^\alpha_n$: we can see on an example that indeed the greater the value of n is, the closer the n-th approximation function is to T^α. We need now to prove that this is always the case, i.e. that the $\mathcal{C}^\alpha_n$ are an approximation by contexts of the threshold tests. To keep the notations readable, we introduce functions $f_{n,\alpha} : [0,1] \to [0,1]$ defined by $f_{n,\alpha}(x) = (1-(1-x)^n)^{r(n,\iota)}$, i.e. such that:

$$\forall M, \quad \mathrm{Obs}(\mathcal{C}^\alpha_n[M]) = f_{n,\alpha}(\mathrm{Obs}(M)).$$

Accordingly, we can reformulate our goal as follows: we want to show that for every $\alpha \in (0,1)$, and $t \in [0,1] \setminus \{\alpha\}$, $\lim_{n\to\infty} f_{n,\alpha}(t) = T^\alpha(t)$. The proof will use tools from real analysis, essentially the Bounded Convergence Theorem, that we recall below. The statement and the proof can for instance be found in [100]. It is a particular instance of a number of theorems in classical analysis, that give conditions to permute limit and integration.

Theorem 6.2.1 (Bounded Convergence Theorem) *If k_n is a sequence of uniformly bounded real-valued measurable functions which converges pointwise on a bounded measure space (S,Σ,μ) to a function k, then the limit k is an integrable function and: $\int_S \lim_{n\to\infty} k_n = \lim_{n\to\infty} \int_S k_n$.*

The bounded convergence theorem is relevant here, because we can express the approximations functions $f_{n,\alpha}$ using an integral. In order to simplify the proof, we will work with the *reverse approximation functions* $g_{n,\alpha} : \mathbb{R} \to \mathbb{R}$ defined as $g_{n,\alpha}(x) := f_{n,1-\alpha}(1-x)$. We can express the $g_{n,\alpha}$ as an integral: indeed, the function g is derivable, and as a consequence,

for every $x \in (0,1)$:

$$g_{n,\alpha}(x) = \begin{cases} \int_0^x g'_{n,\alpha}(t)dt + 1 & \text{if } x < \alpha \\ -\int_x^1 g'_{n,\alpha}(t)dt & \text{if } x > \alpha \end{cases} \tag{6.1}$$

We are now going to fix α, and use the Bounded Convergence Theorem by taking the k_n as $g'_{n,\alpha}$.

Lemma 6.2.2 *For every $\alpha \in (0,1)$, it holds that:*

$$\lim_{n\to\infty} \int_0^x g'_{n,\alpha}(t)dt = 0 \text{ when } x < \alpha$$
$$\lim_{n\to\infty} \int_x^1 g'_{n,\alpha}(t)dt = 0 \text{ when } x > \alpha.$$

Proof. We first fix an $\alpha \in [0,1]$, $x_- \in [0,\alpha[$, $x_+ \in]\alpha, 1]$. Our goal now is to show that both $(g'_{n,\alpha} : [0,x_-] \to \mathbb{R})_{n\in\mathbb{N}}$ and $(g'_{n,\alpha} : [x_+,1] \to \mathbb{R})_{n\in\mathbb{N}}$ are indeed two sequences of uniformly—i.e. by a constant that does not depend from n—bounded real-valued measurable functions which converge pointwise. Observe first that the measurability assumption obviously holds—we endow as usual both $[0,x_-]$ and $[x_+,1]$ with the Lebesgue measure—since the $g'_{n,\alpha}$ are continuous functions. Let us now look for uniform bounds. First, we can give a closed form for the function $g'_{n,\alpha}$: indeed it holds that for every $t \in [0,1]$:

$$g'_{n,\alpha}(t) = -r(n,1-\alpha)\rceil \cdot n \cdot (1-t^n)^{r(n,1-\alpha)-1} \cdot t^{n-1}$$

Recall that $r(n,1-\alpha) = \lceil (\frac{1}{\alpha})^n \rceil$, and that moreover $0 < \alpha < 1$. As a consequence, there exists N, such that $\forall n \geq N$, $r(n,1-\alpha) \leq \left(\frac{1}{\alpha}\right)^n + 1 \leq \left(\frac{1}{\alpha}\right)^{n+1}$. From there—and since $0 \leq (1-t^n) \leq 1$—we can deduce the following bound on the absolute value of the $g'_{n,\alpha}$:

$$\forall t \in [0,1],\ |g'_{n,\alpha}(t)| \leq \frac{1}{\alpha^2} \cdot \left(\frac{t}{\alpha}\right)^{n-1} \cdot n \cdot (1-t^n)^{(\frac{1}{\alpha})^n - 1} \tag{6.2}$$

From there, we consider separately the domains $S_1 = [0,x_-]$ and $S_2 = [x_+,1]$.

- We first consider the sequence $g'_{n,\alpha} : S_1 \to \mathbb{R}$. From (6.2), we see that for every $t \in S_1$, $\mid g'_{n,\alpha}(t) \mid \leq K \cdot n \cdot \left(\frac{t}{\alpha}\right)^{n-1}$, where K is a constant. We need now to use the following well-know result of asymptotic hierarchy.

  $$\lim_{n\to\infty} n^m \cdot e^{-n\cdot b} = 0 \text{ for every } m \in \mathbb{N}, b > 0. \tag{6.3}$$

  Observe that (6.3) can also be retrieved by seeing that $\lim_{y\to\infty} (\ln(y) - \alpha \cdot y) = -\infty$ for every $\alpha > 0$. We can now use (6.3) to see that $g'_{n,\alpha} : S_1 \to \mathbb{R}$ converge pointwise to the 0-function. We're now going to show that this sequence of functions admits a *uniform* bound.

  $$\begin{aligned} \mid g'_{n,\alpha}(t) \mid &\leq K \cdot n \cdot \left(\frac{x_-}{\alpha}\right)^{n-1} && \text{since } t \leq x_-. \\ &\leq K \cdot \sup_{n\in\mathbb{N}} (n \cdot \left(\frac{x_-}{\alpha}\right)^{n-1}) < \infty && \text{since (6.3) and } x_- < \alpha \end{aligned}$$

  Thus we have shown that the sequence $g'_{\alpha,n} : S_1 \to \mathbb{R}$ is a sequence of uniformly bounded real-valued measurable functions that converges pointwise toward the 0 function. Consequently, we can apply the Bounded Convergence Theorem, and we obtain that $\lim_{n\to\infty} \int_0^{x_-} g'_{n,\alpha}(t)dt = 0$.

- We now consider the sequence of functions $g'_{n,\alpha} : S_2 \to \mathbb{R}$. We start from (6.2), and we use the fact that since $t \in [0,1]$, $\ln(1-t^n) \leq -t^n$. We moreover assume that we take $n \geq N_0$, where N_0 is the first natural number with $(\frac{1}{\alpha})^n - 1$—it eventually holds, since $0 < \alpha < 1$. We obtain that for every $t \in S_2$:

$$|g'_{n,\alpha}(t)| \leq \frac{1}{\alpha^2} \cdot \left(\frac{t}{\alpha}\right)^{n-1} \cdot n \cdot e^{\left(\left(\frac{1}{\alpha}\right)^n - 1\right)\cdot(-t^n)} \tag{6.4}$$

$$\leq \frac{1}{\alpha^2} \cdot \left(\frac{t}{\alpha}\right)^{n-1} \cdot n \cdot e^{-\left(\frac{t}{\alpha}\right)^n} \cdot e^{t^n} \leq \frac{1}{\alpha^2} \cdot \left(\frac{1}{\alpha}\right)^{n-1} \cdot n \cdot e^{-\left(\frac{x_+}{\alpha}\right)} \cdot e \tag{6.5}$$

  It is another classical result from the asymptotic hierarchy—that can be retrieved from 6.3—that for every $a, b > 1$, it holds that:

$$\lim_{m\to\infty} a^m \cdot m \cdot exp(-b^m) = 0.$$

  From this equation, we can deduce that:

$$\lim_{n\to\infty} \frac{1}{\alpha^2} \cdot \left(\frac{1}{\alpha}\right)^{n-1} \cdot n \cdot e^{-\left(\frac{x_+}{\alpha}\right)^n} \cdot e = 0 \tag{6.6}$$

  Now, please observe that combining (6.5) and (6.6) gives us both the pointwise limit and the uniform bound (since $g_{n,\alpha}(t)$ is bounded by a sequence that doesn't depends on t, and has a finite limit when n goes towards infinity). So we can apply the bounded convergence theorem, and we obtain that $\lim_{n\to\infty} \int_{x_+}^1 g'_{n,\alpha}(t)dt = 0$, and this concludes the proof of Lemma 6.2.2.

□

Lemma 6.2.3 *Let $n \in \mathbb{N}$, and $\alpha \in (0,1)$. Then for every $x \in (0,1) \setminus \{\alpha\}$ it holds that* $\lim_{n\to\infty} f_{n,\alpha}(x) = T^\alpha(x)$.

Proof. By combining Equation (6.1) and Lemma 6.2.2, we obtain that:

$$\lim_{n\to+\infty} g_{n,\alpha}(x) = \begin{cases} 1 & \text{if } x < \alpha \\ 0 & \text{if } x > \alpha \end{cases} \tag{6.7}$$

Since $g_{n,\alpha}(x) = f_{n,1-\alpha}(1-x)$, it concludes the proof. □

Lemma 6.2.3 means that for every $\alpha \in (0,1)$, the family of $\Lambda^{\parallel}_{\oplus}$ contexts $(\mathcal{C}^\alpha_n)_{n\in\mathbb{N}}$ is indeed an approximation by contexts of T^α on $[0,1] \setminus \{\alpha\}$—in the sense formally stated in Definition 6.1.5. As a consequence, we can now apply Lemma 6.1.1, and we obtain trivialization of the observational distance on $\Lambda^{\parallel}_{\oplus}$, as stated below.

Theorem 6.2.4 $\delta^{ctx}_{\Lambda^{\parallel}_{\oplus}}$ *trivializes.*

6.3 Trivialization in $\mathbb{T}_{\oplus}$.

We now consider the language $\mathbb{T}_{\oplus}$, introduced by Breuvart, Dal Lago and Herrou [19], and that we have presented in Section 1.3.2 of Chapter 1. Recall that this language is AST, i.e. that for every term M, $|[\![M]\!]| = 1$.

Since $\mathbb{T}_\oplus$ is a typed language, the relevant notion of observational distance is the one we gave in Definition 3.1.4 from Chapter 3 for typed observable programming languages. Recall that, following our formalism for a generic typed observable language, we need to fix a set of *observable types*: we choose the natural number type N, and if M is a program of type N, we take $\mathrm{Obs}(M) = [\![M]\!](\underline{0})$, i.e. the probability that the outcome of the execution of M will be 0. Observe that since this language is AST, we are forced to take a *non termination-based* notion of observation.

$$(\delta^{\mathrm{ctx}}_{\mathbb{T}_\oplus})_\sigma(M,N) = \sup_{\mathcal{C} \text{ s.t. } [\cdot]:\sigma\vdash\mathcal{C}:\mathrm{N}} |[\![\mathcal{C}[M]]\!](\underline{0}) - [\![\mathcal{C}[N]]\!](\underline{0})|$$

In the following, our goal is to show trivialization of context distance on $\mathbb{T}_\oplus$, as defined above. To do so, we will use Lemma 6.1.2 based on a double-layered approximation of the threshold tests T^α: the first step consists in building for every $\alpha \in (0,1)$ a sequence of real functions that converge pointwise to T^α, while in the second step we need to exhibit an approximation by contexts for each of those functions. Our approach to the second step is based on Bernstein's constructive proof [13] of the Stone-Weierstrass theorem which states that every *continuous* function $[0,1] \to \mathbb{R}$ can be *uniformly* approximated by polynomials. Bernstein's proof consists in building, for any fixed continuous function $f : [0,1] \to \mathbb{R}$, a *concrete* sequence of polynomials that indeed uniformly converges towards f. More precisely, our overall approach is as follows: we first construct contexts that are able to simulate Bernstein's polynomials for some class of functions f, and then we exhibit in this class of functions a particular sequence f^α_n that *approximates* the Threshold test T^α.

6.3.1 Simulating Bernstein's Polynomials.

Bernstein's construction is as follows: the *n-th Bernstein's polynomial with respect to f* is defined as:

$$P^f_n(x) = \sum\nolimits_{0\le k\le n} f\left(\frac{k}{n}\right)\cdot B^n_k(x), \text{ where } B^n_k(x) = \binom{n}{k}\cdot x^k\cdot(1-x)^{n-k}. \tag{6.8}$$

We illustrate on Figure 6.5 how Bernstein's polynomials approximate a particular continuous function f built as follows: it is zero before some real number α, 1 after some real number $\beta > \alpha$, and an affine function between α and β. Our goal is to show that the contexts in $\mathbb{T}_\oplus$ are able to *simulate* the Bernstein's Polynomials for some continuous functions that globally looks like the f of Figure 6.5, i.e. that verifies the following specification: it must be a *partial test separating α and β*, i.e. that returns 0 if $x \le \alpha$, and 1 if $x \ge \beta$. Observe that this test is partial, in the sense where we do not specify what happens when $x \in (\alpha,\beta)$.

In the following, even if our base data are natural numbers, we will consider them as booleans, where $\underline{0}$ is the encoding of **true**, and any other $\underline{n}$ the encoding of **false**.

We are going to build now step by step the contexts that we will use to simulate Bernstein's polynomials. The first step consists in a context $\mathcal{C}$ such that for any program $M_1,\dots,M_n$, $\mathcal{C}[M_1,\dots,M_n]$ has the following behavior: it counts the number of its arguments that are equal to **true**, and then executes the corresponding M_i.

Definition 6.3.1 (Counting Contexts) *For every $n \in \mathbb{N}$, we define a $\mathbb{T}_\oplus$ context $\mathcal{C}$*

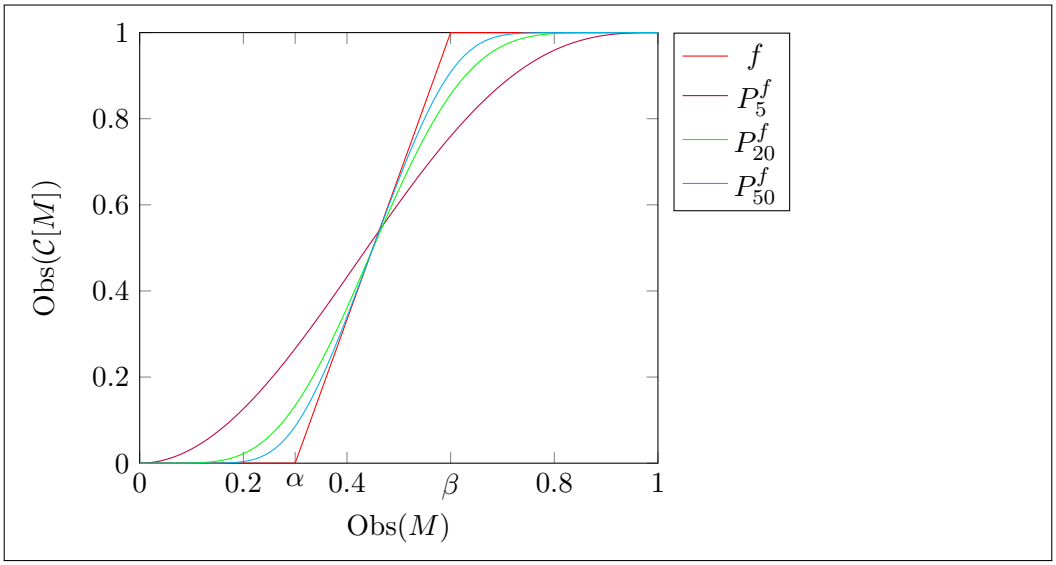


Figure 6.5: f and its approximations P_n^f for $n \in \{5, 20, 50\}$.

with $n+1$ holes, as follows:

$$\begin{aligned}\mathcal{C}[[\cdot]_0, \ldots, [\cdot]_n] = \lambda x_1.\lambda x_2. \ldots \lambda x_n.\\ \textit{let } i = \&_{0\leq i\leq n} x_i \textit{ in}\\ \textit{if } (i == 0) \textit{ then } [\cdot]_0\\ \textit{if } (i == 1) \textit{ then } [\cdot]_1\\ \vdots\\ \textit{if } (i == n) \textit{ then } [\cdot]_n.\end{aligned}$$

Remark that local variables definitions are *not* a primitive of $\mathbb{T}_\oplus$. However, we used them in Definition 6.3.1 as a notation for nested if-then-else procedure, to keep the definition above readable. For instance, when $n = 1$, the real context $\mathcal{C}$ is as follows:

$$\begin{aligned}\mathcal{C}[[\cdot]_0, [\cdot]_1] = \lambda x_1.\lambda x_2.\ \text{if } x_0 \text{ then } (\ \text{if } x_1 \text{ then } [\cdot]_2 \text{ else } [\cdot]_1)\\ \text{else } (\ \text{if } x_1 \text{ then } [\cdot]_1 \text{ else } [\cdot]_0).\end{aligned}$$

An important point guaranteed in the unfolded context is that during an execution each variable x_i is evaluated *exactly once*.

Observe that for every type σ, it holds that

$$[\cdot]_1 : \sigma, \ldots, [\cdot]_n : \sigma \vdash \mathcal{C}[[\cdot]_0, \ldots, [\cdot]_n] : \mathrm{N} \to \ldots \to \mathrm{N} \to \sigma.$$

We formalize now the operational semantics of $\mathcal{C}$.

Lemma 6.3.1 *We fix σ a type in $\mathbb{T}_\oplus$. Then for every n-uple $M_1, \ldots, M_n$ of programs of type N, and every $(n+1)$-uple $(N_0, \ldots, N_n)$ of programs of type σ, it holds that:*

$$[\![\mathcal{C}[M_1, \ldots, M_n]N_0, \ldots, N_n]\!] = \sum_{0\leq j\leq n} p_j \cdot [\![N_j]\!],$$

where for every j, the coefficient $p_j \in [0, 1]$ is as follows:

$$p_j = \sum_{\{k_1,\ldots,k_i\}\subseteq\{1,\ldots,n\}} \prod_{j\in\{k_1,\ldots,k_n\}} Obs(M_j) \prod_{j\in\{1,\ldots,n\}\setminus\{k_1,\ldots,k_n\}} (1 - Obs(M_j)).$$

Proof. Intuitively—and looking at $\mathcal{C}$ as given in the Definition 6.3.1—it is because the probability that the local variable i is equal to j is exactly p_j: indeed, it is the probability that exactly j among the programs M_j are true, and that the remaining $n-j$ are false. Formally, we have to show the result by unfolding the if-then-else procedure, and then doing the proof by induction on n. □

Actually, for our purpose, we will only ever fills the counting context $\mathcal{C}$ with n identical copies of some program M. In this setting, we can give a more compact formulation of Lemma 6.3.1, that highlight the link with Bernstein's polynomials.

Lemma 6.3.2 *For every M program of type* N*, and every $(n+1)$-uple $(N_0, \ldots, N_n)$ of programs of same type σ, it holds that:*

$$[\![\mathcal{C}[M, \ldots, M]N_0, \ldots, N_n]\!] = \sum_{0 \leq j \leq n} B_j^n(Obs(M)) \cdot [\![N_j]\!],$$

where B_j^n are the polynomials defined in (6.8)*—as a building block for Bernstein's polynomials.*

The next step is to look at the N_j programs. First, since we want the program $\mathcal{C}[M, \ldots, M][N_0, \ldots, N_n]$ to be observable, it needs to be of N type, which means that the N_j themselves must be all natural numbers. We now use Lemma 6.3.2 in order to compute the observable of the program $\mathcal{C}[M, \ldots, M][N_0, \ldots, N_0]$.

Lemma 6.3.3 *Let M be a program of type* N*, and $(N_0, \ldots, N_n)$ of $n+1$ programs of type* N*. Then:*

$$Obs(\mathcal{C}[M, \ldots, M]N_0, \ldots, N_n) = \sum_{0 \leq j \leq n} B_j^n(Obs(M)) \cdot Obs(N_j),$$

Proof. It is a direct consequence of Lemma 6.3.2, and from the definition of observable for $\mathbb{T}_\oplus$. □

Looking now at Lemma 6.3.3 and at the shape of Bernstein's polynomials in (6.8), we see that the last step before being able to *simulates* —in the sense of Definition 6.1.3—Bernstein's polynomials P_n^f, is to exhibit $\mathbb{T}_\oplus$ programs N_j such that $\mathrm{Obs}(N_j) = f(\frac{j}{n})$. A natural candidate is given by $N_j = \mathbf{false} \oplus^{f(\frac{j}{n})} \mathbf{true}$. However, recall that this term is only defined when $f(\frac{j}{n})$ is a dyadic number—since the construction in $\Lambda_\oplus$ of a p-biased probabilistic choice in Example 7.4.1 of Chapter 7 is valid only when $p \in \mathbb{D}$. Observe that it is not guaranteed for instance for the function f represented in Figure 6.5.

Proposition 6.3.4 *Let $f : [0,1] \to [0,1]$ be a continuous function, such that $f(\mathbb{Q}) \subseteq \mathbb{D}$. Then there exists a $\mathbb{T}_\oplus$ context that simulates P_n^f.*

We call *reachable functions*, and we denote by $\mathcal{R}$ the class of such continuous functions $[0,1] \to [0,1]$ such that $f(\mathbb{Q}) \subseteq \mathbb{D}$.

6.3.2 Approximating T^α with reachable functions

We can first remark that the existence of *one* function in this class can be established by looking at Minkowski's *question mark function* ?, which is a continuous function $[0,1] \to [0,1]$ that sends $\mathbb{Q}$ into $\mathbb{D}$. We don't recall here the concrete definition of the question-mark function, but the interested reader can find a detailed description of its definition and properties for instance in [121].

Using this question mark function, we now define a sequence of *reachable functions* that converge pointwise towards T^α on $[0,1] \setminus \{\alpha\}$.

Definition 6.3.2 *Let $\alpha \in (0,1)$. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, there exists two sequences $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in\mathbb{N}}$ of elements in $\mathbb{Q}$ such that for every $n \in \mathbb{N}$, $a_n < \alpha < b_n$, and moreover $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n = \alpha$. We define the function $r_n^\alpha : [0,1] \to [0,1]$ as:*

$$r_n^\alpha(x) = \begin{cases} 0 & \text{if } x \leq a_n \\ 1 & \text{if } x \geq b_n \\ ?(\frac{x-a_n}{b_n-a_n}) & \text{otherwise.} \end{cases}$$

Observe that for every $n \in \mathbb{N}$, the function r_n^α is a reachable function, and moreover that the sequence r_n^α converges pointwise towards T^α. Using this result, and Proposition 6.3.4—which says that the Bernstein's polynomials of any reachable functions can be simulated by $\mathbb{T}_\oplus$ contexts—we see that the requirements of Lemma 6.1.2 are met, and so the observational distance trivializes on $\mathbb{T}_\oplus$.

Theorem 6.3.5 *The context distance $\delta^{ctx}_{\mathbb{T}_\oplus}$ trivializes.*

Proof. We see that the requirements of Lemma 6.1.2 hold:

- for every $t \in [0,1] \setminus \{\alpha\}$, $\lim_{n\to\infty} r_n^\alpha(t) = T^\alpha(t)$;
- for every $n \in \mathbb{N}$, there exists—by Proposition 6.3.4 a family of contexts $\mathcal{C}_n^m$ that simulates the Bernstein's polynomials of r_n. From there, Bernstein's theorem tells us that the Bernstein's polynomials of r_n converge towards r_n, and consequently that the $\mathcal{C}_n^m$ are a family of approximations for r_n.

As a consequence, Lemma 6.1.2 tells us that the observational distance trivializes. □

Chapter 7

$\Lambda^!_\oplus$: A λ-calculus with Explicit Copying

As we have seen in Chapter 6, the behaviour of the context distance depends strongly on the copying abilities of the language. For this reason, we will use in this thesis a variant of probabilistic λ-calculus, that we call *Linear Probabilistic λ-Calculus* ($\Lambda^!_\oplus$), designed to give us more control on the copying mechanisms: we are able to specify in $\Lambda^!_\oplus$ which function arguments can be duplicated, and which one cannot. The present chapter is devoted to the presentation of $\Lambda^!_\oplus$.

7.1 The Language $\Lambda^!_\oplus$.

7.1.1 Simpson Surface Calculus

In [114], Simpson introduced a deterministic λ-calculus called *Surface Calculus*, which we denote $\Lambda^!$. We give here a very short overview of this programming language, since we designed $\Lambda^!_\oplus$ as a probabilistic extension of it. The language is based on a distinction *in the syntax* between those subterms that can be copied, and those that cannot. To that end, two λ-abstractions coexist in the syntax: a *linear* one $\lambda x.M$, where we ask x to be used *exactly once* in M, and a *non-linear* (or *exponential*) one $\lambda !x.M$, where x may appear *several times* in M. In the syntax, there is also a *box operator* !: the term $!M$ represents the term M endowed with the ability to be duplicated infinitely often. To an exponential λ-abstraction $\lambda !x.M$, we can only pass as argument a box $!N$. In [114], two different ways of enforcing this linear/ non-linear policy are presented: the definition of criterion on variables assuring that terms are well-formed, and a type system with !-types.

To fix ideas, we give here the syntax of $\Lambda^!$:

$$M, N ::= x \;\Big|\; MN \;\Big|\; \lambda x.M \;\Big|\; \lambda !x.M \;\Big|\; !M \qquad \text{where } x \in \mathbf{V}.$$

It is endowed in [114] by an operational semantics. Since there are two different λ-abstractions, there are also two different kinds of β-redexes:

$$(\lambda x.M)N \to_\beta M\{N/x\}$$
$$(\lambda !x.M)!N \to_\beta M\{N/x\}$$

Observe that, as a consequence, only functions of the form $\lambda !x.N$ can open a box, and thus destroy it.

Reduction in [114] is *strong*, i.e. it goes under the λ-abstractions, and *surface*, in the sense where it doesn't go inside a box $!M$. The idea is that boxes can be copied, but the content of the box cannot be executed until the box is opened. It is shown in [114] that this reduction is *confluent*, hence CBV and CBN evaluation strategies are equivalent.

Surface Calculus is an *untyped* language, but it is inspired by the *explicitely* typed language Lily, introduced by Bierman, Pitts and Russo in [14]. We will inspire ourselves loosely of Lily type system when we will design a type system for $\Lambda^!_\oplus$. The main difference is that, since Lily has also a recursive operator *in the syntax*, the variant of system $\mathbb{F}$ with !-types given as type system for Lily in [14] gives an expressive enough calculus, in particular with infinite behaviour. For $\Lambda^!_\oplus$, we will need to incorporate also recursive types.

7.1.2 The Language $\Lambda^!_\oplus$.

While there is no computationnal effects in Surface Calculus, managing copying through boxes interacts well with probabilistic execution, since delaying the opening of the box allows us also to delay the moment when we do the probabilistic choice. Here, we are going to present $\Lambda^!_\oplus$ as a probabilistic extension of $\Lambda^!$.

Syntax of $\Lambda^!_\oplus$.

The syntax of $\Lambda^!_\oplus$ is an extension of both $\Lambda_\oplus$ and Surface Calculus. Together with λ-abstractions, applications and the operator $\oplus$ for fair probabilistic choice, we take the same operators as in Surface Calculus to manage copying: the operator $!$, which is the constructor for boxes, and the *exponential λ-abstraction* $\lambda!x.\cdot$, which is the destructor for boxes—functions of the form $\lambda!x.M$ accept only boxes as argument, and destroy the box passed to them in the application process. Therefore, as in Surface Calculus, we have *two* distinct λ-abstractions in the syntax. However, we modify here slightly the meaning of $\lambda x.M$: we do not want the variable x to appear in M *exactly* once anymore, but *at most once*. It means that the resource x in M is now *affine* instead of being linear. We will enforce this by a type system, that we will present in Section 7.2.

Definition 7.1.1 *We assume a countable set of variables $\boldsymbol{V}$. The set of* terms *of $\Lambda^!_\oplus$ is defined by the following grammar:*

$$M, N, L \ldots \in \Lambda^!_\oplus ::= x \;\Big|\; MN \;\Big|\; \lambda x.M \;\Big|\; \lambda!x.M \;\Big|\; !M \;\Big|\; M \oplus N$$

$$\text{where } x \in \boldsymbol{V}.$$

As in Chapter 1, we call *programs* those terms in $\Lambda^!_\oplus$ that are closed, i.e have no free variables.

Operationnal Semantics of $\Lambda^!_\oplus$.

We define now an operationnal semantics for $\Lambda^!_\oplus$ programs, following the approach presented in Section 1.2 for $\Lambda_\oplus$: we first define the *one-step reduction relation* between programs and finite sequences of programs modelizing one execution step, then we use it to associate to a program M a family of distribution over programs–the *approximation semantics* of M—and finally we take the *semantics* of M as the supremum of its approximation semantics.

As in [114], the reduction in $\Lambda^!_\oplus$ is going to be surface—we do not reduce inside boxes. However it is also going to be weak—we don't reduce under λ-abstractions either, in order

to be in line with the presentation of $\Lambda_\oplus$ in [33], as well as with the work on coinductively defined equivalences for $\Lambda_\oplus$ done in [32]. Unfortunately, this choice of weak reduction stop us to have confluence, therefore we have to choose a non-ambiguous evaluation strategy. We choose the call-by-value paradigm: this semmingly arbitrary choice is justified by the fact that, as we will show later, we can embedd both CBN $\Lambda_\oplus$ and CBV $\Lambda_\oplus$ in $\Lambda^!_\oplus$ endowed with our CBV semantics.

The One-Step Reduction Relation for $\Lambda^!_\oplus$.

The one-step reduction we take now depends on the evaluation strategy we want: actually, as highlighted in Chapter 1 for a generic probabilistic calculus, the one-step reduction completely determines the language operational semantics. We are going to define it using the *reduction semantics* approach based on evaluation contexts, that we have presented in Chapter 1 for the probabilistic λ-calculus $\Lambda_\oplus$. Since we are in a call-by-value framework, the first step consists of defining the set of *values*—programs, that we want to be non-reducible under our reduction strategy. Since we want our reduction to be weak and surfaces, they are λ-abstraction, and boxes.

Definition 7.1.2 *We define the set $\mathcal{V}_{\Lambda^!_\oplus}$ of $\Lambda^!_\oplus$ values as follows:*

$$V \in \mathcal{V}_{\Lambda^!_\oplus} ::= \lambda x.M \;\Big|\; \lambda !x.M \;\Big|\; !M.$$

We need now to define what our redexes are, and to give rules specifying how to reduce them. We have three different kinds of redexes: the β-reduction of linear abstraction, the β-reduction of a !-abstraction and the fair probabilistic choice.

Definition 7.1.3 *We define the sets of $\Lambda^!_\oplus$-redexes $\mathscr{R}_{\Lambda^!_\oplus}$ as follows:*

$$R, S \in \mathscr{R}_{\Lambda^!_\oplus} ::= (\lambda x.M)V \;\Big|\; (\lambda !x.M)!N \;\Big|\; M \oplus N,$$

where $M, N, L \in \Lambda^!$, $V \in \mathcal{V}_{\Lambda^!_\oplus}$ and $x, y \in \boldsymbol{V}$. We define the reduction $\to_{\mathscr{R}_{\Lambda^!_\oplus}}$ for redexes, as a relation between elements in $\mathscr{R}_{\Lambda^!_\oplus}$ and sequences of programs in $\Lambda^!_\oplus$:

$$(\lambda x.M)V \to_{\mathscr{R}_{\Lambda^!_\oplus}} M\{V/x\} \qquad M \oplus N \to_{\mathscr{R}_{\Lambda^!_\oplus}} M, N$$
$$(\lambda !x.M)!N \to_{\mathscr{R}_{\Lambda^!_\oplus}} M\{N/x\}.$$

Observe that the two first rules are the same as for $\Lambda_\oplus$ in call-by-value, while the third one is the same as the opening of a box in Surface Calculus.

Following the reduction semantics approach, we need now to define the evaluations contexts: it specifies where in a term we are allowed to reduce redexes. As in $\Lambda_\oplus$, the execution of a program is done left-first, i.e. a function is evaluated before its argument. Since we are in call-by-value, the argument has to be evaluated too before reducing the redex. We sum up these considerations by giving the following grammar for evaluation contexts:

$$\mathcal{E}, \mathcal{F} ::= [\cdot] \;\Big|\; \mathcal{E}M \;\Big|\; V\mathcal{E} \qquad \text{with } M \in \Lambda^!, V \in \mathcal{V}_{\Lambda^!}.$$

Approximate Semantics for $\Lambda^!_\oplus$.

We then define the approximate operationnal semantics $M \Rightarrow \mathscr{D}$ by applying Definition 1.2.4 of Chapter 1, which has been given for a generic probabilistic language endowed by a reduction semantics. For any program M, it gives us a family of *approximation semantics*. We would like now be able to say that this family has a supremum. Recall that Lemma 1.2.1 of Chapter 1 tells us that it is the case, provided that the one step reduction relation is unambiguous.

Lemma 7.1.1 *The relation $\rightarrow$ induced by redexes, redexes rules and evaluation contexts is unambiguous.*

Proof. It is enough to see that for every M, there exists at most one $\mathcal{E}$, and R, such that $M = \mathcal{E}[R]$. □

As a consequence, we see using Lemma 1.2.1 that $\sup\{\mathscr{D} \mid M \Rightarrow \mathscr{D}\}$ is a directed set. Following Definition 1.2.6, we define for any closed term M:

$$[\![M]\!] = \sup\{\mathscr{D} \mid M \Rightarrow \mathscr{D}\}.$$

Example 7.1.1 *We adapt the non-terminating term given in Example 1.1.1 to our explicit copying setting: we need to replace the λ-abstractions by exponential λ-abstractions, since the variables are used several times. Hence, we define:*

$$\Omega = (\lambda!x.x(!x))(!\lambda!x.x(!x)).$$

We look now at the operational semantics of Ω:

$$\Omega \rightarrow_\beta \Omega$$

As a consequence, we see that the only approximate denotational semantics for Ω is given by the (Empty) rule of Definition 1.2.4: $\Omega \Rightarrow \emptyset$: it means that $[\![\Omega]\!] = \emptyset$.

We will see later that Ω can be typed in the type system for $\Lambda^!$ that we will give in Section 7.2.

Recall that we have also defined a recursion operator Y in Example 1.1.4, which has a shape similar as Ω. We are going to do here the same thing as we have done for Ω: modify it in order to obtain a program in $\Lambda^!$ respecting our policy on affine/exponential variables.

Example 7.1.2 *We are going to present here a variant of Y fixpoint combinator for call-by-name Λ, which we adapted to our affine/exponential policy, as well as to our evaluation strategy. We define $Y^!$ the* fixpoint combinator in $\Lambda^!_\oplus$ *as:*

$$Y^! = \lambda!x.\,((\lambda!y.x!(y(!y)))!\,(\lambda!y.x!(y(!y))))\,.$$

We are now going to highlight how we can see $Y^!$ as a fixpoint operator. Observe that we have to pass as argument to $Y^!$ a term in a box: it corresponds to the fact that the variable x is used twice. Let be M in $\Lambda^!_\oplus$. We define

$$L = (\lambda!y.M!(y(!y)))!\,(\lambda!y.M!(y(!y)))\,.$$

Then we can see that $Y^!(!M) \rightarrow L$ and moreover when we do one step of execution from L, we obtain the function M applied to L: indeed it holds that $L \rightarrow M!L$.

7.2 Type System for $\Lambda^!_\oplus$.

As explained before, when we write a program of the form $\lambda x.M$, the variable x is meant to be *affine* i.e. it has to be used *at most once* in M. However, we are able to write terms in $\Lambda^!_\oplus$ that do not respect this rule. To leave such terms out, we introduce now a type system to enforce our affine/exponential policy on variables. Besides, this type system will also allow us to avoid *deadlocks*, i.e. terms that are not values, but are not reductible by the reduction $\cdot \to \cdot$ either: for instance terms as $(\lambda !x.x)\lambda x.x$, where the left-hand side wait for a term in a box, but is given a λ-abstraction.

Observe that we do not want our type system to be too restrictive: we want to enforce our policies on the use of resources and the absence of deadlocks, but we also want to be still able to encode $\Lambda_\oplus$. To do that, we use *first-order* recursive types.To sum up, our approach consists in combining in our type system tools to manage resources (using affine and exponentials variables, as well as a ! operator on types to denote the ability to be copied infinitely often), and tools to increase expressiveness (recursive types), thus enabling us to encode $\Lambda_\oplus$. We first give a presentation of generic first-order recursive types for λ-calculi, following [17]. We give it a brief and presentation of it, see [9] for a more complete account.

7.2.1 Types for $\Lambda^!_\oplus$

Our $\Lambda^!_\oplus$-type system is *à la Curry*—we don't add typing anotations to the syntax, but instead define typing judgments for terms without typing annotations. We have a functionnal type constructor $\to$, a pair type constructor $\times$, a constructor ! to denote the ability to be copied, and the fixpoint type operator $\mu\alpha.\sigma$.

The fixpoint type operator μ should be seen as a *finite* notation for a solution of a *recursive type equation*. For instance, the type $\sigma = \mu\alpha.\alpha \to \alpha$ represents a solution to the recursive type equation: $\mathsf{A} = \mathsf{A} \to \mathsf{A}$.

Definition 7.2.1 (Types for $\Lambda^!_\oplus$) *We consider here a fixed countable set of type variables VT. The set $\mathscr{A}_{\Lambda^!_\oplus}$ of types for $\Lambda^!_\oplus$ is defined by the following grammar:*

$$\sigma, \tau \ldots \in \mathscr{A}_{\Lambda^!_\oplus} ::= \alpha \;\big|\; \sigma \to \tau \;\big|\; !\sigma \;\big|\; \mu\alpha.\sigma$$
$$\text{with } \alpha, \beta \in VT$$

The least fixpoint operator $\mu\alpha.\sigma$ bound the variable α in σ. From now on, we consider types up to α-congruence on the bound type variables.

We have to formalize in which sense a type $\mu\alpha.\sigma$ can be seen as a solution of an associated recursive type equation. We follow here [7], and do this by interpreting every type σ in $\mathscr{A}_{\Lambda^!}$ with its (possibly) infinite unfolding of every recursive equation in it: thus we obtain a (possibly) infinite tree $T(\sigma)$. We denote by *Tree* the set of all *regular*—i.e with a finite set of subtrees—trees whose leaves are in $VT \cup \{\bot\}$, and whose nodes are the non-recursive operators of the language, i.e $O = \{!, \to, \times, \forall\alpha\}$.

Definition 7.2.2 *Let be $\sigma \in \mathscr{A}_{\Lambda^!}$. We define co-inductively $T(\sigma) \in Tree$ as follows:*

- $T(\alpha) = \alpha$;
- $T(\sigma \to \tau) = Node(\to, T(\sigma), T(\tau))$;
- $T(!\sigma) = Node(!, T(\sigma))$;

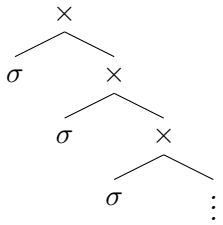
- $T(\mu\alpha.\sigma) = \begin{cases} \bot \text{ if } \sigma = \mu\beta_1 \dots \mu\beta_n \cdot \alpha \\ T(\sigma\{\mu\alpha.\sigma/\alpha\}) \text{ otherwise.} \end{cases}$.

We consider the set *Tree* up to α-equivalence. We illustrate below how first-order recursive typescan be used for encoding infinite data types. For the purpose of this example, we add a product type $\times$ to our type constructors. Observe however that our language $\Lambda^!_\oplus$ has no constructors for pairs, and accordingly our type system for $\Lambda^!_\oplus$ has no product operator.

Example 7.2.1 *Let be $\alpha \in VT$, and σ such that α doesn't occur freely in σ. We consider the type:*

$$\mathit{InfiniteList}_\sigma = \mu\alpha.(\sigma \times \alpha).$$

Observe that this type is indeed designed to model infinite lists of elements of type σ, since every element of type $\mathit{InfiniteList}_\sigma$ is the given of an element of type σ—the head element —and another element of type $\mathit{InfiniteList}_\sigma$ —the tail. We see that $T(\mathit{InfiniteList}_\sigma)$ is the following infinite tree:



Observe that two *different* $\Lambda^!_\oplus$ types can be the finite representation of the *same* infinite tree. For instance, the type $\mu\alpha.(\sigma \times (\sigma \times \alpha))$ has the same associated tree as $\text{InfiniteList}_\sigma$. It is an intrinsic feature of recursive types, since the same type can be a solution of several different recursive equations. Since we are actually interested in the tree itself, and not its finite representation, we will consider $\Lambda^!_\oplus$ types modulo equality of their associated tree, following [7].

Definition 7.2.3 *$=^{\mathscr{A}}$ is the equivalence relation on $\mathscr{A}_{\Lambda^!_\oplus}$ defined as: $\sigma =^{\mathscr{A}} \tau$ if $T(\sigma) = T(\tau)$.*

With this definition, we can now formalize our earlier idea of recursive equations: if we consider the type $\text{InfiniteList}_\sigma$ of Example 7.2.1, we see that it indeed holds that: $\text{InfiniteList}_\sigma =^{\mathscr{A}} \sigma \times \text{InfiniteList}_\sigma$. In the following, we will always consider implicitly the types up to this equality relation.

Lemma 7.2.1 *The equality relation on types $=^{\mathscr{A}}$ verify the rules given in Figure 7.1.*

7.2.2 Typing System for $\Lambda^!_\oplus$

Now that we have defined a set of types $\mathscr{A}_{\Lambda^!_\oplus}$, we are going to specify the typing rules. The typing rules we are going to give here are inspired of the one for Lily (see [14]). We want to establish *typing judgments*, that guarantee that a term M of $\Lambda^!_\oplus$ is well-typed under some typing environment Γ: this typing environment specify the type of the free variable in M. More precisely, a typing judgment will be of the form $\Gamma \vdash M : \sigma$: it means that when the free variables of M are replaced by terms having the type specified by the typing environment Γ, then the resulting term is of type σ.

Recall that in a term M of $\Lambda^!_\oplus$, free variables may have two different meanings: they may be meant to be affine, and in this case they must appear at most once in M, or they

$$\text{Fold-Unfold}\ \frac{}{\mu\alpha.\mathsf{A} =^{\mathscr{A}} \mathsf{A}[\alpha \to \mu\alpha.\mathsf{A}]} \qquad \frac{\mathsf{B} =^{\mathscr{A}} \mathsf{A}}{\mu\alpha.\mathsf{A} =^{\mathscr{A}} \mu\alpha.\mathsf{B}}\ \mu\text{-compat}$$

$$\frac{\mathsf{A}_i =^{\mathscr{A}} \mathsf{B}_i \text{ for } 1 \le i \le n \qquad \mathrm{op} \in O = \{\to, \times, !, \forall\alpha.\}}{\mathrm{op}(\mathsf{A}_1, \ldots, \mathsf{A}_n) =^{\mathscr{A}} \mathrm{op}(\mathsf{B}_1, \ldots, \mathsf{B}_n)}$$

$$\text{Contract}\ \frac{\sigma =^{\mathscr{A}} \mathsf{A}[\alpha \to \sigma] \qquad \tau =^{\mathscr{A}} \mathsf{A}[\alpha \to \tau] \qquad \mathsf{A} \text{ contractive in } \alpha}{\sigma =^{\mathscr{A}} \tau}$$

where $\sigma[\alpha \to \tau]$ is the type obtained by substituting all free occurrences of α by τ in σ;
A is *contractive in* α when every free occurence of α arrive under an operator in O.

Figure 7.1: Rules verified by $=^{\mathscr{A}}$ Relation

may be meant to be exponential, and then they may appear several times. Accordingly, there will be two kind of elements in our typing environment: *affine* pairs of the form (x, σ) which specify that x is an affine free variable of type σ, and *exponential* pairs of the form $(!x, !\sigma)$, that specify that x is an *exponential* free variable of type σ.

Definition 7.2.4 *An* environment *is a set of expressions in the form* $x : \sigma$ *or* $!x :!\sigma$*, where* $x \in \boldsymbol{V}$ *and* $\sigma \in \mathscr{A}_{\Lambda^!_\oplus}$*, and in which any variable* x *occurs at most once.*

To make notations more readable, we will indicate the exponential part with metavariables like $!\Gamma$, which stands for an environment with only exponential pairs $(!x, !\sigma)$, and the affine part by metavariables like Δ standing for an environment where all the elements are affine.

Definition 7.2.5 *We say that a typing judgment* $!\Gamma, \Delta \vdash M : \sigma$ *is* valid *it it can be derived from the rules given in Figure 7.2.*

$$\frac{}{!\Gamma, !x :!\sigma, \Delta \vdash x : \sigma}\ !\text{-V} \qquad \frac{}{!\Gamma, \Delta, x : \sigma \vdash x : \sigma}\ \text{V}$$

$$\frac{!\Gamma, x : \sigma, \Delta \vdash M : \tau}{!\Gamma, \Delta \vdash \lambda x.M : \sigma \to \tau}\ \text{Abstr} \qquad \frac{!x :!\sigma, !\Gamma, \Delta \vdash M : \tau}{!\Gamma, \Delta \vdash \lambda !x.M :!\sigma \to \tau}\ !\text{-Abstr}$$

$$\frac{!\Gamma, \Delta \vdash M : \sigma \to \tau \qquad !\Gamma, \Theta \vdash N : \sigma}{!\Gamma, \Delta, \Theta \vdash MN : \tau}\ \text{App} \qquad \frac{!\Gamma \vdash M : \sigma}{!\Gamma \vdash !M :!\sigma}\ \text{Box}$$

$$\frac{!\Gamma, \Delta \vdash M : \sigma \qquad !\Gamma, \Delta \vdash N : \sigma}{!\Gamma, \Delta \vdash M \oplus N : \sigma}\ \text{Choice}$$

Figure 7.2: Typing Rules

We give now some explanations on the typing rules given in Figure 7.2. First, observe that the (Var) and (!-Var) rules imply the ability to *weaken* variables, in the sense that non-used variables (both affine or exponential) can be discarded.

Then, we can see that we have two rules (Abstr) and (!-Abstr), respectively, for the affine and exponential abstractions. It us an adaption to our affine/ exponential framework of the standard abstraction rule in typed lambda-calculus.

Let us now look at the application rule (App). We see a difference in the treatment of the affine typing context and the one of the exponential typing context: when exponential variables can be used to type *both* M and N, we have to choose for every affine variable whether it is going to be used to type M *or* to type N. This specifies that exponential variables can be *duplicated*, while affine variables cannot.

The rule (Box) allow to type terms of the shape $!M$. It says that if we are able to type M with the type σ using a typing context *with only duplicable variables*, then we can put M in an exponential box, and we obtain a term of type $!\sigma$. The idea is that, since all the free variables of M may be duplicated, we are also able to duplicate M.

The rule (Choice) for probabilistic choice says that we can only do a probabilistic choice between two programs of *the same* type. Crucially, an affine variable x can be used to type *both* components of a probabilistic choice: it is because it *is not* a duplication, but simply the consequence of the fact that x can be used in every probabilistic execution path.

Definition 7.2.6 *We say that $M \in \Lambda^!_\oplus$ is* a program of type σ, *if $(\vdash M : \sigma)$ is a valid typing judgment. We denote by $\mathscr{P}^\sigma$ the set of programs of type σ. We denote by $\mathscr{V}^\sigma$ the set of programs of type σ which are also values.*

We are now going show that there exist valid typing judgments for the terms of $\Lambda^!_\oplus$ we gave as examples in the previous section.

Example 7.2.2 (Type for Ω) *We are going to show here that for every type $\sigma \in \mathscr{A}_{\Lambda^!_\oplus}$, it holds that $(\vdash \Omega : \forall\alpha.\alpha)$ is a valid typing judgment.*

$$\Omega = (\lambda!x.x(!x))(!\lambda!x.x(!x)).$$

In the proof tree, we will use the type $\tau = \mu\alpha.!\alpha \to \sigma$. The relevance of τ comes from the fact that looking at the fold-unfold *equation in Figure 7.1, we can see that $\tau =^{\mathscr{A}} !\tau \to \sigma$.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau =^{\mathscr{A}} !\tau \to \sigma}{!x :!\tau \vdash x :!\tau \to \sigma}\,!\text{-}V \quad \dfrac{\dfrac{!x :!\tau \vdash x : \tau}{!x :!\tau \vdash !x :!\tau}\,!\text{-}V}{}\,Box}{!x :!\tau \vdash x(!x) : \sigma}\,Appl}{\vdash \lambda!x.x(!x) :!\tau \to \sigma}\,!\text{-}Abstr \qquad \dfrac{\dfrac{\vdots}{\vdash \lambda!x.x(!x) : \tau =^{\mathscr{A}} !\tau \to \sigma}}{\vdash !(\lambda!x.x(!x)) :!\tau}\,Box}{\vdash (\lambda!x.x(!x))(!\lambda!x.x(!x)) : \sigma}\,App$$

where we take $\tau ::= (\mu\alpha.!\alpha \to \sigma)$.

Observe that, on the right, it remains to be shown that $\vdash \lambda!x.x(!x) :!\tau \to \sigma$. However, we see that it is sufficient to use again the same derivation tree as in the left part.

Recall that we have also defined a recursion operator Y for $\Lambda^!_\oplus$ in Example 7.1.2. We are now going to see that we can adapt the proof of typability of Ω to Y.

Example 7.2.3 (Type for the fixpoint combinator Y.) *Recall the fixpoint combinator Y that we have defined in Example 7.1.2:*

$$Y = \lambda!x.\,((\lambda!y.x!(y(!y)))!\,(\lambda!y.x!(y(!y))))\,.$$

We are also able to type Y in our type system. Indeed, it holds that for every type $\sigma \in \mathscr{A}_{\Lambda^!_\oplus}$: $\vdash Y : !(!\sigma \to \sigma) \to \sigma$.

In the proof, we use the same auxilliary type $\tau = \mu\alpha.!\alpha \to \sigma$ as in Example 7.2.2: recall that it verifies $\tau =^{\mathscr{A}} !\tau \to \sigma$. We also use the fact that we have shown in the proof of Example 7.2.2 that $!y :!\tau \vdash y!y : \sigma$ (it is the premise of a sub-derivation in the proof tree presented in Example 7.2.2).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!x :!(!\sigma \to \sigma), !y :!\tau \vdash x :!\sigma \to \sigma}\,\text{!-V} \quad \dfrac{\dfrac{\text{(see Example 7.2.2)}}{!y :!\tau \vdash y!y : \sigma}}{!y :!\tau \vdash !(y!y) :!\sigma}\,\text{Box}}{!x :!(!\sigma \to \sigma), !y :!\tau \vdash x!(y(!y)) : \sigma}\,\text{App}}{!x :!(!\sigma \to \sigma) \vdash L_x :!\tau \to \sigma}\,\text{!-Abstr} \quad \dfrac{\dfrac{\vdots}{!x :!(!\sigma \to \sigma) \vdash L_x : \tau =^{\mathscr{A}} !\tau \to \sigma}}{!x :!(!\sigma \to \sigma) \vdash !L_x :!\tau}\,\text{Box}}{!x :!(!\sigma \to \sigma) \vdash L_x!L_x : \sigma}\,\text{App}}{\vdash \lambda!x.L_x!L_x :!(!\sigma \to \sigma) \to \sigma}\,\text{!-Abstr}}{\vdash Y = \lambda!x.L_x!L_x : \forall\sigma.!(!\sigma \to \sigma) \to \sigma}\,\forall\text{ intro}$$

where we take $\tau ::= (\mu\alpha.!\alpha \to \sigma)$.
and $L_x ::= (\lambda!y.x!(y(!y)))$

Figure 7.3: Proof Tree of $\vdash Y : !(!\sigma \to \sigma) \to \sigma$

Since we are able to type the fixpoint operator Y, we may now use it to express program recursively defined.

In the following, it will be sometimes useful to type evaluation contexts too. We write $[\cdot] : \sigma \vdash \mathcal{E}$ to express the fact that, if we fill some evaluation context $\mathcal{E}$ by any program of type σ, then we will obtain a well-typed program.

Notation 7.2.2 (Typing Judgment for Contexts) *Let $\mathcal{C}$ be a context. We write $[\cdot] : \sigma \vdash \mathcal{C} : \tau$ when $x : \sigma \vdash \mathcal{E}[x] : \tau$. When we don't need to specify the result type, we will note $[\cdot] : \sigma \vdash \mathcal{E}$, whenever there exists a type ι such that $[\cdot] : \sigma \vdash \mathcal{C} : \iota$ is a valid typing judgment.*

7.2.3 Soundness of the Type System for $\Lambda^!_\oplus$.

We are now going to study which properties on programs are enforced by our type system. We first want to ensure that no deadlock can occur when we execute a well-typed program. To show that, we need the following two lemmas, that ensure that our type system is well behaved. The first one is the probabilistic counterpart of the Subject Reduction property for deterministic typed λ-calculi: it says that, whenever we do *one* execution step from a program of type σ, every program we may obtain as a result is also of type σ.

Lemma 7.2.3 (Subject Reduction) *Let be M a program such that $\vdash M : \sigma$, and moreover $M \to N_1, \ldots, N_n$. Then for every $i \in \{1, \ldots, n\}$, it holds that $\vdash N_i : \sigma$.*

Proof. As in the deterministic case, the proof of Lemma 7.2.3 is based on an auxilliary Substitution Lemma:

Lemma 7.2.4 *Let be M a program such that $\vdash M : \sigma$. Then:*

- *if $x : \sigma \vdash N : \tau$, then it holds that $\vdash N\{M/x\} : \tau$.*
- *if $!x :!\sigma \vdash N : \tau$, then it holds that $\vdash N\{M/x\} : \tau$.*

Proof. The substitution lemma is shown by induction on the structure of N. □

We conclude the proof of the Subject Reduction lemma by case distinction on redex rules. □

We show now that our type system enjoys the *Progress Property*, that says that if a well typed program cannot be reduced anymore, then it is a value, and not merely a deadlocked term.

Lemma 7.2.5 (Progress) *If M is a typable program which is a normal form for $\to$, then it is a value.*

We are now able to use Subject Reduction and Progress to show that our $\Lambda^!_\oplus$ type system is sound: if an execution path with a non-zero probability stops at a normal form, then this normal form is a value of the same type as the original program. We can interpret this property by saying that it guarantees that well-typed programs are *well-behaved*.

Proposition 7.2.6 (Soundness of the Type System.) *Let be M a program of type σ. Then for every term N in the support of $[\![M]\!]$, it holds that N is a value of type σ.*

Proof. Let be M a program of type σ. Recall that $[\![M]\!] = \sup\{\mathscr{D} \mid M \Rightarrow \mathscr{D}\}$. It means that to show Propsition 7.2.6, it is sufficient to show that whenever $M \Rightarrow \mathscr{D}$, $\mathsf{S}(\mathscr{D}) \subseteq \mathscr{V}^\sigma$. The proof is done by induction on the derivation of $M \Rightarrow \mathscr{D}$.

- If the derivation uses the (Empty) rule as last rule, it means that it is of the form:

$$\frac{}{M \Rightarrow \emptyset = \mathscr{D}},$$

  and since the support of the empty distribution is empty, the result holds.

- If the derivation uses the (NF) rule as last rule, it means that it is of the form:

$$\mathrm{Val}\,\frac{V \text{ is a } \to \text{ normal form.}}{M = V \Rightarrow \{V^1\} = \mathscr{D}}.$$

  It means that the only term in the support of $\mathscr{D}$ is M itself, and moreover it is already in normal form. Using Lemma 7.2.5, we see that it implies that M is a value.

- If the derivation use the (Step) rule as last rule, it is of the form:

$$\mathrm{Step}\,\frac{M \to N_1, \ldots, N_n \qquad (N_i \Rightarrow \mathscr{E}_i)_{1 \le i \le n}}{M \Rightarrow \mathscr{D} = \sum_{1 \le i \le n} \frac{1}{n} \cdot \mathscr{E}_i}.$$

  First, we apply Lemma 7.2.3: it tells us that each one of the program $N_1, \ldots, N_n$ is of type σ. We can apply now the induction hypothesis on every one of the sub-derivations $N_i \Rightarrow \mathscr{E}_i$. As a consequence, we know already that $\mathsf{S}(\mathscr{E}_i) \subseteq \mathscr{V}^\sigma$ for every $i \in \{1, \ldots, n\}$. Since $\mathscr{D} = \sum_{1 \le i \le n} \frac{1}{n} \cdot \mathscr{E}_i$, it holds that $\mathsf{S}(\mathscr{D}) = \bigcup_{1 \le i \le n} \mathsf{S}(\mathscr{E})_i$. We can see that it leads to the result.

□

While our type system for $\Lambda^!_\oplus$ is sound and allows us to enforce our ressource policy on variables, we should also note that type inference is undecidable.

Example 7.2.4 (Boolean in $\Lambda^!_\oplus$.) *Recall the terms* $\boldsymbol{true} = \lambda x.\lambda y.x$, $\boldsymbol{false} = \lambda x.\lambda y.y$, *and* $ITE = \lambda x.\lambda y.\lambda z.xyz$, *that we presented in Example 9.2.1 to simulate the boolean data type in* Λ. *For every type* σ, *we note* $\mathbb{B}_\sigma := \sigma \to \sigma \to \sigma$. *Then for every type* σ, *we have:*

$$\vdash \boldsymbol{true} : \mathbb{B}_\sigma \qquad \vdash \boldsymbol{false} : \mathbb{B}_\sigma \qquad \vdash ITE : (\mathbb{B}_\sigma \to \sigma \to \sigma)$$

Since we have fixed a CBV strategy, it means in particular that when evaluating $ITE\,M\,N\,L$ *both conditional branches* N *and* L *are executed. To obtain a more intuitive version of conditional, we use the term* ITE *to define the* if-then-else *procedure, which is a* $\Lambda^!_\oplus$ *context with three holes, defined as:*

$$\textit{if } ([\cdot]_1) \textit{ then } [\cdot]_2 \textit{ else } [\cdot]_3 := (\ ITE\ [\cdot]_1(\lambda x.[\cdot]_2)(\lambda x.[\cdot]_3))I$$

7.3 Embeddings from $\Lambda_\oplus$ into $\Lambda^!_\oplus$

We see now that $\Lambda^!_\oplus$ allows to encode the untyped probabilistic λ-calculus $\Lambda_\oplus$. To make this connection more precise, we will define encodings $[\cdot]^{\mathcal{L}}_\oplus : \mathcal{L}_\oplus \to \Lambda^!_\oplus$, for $\mathcal{L}_\oplus \in \{\Lambda^{\text{cbn}}_\oplus, \Lambda^{\text{CBV}}_\oplus\}$, and we will show that these encoddings preserve operational semantics.

Here, we present the embedding $[]^{\text{cbn}}$ from $\Lambda_\oplus$ endowed with its CBN semantics as presented in Chapter 1, into $\Lambda^!_\oplus$. To design it, we made use of the connexion between $\Lambda^!_\oplus$ and Linear Logic (LL). Linear Logic is a logic which talk about the use of *ressources.* $\Lambda^!_\oplus$ can be seen as a *linear calculus*, in the sens that the syntactic construct of the language (apart from the probabilitic choice) correspond to the connector of the MELC (Multiplicative Exponential Linear Logic) fragment of LL. In [51], Girard looked at standard ways to translate the intuitionnistic logic into intuitionnistic linear logic. Using the connection between logic and lambda-calculi, it gives equivalently a way to translate the simply typed λ-calculus in the linear term calculus. Girard proposed actually two translations, which were shown to correspond to a CBN and CBV evaluation strategy. Here, we are going to adapt these translations to translations from the *untyped* probabilistic lambda-calculus $\Lambda_\oplus$ into $\Lambda^!_\oplus$, that can be seen as a probabilistic linear language *enhanced* with recursive types.

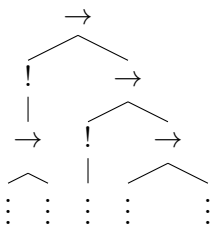
The Call-By-Name Embedding

We adapt here this translation to an embedding $[\cdot]^{\text{cbn}} : \Lambda_\oplus \to \Lambda^!_\oplus$.

Definition 7.3.1 *We define* $\sigma^{cbn} = \mu\alpha.!\alpha \to \alpha$, *and* $[\cdot]^{cbn} : \Lambda_\oplus \to \Lambda^!_\oplus$ *as:*

$$[x]^{cbn} = x \qquad [\lambda x.M]^{cbn} = \lambda !x.[M]^{cbn}$$
$$[MN]^{cbn} = [M]^{cbn}![N]^{cbn} \qquad [M \oplus N]^{cbn} = [M]^{cbn} \oplus [N]^{cbn}$$

Observe that the type σ^{cbn} is a solution to the recursive equation $\sigma^{\text{cbn}} = !\sigma^{\text{cbn}} \to \sigma^{\text{cbn}}$.We illustrate it by giving the representation of its infinite unfolding:



Proposition 7.3.1 *Let be $M \in \Lambda_{\oplus}$, with $FV(M) \subseteq \{x_1, \ldots, x_n\}$. Then it holds that $!x_1 :!\sigma^{cbn}, \ldots, !x_n :!\sigma^{cbn} \vdash [M]^{cbn} : \sigma^{cbn}$.*

A positive point about this embedding, that will be relevant in the following, is that it is *compositionnal*, i.e. it preserves the structure of the term. We formalize this idea in Lemma 7.3.2 below.

Lemma 7.3.2
- *Let be $M, N \in \Lambda_{\oplus}$, and $z \in Vars()$. Then $[M\{N/z\}]^{cbn} = [M]^{cbn}\{[N]^{cbn}/z\}$;*
- *Let be $M \in [\Lambda_{\oplus}]^{cbn}$, such that there exists an evaluation context $\mathcal{E}$, and $N \in \Lambda_{\oplus}$, verifying $[M]^{cbn} = \mathcal{E}[[N]^{cbn}]$. There for any term $L \in \Lambda_{\oplus}$, it holds that $\mathcal{E}[[L]^{cbn}] \in [\Lambda_{\oplus}]^{cbn}$.*

The operational semantics of CBN $\Lambda_{\oplus}$ is preserved by the embedding in the following sense:

Proposition 7.3.3 *Let be M a closed term in $\Lambda_{\oplus}$. Then $[[\![M]\!]]^{cbn} = [\![[M]^{cbn}]\!]$.*

The Call-By-Value Embedding

Girard gives actually another translation from intutionnistic logic into ILL, that lead to a CBV evaluation strategy.

Definition 7.3.2 *We define $\sigma^{cbv} = \mu\alpha.!(\alpha \to \alpha)$, and $[\cdot]^{cbv} : \Lambda_{\oplus} \to \Lambda_{\oplus}^{!}$ as:*

$$[x]^{cbv} = !x \qquad [\lambda x.M]^{cbv} = !\lambda !x.[M]^{cbv}$$
$$[MN]^{cbv} = (\lambda !x.x[N]^{cbv})[M]^{cbv} \qquad [M \oplus N]^{cbv} = [M]^{cbv} \oplus [N]^{cbv}$$

7.4 $\Lambda_{\oplus}^{\leq 1}$: An affine λ-calculus

Here, we are going to define a λ-calculus which is affine, i.e. no variable can be duplicated. This language is designed to model functions using their arguments at most once.

Observe that we could obtain such an affine higher-order language by looking at the fragment of $\Lambda_{\oplus}^{!}$ obtained when removing the exponential constructs of the syntax of the language, and similarly the exponential type operator ! of the syntax for types. However, we instead opt for a presentation with *affinity constraints*, which is the one we used in the joint work with Dal Lago in [29]. The $\Lambda_{\oplus}^{\leq 1}$ terms are generated by the following grammar:

$$M ::= x \;\Big|\; \lambda x.M \;\Big|\; MM \;\Big|\; M \oplus M \;\Big|\; \Omega,$$

where Ω models divergence—since we only consider affine terms, we cannot encode divergence by the usual constructions of λ-calculus, and x ranges as usual over a countable set $\mathbf{V}$ of variables. The *valid terms* of $\Lambda_{\oplus}^{\leq 1}$ are isolated by way of a formal system, whose judgements are in the form $\Gamma \vdash M$ (where Γ is any finite set of variables) and whose rules are the following (where Γ, Δ stands for the union of two disjoints contexts):

$$\frac{}{\Gamma, x \vdash x} \qquad \frac{\Gamma, x \vdash M}{\Gamma \vdash \lambda x.M} \qquad \frac{\Gamma \vdash M \qquad \Delta \vdash N}{\Gamma, \Delta \vdash MN}$$
$$\frac{\Gamma \vdash M \qquad \Gamma \vdash N}{\Gamma \vdash M \oplus N} \qquad \frac{}{\Gamma \vdash \Omega}$$

A *program* is a term such that $\emptyset \vdash M$, and $\mathbf{P}_{\Lambda_{\oplus}^{\leq 1}}$ is the set of all such terms. We will call them *closed terms*. We say that a program is a *value* if it is of the form $\lambda x.M$, and $\mathcal{V}_{\Lambda_{\oplus}^{\leq 1}}$ is the set of all such programs. The semantics of the just defined calculus is expressed as a binary relation $\Downarrow$ between programs and *value subdistributions*.

Definition 7.4.1 *The relation $\Downarrow$ is inductively defined by the following rules:*

$$\frac{}{\Omega \Downarrow \emptyset} \qquad \frac{V \in \mathcal{V}}{V \Downarrow \{V^1\}} \qquad \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2}\mathscr{D} + \frac{1}{2}\mathscr{E}}$$

$$\frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E} \qquad \{L\{V/x\} \Downarrow \mathscr{F}_{L,V}\}_{\lambda x.L \in \mathsf{S}(\mathscr{D}), V \in \mathsf{S}(\mathscr{E})}}{MN \Downarrow \sum \mathscr{D}(\lambda x.L) \cdot \mathscr{E}(V) \cdot \mathscr{F}_{L,V}}$$

The divergent program Ω, as expected, evaluates to the *empty* value distribution $\emptyset$ which assigns 0 to any value. Recall that the expression $\{V^1\}$ stands for the Dirac's value distribution on V; more generally the expression $\{V_1^{p_1}, \ldots, V_n^{p_n}\}$ indicates the value distribution assigning probability p_i to each V_i (and 0 to any other value). For every program M, there exists precisely *one* value distribution $\mathscr{D}$ such that $M \Downarrow \mathscr{D}$, that we note $[\![M]\!]$. This holds only because we restrict ourselves to affine terms. Moreover, $[\![M]\!]$ is always a finite distribution. The rule for application expresses the fact that the semantics is call-by-value: the argument is evaluated before being passed to the function. There is no special reason why we adopt call-by-value here, and all we are going to say also holds for (weak) call-by-name evaluation.

The way $\Downarrow$ is defined means that it is a *big-step* notion of semantics. In some circumstances, we need to have a more local view of how programs behave. We can define *small-step semantics*, again as a relation $\Rightarrow$ between programs and value distributions (itself defined on top of a relation $\rightarrow$ between programs and *program* distributions capturing a single evaluation step).

$\Lambda_{\oplus}^{\leq 1}$, seen as a fragment of $\Lambda_{\oplus}^{!}$, is stable by the $\Lambda_{\oplus}^{!}$ reduction relation $\rightarrow$. It means that we can endow $\Lambda_{\oplus}^{\leq 1}$ with an operational semantics $[\![\,]\!]$ simply by taking the restriction of the $\Lambda_{\oplus}^{!}$-operational semantics.

However, we can also give a simpler caracterisation of $[\![\,]\!]$ for $\Lambda_{\oplus}^{\leq 1}$ programs. Indeed, enforcing affinity in terms leads to removind recursion. As a consequence, the set of *approximation semantics* of a program in $\Lambda_{\oplus}^{\leq 1}$ becomes finite. Recall that the operational semantics of a program is defined as the supremum of its approximation semantics: the previous observation means that we can give a direct inductive caracterisation of the $\Lambda_{\oplus}^{\leq 1}$-operational semantics.

Proposition 7.4.1 *The operational semantics in $\Lambda_{\oplus}^{\leq 1}$ can be caracterised by the following inductive definition:*

$$NF\,\frac{V \text{ is a } \rightarrow \text{ normal form.}}{[\![V]\!] = \{V^1\}} \qquad Step\,\frac{M \rightarrow N_1, \ldots, N_n}{[\![M]\!] = \sum_{1 \leq i \leq n} \frac{1}{n} \cdot [\![N_i]\!]}$$

Observe that the rules are the same as for defining approximation semantics (see Chapter 1 Definition 1.2.4), except that we removed the (Empty) rule.

Proof. What we have to show is that for every program $M \in \mathbf{P}^{\leq 1}$, there exists $\mathscr{D}$ such that we can show $M \Rightarrow \mathscr{D}$ using the rules of $\Lambda_{\oplus}^{!}$ for the approximation semantics, but without using the (Empty) rule.

The proof is based on the fact that we can define a function $c : \mathbf{P}^{\leq 1} \rightarrow \mathbb{N}$, such that, if M is a well-typed affine program, and $M \rightarrow N_1, \ldots, N_n$, it holds that for every $i \in \mathbb{N}$,

$c(M) > c(N_i)$. We define $c(M)$ inductively on the structure of M, as follows:

$$\begin{aligned}
&c(x) = 0 && c(\lambda x.N) = c(N)\\
&c(\Omega) = 1 && c(NL) = 1 + c(N) + c(L)\\
&c(M \oplus N) = 1 + \max\{c(M), c(N)\} && c(\langle M, N\rangle) = c(M) + c(N)\\
&c(\texttt{let } \langle x, y\rangle = M \texttt{ in } L) = c(M) + c(N) + 1
\end{aligned}$$

The idea is that we take $c(M)$ as the maximal number of redexes on every possible execution path. We first show the following auxilliary lemma:

Lemma 7.4.2 *Let be M such that $x_1 : \tau_1, \ldots, x_m : \tau_m \vdash M : \sigma$ is a valid affine typing judgment, and a family of terms N such that $\vdash N_i : \tau_i$. Then it holds that*

$$c(M\{N_1/x_1\}\ldots\{N_m/x_m\}) \leq c(M) + \sum_{1\leq i\leq m} c(N_i)$$

Proof. The proof of Lemma 7.4.2 is done by induction on the structure of M. We do explicitely here the application case: suppose that $M = KL$. Using the typing rules, we see that we can split the typing context $\Delta = x_1 : \tau_1, \ldots, x_m : \tau_m$ into two disjoints typing contexts $\Delta_1 = x_{i_1} : \tau_{i_1}, \ldots, x_{i_k} : \sigma_{i_k}$, and $\Delta_2 = x_{j_1} : \tau_{j_1}, \ldots, x_{j_l} : \tau_{j_l}$, such that $\Delta_1 \vdash N : \iota \to \sigma$, and $\Delta_2 \vdash L : \iota$. Moreover, since Δ_1 and Δ_2 are disjoint, a free variable x_i of M is substituted either in K, or in L, but not in both. It means that we can write:

$$\begin{aligned}
M\{N_1/x_1\}\ldots\{N_m/x_m\} &= K'L'\\
\text{with } K' &= K\{N_{i_1}/x_{i_1}\}\ldots\{N_{i_k}/x_{i_k}\}\\
\text{and } L' &= L\{N_{j_1}/x_{j_1}\}\ldots\{N_{j_l}/x_{j_l}\}
\end{aligned}$$

Using the definition of c, it tells us that

$$c(M\{N_1/x_1\}\ldots\{N_m/x_m\}) = 1 + c(K') + c(L'). \tag{7.1}$$

We apply now the induction hypothesis to both K and L: we obtain that:

$$c(K') \leq c(K) + \sum_{p\in\{i_1,\ldots,i_k\}} c(N_p) \tag{7.2}$$

$$\text{and } c(L') \leq c(L) + \sum_{p\in\{j_1,\ldots,j_l\}} c(N_p) \tag{7.3}$$

$$\tag{7.4}$$

Since $c(M) = c(K) + c(L)$ by the definition of c, we see that by combining (7.1), (7.2) and (7.3), we obtain the result.

□

Using 7.4.2, we are now able to show that the function c strictly decrease at each step of execution. Recall however that we still nonetheless have *non-convergence* in the calculus $\Lambda_{\oplus}^{\leq 1}$, but it is not modelled as a program with an execution that never stops as in Λ or $\Lambda_{\oplus}$, but by the primitive program Ω whose reduction rule is $\Omega \to \emptyset$.

Lemma 7.4.3 *M is a well-typed affine program, and $M \to N_1, \ldots, N_n$, it holds that for every $i \in \mathbb{N}$, $c(N_i) \leq c(M) - 1$.*

Proof. The proof is by case distinction on the redex which is actually reduced by $\rightarrow$. $\square$

We can now end the proof of Proposition 7.4.1 using Lemma 7.4.3. Indeed, recall that we have to show that there exists a finite proof derivation of $M \Rightarrow \mathscr{D}$ (seen as the operational semantics of $\Lambda_{\oplus}^{!}$) which does not use the (Empty) rule. Suppose by contradiction that this is not the case. Since we have a progress Lemma, the fact that there is no finite proof derivation implies that there exists a family of programs $(M_n)_{n\in\mathbb{N}}$, such that $M = M_0$, and for every $i \in \mathbb{N}$, it holds that $M_i \rightarrow N_1, \dots N_k$, where M_{i+1} is one of the N_j. But then we have build an infinite strictly decreasing sequence of natural numbers by considering $c(M_i)$, which does not exist. $\square$

Observe that since we don't have recursion anymore, the program $\Omega^{!}$ is not in $\Lambda_{\oplus}^{\leq 1}$. However, we still have non-terminating programs: indeed the generic non-terminating term Ω is still a constant in the syntax. It is actually our motivation for introducing Ω in the syntax: we wanted to keep the possiblilty to express non-termination in the affine fragment of $\Lambda_{\oplus}^{!}$: indeed recall that we take as our notion of *observation* on programs—for defining context equivalence or context distance—the probability of non-termination (see Chapters 2 and 3).

Corollary 7.4.1 *For any $\Lambda_{\oplus}^{\leq 1}$-program M, $[\![M]\!]$ is a sub-distribution with finite support.*

Example 7.4.1 (Dyadic Choice.) *Let $\mathbb{D}$ be the set of dyadic numbers (i.e. those rational numbers in the form $\frac{n}{2^m}$ (with $n, m \in \mathbb{N}$ and $n \leq 2^m$). It is easy to derive, for every $\varepsilon \in \mathbb{D}$, a new binary operator on terms $\cdot \oplus^{\varepsilon} \cdot$ such that $[\![M \oplus^{\varepsilon} N]\!] = (1-\varepsilon)[\![M]\!] + \varepsilon[\![N]\!]$ for every closed M, N. It can be defined in $\Lambda_{\oplus}^{\leq 1}$, e.g., as follows by induction on m:*

$$
\begin{aligned}
M \oplus^{0} N &= M \\
M \oplus^{1} N &= N \\
M \oplus^{\frac{n}{2^m}} N &= \begin{cases} M \oplus (M \oplus^{\frac{n}{2^{m-1}}} N) & \text{if } n \leq 2^{m-1} \\ N \oplus (M \oplus^{\frac{n-2^{m-1}}{2^{m-1}}} N) & \text{if } n > 2^{m-1} \end{cases}
\end{aligned}
$$

Definition 7.4.2 *Contexts in $\Lambda_{\oplus}^{\leq 1}$ We call* open $\Lambda_{\oplus}^{\leq 1}$-contexts *the terms $\mathcal{C}$ build using the grammar of $\Lambda_{\oplus}^{\leq 1}$, with $Vars()$ as set of potential bounded variables, and $[\cdot] \cup Vars()$ as set of potential free variables, and such that there exists $X \subseteq Vars()$ with $X \cup [\cdot] \vdash \mathcal{C}$. We call $\Lambda_{\oplus}^{\leq 1}$-context the open $\Lambda_{\oplus}^{\leq 1}$-context such that $[\cdot] \vdash \mathcal{C}$.*

Lemma 7.4.4 *We can caracterise the open $\Lambda_{\oplus}^{\leq 1}$-contexts as the terms generated by the following grammar:*

$$\mathcal{C} ::= [\cdot] \;\Big|\; M\mathcal{C} \;\Big|\; \mathcal{C}M \;\Big|\; \lambda x.\mathcal{C} \;\Big|\; \mathcal{C} \oplus \mathcal{D} \;\Big|\; M,$$

and such that $[\cdot] \vdash \mathcal{C}$.

Chapter 8

Applicative Trace Distances for Higher-order Probabilistic Calculi.

In Chapter 4 and 5 we studied two extensions of Abramsky's applicative bisimilarity in the setting of higher-order *probabilistic* calculi, that were both introduced by Dal Lago, Sangiorgi and Alberti [32]: trace equivalence—that was shown by the authors of [32] to be fully abstract for CBN $\Lambda_{\oplus}$—and the probabilistic applicative bisimilarity—that we have shown in Chapter 5 to be fully abstract for CBV $\Lambda_{\oplus}$. We now want to extend this line of reasoning to the quantitative case, i.e. to build *distances* on programs that are easier to handle than the observational distance, while still allowing us to gather as much useful information as possible about the observational distance. In the first part of the present Chapter, we revisit the properties for equivalence relations on programs that we presented in Chapters 4 and 5 —i.e. *observational correctness*, *compositionality*, and *soundness*—by adapting them to distances, in the setting of a generic observable probabilistic programming language. In the second part of the present chapter, our goal thus becomes to extend trace equivalence into a quantitative notion, thus defining sound *trace distances* on probabilistic higher-order programs. In the next chapter, we will do the same for probabilistic applicative bisimilarity, thus obtaining *applicative bisimilarity distances*.

When introducing trace equivalence for CBN $\Lambda_{\oplus}$, Dal Lago, Sangiorgi and Alberti used a presentation based on applicative contexts. Recall that in Chapter 4, we give also an alternative presentation using WLTSs, that are deterministic Labeled Transition Systems with weight. With this viewpoint, defining a trace equivalence for some higher-order probabilistic language means giving a WLTS that expresses interactively the operational semantics of this language, and then to use bisimilarity on this WLTS to build an equivalence on programs.

Our approach towards a quantitative extension of trace equivalence is as follows: we first adapt the notion of WLTS-bisimilarity into a quantitative notion of *bisimilarity distance* for WLTSs, and then as in the equivalence case, we build a distance on the set of programs of a particular language by using the bisimilarity distance of its associated WLTS. Observe however that while this approach can *a priori* be used for any higher-order probabilistic programming language, there is *no guarantee* that the resulting equivalences and distances on programs will be *sound* with respect to the observational equivalence and the observational distance of the language respectively. To illustrate this point, let us recall what happens for the language $\Lambda_{\oplus}$: while the WLTS $\mathscr{L}({\Lambda_{\oplus}}^{\mathrm{cbn}})$—the applicative WLTS for *CBN* $\Lambda_{\oplus}$ that we defined in Definition 4.2.4 in Chapter 4 —leads to a sound trace equivalence, that is not the case anymore if we use a similarly built WLTS for *CBV*

$\Lambda_\oplus$, as we mentioned in Section 5.3 in Chapter 5.

Actually, the situation becomes *worse* in the quantitative case, since—as we will show in this Chapter—the trace distance on CBN $\Lambda_\oplus$ induced by $\mathscr{L}({\Lambda_\oplus}^{\mathrm{cbn}})$ *is not* sound. As we will highlight, this *gap* between the 2-valued case and the quantitative case comes actually from the *copying phenomenon*. For this reason, we will first consider the affine language $\Lambda_\oplus^{\leq 1}$, in which no copying can occur: we will adapt to this language the applicative WLTS used for CBN $\Lambda_\oplus$, and show that this WLTS indeed leads to a sound—and even fully abstract—distance for $\Lambda_\oplus^{\leq 1}$ programs. The last part of this Chapter deals then with the copying phenomenon: we look at the language $\Lambda_\oplus^!$ with ! explicit that we presented in Chapter 7: the presence of this additional syntactic structure allows us to design a WLTS specifically to get around of the copying problem. We then show that the distance for $\Lambda_\oplus^!$-programs obtained with this WLTS is sound—and even fully abstract.

The work we present here was published jointly with Ugo Dal Lago in [29, 26], and our presentation here follows closely the structure of these papers.

8.1 Well-behaved Distances for Higher-Order Languages

We have defined in Chapter 3 the *observational distance* for a generic observable probabilistic programming language—that is any language equipped with a notion of observation and a notion of contexts as formalized in Definition 2.3.1 for untyped languages, and in Definition 2.3.2 for typed languages. In this section, we want to look at a larger class of *distances* for such generic observable programming language, and at how we can describe their behavior with respect to the observational distance.

The distances that we are going to consider are—contrary to the observational distance—defined on the set of *programs*, instead of all terms of the language; recall that it is also the case of the coinductively defined equivalences that we have built in Chapter 4 and 5. In the same way that we used open extension to go from a relation on programs to a relation on all terms in the equivalence case, we will also be able to go from a distance on programs to a distance on all terms, as we will illustrate in the next chapter. We ask our distances to be essentially *pseudo-metrics*—see Definition 3.1.3 of Chapter 3—on programs, but when we work with a programming language which is typed, we need to take also into account the fact that we can compare only programs that have the same type. Summing up, we formalize below in Definition 8.1.1 the class of distances for any observable probabilistic programming language.

Definition 8.1.1 *Let $\mathcal{L}_\oplus$ be an observable programming language. A* distance on programs *is:*

- *if $\mathcal{L}_\oplus$ is untyped, a pseudo-metric $\mu : \boldsymbol{P} \times \boldsymbol{P} \to [0,1]$, where $\boldsymbol{P}$ is the set of programs of $\mathcal{L}_\oplus$;*
- *if $\mathcal{L}_\oplus$ is typed, a family of pseudo-metrics indexed by the types: $(\mu_\sigma)_{\sigma \in \mathscr{A}_{\mathcal{L}_\oplus}}$ with $\mu_\sigma : \boldsymbol{P}_\sigma \times \boldsymbol{P}_\sigma \to [0,1]$, where $\mathscr{A}_{\mathcal{L}_\oplus}$ is the set of $\mathcal{L}_\oplus$-types, and for every $\sigma \in \mathscr{A}_{\mathcal{L}_\oplus}$, $\boldsymbol{P}_\sigma$ is the set of all programs of type σ.*

As we have explained in Chapter 4, the primary requirement enforced in the literature on an equivalence relation R on programs is *soundness* with respect to observational equivalence, i.e. that $M \equiv^{\mathrm{ctx}}_{\mathcal{L}_\oplus} N$ whenever $(M,N) \in R$. We give here the quantitative counterpart of this requirement: we ask to a distance μ to move apart programs *at least as much as* the observational distance does, in the sense formalized in Definition 8.1.2 below.

Definition 8.1.2 *Let $\mathcal{L}_\oplus$ be an observable programming language, and μ a distance on programs. We say that μ is* sound with respect to the observational distance *when the following conditions hold:*

- *if $\mathcal{L}_\oplus$ is an untyped language, we ask that for every programs M and N, $\delta^{ctx}(M,N) \leq \mu(M,N)$.*
- *if $\mathcal{L}_\oplus$ is a typed language, we ask that for every $\sigma \in \mathscr{A}$, $M, N \in \boldsymbol{P}_\sigma$, $\delta^{ctx}_\sigma(M,N) \leq \mu_\sigma(M,N)$.*

Observe that this soundness requirement on distances is a generalization of soundness for equivalence relations, in the sense that whenever a pseudo-metric is sound w.r.t observational distance, then its kernel is sound w.r.t. observational equivalence. Proving soundness of an equivalence relation on programs for some higher-order language usually consists in proving both observational correctness and compositionality; this approach was for instance followed [32] for trace equivalence and applicative probabilistic bisimilarity for CBN $\Lambda_\oplus$, and we also illustrated it here when proving soundness of applicative probabilistic bisimilarity for CBV $\Lambda_\oplus$ in Chapter 5. We first look at how to generalize *observational correctness* to the quantitative case. Recall that an equivalence relation R is observationally correct when:

$$M \, R \, N \Rightarrow \mathrm{Obs}(M) = \mathrm{Obs}(N).$$

It means that whenever M and N are equivalent with respect to R, the dummy context $[\cdot]$ that simply executes its argument should not be able to distinguish between M and N. Similarly, we say that a distance μ is *observationally correct* when it is *at least* as informed as this dummy context, in the sense that it should separate programs at least as much.

Definition 8.1.3 *Let $\mathcal{L}_\oplus$ be an observable programming language, and μ a distance on programs. We say that μ is an* observationally correct distance *if the following conditions hold:*

- *if $\mathcal{L}_\oplus$ is an untyped language, we ask that for all programs $M, N \in \boldsymbol{P}$, $\mu(M,N) \geq |Obs(M) - Obs(N)|$;*
- *if $\mathcal{L}_\oplus$ is a typed language, we ask that for every observable type $\sigma \in \mathscr{A}_{obs}$, and every programs $M, N \in \boldsymbol{P}_\sigma$: $\mu_\sigma(M,N) \geq |Obs_\sigma(M) - Obs_\sigma(N)|$., where $\mathscr{A}_{obs}$ is the set of observable types of $\mathcal{L}_\oplus$.*

Observe that observational correctness is weaker than soundness, since being sound means to be at least as informed as *all* possible contexts. As developed in Chapter 5—see Proposition 5.1.2—if a relation is an observationally correct equivalence, then we know it is sound with respect to observational equivalence *as soon as* it is also compositional, i.e. :

$$M \, R \, N \wedge \mathcal{C} \text{ a context} \quad \Rightarrow \quad (\mathcal{C}[M]) \, R \, (\mathcal{C}[N]).$$

We now want to generalize compositionality to the quantitative case. We choose here to ask for the distance μ to be *non-expansive*, i.e. that an environment cannot increase the μ-distance between two programs by filling them into some context. It is the notion which is used for instance by Reed and Pierce [97] to do compositional reasoning for *differential privacy*, as well as by Azevedo et al [6] when developing denotational semantics that are *distance-based*. We formalize in our setting this non-expansiveness requirement below.

Definition 8.1.4 *Let $\mathcal{L}_\oplus$ be an observable programming language, and μ a distance on programs. We say that μ is* non-expansive*, when:*

- *if $\mathcal{L}_\oplus$ is an untyped language, we ask that for every programs $M, N \in \boldsymbol{P}$, and every context $\mathcal{C}$, it holds that $\mu(\mathcal{C}[M], \mathcal{C}[N]) \leq \mu(M, N)$;*
- *if $\mathcal{L}_\oplus$ is a typed language, we ask that for every $\sigma, \tau \in \mathscr{A}$, every programs $M, N \in \boldsymbol{P}_\sigma$, and every context $\mathcal{C} \in \mathbf{C}_{(\emptyset,\sigma)\to\tau}$, it holds that $\mu_\tau(\mathcal{C}[M], \mathcal{C}[N]) \leq \mu_\sigma(M, N)$.*

It can be interesting to note that non-expansiveness is not the only quantitative generalization of compositionality considered in the literature. Gebler and Tini [49], for instance, ask in their work on distances on a *first-order* probabilistic language for *uniform continuity*, i.e. that if we fix a context $\mathcal{C}$, and an objective $\varepsilon \in [0, 1]$, to be sure that $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are not farther away than our objective ε, it is enough to know that M and N are closer than some quantity δ. Formally, it means that for every small $\varepsilon \in (0, 1]$, and for every context $\mathcal{C}$, there exists a $\delta \in [0, 1]$, such that for *all* programs M, N $\mu(M, N) \leq \delta \Rightarrow \mu(\mathcal{C}[M], \mathcal{C}[N]) \leq \varepsilon$. Observe that our non-expansiveness requirement is *stronger* than uniform continuity.

In Proposition 5.1.2 of Chapter 5, we showed that when an equivalence relation on program is both observationally correct and compositional, then it is sound with respect to the observational equivalence. We prove now in Proposition 8.1.1 below that the quantitative counterpart is also true, i.e. as soon as we have established both observational correctness and non-expansiveness for some distance μ on programs, we obtain that μ is sound with respect to the observational distance. Observe that the non-expansiveness requirement is also interesting *on its own*, since it allows us to reason by compositionality when computing the distance between two programs.

Proposition 8.1.1 *Let $\mathcal{L}_\oplus$ be an observable programming language, and μ a distance on programs. If μ is both observationally correct and non-expansive, then it is sound with respect to the observational distance.*

Proof. We show Proposition 8.1.1 in the case where $\mathcal{L}_\oplus$ is an untyped language, but the reasoning is similar in the typed case. Let μ be a distance on programs, that is both observationally correct and non-expansive. Let M, N be two programs. Recall the definition of observational distance:

$$\delta^{\mathrm{ctx}}(M, N) = \sup_{\mathcal{C} \text{ a context}} |\mathrm{Obs}(\mathcal{C}[M]) - \mathrm{Obs}(\mathcal{C}[N])|.$$

It means that it is enough to show that for every context $\mathcal{C}$, $|\mathrm{Obs}(\mathcal{C}[M]) - \mathrm{Obs}(\mathcal{C}[N])| \leq \mu(M, N)$. Since μ is observationally correct, we see that $|\mathrm{Obs}(\mathcal{C}[M]) - \mathrm{Obs}(\mathcal{C}[N])| \leq \mu(\mathcal{C}[M], \mathcal{C}[N])$, and since μ is non-expansive, $\mu(\mathcal{C}[M], \mathcal{C}[N]) \leq \mu(M, N)$. By combining these two inequalities, we can conclude the proof. □

8.2 Bisimilarity Distance for WLTSs

Recall that our presentation of sound equivalences for $\Lambda_\oplus$ in Chapter 4 and Chapter 5 relied on built-in notions of equivalence on various classes of quantitative transition systems. As highlighted there, WLTSs—Weighted Labeled Transition Systems, that we have introduced in Section 4.2.1 of Chapter 4—is the class of transition systems that allows to express trace equivalences for probabilistic higher-order languages. In this section, we focus on WLTSs, and we see that bisimilarity on WLTS can be generalized in a natural way into *bisimilarity distance*. Recall that we have gave two equivalent definitions of

bisimilarity for WLTSs: the first one is inherently coinductive, while the other one characterizes bisimilarity inductively by using linear tests. We follow a similar path here: we first give a coinductive definition of bisimilarity distance, and then we show that we can also characterize it by linear tests.

8.2.1 A Coinductive Definition of Bisimilarity Distance for WLTSs.

We look here for a quantitative generalization of the coinductive definition of WLTS-bisimilarity as the largest WLTS-bisimulation—see Definition 4.2.5 of Chapter 4. Observe that a necessary condition for two states s and t to be bisimilar is that $w(s)$ and $w(t)$ must be *equal*. When dealing with distances, however, we want to be able to express the fact that two states —that can possibly have *distinct* images by $w(\cdot)$—are *not farther away* than some small quantity ε. This idea leads us to a relaxed notion of bisimulation: for each $\varepsilon \in [0,1]$, we define below the *ε-bisimulation requirement*, designed to guarantee that whenever two states s, t are related, their w-valuation will never be farther away than ε no matter the evolution of the system.

Definition 8.2.1 *Let $\mathscr{L}^w = (w, (\mathcal{S}, \mathcal{L}, \rightarrow))$ be a WLTS. Let R be a binary relation on $\mathcal{S}$, and $\varepsilon \in [0,1]$. We say that R is a ε-bisimulation whenever the following two conditions hold:*

1. *R is a bisimulation on the LTS $\mathscr{L} = (\mathcal{S}, \mathcal{L}, \rightarrow)$;*
2. *if $s\,R\,t$, then $|w(s) - w(t)| \leq \varepsilon$.*

When only the condition (2) *holds, we will say that the relation R is* ε-bounded. *For every $\varepsilon \in [0,1]$, there exists a largest ε-bisimulation, that we indicate as $R^{\varepsilon}_{\mathscr{L}^w}$, and call* ε-bisimilarity on $\mathscr{L}^{\mathrm{w}}$.

Observe that a 0-bisimulation is simply a WLTS-bisimulation, so the 0-bisimilarity $R^0_{\mathscr{L}^{\mathrm{w}}}$ coincides with the bisimilarity on $\mathscr{L}^{\mathrm{w}}$: in that sense our relaxed notion of ε-bisimilarity is indeed a quantitative extension of bisimilarity. If we now look at the 1-bisimilarity, we see that since the image of w is contained in $[0,1]$, $R^1_{\mathscr{L}^{\mathrm{w}}}$ is the bisimilarity on the underlying LTS $\mathscr{L}$.In particular, it is an equivalence relation: we say that two states are *comparable* when they are in the same equivalence class for $R^1_{\mathscr{L}^{\mathrm{w}}}$, and we denote by $\mathscr{E}(\mathscr{L}^{\mathrm{w}})$ the set of equivalence classes for $R^1_{\mathscr{L}^{\mathrm{w}}}$. We will often in the following ask binary relations on states to be *clustered*, i.e. to respect the equivalence class of R^1. We formalize this notion in Definition 8.2.2 below.

Definition 8.2.2 *Let $\mathscr{L}^w$ be a WLTS. We say that a relation R is* clustered *if $s\,R\,t$ implies that there exists $E \in \mathscr{E}(\mathscr{L})$, such that both $s \in E$ and $t \in E$.*

When R is a clustered relation, we use the following notation: for every $E \in \mathscr{E}(\mathscr{L}^{\mathrm{w}})$, we will denote $s\,R_E\,t$ to mean that $s, t \in E$, and $s\,R\,t$. Observe that as soon as R is a ε-bisimulation for some $\varepsilon \in [0,1]$, it holds that R is a clustered binary relation. If we look now at the ε-relaxed version of bisimilarity $R^{\varepsilon}_{\mathscr{L}^{\mathrm{w}}}$, we see that it is not necessarily an equivalence relation (since it is not transitive). It is however an reflexive and symmetric binary relation, as we show below:

Lemma 8.2.1 *For every $\varepsilon \in [0,1]$, it holds that $R^{\varepsilon}_{\mathscr{L}^w}$ is reflexive and symmetric.*

Proof. The proof is similar to the one we did to show that the bisimilarity on some LTS is an equivalence relation: we show that $\{(s,s) \mid s \in \mathcal{S}\}$ is a 0-bisimulation; and that if R is a ε bisimulation, then also R^{-1} is a ε-bisimulation. □

We illustrate on an example the notion of ε-bisimulation, by looking at the WLTS $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$, which is the applicative WLTS for CBN $\Lambda_\oplus$ that we used to present trace equivalence in Chapter 4. We want to compare the two programs $M = \lambda x.I$, and $N = (\lambda x.I \oplus \lambda x.\Omega)$. Recall that we have shown—in Example 4.2.4 of Chapter 4—that their corresponding states $s(M) := \{M^1\}$ and $s(N) := \{N^1\}$ are not bisimilar on $\mathscr{L}({\Lambda_\oplus}^{\text{cbn}})$; in other words there is no 0-bisimulation that connects them. We show in Example 8.2.1 below that we can however build a $\frac{1}{2}$-bisimulation that connects those states.

Example 8.2.1 *We consider the two $\mathscr{L}({\Lambda_\oplus}^{cbn})$-states $s(M) := \{M^1\}$ and $s(N) := \{N^1\}$, with $M = \lambda x.I$, and $N = (\lambda x.I \oplus \lambda x.\Omega)$. We represent the fragment of $\mathscr{L}({\Lambda_\oplus}^{cbn})$ relevant for these two states in Figure 4.3 of Chapter 4. We build now a $\frac{1}{2}$-bisimulation that connect $s(M)$ and $s(N)$: recall that the set of states for this WLTS is $\mathcal{S}_{\mathscr{L}({\Lambda_\oplus}^{cbn})} = \Delta(\boldsymbol{P}_{\Lambda_\oplus}) \uplus \Delta(\widehat{\mathcal{V}}_{\Lambda_\oplus})$, i.e. consists of both the distributions over the set of $\Lambda_\oplus$ terms, and the distributions over the set of $\Lambda_\oplus$ distinguished values; we define a relation $R \subseteq \mathcal{S}_{\mathscr{L}({\Lambda_\oplus}^{cbn})} \times \mathcal{S}_{\mathscr{L}({\Lambda_\oplus}^{cbn})}$ as:*

$$R := \{(\{M^1\},\{N^1\}),(s_1,t_1),(s_2,t_2)\} \cup \{(\mathscr{D}, \frac{1}{2}\cdot\mathscr{D}) \mid \mathscr{D} \in \Delta(\boldsymbol{P}_{\Lambda_\oplus}) \uplus \Delta(\widehat{\mathcal{V}}_{\Lambda_\oplus})\},$$

where s and t are as represented in Figure 4.3. We can then check that R is indeed a $\frac{1}{2}$-bisimulation over $\mathscr{L}({\Lambda_\oplus}^{cbn})$.

Intuitively, each of the ε-bisimilarity $R^\varepsilon_{\mathscr{L}^{\text{w}}}$ is designed to contain all the pairs (s,t) such that the states s and t are *not farther away* as ε. When looking at this the other way around, it means that the distance between s and t should be the smallest ε such that $(s,t) \in R^\varepsilon_{\mathscr{L}^{\text{w}}}$. Accordingly with this intuition, we define now the *bisimilarity distance* on the states of a WLTS.

Definition 8.2.3 *Let be $\mathscr{L}^w = (w,(\mathcal{S},\mathcal{L},\to))$ a WLTS. We define the* bisimulation distance for the WLTS $\mathscr{L}^{\text{w}}$ *as the family of quantitative valuations $(\delta^{\mathscr{L}^w}_E)_{E\in\mathscr{E}(\mathscr{L}^w)}$ where $\delta^{\mathscr{L}^w}_E : E \times E \to [0,1]$ is defined as:*

$$\delta^{\mathscr{L}^w}_E(s,t) = \inf\{\varepsilon \mid s\, R^\varepsilon_{\mathscr{L}^w}\, t\}.$$

Observe that the definition above is indeed well-posed, since whenever s and t are comparable, the set $\{\varepsilon \mid s\, R^\varepsilon_{\mathscr{L}^{\text{w}}}\, t\} \subseteq [0,1]$ contains at least 1—since $s\, R^1_{\mathscr{L}^{\text{w}}}\, t$ by definition of comparable states—so it is not empty.

Lemma 8.2.2 *For every class $E \in \mathscr{E}(\mathscr{L}^w)$, it holds that $\delta^{\mathscr{L}^w}_E$ is a pseudo-metric.*

Proof. Using Lemma 8.2.1, we see that $\delta^{\mathscr{L}^{\text{w}}}_E$ is reflexive and symmetric. To see that the triangular inequality holds, we use the fact that if R is a ε_1-bisimulation, and S is a ε_2 bisimulation, then $R \circ S$—i.e. the composition of R and S, see its definition in Chapter 4—is a $(\varepsilon_1+\varepsilon_2)$-bisimulation. □

Moreover, the infinum in the definition of $\delta^{\mathscr{L}^{\text{w}}}_E$ is actually a *minimum*, as we show in Lemma 8.2.3 below.

Lemma 8.2.3 *Let $\mathscr{L}^w = (w,(\mathcal{S},\mathcal{L},\to))$ be a WLTS, $E \in \mathscr{E}(\mathscr{L}^w)$, and $s,t \in E$. Then $\delta^{\mathscr{L}^w}_E(s,t) \le \varepsilon$ if and only if $s\, R^\varepsilon_{\mathscr{L}^w}\, t$.*

Proof. The right-to-left implication is given by the definition of $\delta^{\mathscr{L}^{\text{w}}}_E$. The other one comes from the fact that for every $\varepsilon \in [0,1]$, the relation $R := \{(s,t) \mid \delta^{\mathscr{L}^{\text{w}}}_E(s,t) \le \varepsilon\}$ is a ε-bisimulation. □

The coinductive definition of the bisimilarity distance is well-adapted to show *upper bounds* on the distance between two states: indeed, as we illustrate in the Example below, it is enough to build *one* ε-bisimulation that contains (s,t) in order to show that $\delta_E^{\mathscr{L}^{\mathrm{w}}}(s,t) \leq \varepsilon$.

Example 8.2.2 *We consider once again the WLTS* $\mathscr{L}({\Lambda_\oplus}^{cbn})$. *Observe first that* $R^1_{\mathscr{L}({\Lambda_\oplus}^{cbn})}$ *has two equivalence classes:* $\mathscr{E}(\mathscr{L}({\Lambda_\oplus}^{cbn})) = \{E_{\boldsymbol{P}}, E_{\widehat{\mathcal{V}}}\}$ *with* $E_{\boldsymbol{P}} := \Delta(\boldsymbol{P}_{\Lambda_\oplus})$, $E_{\widehat{\mathcal{V}}} := \Delta(\widehat{\mathcal{V}}_{\Lambda_\oplus})$. *It means that we can compare either two states that are distributions over programs, or two states that are distributions over distinguished values. Recall that we have shown in Example 8.2.1 the existence of a* $\frac{1}{2}$*-bisimulation, that connects* $\{(\lambda x.I) \oplus (\lambda x.\Omega)^1\}$ *and* $\{(\lambda x.I)^1\}$. *As a consequence:*

$$\delta_{E_{\boldsymbol{P}}}^{\mathscr{L}({\Lambda_\oplus}^{cbn})}(\{(\lambda x.I) \oplus (\lambda x.\Omega)^1\}, \{(\lambda x.I)^1\}) \leq \frac{1}{2}.$$

8.2.2 An Inductive Characterization of Bisimilarity Distance for WLTS

In Section 4.2.1 of Chapter 4, we also characterized inductively bisimilarity for WLTSs, by using linear tests that are represented by traces, i.e. finite sequences of actions. Recall that for each state s, we defined $\mathcal{T}(s)$ the set of traces that we can execute with s as point of departure, and for every trace $\alpha \in \mathcal{T}(s)$, we defined the success probability $\mathrm{Prob}(\alpha\downarrow)(s)$ designed to represent the probability that α succeeds when applied to the state s. Then we showed—in Proposition 4.2.3—that two states are bisimilar if and only if they have the same set of admissible traces, and for each such trace the probability of success is the same for the two states. Here, we keep the same class of tests, and we show that they also characterize the bisimilarity distance: indeed, as stated formally in Proposition 8.2.4 below, the bisimilarity distance between two comparable states s and t coincides with the supremum of the *separation* induced by any trace α on (s,t).

Proposition 8.2.4 *Let* $\mathscr{L}^w = (w, (\mathcal{S}, \mathcal{L}, \rightarrow))$ *be a WLTS. For any set of traces* $A \subseteq \mathcal{T}_{\mathscr{L}^w}$, *we define:* $E_A := \{s \mid \mathcal{T}(s) = A\}$. *Then it holds that the equivalence classes of* $R^1_{\mathscr{L}^w}$ *are the* E_A, *i.e. more precisely* $\mathscr{E}(\mathscr{L}^w) = \{E_A \mid A \subseteq \mathcal{T}_{\mathscr{L}^w}, E_A \neq \emptyset\}$. *Moreover, when* $A \subseteq \mathcal{T}_{\mathscr{L}^w}$ *and* $s,t \in E_A$, *it holds that:*

$$\delta_{E_A}^{\mathscr{L}^w}(s,t) = \sup_{\alpha \in A} |\mathit{Prob}(\alpha\downarrow)(s) - \mathit{Prob}(\alpha\downarrow)(t)|.$$

Proof. We show separately the two inequalities:

- First, if two states s and t are related by an ε-bisimulation, we can see that their set of admissible traces coincide, and moreover $|\mathrm{Prob}(\alpha\downarrow)(s) - \mathrm{Prob}(\alpha\downarrow)(t)| \leq \varepsilon$ (the proof is by induction on the length of the trace α, since the states obtained after having done every action in α are still ε-bisimilar). As a consequence: $\delta_{E_A}^{\mathscr{L}^{\mathrm{w}}}(s,t) \geq \sup_{\alpha\in A}|\mathrm{Prob}(\alpha\downarrow)(s) - \mathrm{Prob}(\alpha\downarrow)(t)|$.
- To show the other implication, we define for every ε the binary relation on states $R_\varepsilon := \{(s,t) \mid \exists A \subseteq \mathcal{T}_{\mathscr{L}^{\mathrm{w}}}, s,t \in E_A \wedge \forall \alpha \in A, |\mathrm{Prob}(\alpha\downarrow)(s) - \mathrm{Prob}(\alpha\downarrow)(t) \leq \varepsilon\}$. We can see that R_ε is indeed a ε-bisimulation, by remarking that if $s \xrightarrow{a} s'$, it holds that $\mathcal{T}(s') = \{\alpha \mid a\cdot\alpha \in \mathcal{T}(s)\}$, and for $\alpha \in \mathcal{T}(s')$, $\mathrm{Prob}(\alpha\downarrow)(s') = \mathrm{Prob}(a\cdot\alpha\downarrow)(s)$. It means that two states s, t with the same admissible traces are ε-bisimilar as soon as the difference between their success probabilities for traces is never greater that ε. As a consequence, $\delta_{E_A}^{\mathscr{L}^{\mathrm{w}}}(s,t) \leq \sup_{\alpha\in A}|\mathrm{Prob}(\alpha\downarrow)(s) - \mathrm{Prob}(\alpha\downarrow)(t)|$.

□

An immediate corollary of Proposition 8.2.4 is that $\mathcal{T}(s)$ and $\mathcal{T}(t)$ coincide as soon as s, t are comparable. Accordingly, we will denote for $E \in \mathscr{E}(\mathscr{L}^{\mathrm{w}})$, $\mathcal{T}(E) := \mathcal{T}(s)$ with s any element of E. Recall that the coinductive definition of bisimilarity distance is well-adapted to show *upper bounds* on the distance between states, as we have illustrated in Example 8.2.2. By contrast, the inductive characterization allows to easily show lower bounds, as we can see in Example 8.2.3 below.

Example 8.2.3 *We consider once again the $\Lambda_{\oplus}$ programs $M := \lambda x.I$, and $N := \lambda x.I \oplus \lambda x.\Omega$, and we look at what we can learn using the trace characterization of the bisimilarity distance on the WLTS $\mathscr{L}({\Lambda_{\oplus}}^{cbn})$. Recall that we have shown in Example 4.2.4 of Chapter 4 the existence of a trace α with $Prob(\alpha\downarrow)((\{M^1\})) = 1$, while $Prob(\alpha\downarrow)((\{N^1\})) = \frac{1}{2}$. As a consequence, Proposition 8.2.4 tells us that:*

$$\delta_{E_{\boldsymbol{P}}}^{\mathscr{L}({\Lambda_{\oplus}}^{cbn})}(\{M^1\}, \{N^1\}) \geq \frac{1}{2}.$$

By combining this result with the reverse inequality that we have shown in Example 8.2.2, we see that the bisimilarity distance between $\{M^1\}$ and $\{N^1\}$ in the WLTS $\mathscr{L}({\Lambda_{\oplus}}^{cbn})$ is $\frac{1}{2}$.

8.3 The Trace Distance in $\Lambda_{\oplus}^{\leq 1}$.

We want now to build well-behaved—i.e. at least sound—distances for higher-order probabilistic programming languages, by using the built-in notion of bisimilarity distance for WLTSs. Since they are characterized by linear tests, and by analogy with trace equivalences, we will call such distances *trace distances*. At this point, a natural thing to do would be to look once again at the language CBN $\Lambda_{\oplus}$, since we know already that the WLTS $\mathscr{L}({\Lambda_{\oplus}}^{\mathrm{cbn}})$ leads to a sound—and even fully abstract—notion of trace equivalence for this language, as shown by Dal Lago, Sangiorgi, Alberti [32]. However, a major problem occurs when we try to do so: the resulting trace distance on programs would be *unsound* with respect for the context distance on $\Lambda_{\oplus}$, as we state in Lemma 8.3.1 below.

Lemma 8.3.1 *There exist M, N programs in $\Lambda_{\oplus}^{cbn}$ such that $\delta^{\mathscr{L}({\Lambda_{\oplus}}^{cbn})}(\{M^1\}, \{N^1\}) < \delta^{ctx}(M, N)$.*

Proof. We take $M = (\lambda x.I) \oplus (\lambda x.\Omega)$ and $N = \lambda x.I$. We can see from Example 8.2.3 that the bisimilarity distance—on the WLTS $\mathscr{L}({\Lambda_{\oplus}}^{cbn})$—between $\{M^1\}$ and $\{N^1\}$ is equal to $\frac{1}{2}$. However, as illustrated in Chapter 3, the context distance between these two $\Lambda_{\oplus}$ programs is 1—for both the CBV and the CBN reduction theory: it is the usual trivialization phenomenon. □

As argued in Chapter 6, trivialization arises primarily from the copying ability of the language. For this reason, as a first step to look at how we can build trace distances for higher-order probabilistic languages, we ban copying altogether, by considering the affine language $\Lambda_{\oplus}^{\leq 1}$—that we introduced in Section 7.4 of Chapter 7. So we first build a WLTS that expresses interactively the operational semantics for this affine language. There are two main differences with respect to the WLTS for CBN $\Lambda_{\oplus}$. First, since the reduction strategy on $\Lambda_{\oplus}^{\leq 1}$ is *call-by-value*, the applicative labels are going to be only values, instead of all programs. Second, the affinity constraint on the language enforces that the operational semantics of a program has always *finite support*—see Corollary 7.4.1 in Chapter 7—so the states of the WLTS are not all sub-distributions over programs, but only those sub-distributions that have finite support.

Definition 8.3.1 *We define the* affine LTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1}) := (\mathcal{S}_{\Lambda_{\oplus}^{\leq 1}}, \mathcal{L}_{\Lambda_{\oplus}^{\leq 1}}, \to)$ *by taking its set of states as:* $\mathcal{S}_{\Lambda_{\oplus}^{\leq 1}} := \{\mathscr{D} \mid \mathscr{D} \in \Delta(\boldsymbol{P}_{\Lambda_{\oplus}^{\leq 1}}) \uplus \Delta(\widehat{\mathcal{V}}_{\Lambda_{\oplus}^{\leq 1}}) \wedge \mathscr{D} \text{ has finite support.}\}$*, its set of labels as* $\mathcal{L}_{\Lambda_{\oplus}^{\leq 1}} := \mathcal{V}_{\Lambda_{\oplus}^{\leq 1}} \uplus \{eval\}$*, and the transition relation* $\to$ *as follows:*

- $\forall \mathscr{D} \in \Delta(\boldsymbol{P}_{\Lambda_{\oplus}^{\leq 1}}), \quad \mathscr{D} \xrightarrow{eval} \sum_{M \in \boldsymbol{P}_{\Lambda_{\oplus}^{\leq 1}}} \mathscr{D}(M) \cdot \widehat{[\![M]\!]}$;
- $\forall \mathscr{D} \in \Delta(\widehat{\mathcal{V}}_{\Lambda_{\oplus}^{\leq 1}}), V \in \mathcal{V}_{\Lambda_{\oplus}^{\leq 1}}, \quad \mathscr{D} \xrightarrow{V} \sum_{\lambda x.M \in \mathcal{V}_{\Lambda_{\oplus}^{\leq 1}}} \mathscr{D}(\widehat{\lambda x.M}) \cdot \{M\{V/x\}^1\}$.

We turn $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ *into a WLTS by adding a weight function* $w : \mathcal{S}_{\Lambda_{\oplus}^{\leq 1}} \to [0,1]$*, taken as:* $w(\mathscr{D}) := |\mathscr{D}|$.

We first want to know which states s, t of $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ are *comparable*, i.e. what are the equivalence classes in $\mathscr{E}(\mathscr{L}(\Lambda_{\oplus}^{\leq 1}))$. As stated in Lemma 8.3.2 below, the situation is similar to the case of $\mathscr{L}({\Lambda_{\oplus}}^{\text{cbn}})$: we can compare two states when either they are both sub-distribution over programs, or they are both sub-distributions over distinguished values. Recall that we called $E_{\mathbf{P}}$ and $E_{\widehat{\mathcal{V}}}$ the two equivalence classes of $R^1_{\mathscr{L}({\Lambda_{\oplus}}^{\text{cbn}})}$. To keep the notations readable, we will also write $E_{\mathbf{P}}$ and $E_{\widehat{\mathcal{V}}}$ for the two elements in $\mathscr{E}(\mathscr{L}(\Lambda_{\oplus}^{\leq 1}))$, without specifying the language considered—unless otherwise specified, in the following we will be referring to the affine language.

Lemma 8.3.2 *The equivalence relation* $R^1_{\mathscr{L}(\Lambda_{\oplus}^{\leq 1})}$ *has two equivalence classes, that are* $E_{\boldsymbol{P}} := \{\mathscr{D} \in \Delta(\boldsymbol{P}_{\Lambda_{\oplus}^{\leq 1}}) \mid \mathscr{D} \text{ has finite support}\}$ *and* $E_{\widehat{\mathcal{V}}} := \{\mathscr{D} \in \Delta(\widehat{\mathcal{V}}_{\Lambda_{\oplus}^{\leq 1}}) \mid \mathscr{D} \text{ has finite support}\}$.

Proof. The proof is similar to the one we did for $\mathscr{L}({\Lambda_{\oplus}}^{\text{cbn}})$—see Example 8.2.2 in Section 8.2. □

We now associate to every program M in $\Lambda_{\oplus}^{\leq 1}$ a $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$-state—in the same way as when characterizing trace equivalence for CBN $\Lambda_{\oplus}$, by taking $s(M) := \{M^1\} \in E_{\mathbf{P}}$. From there, we define the *trace distance* between two programs as the bisimilarity distance on $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$—as given in Definition 8.2.3—between their associated states.

Definition 8.3.2 *We define the* applicative trace distance *as the quantitative valuation* $\delta^{tr}_{\Lambda_{\oplus}^{\leq 1}} : \boldsymbol{P}_{\mathscr{L}(\Lambda_{\oplus}^{\leq 1})} \times \boldsymbol{P}_{\mathscr{L}(\Lambda_{\oplus}^{\leq 1})} \to [0,1]$ *taken as:*

$$\delta^{tr}_{\Lambda_{\oplus}^{\leq 1}}(M, N) := \delta^{\mathscr{L}(\Lambda_{\oplus}^{\leq 1})}_{E_{\boldsymbol{P}}}(\{M^1\}, \{N^1\}).$$

Observe that $\delta^{\text{tr}}_{\Lambda_{\oplus}^{\leq 1}}$ is indeed a distance on programs for the untyped observable language $\Lambda_{\oplus}^{\leq 1}$—in the sense of Definition 8.1.1—since Lemma 8.2.2 tells us it is a pseudo-metric. Recall that in Example 8.2.3 we saw that the trace distance between the CBN $\Lambda_{\oplus}$-programs $\lambda x.I$ and $(\lambda x.I \oplus \lambda x.\Omega)$—i.e. the bisimilarity distance between their associated states in the WLTS $\mathscr{L}({\Lambda_{\oplus}}^{cbn})$—is $\frac{1}{2}$. In Example 8.3.1 below we show that the situation is similar when we look at those programs as $\Lambda_{\oplus}^{\leq 1}$-programs.

Example 8.3.1 *We consider here the* $\Lambda_{\oplus}^{\leq 1}$ *programs* $M := (\lambda x.I \oplus \lambda x.\Omega)$ *and* $N := \lambda x.I$*—recall that* Ω *is here a primitive construct of* $\Lambda_{\oplus}^{\leq 1}$*, designed as a tool to add divergence to our simplified affine language. To compute the trace distance between* M *and* N*,*

we need to consider the $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$-states $s = \{M^1\}$, and $t = \{N^1\}$. We can show that the bisimilarity distance between s and t is $\frac{1}{2}$: the proof is similar to the one we did for s and t seen as states of $\mathscr{L}({\Lambda_{\oplus}}^{cbn})$, in Example 8.2.2 and Example 8.2.3. From there, we conclude:

$$\delta^{tr}_{\Lambda_{\oplus}^{\leq 1}}(M, N) = \frac{1}{2}.$$

8.3.1 A Roadmap to Soundness.

We want now to show that the trace distance on $\Lambda_{\oplus}^{\leq 1}$ is sound with respect to the observational distance, i.e:

$$\forall M, N \text{ programs}, \quad \delta^{\mathrm{ctx}}_{\Lambda_{\oplus}^{\leq 1}}(M, N) \leq \delta^{\mathrm{tr}}_{\Lambda_{\oplus}^{\leq 1}}(M, N).$$

To do that, we use Proposition 8.1.1—see Section 8.1—that tells us it is enough to show that the trace distance is both observationally correct and non-expansive. The first step is easily done, by using the inductive characterization by linear tests.

Proposition 8.3.3 *The distance on programs $\delta^{tr}_{\Lambda_{\oplus}^{\leq 1}}$ is observationally correct for the language $\Lambda_{\oplus}^{\leq 1}$.*

Proof. We can see it immediately by using the inductive characterization of bisimilarity distance on the WLTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$. Indeed, for every program M, $\mathrm{Obs}(M)$ can be computed as the success probability of the empty trace ϵ on the state s_M . □

The proof of non-expansiveness, however, is considerably more involved. Since the distance on programs is actually defined using the bisimilarity distance on a WLTS, the first step consists in expressing non-expansiveness directly as a WLTS-property. More precisely, we extend the notion of *contexts*, by transforming the $\Lambda_{\oplus}^{\leq 1}$-contexts—that represent an algorithm that takes in input a program and returns another program—into *state-contexts* that are algorithm that take in input a state of $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ and return another state. Since the structure of this non-expansiveness proof is similar to the one of the proof for the trace distance on $\Lambda_{\oplus}^{!}$—that we will introduce in the next section—we will present the next steps in the setting of a generic WLTS *equipped with state-contexts*, in the following sense.

Definition 8.3.3 (WLTS equipped with States-Contexts) *Let $\mathscr{L}^w$ be a WLTS. We say that we* equip $\mathscr{L}^{\mathrm{w}}$ with state-contexts *when specifying for each pair (E_1, E_2) of elements in $\mathscr{E}(\mathscr{L}^w)$, a set of functions $E_1 \to E_2$, that we denote $\mathbf{C}(E_1 \to E_2)$, such that:*

$$\forall \mathcal{C} \in \mathbf{C}(E_1 \to E_2), \forall s \in E_1,\ w(\mathcal{C}(s)) = w(s).$$

By analogy with the contexts of a programming language, we will use $\mathcal{C}, \mathcal{D} \ldots$ to denote the elements of $\mathbf{C}(E_1 \to E_2)$, and if $s \in E_1$, we will note $\mathcal{C}[s]$ for $\mathcal{C}(s)$. There is a slight abuse in notation here, in the sense that it also depends on the type B that we specify as output type: the same state-context can be both in $\mathbf{C}(A \to B_1)$ and $\mathbf{C}(A \to B_2)$. We are now ready to formalize the *non-expansiveness property* on *WLTSs* that are equipped with states-contexts: we ask that no state-context can increase the bisimilarity distance between two states.

Definition 8.3.4 (Non-Expansiveness for WLTS) *Let $\mathscr{L}^w$ be a WLTS equipped with state contexts. We say that $\mathscr{L}^w$ is* non-expansive*, if for every s, t in the same equivalence class E, for any state-context $\mathcal{C} \in \mathbf{C}(E \to E')$, it holds that:*

$$\delta_{E'}^{\mathscr{L}^w}(\mathcal{C}[s], \mathcal{C}[t]) \leq \delta_{E}^{\mathscr{L}^w}(s, t).$$

We now equip our WLTs $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ with state-contexts: in this case, a state-context is simply a context of the language $\Lambda_{\oplus}^{\leq 1}$. Recall that here the states are sub-distributions over $\Lambda_{\oplus}^{\leq 1}$-programs; in this setting, applying a context to a state consists in applying it to all the programs in the support of the sub-distribution. We have however to be careful, because we have two equivalence classes in $\mathscr{E}(\mathscr{L}(\Lambda_{\oplus}^{\leq 1}))$: the class $E_{\mathbf{P}}$, that contains those states that are sub-distributions over programs, and the class $E_{\mathcal{V}}$ that contains those states that are sub-distribution over distinguished values. It means that when considering $\mathbf{C}(E_{\mathbf{P}} \to E_{\mathcal{V}})$, we keep only the $\Lambda_{\oplus}^{\leq 1}$ contexts that return always a value—even when they are filled by a program that is not a value. We sum up these considerations in Definitions 8.3.5 and 8.3.6 below.

Definition 8.3.5 *For every pair (E_1, E_2) of equivalence classes in $\mathscr{E}(\mathscr{L}(\Lambda_{\oplus}^{\leq 1}))$, we define the set of* state-contexts with input E_1 and output in E_2*—denoted $\mathbf{C}(E_1 \to E_2)$—as follows:*

- *If $E_2 = E_{\boldsymbol{P}}$, $\mathbf{C}(E_1 \to E_2) := \mathbf{C}\Lambda_{\oplus}^{\leq 1}$;*
- *If $E_1 = E_2 = E_{\widehat{\mathcal{V}}}$, $\mathbf{C}(E_1 \to E_2) := \{\mathcal{C} \in \mathbf{C}\Lambda_{\oplus}^{\leq 1} \mid \mathcal{C} = [\cdot] \vee \mathcal{C} = \lambda x.\mathcal{D}\}$;*
- *If $E_1 = E_{\boldsymbol{P}}$, and $E_2 = E_{\widehat{\mathcal{V}}}$, $\mathbf{C}(E_1 \to E_2) := \{\mathcal{C} \in \mathbf{C}\Lambda_{\oplus}^{\leq 1} \mid \mathcal{C} = \lambda x.\mathcal{D}\}$.*

We now formalize what happens when we fill a $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ state-context in $\mathbf{C}(E_1 \to E_2)$ by a state in E_1.

Definition 8.3.6 *Let E_1, E_2 be two equivalence classes in $\mathscr{E}(\mathscr{L}(\Lambda_{\oplus}^{\leq 1}))$, and $s \in E_1$. Let $\mathscr{D}$ be the sub-distribution obtained by taking either $\mathscr{D} = s$ if $E_1 = \boldsymbol{P}$, or $\mathscr{D}$ such that $\widehat{\mathscr{D}} = s$, if $E_1 = \widehat{\mathcal{V}}$. Let $\mathcal{C}$ be a state-context in $\mathbf{C}(E_1 \to E_2)$. We define the state $\mathcal{C}[s]$ in E_2 as follows:*

- *if $E_2 = \boldsymbol{P}$, we take $\mathcal{C}[s] = \sum_M \mathscr{D}(M) \cdot \{\mathcal{C}[M]^1\}$;*
- *if $E_2 = \widehat{\mathcal{V}}$, we take $\mathcal{C}[s] = \sum_M \mathscr{D}(M) \cdot \{\widehat{\mathcal{C}[M]}^1\}$;*

In Proposition 8.3.4 below, we show that our approach is *valid*, in the sense that as soon as we are able to show that $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ is non-expansive, it also holds that the trace distance on $\Lambda_{\oplus}^{\leq 1}$-programs is non-expansive.

Proposition 8.3.4 *We suppose that the non-expansiveness property holds for $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$, equipped with state-contexts as specified in Definitions 8.3.5 and 8.3.6. Then the trace metric $\delta^{tr}_{\Lambda_{\oplus}^{\leq 1}}$ on $\Lambda_{\oplus}^{\leq 1}$ programs is non-expansive.*

Proof. Let M, N be two $\Lambda_{\oplus}^{\leq 1}$-programs, and $\mathcal{C}$ a context in $\Lambda_{\oplus}^{\leq 1}$. Then $\mathcal{C}$ can also be seen as a state-context in $\mathbf{C}(E_{\mathbf{P}} \to E_{\mathbf{P}})$. Since the WLTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ is non expansive, we obtain that: $\delta_{E_{\mathbf{P}}}^{\mathscr{L}(\Lambda_{\oplus}^{\leq 1})}(\mathcal{C}[s_M], \mathcal{C}[s_N]) \leq \delta_{E_{\mathbf{P}}}^{\mathscr{L}(\Lambda_{\oplus}^{\leq 1})}(s_M, s_N)$. Observe that moreover, $\mathcal{C}[s_M] = s_{\mathcal{C}[M]}$ and similarly $\mathcal{C}[s_N] = s_{\mathcal{C}[N]}$. From there, we obtain that $\delta_{E_{\mathbf{P}}}^{\mathscr{L}(\Lambda_{\oplus}^{\leq 1})}(s_{\mathcal{C}[M]}, s_{\mathcal{C}[N]}) \leq \delta_{E_{\mathbf{P}}}^{\mathscr{L}(\Lambda_{\oplus}^{\leq 1})}(s_M, s_N)$, and it concludes the proof. □

8.3.2 Contexts Composition for Relations on States

Our objective now thus becomes to show non-expansiveness for $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$. We see that we can reformulate the non-expansiveness for a WLTS by asking that all ε-bisimilarities must be *compatible*, i.e. that as soon as two states s, t are ε-bisimilar, then for every context $\mathcal{C}$ that can be filled with s, t, it holds that also $\mathcal{C}[s]$ and $\mathcal{C}[t]$ are ε-bisimilar. As a consequence, a first idea would be to look at the binary relation we obtain by *composing R by state-contexts* and to show that whenever R is an ε-bisimulation, the relation we obtain this way is also a ε-bisimulation. Let us first formalize how we *compose by states-contexts* a clustered—see Definition 8.2.2—binary relation on states.

Definition 8.3.7 *Let $\mathscr{L}^w$ be a WLTS equipped with state-contexts. Let R be a clustered binary relation on states. We call* compositions of R by states-contexts*, and we denote by $\mathbf{C}[R]$ the binary relation on states defined as follows: for every equivalence class E for $R^1_{\mathscr{L}^w}$, $s\,(\mathbf{C}[R]_E)\,t$ when there exists E', $\mathcal{C} \in \mathbf{C}(E' \to E)$ and two states s', t' such that $s'\,R_E\,t'$ and $s = \mathcal{C}[s']$, $t = \mathcal{C}[t']$.*

Observe that the relation $\mathbf{C}[R]$ is also a clustered relation. Moreover, if R is ε-bounded for some $\varepsilon \in [0, 1]$, that is also the case for $\mathbf{C}[R]$. As stated in Proposition 8.3.5 below, we are now able to re-frame our objective of proving non-expansiveness for $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ using the composition by contexts of a relation on states.

Proposition 8.3.5 *Let $\mathscr{L}^w$ be a WLTS equipped with state-contexts. If for every ε-bisimulation R, there exists a ε-bisimulation S such that $\mathbf{C}[R] \subseteq S$, then non-expansiveness holds for $\mathscr{L}^w$.*

As said before, we would like to establish that composition by contexts preserves ε-bisimulations, i.e. that when R is a ε-bisimulation on $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$, then $\mathbf{C}[R]$ itself is also an ϵ-bisimulation. This is unfortunately not the case, as we can see in Example 8.3.2 below.

Example 8.3.2 *We consider the states $s = \{(I \oplus \Omega)^1\}$ and $t = \{I^1\}$. Using these two states, we build an $\frac{1}{2}$-bisimulation:*

$$R = \{(s,t)\} \cup \{(\frac{1}{2} \cdot \widehat{\mathscr{D}}, \widehat{\emptyset}) \mid \widehat{\mathscr{D}} \in E_{\widehat{\mathcal{V}}}\} \cup \{(\frac{1}{2} \cdot \mathscr{D}, \emptyset) \mid \mathscr{D} \in E_{\boldsymbol{P}}\}.$$

We consider now the relation $\mathbf{C}[R]$, and we show that it is not *a $\frac{1}{2}$-bisimulation. We start from the context $\mathcal{C} = [\cdot] \oplus \lambda x.[\cdot]$; we can see $\mathcal{C}$ as a state-context in $\mathbf{C}(E_{\boldsymbol{P}} \to E_{\boldsymbol{P}}$, hence we can fill $\mathcal{C}$ with s and t, and $\mathcal{C}[s]$ and $\mathcal{C}[t]$ in $E_{\boldsymbol{P}}$ are as follows:*

$$\mathcal{C}[s] = (I \oplus \Omega) \oplus \lambda x.(I \oplus \Omega); \qquad \mathcal{C}[t] = (\Omega) \oplus \lambda x.(\Omega).$$

We now look at the effect of the eval action on $\mathcal{C}[s]$ and $\mathcal{C}[t]$. We see that $\mathcal{C}[s] \xrightarrow{eval} u$, $\mathcal{C}[t] \xrightarrow{eval} v$, where the states $u, v \in E_{\widehat{\mathcal{V}}}$ are as follows:

$$u := \frac{1}{4} \cdot \{\widehat{I}^1\} + \frac{1}{2}\{\widehat{\lambda x.I \oplus \Omega}^1\}; \qquad v := \frac{1}{2}\{\widehat{\lambda x.\Omega}^1\}.$$

We show by contradiction that $(u, v) \notin \mathbf{C}[R]$. Indeed, let us suppose that $(u, v) \in \mathbf{C}[R]$. Looking at the definition of $\mathbf{C}[R]$, we see that one among the three following statements must hold:

$$\exists \mathcal{C} \in \mathbf{C}(E_{\widehat{\mathcal{V}}} \to E_{\widehat{\mathcal{V}}}),\ u = \mathcal{C}[\frac{1}{2} \cdot \widehat{\mathscr{D}}],\ v = \mathcal{C}[\widehat{\emptyset}] \text{ with } \mathscr{D} \in E_{\widehat{\mathcal{V}}} \tag{8.1}$$

$$\exists \mathcal{C} \in \mathbf{C}(E_{\boldsymbol{P}} \to E_{\widehat{\mathcal{V}}}),\ u = \mathcal{C}[s],\ v = \mathcal{C}[t] \tag{8.2}$$

$$\exists \mathcal{C} \in \mathbf{C}(E_{\boldsymbol{P}} \to E_{\widehat{\mathcal{V}}}),\ u = \mathcal{C}[\frac{1}{2} \cdot \mathscr{D}],\ v = \mathcal{C}[\emptyset] \text{ with } \mathscr{D} \in E_{\boldsymbol{P}} \tag{8.3}$$

By looking at Definition 8.3.6 that specifies how to fill a state-context by a state in $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$, we see that we can rewrite the cases (8.1), (8.2), (8.3) *respectively as follows:*

$$\exists \mathcal{C} \in \mathbf{C}(E_{\widehat{\mathcal{V}}} \to E_{\widehat{\mathcal{V}}}),\ u = \frac{1}{2}\sum_{V} \mathscr{D}(V) \cdot \{\widehat{\mathcal{C}[V]}^{1}\},\ v = \widehat{\emptyset} \text{ with } \mathscr{D} \in E_{\widehat{\mathcal{V}}}$$

$$\exists \mathcal{C} \in \mathbf{C}(E_{\boldsymbol{P}} \to E_{\widehat{\mathcal{V}}}),\ u = \{\widehat{\mathcal{C}[I]}^{1}\},\ v = \{\widehat{\mathcal{C}[\Omega]}^{1}\}$$

$$\exists \mathcal{C} \in \mathbf{C}(E_{\boldsymbol{P}} \to E_{\widehat{\mathcal{V}}}),\ u = \frac{1}{2}\sum_{M} \mathscr{D}(M) \cdot \{\widehat{\mathcal{C}[M]}^{1}\},\ v = \widehat{\emptyset} \text{ with } \mathscr{D} \in E_{\boldsymbol{P}}$$

Since none of the three cases above holds, we obtain a contradiction.

8.3.3 Convex Structure for a WLTS

Recall that our goal is to find a way to build from an ε-bisimulation R, a relation which is both a ε-bisimulation and contains $\mathbf{C}[R]$. Example 8.3.2 tells us that this relation must be larger that $\mathbf{C}[R]$. To this end, we draw on the inherent convex structure of $\mathcal{S}_{\Lambda_{\oplus}^{\leq 1}}$—i.e. the fact that we are able to take convex combination of states, since they are sub-distributions over programs. We formalize this idea by defining a class of WLTSs that have this property, in Definition 8.3.8 below. We use there the concept of *cone*, that can be found in Definition 9.3.1 of Chapter 9—where we use it for totally different purposes, i.e. when presenting a denotational model developed by Ehrhard, Pagani and Tasson of the language PCF enriched with continuous probability. A cone C is essentially a $\mathbb{R}_+$-semimodule with a norm $\|\cdot\|_C$, that allows to talk about its unit ball $\mathscr{B}C$.

Definition 8.3.8 *We say that a WLTS $\mathscr{L}^w$ is* equipped with a convex structure*, if for every $E \in \mathscr{E}(\mathscr{L}^w)$, there exists a cone C_E such that $E = \mathscr{B}C_E$, and w and $\|\cdot\|_{C_E}$ coincide on $\mathscr{B}C_E$.*

In Example 8.3.3 below we equip $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ with a convex structure: we use the fact that the set of finite measures—i.e. distributions-like object with weight in $\mathbb{R}_+$, see Definition 1.4.2 of Chapter 1—with finite support over some countable set is stable under addition and multiplication by a non-negative scalar.

Example 8.3.3 *For $E \in \{E_{\boldsymbol{P}}, E_{\widehat{\mathcal{V}}}\}$, we build a cone C_E as follows:*

$$C_{E_{\boldsymbol{P}}} := \{\mu \text{ finite measure over } \boldsymbol{P} \mid \mathit{card}(\mathsf{S}(\mu)) < \infty\}; \qquad \|\mu\|_C = \mu(\boldsymbol{P})$$

$$C_{E_{\widehat{\mathcal{V}}}} := \{\mu \text{ finite measure over } \widehat{\mathcal{V}} \mid \mathit{card}(\mathsf{S}(\mu)) < \infty\}; \qquad \|\mu\|_C = \mu(\widehat{\mathcal{V}})$$

We take as $+$, and $.$ respectively the usual addition and multiplication by a scalar for finite measures. We can check that $C_{E_{\boldsymbol{P}}}$ and $C_{E_{\widehat{\mathcal{V}}}}$ are indeed cones, and that moreover $\mathscr{B}C_E = E$ for $E \in \{E_{\boldsymbol{P}}, E_{\widehat{\mathcal{V}}}\}$. It means that this way we have indeed equipped $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ with a convex structure.

If $\mathscr{L}^{\mathrm{w}}$ is a WLTS equipped with a convex structure, we can define for every clustered binary relation R on states its *convex closure*.

Definition 8.3.9 *Let $\mathscr{L}^w$ be a WLTS equipped with a convex structure. Let R be a clustered binary relation on states. We say that R is* convex closed*, if for every $E \in \mathscr{E}(\mathscr{L}^w)$, for every $\alpha, \beta \in [0,1]$ with $\alpha + \beta \leq 1$, for every s_1, s_2, t_1, t_2 such that $s_1 \,R_E\, t_1$ and $s_2 \,R_E\, t_2$ it holds that $(\alpha \cdot s_1 + \beta \cdot s_2) \,R_E\, (\alpha \cdot t_1 + \beta \cdot t_2)$. We write $\overline{(R)}^{+}$ for the smallest relation that is convex closed and that contains R.*

Observe that since each $E \in \mathscr{E}(\mathscr{L}^{\mathrm{w}})$ is stable by convex sums, it holds that the convex closure of R is also a clustered relation. We are in fact able to build explicitly $\overline{(R)}^{+}$ starting from R, as stated in Lemma 8.3.6 below.

Lemma 8.3.6 *Let $\mathscr{L}^{w}$ be a WLTS equipped with a convex structure, and R be a clustered binary relation on states. Then $s\,\overline{(R)}^{+}\,t$ if and only if there exists a finite sequence $(s_n, t_n)_{1\leq n\leq N}$ of pairs of states in R, and $(\alpha_n)_{1\leq n\leq N}$ a finite sequence of non-negative real numbers, such that $s = \sum_{1\leq n\leq N} \alpha_n \cdot s_n$ and $t = \sum_{1\leq n\leq N} \alpha_n \cdot t_n$.*

Our goal in the following is to show that if R is a ε-bisimulation on $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$, then $\overline{(\mathbf{C}[R])}^{+}$—the convex closure of the relation obtained by composing R by contexts—also is an ε-bisimulation and contains $\mathbf{C}[R]$—thus enabling us to apply Proposition 8.3.5. We see immediately that $\overline{(\mathbf{C}[R])}^{+}$ contains $\mathbf{C}[R]$, so from now on we will work towards proving that this combination of contexts and convex closure preserves ε-bisimulations. We first look back at the $\frac{1}{2}$-bisimulation R considered in Example 8.3.2, and we see that taking the context closure of $\mathbf{C}[R]$ allows us to overcome the problem highlighted there. Indeed, recall that our proof of $\mathbf{C}[R]$ not being a ε-bisimulation consisted of exhibiting a pair of states $(s,t) \in R$, and a context $\mathcal{C}$ such that $(u,v) \notin \mathbf{C}[R]$ with u and v defined as $\mathcal{C}[s] \xrightarrow{\text{eval}} u$, $\mathcal{C}[t] \xrightarrow{\text{eval}} v$. In Example 8.3.4 below, we show that $(u,v) \in \overline{(\mathbf{C}[R])}^{+}$: it means that in this particular case we are indeed able to overcome the problem by considering $\overline{(\mathbf{C}[R])}^{+}$ instead of $\mathbf{C}[R]$.

Example 8.3.4 *We keep the notation of Example 8.3.2. We can split u and v as follows:*

$$u = \frac{1}{2}(\lambda x.[\cdot])[s] + \frac{1}{2}([\cdot])[\frac{1}{2}\cdot\{\widehat{I^1}\}]; \qquad v = \frac{1}{2}(\lambda x.[\cdot])[t] + \frac{1}{2}([\cdot])[\widehat{\emptyset}].$$

Moreover it holds that both the pair (s,t) and the pair $(\frac{1}{2}\cdot\{\widehat{I^1}\}, \widehat{\emptyset}) \in R$ are in R, and $\mathcal{C}_1 = \lambda x.[\cdot] \in \mathbf{C}(E_{\widehat{\mathcal{V}}} \to E_{\widehat{\mathcal{V}}}), \mathcal{C}_2 = [\cdot] \in \mathbf{C}(E_{\boldsymbol{P}} \to E_{\boldsymbol{P}})$. As a consequence, we obtain that $u\,\overline{(\mathbf{C}[R])}^{+}\,v$.

We look more in depth to the behavior of the convex closure: we are going to show that it leads actually to an *up-to technique* for WLTS. A survey of *up-to* reasoning on transition structures can for instance be found in [104]. The general idea is as follows: we want to show that two states are bisimilar, without going all the way to find a bisimulation that relates them. If we have a valid up-to technique, it becomes enough to show that the two states are connected by a relation which is a *bisimulation up-to* some operator on relation, which is a weaker requirement as being a bisimulation.

Notation 8.3.7 *Let $\mathscr{L}^{w} = ((\mathcal{S}, \mathcal{L}, \to), w)$ be a WLTS, and $R, S \subseteq \mathcal{S}\times\mathcal{S}$. Then we write $R \rightarrowtail S$ when: $\forall (s,t) \in R, Act(s) = Act(t)$, and $\forall a \in Act(s)$, $(u,v) \in S$ with $s \xrightarrow{a} u$ and $t \xrightarrow{a} v$.*

With this notation, we can rewrite the ε-bisimulation requirement: R is a ε-bisimulation if and only if it is ε-bounded, and $R \rightarrowtail R$.

Definition 8.3.10 *Let $\mathscr{L}^{w} = ((\mathcal{S}, \mathcal{L}, \to), w)$ be a WLTS. We call* operator on $\mathscr{L}^{\mathrm{w}}$-relations *a function $\mathcal{O} : \{R \subseteq \mathcal{S}\times\mathcal{S}\} \to \{R \subseteq \mathcal{S}\times\mathcal{S}\}$, non-decreasing for the inclusion order. If $\mathcal{O}$ is an operator on $\mathscr{L}^{w}$-relations, we say that a binary relation $R \subseteq \mathcal{S}\times\mathcal{S}$ is a* ε-bisimulation up-to $\mathcal{O}$*, if it is ε-bounded, and $R \rightarrowtail \mathcal{O}(R)$.*

In [107], Vignudelli and Sangiorgi studied up-to techniques for environmental bisimulations—that are equivalences defined using bisimilarity on structures similar to our WLTSs—on probabilistic λ-calculi. Here, since we are interested in distances, we look at *quantitative* up-to techniques, that give us proof techniques for ε-bisimulation.

Definition 8.3.11 *Let $\mathscr{L}^w = ((\mathcal{S}, \mathcal{L}, \to), w)$ be a WLTS. We say that $\mathcal{O}$ an operator on $\mathscr{L}^w$-relations is a* valid quantitative up-to technique*, if every R which is a ε-bisimulation up-to $\mathcal{O}$ is also contained in a ε-bisimulation.*

We are now going to exhibit a class of WLTSs with convex structure, such that taking the convex closure is a valid quantitative up-to technique.

Definition 8.3.12 *We say that a WLTS $\mathscr{L}^w$ is equipped with a* well-behaved convex structure *if it is equipped with a convex structure such that:*

1. *$\forall s, t$ states, and α, β with $\alpha + \beta \leq 1, w(\alpha \cdot s + \beta \cdot s) = \alpha \cdot w(s) + \beta \cdot w(t)$.*
2. *for every state s that can be decomposed as $s = \sum_{1 \leq i \leq N} \alpha_i \cdot t_i$ with $\alpha_i > 0$, then:*
   - *$Act(s) = Act(t_i)$ for every $i \in \mathbb{N}$.*
   - *for $a \in Act(s)$, it holds that $v \xrightarrow{a} \sum_{1 \leq i \leq N} \alpha_i \cdot u_i$, where the u_i are defined by $t_i \xrightarrow{a} u_i$*

We can see easily that the convex structure on the WLTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$—as described in Example 8.3.3—is indeed well-behaved.

Lemma 8.3.8 *Let $\mathscr{L}^w$ be a WLTS equipped with a well-behaved convex structure. If R is a ε-bisimulation then $\overline{(R)}^+$ is also a ε-bisimulation.*

Proof. The proof consists of two steps:
- if a clustered relation R is ε-bounded, then also $\overline{(R)}^+$ is ε-bounded—it comes from Condition 1 in our definition of a well-behaved convex structure;
- if $R \rightarrowtail \overline{(R)}^+$ then also $\overline{(R)}^+ \rightarrowtail \overline{(R)}^+$—it comes from Condition 2 in our definition of a well-behaved convex structure.

□

An immediate consequence of Lemma 8.3.8 is that taking the convex closure is indeed a valid quantitative up-to-technique for the WLTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$.

Proposition 8.3.9 *If $\mathscr{L}^w$ is a WLTS equipped with a well-behaved convex structure, the convex closure $R \mapsto \overline{(R)}^+$ is a valid quantitative up-to technique.*

8.3.4 Combining Composition by Contexts and Convex Closure.

Recall that our goal in the present Section is to show that if R is a ε-bisimulation, and (s, t) are in $\mathbf{C}[R]$, then s and t are ε-bisimilar. Since taking the convex closure is a valid quantitative up-to technique, it is enough to show that $\mathbf{C}[R]$ is ε-bounded—which is apparent from the definition of $\mathbf{C}[R]$—and that $\mathbf{C}[R] \rightarrowtail \overline{(\mathbf{C}[R])}^+$. So we fix R an ε-bisimulation, and we show that $\mathbf{C}[R] \rightarrowtail \overline{(\mathbf{C}[R])}^+$, i.e. :

1. if s, t are in $E_{\widehat{\mathcal{V}}}$, with $s \, \mathbf{C}[R] \, t$ then the pair of states we obtain after doing a V action —for any value V—is again in $\mathbf{C}[R]$.

2. if s, t are in $E_{\mathbf{P}}$ with $s\,\mathbf{C}[R]\,t$ then the pair of states we obtain after doing a eval action is in $\overline{(\mathbf{C}[R])}^{+}$.

Requirements (1) and (2) are represented respectively on the right and the left part of Figure 8.1. We first show the requirement (1), that deals with the case of applicative

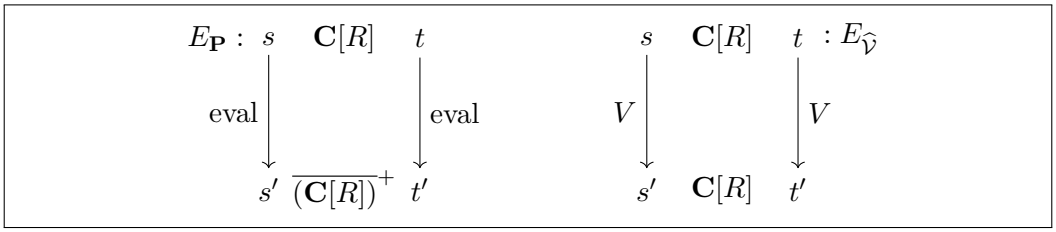


Figure 8.1: Overview of the proof that $\overline{(\mathbf{C}[R])}^{+}$ is a ε-bisimulation.

actions. The proof is based on the fact that when the environment performs an applicative action V on a state s of the form $s = \mathcal{C}[s']$, it is going to trigger *at most one* action on the internal state s.

Lemma 8.3.10 *Let $s, t \in E_{\widehat{\mathcal{V}}}$, with $s\,\mathbf{C}[R]\,t$. Let $V \in \widehat{\mathcal{V}}$, and let s', t' be such that $s \xrightarrow{V} s'$ and $t \xrightarrow{V} t'$. Then it holds that $s'\,\mathbf{C}[R]\,t'$*

Proof. Let V be a $\Lambda_{\oplus}^{\leq 1}$-value, and $s, t \in E_{\widehat{\mathcal{V}}}$ such that $s\,\mathbf{C}[R]\,t$. It means that there exists a pair of states $(s, t) \in R$, with $s = \mathcal{C}[s']$, $t = \mathcal{C}[t']$. Recall that moreover $\mathcal{C}$ must be in $\mathbf{C}(E, E_{\widehat{\mathcal{V}}})$, for some $E \in \{E_{\mathbf{P}}, E_{\widehat{\mathcal{V}}}\}$. We consider separately the case where $E = E_{\widehat{\mathcal{V}}}$ and the case where $E = E_{\mathbf{P}}$.

- We first suppose that $E = E_{\mathbf{P}}$. It means that s' and t' are sub-distributions over programs, and that $\mathcal{C}$ is of the form $\mathcal{C} = \lambda x.\mathcal{D}$, where $\mathcal{D}$ is an open context for the language $\Lambda_{\oplus}^{\leq 1}$, with x as only possible free variable —see how we defined $\mathbf{C}(E \to E')$ in Definition 8.3.5. We now spell out what it implies for the states s and t:

$$s = \sum s'(M) \cdot \{\lambda x.\mathcal{D}[M]^1\};$$
$$t = \sum t'(M) \cdot \{\lambda x.\mathcal{D}[M]^1\}.$$

  From there, we look at the effect of the applicative action V on s and t. Observe that $\mathcal{D}\{V/x\} \in \mathbf{C}(E_{\mathbf{P}} \to E_{\mathbf{P}})$; as a consequence we can fill $\mathcal{D}\{V/x\}$ by the states s' and t'. By looking at the transition relation in the WLTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$, we can see that $s \xrightarrow{V} u$, and $t \xrightarrow{V} v$ with:

$$u := \{(\widehat{\mathcal{D}\{V/x\}})[s']^1\} \tag{8.4}$$
$$v := \{(\widehat{\mathcal{D}\{V/x\}})[t']^1\}. \tag{8.5}$$

  Since $(s', t') \in R$, Equations (8.4), and (8.5) tell us that $(u, v) \in \mathbf{C}[R]$, hence Lemma 8.3.10 holds.

- We now look at the case where $E = E_{\widehat{\mathcal{V}}}$: then s' and t' are sub-distributions over distinguished values, and $\mathcal{C}$ can be either of the form $\mathcal{C} = \lambda x.\mathcal{D}$—as in the previous

case—or of the form $[\cdot]$. In the first case, we obtain the result in the same way as when $E = E_{\mathbf{P}}$. From now on, we suppose that $\mathcal{C} = [\cdot]$. It means that $s' = s$, and $t' = t$. Since R is a ε-bisimulation, and $s' \, R \, t'$, we see that $s \xrightarrow{V} u$ and $t \xrightarrow{V} v$, with $u \, R \, v$. Since $R \subseteq \mathbf{C}[R]$, it implies that the result of Lemma 8.3.10 holds.

□

We have now to look at the eval action, and to show that Requirement (2) is also satisfied. This case is harder to handle that the case of applicative actions, because when the environment performs an action eval on a state of the form $t = \mathcal{C}[s']$, it can trigger a variety of possible sequences of actions on the state s'—depending on the form of the context $\mathcal{C}$. We first look at the simple case where every program in the support of the states s, t is actually already a value: it means that the action eval for those states consists only in transforming values into distinguished values.

Lemma 8.3.11 *Let R be an ϵ-bisimulation on $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$. For every $s, t \in E_{\boldsymbol{P}}$ such that $s \, \overline{(\mathbf{C}[R])}^{+} \, t$, if moreover the support of s and t consists only of values, then it also holds that $\widehat{s} \, \overline{(\mathbf{C}[R])}^{+} \, \widehat{t}$—where $\widehat{u}$ simply refers to the distribution u where every value in the support of u has been replaced by the associated distinguished value.*

Our approach to the general case is as follows: we are going to define an auxiliary operational semantics, where *programs* are replaced by *pairs of the form* $(\mathcal{C}, s)$, that allows us to *split* the action eval into a sequence of actions that have an atomic effect on s'.

8.3.5 A Contexts-Programs WLTS for $\Lambda_{\oplus}^{\leq 1}$.

Definition 8.3.13 *We call $(\mathbf{C} \times \mathcal{S})$ the set of all pairs of the form $(\mathcal{C}, s)$, where there exists $E \in \mathscr{E}(\mathscr{L}(\Lambda_{\oplus}^{\leq 1}))$ such that $s \in E$, $\mathcal{C} \in \mathbf{C}(E \to E_{\boldsymbol{P}})$. We say that $e \in (\mathbf{C} \times \mathcal{S})$ is a $(\mathbf{C} \times \mathcal{S})$*-value *if either $s \in E_{\widehat{\mathcal{V}}}$ and $\mathcal{C} = [\cdot]$, or $\mathcal{C}$ is of the form $\lambda x.\mathcal{D}$, where $\mathcal{D}$ is a $\Lambda_{\oplus}^{\leq 1}$ open context with x as only possible free variable.*

Observe that if $e = (\mathcal{C}, s)$ is a $(\mathbf{C} \times \mathcal{S})$-value, then in particular all the programs in the support of $\mathcal{C}[s]$ are $\Lambda_{\oplus}^{\leq 1}$-values. We call $(\mathbf{C} \times \mathcal{S})$*-state* a sub-distribution over $(\mathbf{C} \times \mathcal{S})$ with finite support, and we denote $\mathcal{S}^{(\cdot,\cdot)}$ the set of all such states. We first formalize how we can retrieve back a $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$-state from any state in $\mathcal{S}^{(\cdot,\cdot)}$.

Definition 8.3.14 *We define the* forgetful function $\mathbf{F} : \mathcal{S}^{(\cdot,\cdot)} \to \mathcal{S}_{\Lambda_{\oplus}^{\leq 1}}$, *as follows:*

$$\mathbf{F}(s) := \sum_{(\mathcal{C},t) \in (\mathbf{C} \times \mathcal{S})} s(\mathcal{C}, t) \cdot \mathcal{C}[t].$$

For any clustered relation R, we build the *explicit composition of R by contexts*, which is a binary relation on $\mathcal{S}^{(\cdot,\cdot)}$. It consists of simply putting side by side R-connected programs and states-contexts.

Definition 8.3.15 *Let R be a clustered binary relation on $\mathcal{S}_{\Lambda_{\oplus}^{\leq 1}}$. Then we define a binary relation $(\mathbf{C} \Diamond R) \subseteq \mathcal{S}^{(\cdot,\cdot)} \times \mathcal{S}^{(\cdot,\cdot)}$ as:*

$$(\mathbf{C} \Diamond R) = \{(\{(\mathcal{C}, s)^1\}, \{(\mathcal{C}, t)^1\}) \mid \exists E \in \{E_{\widehat{\mathcal{V}}}, E_{\boldsymbol{P}}\}, \mathcal{C} \in \mathbf{C}(E \to E_{\boldsymbol{P}}), \, s \, R_E \, t\}.$$

The relation $(\mathbf{C}\Diamond R)$ on $\mathcal{S}^{(\cdot,\cdot)}$ should be seen as a transposition of the relation $\mathbf{C}[R]$ on $\mathcal{S}_{\Lambda_{\oplus}^{\leq 1}}$. In Lemma 8.3.12 below, this observation is formalized using the forgetful function $\mathbf{F}$.

Lemma 8.3.12 *Let R be a clustered binary relation on $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$, and $s, t \in \mathcal{S}^{(\cdot,\cdot)}$ such that $s\,(\mathbf{C}\Diamond R)\,t$. Then it holds that $\mathbf{F}(s)\,\mathbf{C}[R]\,\mathbf{F}(t)$.*

We want now to define an small-step operational semantics on $(\mathbf{C} \times \mathcal{S})$, designed to express how we can execute—in a probabilistic way—the elements of $(\mathbf{C} \times \mathcal{S})$.

Definition 8.3.16 *We define the* one-step transition relation over $(\mathbf{C} \times \mathcal{S})$*, that we note* $\rightarrow_{(\mathbf{C}\times\mathcal{S})} \subseteq (\mathbf{C} \times \mathcal{S}) \times \Delta((\mathbf{C} \times \mathcal{S})$*, as specified in Figure 8.2 below.*

The β-rule of Figure 8.2 is valid only because $\Lambda_{\oplus}^{\leq 1}$ is affine: indeed that $N\{\mathcal{C}/x\}$ is still a context is guaranteed by the $\Lambda_{\oplus}^{\leq 1}$-affinity constraints—see Section 7.4 of Chapter 7. Observe that this one-step reduction is deadlocks-free, in the sense that for an element $e \in (\mathbf{C} \times \mathcal{S})$, there exists $\mathscr{D}$ such that $e \longrightarrow \mathscr{D}$ if and only if e is not a $(\mathbf{C} \times \mathcal{S})$-value. We now extend this one-step reduction to elements of $\mathcal{S}^{(\cdot,\cdot)}$—i.e. sub-distributions over $(\mathbf{C} \times \mathcal{S})$ with finite support.

Definition 8.3.17 *We define $\rightarrow_{\mathcal{S}^{(\cdot,\cdot)}} \subseteq \mathcal{S}^{(\cdot,\cdot)} \times \mathcal{S}^{(\cdot,\cdot)}$, that we call the* one-step transition relation over $\mathcal{S}^{(\cdot,\cdot)}$*, as follows: if $s = \sum_{i\in I} p_i \cdot \{e_i{}^1\} + \mathscr{E}$ where all elements in the support of $\mathscr{E}$ are $(\mathbf{C} \times \mathcal{S})$-values, and the e_i are not $(\mathbf{C} \times \mathcal{S})$-values, and I not empty, then we define $s \rightarrow_{\mathcal{S}^{(\cdot,\cdot)}} \mathscr{D} + \sum_{i\in I} p_i \cdot t_i$ with $e_i \rightarrow_{(\mathbf{C}\times\mathcal{S})} t_i$.*

$$\mathrm{CR}\ \frac{N \to \mathscr{E}}{(N\mathcal{C}, s) \rightarrow_{(\mathbf{C}\times\mathcal{S})} \sum_L \mathscr{E}(L) \cdot \{(L\mathcal{C}, s)^1\}}$$

$$\mathrm{CL}\ \frac{(\mathcal{C}, s) \text{ a } (\mathbf{C} \times \mathcal{S})\text{-value} \qquad N \to \mathscr{E}}{(\mathcal{C}N, s) \rightarrow_{(\mathbf{C}\times\mathcal{S})} \sum_L \mathscr{E}(L) \cdot \{(\mathcal{C}L, s)^1\}}$$

$$\mathrm{SR}\ \frac{(\mathcal{C}, s) \rightarrow_{(\mathbf{C}\times\mathcal{S})} \mathscr{E} \qquad V \in \mathcal{V}}{(V\mathcal{C}, s) \stackrel{\tau}{\rightarrow}_{(\mathbf{C}\times\mathcal{S})} \sum_{\mathcal{D},t} \mathscr{E}(\mathcal{D}, t) \cdot \{(V\mathcal{D}, t)^1\}}$$

$$\beta\ \frac{(\mathcal{C}, s) \text{ a } (\mathbf{C} \times \mathcal{S})\text{-value}}{((\lambda x.N)\mathcal{C}, s) \rightarrow_{(\mathbf{C}\times\mathcal{S})} \{(N\{\mathcal{C}/x\}, s)^1\}}$$

$$\mathrm{Div}\ \frac{}{(\Omega, s) \rightarrow_{(\mathbf{C}\times\mathcal{S})} \emptyset} \qquad \frac{}{(\mathcal{C} \oplus \mathcal{D}, s) \rightarrow_{(\mathbf{C}\times\mathcal{S})} \frac{1}{2} \cdot \{(\mathcal{C}, s)^1\} + \frac{1}{2}\{(\mathcal{D}, s)^1\}}\ \oplus$$

$$\mathrm{eval}\ \frac{s \in E_{\mathbf{P}} \qquad s \stackrel{\mathrm{eval}}{\longrightarrow} t}{([\cdot], s) \rightarrow_{(\mathbf{C}\times\mathcal{S})} \{([\cdot], t)^1\}} \qquad \frac{s \in E_{\widehat{\mathcal{V}}} \qquad s \xrightarrow{V} t}{([\cdot]V, s) \rightarrow_{(\mathbf{C}\times\mathcal{S})} \{([\cdot], t)^1\}}\ V$$

Figure 8.2: The one-step reduction relation on $(\mathbf{C} \times \mathcal{S})$.

We say that a state $s \in \mathcal{S}^{(\cdot,\cdot)}$ is *in normal form* if there is no t such that $s \rightarrow_{(\mathbf{C}\times\mathcal{S})} t$. Using the fact that the calculus is *affine*, and that we consider only finite sub-distributions, we are able to show that the one-step reduction relation on $\mathcal{S}^{(\cdot,\cdot)}$ is strongly normalizing, i.e. there exists a normal form.

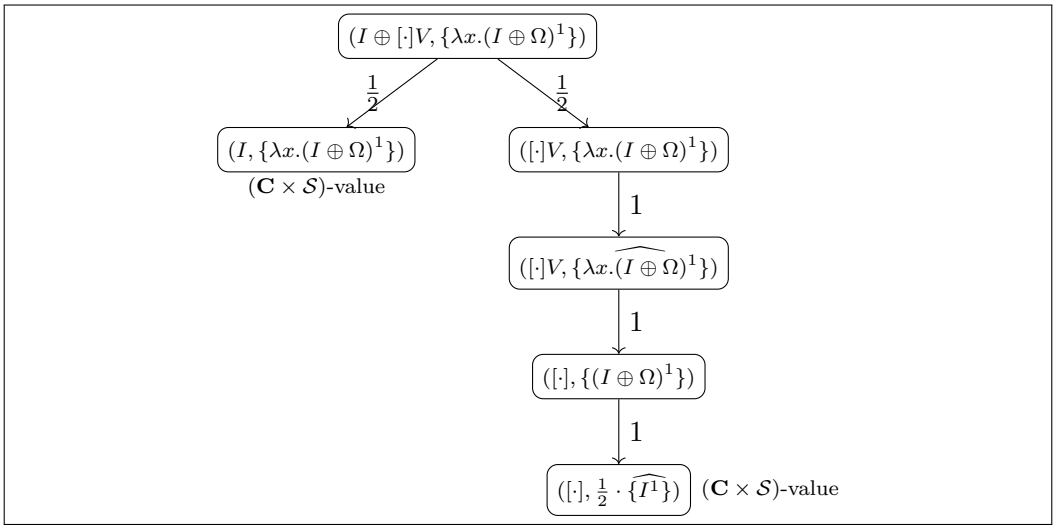


Figure 8.3: Example of the effect of the one-step transition relation for $(\mathbf{C} \times \mathcal{S})$

Lemma 8.3.13 *For every state $s \in \mathcal{S}^{(\cdot,\cdot)}$:*

- *either s is in normal form, and then all $e \in \mathsf{S}(s)$ are $(\mathbf{C} \times \mathcal{S})$-values;*
- *or s is not in normal form, and then there exists a unique t in normal form such that $s \to^{\star}_{\mathcal{S}^{(\cdot,\cdot)}} t$.*

Proof. We can see that the first point holds simply by looking at the rules of Figure 8.2. We now look at the second point. Recall the cost function $c : \text{Terms}_{\Lambda_{\oplus}^{\leq 1}} \to \mathbb{N}$ that we have used when proving termination for $\Lambda_{\oplus}^{\leq 1}$-programs in Proposition 7.4.1 from Chapter 7. We defined c inductively on the structure of terms. We extend here the inductive definition of $\mathcal{C}$ by $c([\cdot]) = 1$, which allows us to see c as a function $\mathbf{C}_{\Lambda_{\oplus}^{\leq 1}} \to \mathbf{C}_{\Lambda_{\oplus}^{\leq 1}}$. We now consider a sequence of reduction: $s = \{(\mathcal{C}, t)^1\} \to_{\mathcal{S}^{(\cdot,\cdot)}} s_2 \to_{\mathcal{S}^{(\cdot,\cdot)}} \ldots$. We are going to show by induction on $c(\mathcal{C})$ that this sequence is finite:

- If $c(\mathcal{C}) = 0$, it means that $\mathcal{C} = \lambda x.\mathcal{D}$, and moreover $c(\mathcal{D}) = 0$. As a consequence, $(\mathcal{C}, s)$ is a $(\mathbf{C} \times \mathcal{S})$-value, and so the result holds.
- We suppose now that the result holds as soon as $c(\mathcal{C}) = n$, and we want to show it when $c(\mathcal{C}) = n + 1$. We first show an intermediary result:

$$(\mathcal{C}, s) \to_{(\mathbf{C} \times \mathcal{S})} \mathscr{D} \Rightarrow \begin{cases} \text{either } \forall (\mathcal{D}, t) \in \mathsf{S}(\mathscr{D}), c(\mathcal{D}) < c(\mathcal{C}) \\ \text{or } s \in E_{\mathbf{P}}, \mathscr{D} = \{\mathcal{C}, t^1\} \text{ with } t \in E_{\widehat{\mathcal{V}}} \end{cases}$$

  To show this intermediary result, we do the proof by induction on the structure of the proof derivation of $(\mathcal{C}, s) \to_{(\mathbf{C} \times \mathcal{S})} s_2$ —i.e. the first reduction step.

  - if the first rule applied is eval, i.e. if $\mathcal{C} = [\cdot]$,then $([\cdot], s) \to_{(\mathbf{C} \times \mathcal{S})} \{([\cdot], t)^1\}$, with $s \xrightarrow{\text{eval}} t$, and indeed $s \in E_{\mathbf{P}}$, $t \in E_{\widehat{\mathcal{V}}}$.
  - if the first rule applied is Div, i.e. if $\mathcal{C} = \Omega$, then $(\mathcal{C}, s) \to_{(\mathbf{C} \times \mathcal{S})} \emptyset$, and the result holds;

- if the first rule applied is CR, i.e. if $\mathcal{C} = N\mathcal{D}$, when N is not a value, we see that $(\mathcal{C}, s) \to_{(\mathbf{C}\times\mathcal{S})} \sum_L \mathscr{D}(L) \cdot \{(L\mathcal{D}, s)^1\}$, with $N \to \mathscr{D}$. Recall that, as shown in Lemma 7.4.3, the cost function c decreases at each step of the execution of programs: it means that for every $L \in \mathsf{S}(\mathscr{D})$, $c(L) \leq c(N) - 1$. From there, we obtain that $c(L\mathcal{D}) = c(L) + c(\mathcal{D}) < c(\mathcal{C})$.
- if the first rule applied is CL, it means that $\mathcal{C} = \mathcal{D}N$, with $(\mathcal{D}, s)$ a $(\mathbf{C} \times \mathcal{S})$-value. Then $s_2 = \sum_{L\in\mathsf{S}(\mathscr{D})} \mathscr{D}(L) \cdot \{(\mathcal{D}L, s)^1\}$, with $N \to \mathscr{D}$. We can see that $c(\mathcal{D}L) \leq c(\mathcal{C}) - 1$ for every $L \in \mathsf{S}(\mathscr{D})$, and we conclude similarly to the previous case.
- if the first rule applied is SR, it means that $\mathcal{C} = V\mathcal{D}$. We apply the induction hypothesis on $(\mathcal{D}, s) \to_{(\mathbf{C}\times\mathcal{S})} \mathscr{E}$, and this way we obtain the result.
- If the first rule applied is β or V, we see that $((\mathcal{C}, s) \to_{(\mathbf{C}\times\mathcal{S})} \{(\mathcal{D}, t)^1\}$ with $c(\mathcal{D}) < c(\mathcal{C})$.

We now use the intermediary result to conclude the proof of Lemma 8.3.13:

- We first suppose that s_2 is such that $\forall(\mathcal{D}, t) \in \mathsf{S}(s_2), c(\mathcal{D}) < c(\mathcal{C})$. If s_2 is actually the empty sub-distribution $\emptyset$, then the result holds since $\emptyset$ is a $(\mathbf{C}\times\mathcal{S})$-value. Otherwise, we can apply the induction hypothesis to all the $e = (\mathcal{D}, t) \in \mathsf{S}(s_2)$: for every e, there exists a reduction sequence $t_1^e = \{e^1\}\to_{\mathcal{S}^{(\cdot,\cdot)}} t_2^e \ldots \to_{\mathcal{S}^{(\cdot,\cdot)}} t_{N_e}^e$, with $t_{N_e}^e$ is a normal form. Then we see that for every n, we can rewrite s_n as: $s_n = \sum_{e\in\mathsf{S}(s_2)|N_e\geq n-1} \cdot t_{n-1}^e + \sum_{e\in\mathsf{S}(s_2)|N_e\leq n-2} \cdot t_{N_e}^e$. Looking at the definition of $\to_{\mathcal{S}^{(\cdot,\cdot)}}$, we see that if the length of the sequence (t_n) is greater than N, it implies that $\{e \mid N_e - 1 \geq N\}$ must be non-empty. Since all the reduction sequences for the $e \in \mathsf{S}(s_2)$ are of finite length, and that $\mathsf{S}(s_2)$ is a finite set, we can conclude that the sequence (t_n) is also of finite length.
- We now suppose that the second alternative of the intermediary result holds, i.e. $s \in E_{\mathbf{P}}$, and $(\mathcal{C}, s) \to_{(\mathbf{C}\times\mathcal{S})} \{\mathcal{C}, t^1\}$ with $t \in E_{\widehat{\mathcal{V}}}$. Then the first alternative holds for $(\mathcal{C}, t)$, so we obtain the result.

□

We define accordingly the *operational semantics* for elements in $(\mathbf{C} \times \mathcal{S})$.

Definition 8.3.18 *Let $e \in (\mathbf{C} \times \mathcal{S})$. We call* operational semantics of e*, and we denote $[\![e]\!]$ the sub-distribution over $(\mathbf{C} \times \mathcal{S})$-values defined as the unique state $s \in \mathcal{S}^{(\cdot,\cdot)}$ in normal form such that $\{e^1\}\to^\star_{\mathcal{S}^{(\cdot,\cdot)}} s$.*

We extend this notation to all states in $\mathcal{S}^{(\cdot,\cdot)}$: if $s \in \mathcal{S}^{(\cdot,\cdot)}$, we denote $[\![s]\!]$ the unique normal form t such that $s\to^\star_{\mathcal{S}^{(\cdot,\cdot)}} t$. This operational semantics on $\mathcal{S}^{(\cdot,\cdot)}$ can be connected with the operational semantics on $\Lambda_{\oplus}^{\leq 1}$-programs: more precisely, executing $(\mathcal{C}, \{M^1\})$ is the same thing as executing the $\Lambda_{\oplus}^{\leq 1}$-program $\mathcal{C}[M]$, but with using a small-step reduction for $\mathcal{C}$, and a big-step reduction for M. We formalize this correspondence in Lemma 8.3.14 below.

Lemma 8.3.14 *Let $s \in \mathcal{S}^{(\cdot,\cdot)}$. Then it holds that: $\sum_{M\in\boldsymbol{P}} \mathbf{F}(s)(M) \cdot [\![M]\!] = \mathbf{F}([\![s]\!])$.*

Proof. In order to simplify the notations, we extend the operational semantics on $\Lambda_{\oplus}^{\leq 1}$-programs to $E_{\mathbf{P}}$, by noting $[\![t]\!] = \sum_{M\in\mathbf{P}} t(M) \cdot [\![M]\!]$, when $t \in E_{\mathbf{P}}$, i.e. t is a sub-distribution over $\Lambda_{\oplus}^{\leq 1}$-programs. With this notation, we can rewrite our goal as: $\forall s \in \mathcal{S}^{(\cdot,\cdot)}$, $[\![\mathbf{F}(s)]\!] = \mathbf{F}([\![s]\!])$. We show this result in two steps:

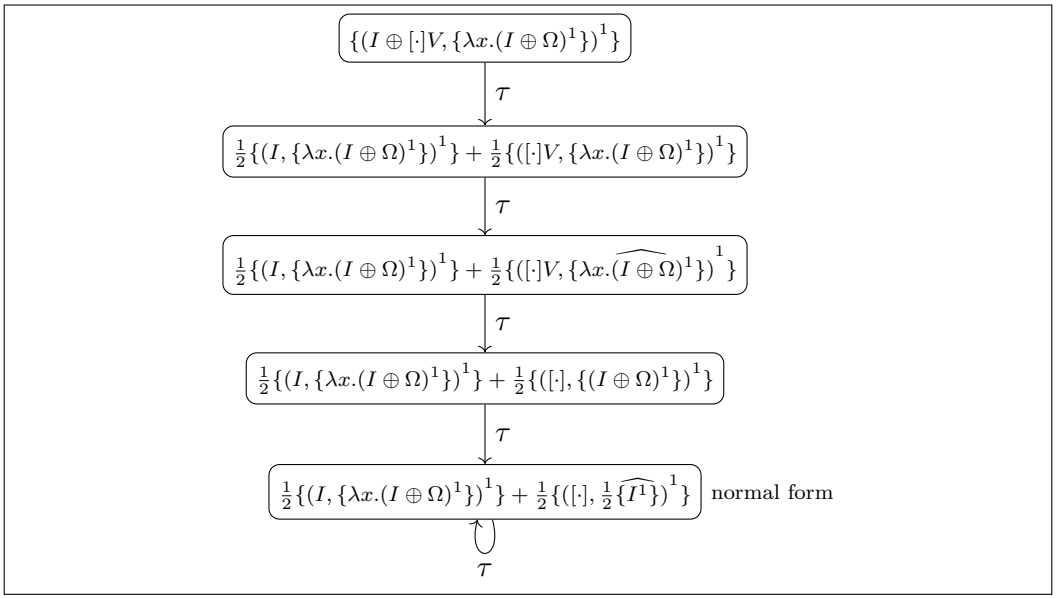


Figure 8.4: A Fragment of the WLTS $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$

- first, we see—by looking at the rule for $\rightarrow_{(\mathbf{C}\times\mathcal{S})}$—that if $s \rightarrow_{\mathcal{S}^{(\cdot,\cdot)}} t$, then the operational semantics of the underlying $\Lambda_{\oplus}^{\leq 1}$-subdistribution is not modified, i.e. $[\![\mathbf{F}(s)]\!] = [\![\mathbf{F}(t)]\!]$;
- then we see that if s is a normal-form, then $[\![\mathbf{F}(s)]\!] = \mathbf{F}(s)$, since all the program in the support of $\mathbf{F}(s)$ are $\Lambda_{\oplus}^{\leq 1}$-values.

We are now able to conclude: let $s \in \mathcal{S}^{(\cdot,\cdot)}$. By iterating the first step we obtain that $[\![\mathbf{F}(s)]\!] = [\![\mathbf{F}([\![s]\!])]\!]$, and then since $[\![s]\!] \in \mathcal{S}^{(\cdot,\cdot)}$ is a normal-form, we can use the second step, and we see that $[\![\mathbf{F}([\![s]\!])]\!] = \mathbf{F}([\![s]\!])$, and we have the result.

□

Our goal now is to start from a binary relation on $\mathcal{S}^{(\cdot,\cdot)}$ of the form $(\mathbf{C}\Diamond R)$—where R is taken as a ε-bisimulation—and to study the binary relation over $\mathcal{S}^{(\cdot,\cdot)}$ that we obtain after one step of reduction, that is $\{(u,v) \mid (s,t) \in (\mathbf{C}\Diamond R), s \rightarrow_{\mathcal{S}^{(\cdot,\cdot)}} u, t \rightarrow_{\mathcal{S}^{(\cdot,\cdot)}} v\}$. We thus introduce a WLTS $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$, which has $\mathcal{S}^{(\cdot,\cdot)}$ as set of states, and whose transition relation is designed to embed the one-step reduction relation on $(\mathbf{C} \times \mathcal{S})$.

Definition 8.3.19 (The WLTS $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$.) *We define the* execution WLTS for $\mathcal{S}^{(\cdot,\cdot)}$ *as* $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})} := (w, (\mathcal{S}^{(\cdot,\cdot)}, \{\tau\}, \longrightarrow))$*, where:*

- *the weight function w is defined as:* $w(\mathscr{D}) := \sum_{(\mathcal{C},s)\in \mathsf{S}(\mathscr{D})} \mathscr{D}(\mathcal{C}, s) \cdot |s|$*;*
- *if $s \rightarrow_{(\mathbf{C}\times\mathcal{S})} t$, then $s \xrightarrow{\tau} t$; and if s is in normal form, $s \xrightarrow{\tau} s$.*

Observe that $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$ has only *one* equivalence class in $\mathscr{E}(\mathscr{L}^{(\mathbf{C}\times\mathcal{S})})$, that contains all its states. Moreover, it has an obvious well-behaved convex structure, based on sum and multiplication by a non-negative scalar for measures with finite support. We illustrate how we build the WLTS $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$ starting from the relation $\rightarrow_{(\mathbf{C}\times\mathcal{S})}$ in Figure 8.4.

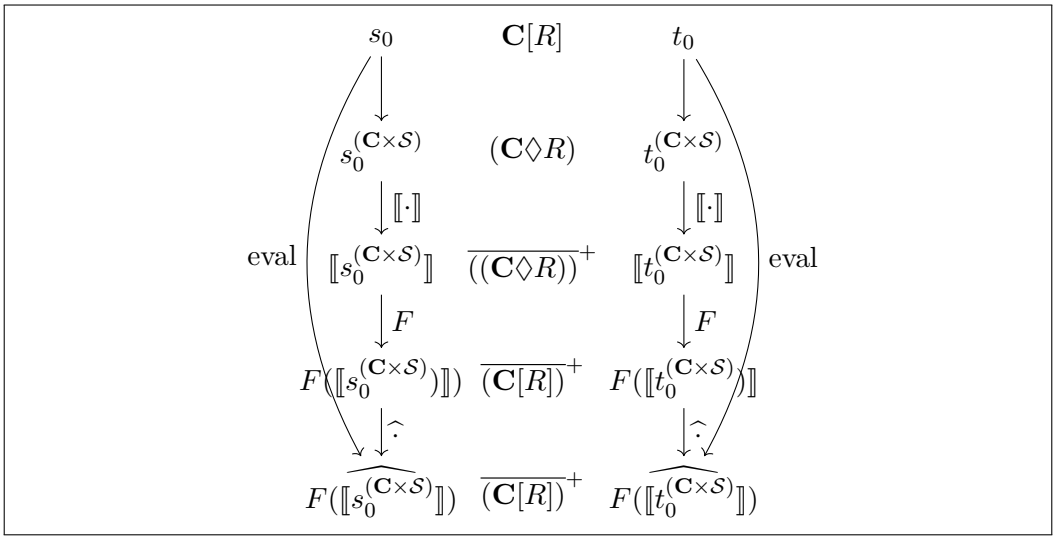


Figure 8.5: Proof Scheme

8.3.6 Using $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$ to conclude the Proof of Non-Expansiveness

We represent graphically in Figure 8.5 our proof strategy, i.e how to use the WLTS $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$ in order to conclude the proof of non-expansiveness for $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$. Recall that what remains to be shown is:

$$\forall s,t \in \mathcal{S}_{\Lambda_{\oplus}^{\leq 1}}, \quad s\,(\mathbf{C}[R])_{\mathbf{P}}\,t \Rightarrow u\,\overline{(\mathbf{C}[R])}^{+}_{\widehat{\mathcal{V}}}\,v, \qquad \text{with } s \xrightarrow{\text{eval}} u,\ t \xrightarrow{\text{eval}} v. \tag{8.6}$$

So we fix $s_0, t_0 \in \mathcal{S}_{\Lambda_{\oplus}^{\leq 1}}$ such that $s_0\,(\mathbf{C}[R])_{\mathbf{P}}\,t_0$. Our approach is as follows: we split the moves $s_0 \xrightarrow{\text{eval}} u_0$ and $t_0 \xrightarrow{\text{eval}} v_0$ into a sequence of transformations with (s_0, t_0) as point of departure, and (u_0, v_0) as end point.

1. In a first step, we translate our $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$-states s_0 and t_0 into $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$-states. By definition of $\mathbf{C}[R]$, we know that there exists $E \in \mathscr{E}(\mathscr{L}(\Lambda_{\oplus}^{\leq 1}))$, $s', t' \in E$, and $\mathcal{C} \in \mathbf{C}(E \to E_{\mathbf{P}})$ such that $s_0 = \mathcal{C}[s']$, and $t_0 = \mathcal{C}[t']$. From there, we define $s_0^{(\mathbf{C}\times\mathcal{S})} := \{(\mathcal{C}, s')^1\}$, and $t_0^{(\mathbf{C}\times\mathcal{S})} := \{(\mathcal{C}, t')^1\}$. Looking at the definition of $(\mathbf{C}\Diamond R)$, we can immediately see that—as represented in Figure 8.5—this transformation guarantees that the resulting pair $(s_0^{(\mathbf{C}\times\mathcal{S})}, t_0^{(\mathbf{C}\times\mathcal{S})})$ is in $(\mathbf{C}\Diamond R)$.

2. Then, we take the operational semantics —as states of $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$—of $s_0^{(\mathbf{C}\times\mathcal{S})}$, and $t_0^{(\mathbf{C}\times\mathcal{S})}$. For this step to work, we need to guarantee that the operational semantics preserves the relation $\overline{((\mathbf{C}\Diamond R))}^{+}$, i.e. that the resulting pair of $\mathcal{S}^{(\cdot,\cdot)}$-elements $([\![s_0^{(\mathbf{C}\times\mathcal{S})}]\!], [\![t_0^{(\mathbf{C}\times\mathcal{S})}]\!])$ is still in $\overline{(\mathbf{C}[R])}^{+}$. We will prove this—which in fact consists in the core of the non-expansiveness proof —in Lemma 8.3.16 later.

3. In the third step, we go back from $\mathcal{S}^{(\cdot,\cdot)}$ to $\mathcal{S}_{\mathscr{L}(\Lambda_{\oplus}^{\leq 1})}$, by applying the forgetful function to $[\![s_0^{(\mathbf{C}\times\mathcal{S})}]\!]$ and $[\![s_0^{(\mathbf{C}\times\mathcal{S})}]\!]$.We know that the resulting pair of states is back in $\overline{(\mathbf{C}[R])}^{+}$ as represented in Figure 8.5: it is an immediate consequence of Lemma 8.3.12.

4. At the end of the third step, we have states that are still in $E_{\mathbf{P}}$—i.e. sub-distributions over programs—but such that all the program in their support are actually values.

The fourth step consists in taking the associated states in $E_{\widehat{\mathcal{V}}}$. This transformation also preserves $\overline{(\mathbf{C}[R])}^{+}$, as we have shown in Lemma 8.3.11.

5. In the last step, we see that the states $F(\widehat{[\![s_0^{(\mathbf{C}\times\mathcal{S})}]\!]})$ and $F(\widehat{[\![t_0^{(\mathbf{C}\times\mathcal{S})}]\!]})$ obtained as described above actually coincide with the states u and v defined by $s_0 \xrightarrow{\text{eval}} u$ and $t \xrightarrow{\text{eval}} v$. It is a consequence of Lemma 8.3.14, and of the fact that $F(s_0^{(\mathbf{C}\times\mathcal{S})}) = s_0$, and $F(t_0^{(\mathbf{C}\times\mathcal{S})}) = t_0$. From there, as represented in Figure 8.5, we can conclude that $u\,\overline{(\mathbf{C}[R])}^{+}\,v$, and as a consequence, we have proved Equation (8.6).

To sum up, it means that what remains to be shown to conclude the proof that $\overline{(\mathbf{C}[R])}^{+}$ is a ε-bisimulation is:

$$\forall s,t \in \mathcal{S}^{(\cdot,\cdot)},\ s\,(\mathbf{C}\Diamond R)\,t \ \Rightarrow\ [\![s]\!]\,(\mathbf{C}\Diamond R)\,[\![t]\!].$$

The proof is based on the fact that $(\mathbf{C}\Diamond R)$ is a ε-bisimulation on the WLTS $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$. We first remark that we have built $(\mathbf{C}\Diamond R)$ in such a way that if R is a $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ ε-bisimulation, then the convex closure of $(\mathbf{C}\Diamond R)$ is a ε-bisimulation on $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$.

Lemma 8.3.15 *Let R be a ε-bisimulation on $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$. Then $\overline{((\mathbf{C}\Diamond R))}^{+}$ is a ε-bisimulation on $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$.*

Proof. The action τ is the only action on the WLTS $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$, and the transition relation is defined using the one-step transition for $\mathcal{S}^{(\cdot,\cdot)}$. It means that it is enough to look at each rule of Figure 8.2, and to see that indeed $\overline{((\mathbf{C}\Diamond R))}^{+}$ is preserved under those rules. □

Lemma 8.3.16 *If $s\,\overline{((\mathbf{C}\Diamond R))}^{+}\,t$, then it holds that also $[\![s]\!]\,\overline{((\mathbf{C}\Diamond R))}^{+}\,[\![t]\!]$.*

Proof. The proof is by induction on the length of the reduction sequence $s{\to}^{\star}_{\mathcal{S}^{(\cdot,\cdot)}}[\![s]\!]$.

- If it is 0, it means that s is a normal form. Then we can see from the definition of $\overline{((\mathbf{C}\Diamond R))}^{+}$ that it implies that t too is in normal form: as a consequence $[\![t]\!] = t$, which shows the result.
- We suppose that the results holds for n, and that the length of the reduction sequence is $n+1$. Then we do one step of reduction for both s and t, and since $\overline{((\mathbf{C}\Diamond R))}^{+}$ is a bisimulation, the resulting states are still related by $\overline{((\mathbf{C}\Diamond R))}^{+}$. From there, we can apply the induction hypothesis, and it concludes the proof.

□

Observe that we have now shown that all implications represented in Figure 8.5 can be derived; we can deduce from there that whenever $s\,\mathbf{C}[R]_{E_{\mathbf{P}}}\,t$, we have also that $s'\,\overline{(\mathbf{C}[R])}^{+}_{E_{\widehat{\mathcal{V}}}}\,t'$, where s' and t' are the states obtained by doing the eval action respectively on s and t. When combining this with the similar result we already have for the applicative actions in Lemma 8.3.10, we obtain Lemma 8.3.17 below.

Lemma 8.3.17 *Let R be a clustered binary relation for $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ such that $R \rightarrowtail R$. Then it holds that $\mathbf{C}[R] \rightarrowtail \overline{(\mathbf{C}[R])}^{+}$.*

Since the composition by contexts preserves ε-boundedness, it means that if R is a ε-bisimulation, then $\mathbf{C}[R]$ is a ε-bisimulation up to convex closure. From there—and since the convex closure is a valid up-to technique—we can conclude—as stated in Proposition 8.3.18 that the WLTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ is non-expansive.

Proposition 8.3.18 *The WLTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$—equipped with state-contexts as specified in Definitions 8.3.5 and 8.3.6—is non-expansive.*

Since we have shown non-expansiveness for $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$, we are now able to apply Proposition 8.3.4, which tells us that the trace distance on $\Lambda_{\oplus}^{\leq 1}$ is non-expansive.

Theorem 8.3.19 *$\delta^{tr}_{\Lambda_{\oplus}^{\leq 1}}$ is a non-expansive distance.*

We know—from Proposition 8.3.3—that the trace distance on $\Lambda_{\oplus}^{\leq 1}$ is observationally correct and—from Theorem 8.3.19—that it is also non-expansive. As a consequence, we are now able to apply Proposition 8.1.1, which allows us to finally conclude that it is sound.

Theorem 8.3.20 *The trace distance $\delta^{tr}_{\Lambda_{\oplus}^{\leq 1}}$ is sound with respect to the observational distance in $\Lambda_{\oplus}^{\leq 1}$.*

8.3.7 Full Abstraction of the Trace Distance on $\Lambda_{\oplus}^{\leq 1}$.

We now show that the trace distance is *fully-abstract*, i.e. that it coincides on programs with the observational distance. Since we already know that it is sound, what remains to be shown is that it is *complete*, i.e.:

$$\forall \text{ programs } M, N, \qquad \delta^{\mathrm{tr}}_{\Lambda_{\oplus}^{\leq 1}}(M,N) \geq \delta^{\mathrm{ctx}}_{\Lambda_{\oplus}^{\leq 1}}(M,N).$$

Our proof is based on the testing characterization of the bisimilarity distance on a WLTS—that we presented in Proposition 8.2.4. More precisely, we show that the traces on $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$ can be *emulated* by $\Lambda_{\oplus}^{\leq 1}$ contexts, that play a similar role to the applicative contexts we used to define the trace equivalence for CBN $\Lambda_{\oplus}$ in Definition 4.2.1 in Chapter 4—or the $\Lambda_{\oplus}$-contexts we used to show completeness of the probabilistic applicative bisimilarity for CBV $\Lambda_{\oplus}$ in Chapter 5—Observe that our construction is closer of the former than of latter, since we are using WLTSs here instead of LMCs. When we look at the set of those traces that are admissible from a state that consists in a sub-distribution over programs—that is, a state in $E_{\mathbf{P}}$—we see that:

$$\begin{aligned}\mathcal{T}(E_{\mathbf{P}}) = \{\epsilon\} \cup \{&\mathrm{eval}.V_1 \dots \mathrm{eval}.V_n.\mathrm{eval} \mid \forall i, V_i \text{ a value}\} \\ &\cup \{\mathrm{eval}.V_1 \dots \mathrm{eval}.V_n \mid \forall i, V_i \text{ a value}\}.\end{aligned}$$

For every trace $\alpha \in \mathcal{T}(E_{\mathbf{P}})$, we build a context that emulates α, i.e. for every program M, $\mathrm{Obs}(\mathcal{C}[M]) = \mathrm{Prob}(\alpha\downarrow)(s(M))$.

Definition 8.3.20 *Let α be a trace in $\mathcal{T}(E_{\boldsymbol{P}})$. We define the $\Lambda_{\oplus}^{\leq 1}$-context emulating α, that we denote $\mathcal{C}_\alpha$, as follows:*

$$\begin{aligned}\mathcal{C}_\epsilon &= [\cdot]; \\ \mathcal{C}_{\mathit{eval}.V_1 \dots \mathit{eval}.V_n.\mathit{eval}} &= [\cdot]V_1 \dots V_n; \\ \mathcal{C}_{\mathit{eval}.V_1 \dots \mathit{eval}.V_n} &= \mathcal{C}_{\mathit{eval}.V_1 \dots \mathit{eval}.V_{n-1}.\mathit{eval}}\end{aligned}$$

Observe that we indeed obtain $\Lambda_{\oplus}^{\leq 1}$-context, since for every trace α, $x \vdash \mathcal{C}[x]$ is indeed a valid affinity judgment. Moreover, we can see that for any program M, and for α admissible trace from $\{M^1\}$, it holds that: $\mathrm{Prob}(\{M^1\}\downarrow)(\alpha) = \mathrm{Obs}(\mathcal{C}_\alpha[M])$—the proof is done by induction on traces, and uses the fact that all applicative actions are deterministic. From there, looking at the inductive definition of bisimulation distance for WLTS, we can show that the context distance is a lower bound on the trace distance.

Proposition 8.3.21 (Completness of the Trace Distance on $\Lambda_{\oplus}^{\leq 1}$) $\delta^{ctx}_{\Lambda_{\oplus}^{\leq 1}} \leq \delta^{tr}_{\Lambda_{\oplus}^{\leq 1}}$.

Proof. Let M, N be two $\Lambda_{\oplus}^{\leq 1}$ programs. Using the trace characterization of bisimilarity distance—Proposition 8.2.4—we see that:

$$\delta^{\mathrm{tr}}_{\Lambda_{\oplus}^{\leq 1}}(M,N) = \sup_{\alpha \in \mathcal{T}(E_{\mathbf{P}})} |\mathrm{Prob}(\{M^1\}\downarrow)(\alpha) - \mathrm{Prob}(\{N^1\}\downarrow)(\alpha)|.$$

If we take $\alpha \in \mathcal{T}(E_{\mathbf{P}})$, we see now that $\mathrm{Prob}(\{M^1\}\downarrow)(\alpha) = \mathrm{Obs}(\mathcal{C}_\alpha[M])$, while $\mathrm{Prob}(\{N^1\}\downarrow)(\alpha) = \mathrm{Obs}(\mathcal{C}_\alpha[N])$. Now we observe that by definition of the observational distance it holds that:

$$|\mathrm{Obs}(\mathcal{C}_\alpha[M]) - \mathrm{Obs}(\mathcal{C}_\alpha[N])| \leq \delta^{\mathrm{ctx}}_{\Lambda_{\oplus}^{\leq 1}}(M,N).$$

Since it is true for every trace $\alpha \in \mathcal{T}(E_{\mathbf{P}})$, it implies that $\delta^{\mathrm{tr}}_{\Lambda_{\oplus}^{\leq 1}}(M,N) \leq \delta^{\mathrm{ctx}}_{\Lambda_{\oplus}^{\leq 1}}(M,N)$, which shows the result. □

When we combine this completeness result with the soundness result from Theorem 8.3.20, we obtain that the trace distance and the observational distance coincide on $\Lambda_{\oplus}^{\leq 1}$-programs.

Theorem 8.3.22 *The trace distance $\delta^{tr}_{\Lambda_{\oplus}^{\leq 1}}$ is fully abstract with respect to observational distance for $\Lambda_{\oplus}^{\leq 1}$.*

8.3.8 ε up-to composition by contexts and convex sums.

We have shown already that taking the convex closure is a valid quantitative up-to technique. We want now to study a more involved up-to technique, in the aim of generalizing to the quantitative case the up-to-contexts line of reasoning, that is a regular occurrence in the study of higher-order programming languages. Since we work in a probabilistic affine setting, we need to combine up-to contexts with up-to convex closure: more precisely, we are going to show that the operator $R \mapsto \overline{(\mathbf{C}[R])}^{+}$ is a valid quantitative up-to technique. First, we can see that this operator preserves ε-boundedness. Recall that in the course of the non-expansiveness proof, we have shown that:

$$R \rightarrowtail R \quad \Rightarrow \quad \mathbf{C}[R] \rightarrowtail \overline{(\mathbf{C}[R])}^{+}.$$

This statement is however weaker than what we need to show now, that is:

$$R \rightarrowtail \overline{(\mathbf{C}[R])}^{+} \quad \Rightarrow \quad \mathbf{C}[R] \rightarrowtail \overline{(\mathbf{C}[R])}^{+}.$$

The proof follows roughly the same path as the non-expansiveness proof. We consider separately the applicative actions and the eval action: we deal with the first ones in Lemma 8.3.23.

Lemma 8.3.23 *Let R be such that $R \rightarrowtail \overline{(\mathbf{C}[\mathcal{R}])}^{+}$. Let $V \in \mathcal{V}$, and s, t be such that $s\, \mathbf{C}[R]_{E_{\widehat{\mathcal{V}}}}\, t$. Then it holds that $u\, \overline{(\mathbf{C}[R])}^{+}_{E_{\boldsymbol{P}}}\, v$, where u, v are the $\mathscr{L}(\Lambda^{\leq 1}_{\oplus})$-states defined as $s \xrightarrow{V} u$, and $t \xrightarrow{V} v$.*

Proof. By looking at our definition of $\mathbf{C}[R]$, we see that there exists $E \in \mathscr{E}(\mathscr{L}(\Lambda^{\leq 1}_{\oplus}))$, $(s', t') \in R_E$, $\mathcal{C} \in \mathbf{C}(E \to E_{\widehat{\mathcal{V}}})$ such that $s = \mathcal{C}[s']$ and $t = \mathcal{C}[t']$. We distinguish two cases:

- either $\mathcal{C}$ is of the form $\lambda x.\mathcal{D}$, and then $u = \mathcal{D}[s']\{V/x\}$, and $v = \mathcal{D}[t']\{V/x\}$, and we see that the result holds.
- or $\mathcal{C} = [\cdot]$, and it implies that $E = E_{\widehat{\mathcal{V}}}$. In this case, we see that $s \xrightarrow{V} u$, $t \xrightarrow{V} v$ and from there we can conclude using the hypothesis on R.

□

We want now to show the counterpart of Lemma 8.3.23 for the action eval. In this end, we need once again to use the WLTS $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$. The main idea is as follows: as in the proof of non expansiveness, we reduce step-by-step an element in $(\mathbf{C} \times \mathcal{S})$ of the form $(\mathcal{C}, s)$, with s a $\mathscr{L}(\Lambda^{\leq 1}_{\oplus})$-state, but we now allow ourselves to also *rearrange* the $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$-state as each step of the reduction. Essentially, a rearrangement is a change in the way we split the underlying global $\mathscr{L}(\Lambda^{\leq 1}_{\oplus})$-state between a context and a program.

Definition 8.3.21 *Let S be a clustered relation on $\mathscr{L}(\Lambda^{\leq 1}_{\oplus})$, and (s, t) a pair of $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$-states in $(\mathbf{C}\Diamond S)$ such that there exist:*

- *I a finite set, a family $(E_i)_{i\in I}$ over $\mathscr{E}(\mathscr{L}(\Lambda^{\leq 1}_{\oplus}))$, and a family $(\mathcal{C}_i)_{i\in I}$ over states-contexts such that $\mathcal{C}_i \in \mathbf{C}(E_i \to E_{\boldsymbol{P}})$;*
- *for every i, a finite set J_i, a family $(F_{i,j})_{j\in I_j}$ over $\mathscr{E}(\mathscr{L}(\Lambda^{\leq 1}_{\oplus}))$, and a family $(\mathcal{D}_{i,j})_{j\in I_j}$ over $\mathbf{C}(F_{i,j} \to E_i)$;*

such that the $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$-states s and t can be written as follows:

$$s = \sum_{i\in I} \alpha_i \cdot \{(\mathcal{C}_i, s_i)^1\}; \qquad \forall i \in I,\ s_i = \sum_{j\in J_i} \beta_{i,j} \cdot \mathcal{D}_{i,j}[s'_{i,j}]; \qquad s'_{i,j} \in F_{i,j};$$
$$t = \sum_{i\in I} \alpha_i \cdot \{(\mathcal{C}_i, t_i)^1\}; \qquad \forall i \in I,\ t_i = \sum_{j\in J_i} \beta_{i,j} \cdot \mathcal{D}_{i,j}[t'_{i,j}]; \qquad t'_{i,j} \in F_{i,j};$$

with $\forall i \in I$, $(w(s_i) = 0 \wedge w(t_i) = 0)$ implies $J_i = \emptyset$. Then we say that the pair of states (u, v) is a meaningful rearrangement for the pair (s, t), *where u, v are defined as:*

$$u := \sum_{i\in I, j\in J} \alpha_i \cdot \beta_{i,j} \cdot \{(\mathcal{C}_i[\mathcal{D}_{i,j}], s'_{i,j})^1\}$$
$$v := \sum_{i\in I, j\in J} \alpha_i \cdot \beta_{i,j} \cdot \{(\mathcal{C}_i[\mathcal{D}_{i,j}], t'_{i,j})^1\}$$

Observe that rearranging does not change the underlying global $\mathscr{L}(\Lambda^{\leq 1}_{\oplus})$-state, that can be retrieved by the forgetful function $\mathbf{F}$: whenever (u, v) is a rearrangement of (s, t), it holds that $\mathbf{F}(s) = \mathbf{F}(u)$, and $\mathbf{F}(t) = \mathbf{F}(v)$. We can also observe that if s, t are both in normal form, then it is also the case that u and v are in normal form. Moreover, rearranging is exactly what is needed to go back to $\overline{(\mathbf{C}\Diamond R)}^{+}$ when we start from two states (u, v) obtained as $s \xrightarrow{\tau} u$, $t \xrightarrow{\tau} v$ with $(s, t) \in \overline{(\mathbf{C}\Diamond R)}^{+}$, as we state in Lemma 8.3.24 below.

Lemma 8.3.24 *Let s,t be two states in $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$ such that $s\,\overline{((\mathbf{C}\Diamond\overline{\mathbf{C}[R]}^{+}))}^{+}\,t$. Then there exists (u,v) a meaningful rearrangement of (s,t) such that $u\,\overline{((\mathbf{C}\Diamond R))}^{+}\,v$.*

We represent our proof technique in Figure 8.6. The idea is that—similarly to what we have done in the non-expansiveness proof—we start with two $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$-states that are in $(\mathbf{C}\Diamond R)$. Then, we reduce these two states step-by-step in parallel, and at each step we rearrange them in order to force them to go back in $\overline{((\mathbf{C}\Diamond R))}^{+}$. In Lemma 8.3.25 below, we show that this process indeed terminates—i.e. we reach $\mathcal{S}^{(\cdot,\cdot)}$-elements that are in normal form—and then in Lemma 8.3.26, we will show that by using $\mathbf{F}$ we can indeed retrieve from those normal forms the $\mathscr{L}(\Lambda^{\leq 1}_{\oplus})$-states produced by the action eval.

Lemma 8.3.25 *Let S be any clustered relation, and $(s,t)\in(\mathbf{C}\Diamond S)$. Then there exists no pair of infinite sequences $(u_n)_{n\in\mathbb{N}},(v_n)_{n\in\mathbb{N}}$ over $\mathcal{S}^{(\cdot,\cdot)}$ such that:*

1. *$u_0=s$, $v_0=t$;*
2. *if i is odd, $u_i\to_{\mathcal{S}^{(\cdot,\cdot)}}u_{i+1}$ and $v_i\to_{\mathcal{S}^{(\cdot,\cdot)}}v_{i+1}$;*
3. *if i is even, (u_{i+1},v_{i+1}) is a meaningful rearrangement of (u_i,v_i).*

Proof. Observe that we had to show in Lemma 8.3.13 a similar—but weaker—result —with the aim of defining the operational semantics on $(\mathbf{C}\times\mathcal{S})$. Recall that a key point there was that if $(\mathcal{C},s)\to_{(\mathbf{C}\times\mathcal{S})}\mathscr{D}$, then for every $(\mathcal{D},t)$ in the support of $\mathscr{D}$, $c(\mathcal{D})<c(\mathcal{C})$, where $c:\mathbf{C}\to\mathbb{N}$ is as defined in Proposition 7.4.1 from Chapter 7. Here, we cannot use the function $c(\cdot)$ anymore, because when rearranging a state $\{(\mathcal{C},s)^1\}$, we can obtain for instance a state $\{(\mathcal{D},t)^1\}$ with $c(\mathcal{C})<c(\mathcal{D})$. Instead, we build a function $p:(\mathbf{C}\times\mathcal{S})\to\mathbb{N}^3$, where $\mathbb{N}^3$ is equipped with the lexicographic order, as follows:

$$p((\mathcal{C},s))=(\sup\{c(M)\mid M\in\mathsf{S}(\mathbf{F}(\mathcal{C}[s]))\},\text{height}(s),\text{is-val}(s)),$$

where is-val$(s)=0$ if $s\in E_{\widehat{\mathcal{V}}}$, 1 if $s\in E_{\mathbf{P}}$; and height$(s)=\sup_{M\in\mathsf{S}(s)}\text{height}(M)$ with height(M) the number of symbols in M. We now associate to each pair $s,t\in\mathcal{S}^{(\cdot,\cdot)}$ a element in $\mathbb{N}^3$, by taking

$$p(s,t)=\sup\{p(\mathcal{C},s')\mid(\mathcal{C},s')\in(\mathsf{S}(s)\cup\mathsf{S}(t))\text{ not a }(\mathbf{C}\times\mathcal{S})\text{-value}\},$$

where the sup is taken with respect to the lexicographical order on $\mathbb{N}^3$. We first show the following auxiliary result: if $(s_n,t_n)_{n\in\mathbb{N}}$ is a sequence such that Conditions 1, 2, 3 hold, then:

1. if i is odd—i.e. there is a step of reduction in $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$—then $p(s_{i+1},t_{i+1})<p(s_i,t_i)$,
2. if i is even—i.e. the step consists in a rearranging—then $p(s_{i+1},t_{i+1})\leq p(s_i,t_i)$.

From this auxiliary result, we will be able to conclude since there can't be any infinite decreasing sequence in $\mathbb{N}^3$ equipped with the lexicographical order. We now show that our auxiliary result indeed holds:

- We first show item 2. We can see by looking at the definition of rearrangement that if $p(s_i,t_i)=(\alpha,\beta,\gamma)$, then the first component of $p(s_{i+1},t_{i+1})$ is also α—since a rearranging cannot change $\mathbf{F}(s_i)$, $\mathbf{F}(t_i)$. We then see that γ can increase only when β decreases, and that β cannot increase, and we have the result.

- We then show item 1. To do that, we see—by looking at the rules of Figure 8.2—that for any $(\mathcal{C}, s) \in (\mathbf{C} \times \mathcal{S})$, $(\mathcal{C}, s) \rightarrow_{(\mathbf{C}\times\mathcal{S})} \mathscr{D}$ implies that for every $(\mathcal{C}', s') \in \mathsf{S}(\mathscr{D})$, it holds that $p(\mathcal{C}', s') < p(\mathcal{C}, s)$.

□

It means that a sequence of pairs (s_i, t_i) as specified in Lemma 8.3.25 must at some point reach a pair (s_N, t_N), such that at least one among s_N, t_N is in normal form for $\rightarrow_{\mathcal{S}^{(\cdot,\cdot)}}$. In Lemma 8.3.26 below, we show that when it happens, then both s_N, t_N are in normal form, and moreover when applying the forgetful function, we get back the pair (u, v) of $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$-states defined as $\mathbf{F}(s_0) \xrightarrow{\text{eval}} \widehat{u}$ and $\mathbf{F}(t_0) \xrightarrow{\text{eval}} \widehat{v}$.

Lemma 8.3.26 *Let S be any clustered relation, and s, t be two states connected by $(\mathbf{C}\Diamond S)$. Let $(un)_{0\leq n\leq N}, (v_n)_{0\leq n\leq N}$ be two maximal finite sequences—i.e. there cannot be extended—such that the conditions 1, 2, 3 of Lemma 8.3.25 hold. Then it holds that u_N and v_N are in normal for for $\rightarrow_{\mathcal{S}^{(\cdot,\cdot)}}$, and moreover $\mathbf{F}(u_N) = [\![\mathbf{F}(s)]\!]$, and $\mathbf{F}(v_N) = [\![\mathbf{F}(s)]\!]$.*

Proof. The proof is similar to the one we did when showing validity for the operational semantics on $(\mathbf{C} \times \mathcal{S})$. □

We are now ready to show the counterpart of Lemma 8.3.23 for the action eval.

Lemma 8.3.27 *Let R be a clustered relation on the states of the WLTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$, such that $R \rightarrowtail \overline{(\mathbf{C}[\mathcal{R}])}^{+}$. Then it holds that, for every s, t such that $s\, \mathbf{C}[R]_{E_{\mathbf{P}}}\, t$, $u\, \overline{(\mathbf{C}[R])}^{+}_{E_{\widehat{\mathcal{V}}}}\, v$, with u, v defined by $s \xrightarrow{eval} u$, and $t \xrightarrow{eval} v$.*

Proof. The proof is as represented in Figure 8.6. First $s\, \mathbf{C}[R]_{E_{\mathbf{P}}}\, t$ means that there exist $E \in \mathscr{E}(\mathscr{L}(\Lambda_{\oplus}^{\leq 1}))$, and a state-context $\mathcal{C} \in \mathbf{C}(E \rightarrow E_{\mathbf{P}})$, such that $s = \mathcal{C}[s']$, and $t = \mathcal{C}[t']$. Looking at the $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$-states $\{(\mathcal{C}, s')^1\}$ and $\{(\mathcal{C}, t')^1\}$, we see—using Lemma 8.3.24—that because $R \rightarrowtail \overline{(\mathbf{C}[\mathcal{R}])}^{+}$, there exists a pair of sequences $(u_n)_{1\leq n\leq N}, (v_n)_{1\leq n\leq N}$ over the states of $\mathscr{L}^{(\mathbf{C}\times\mathcal{S})}$ such that:

- $u_0 = \{(\mathcal{C}, s')^1\}$, $v_0 = \{(\mathcal{C}, t')^1\}$, and if i is odd, $u_i \rightarrow_{(\mathbf{C}\times\mathcal{S})} u_{i+1}$, $v_i \rightarrow_{(\mathbf{C}\times\mathcal{S})} v_{i+1}$, while if i is even, (u_{i+1}, v_{i+1}) is a meaningful rearrangement of (u_i, v_i).
- N is even, and one among the u_N, v_N is in normal-form for $\rightarrow_{(\mathbf{C}\times\mathcal{S})}$;
- if i is even, $(u_i, v_i) \in \overline{(\mathbf{C}\Diamond\overline{(\mathbf{C}[R])}^{+})}^{+}$; if i is odd, $(u_i, v_i) \in \overline{(\mathbf{C}\Diamond R)}^{+}$.

Observe that we can indeed suppose that one among u_N, v_N is in normal form, because otherwise we could do one more step $\rightarrow_{(\mathbf{C}\times\mathcal{S})}$. It means that we can apply Lemma 8.3.26, that tells us: $\widehat{\mathbf{F}(u_N)} = u$, and $\widehat{\mathbf{F}(v_N)} = v$. We moreover know that $(u_N, v_N) \in \overline{(\mathbf{C}\Diamond R)}^{+}$, so when we apply the forgetful function to this pair, we end up in $\overline{(\mathbf{C}[R])}^{+}$, i.e. it means that $(\mathbf{F}(u_N), \mathbf{F}(v_N)) \in \overline{(\mathbf{C}[R])}^{+}$. We are now able to conclude by using Lemma 8.3.11, that tells that $\overline{(\mathbf{C}[R])}^{+}$ is preserved by $\widehat{\ }$. □

Proposition 8.3.28 *The closure up-to convex sum combined with the composition by context $R \mapsto \overline{(\mathbf{C}[R])}^{+}$ is a valid up-to technique on the WLTS $\mathscr{L}(\Lambda_{\oplus}^{\leq 1})$.*

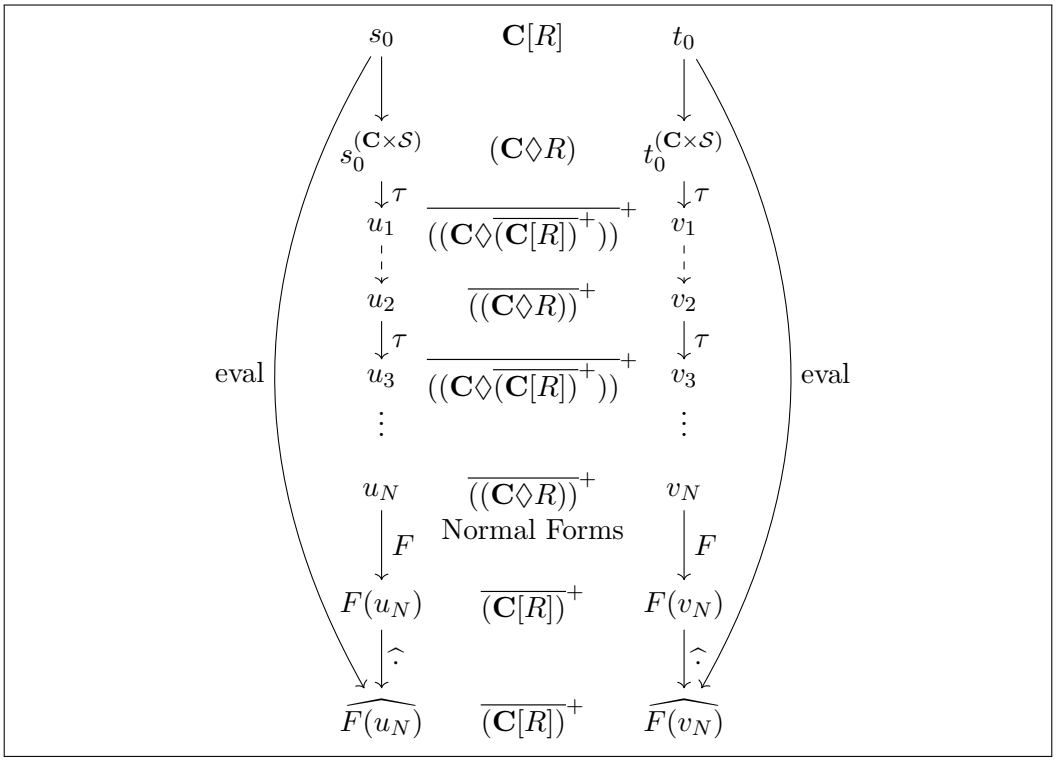


Figure 8.6: Proof Scheme for validity of the up-to technique $R \mapsto \overline{(\mathbf{C}[R])}^{+}$.

Proof. Recall that since we know already that taking the convex closure is a valid up-to technique, it is enough to show that:

$$R \rightarrowtail \overline{(\mathbf{C}[\mathcal{R}])}^{+} \qquad \Rightarrow \qquad \mathbf{C}[R] \rightarrowtail \overline{(\mathbf{C}[R])}^{+}.$$

So we take R such that $R \rightarrowtail \overline{(\mathbf{C}[\mathcal{R}])}^{+}$. Lemma 8.3.23 and Lemma 8.3.27 tell us that the right-hand side holds. □

We now illustrate how the up-to techniques we have just developed help us to compute the trace distance between $\Lambda^{\leq 1}_{\oplus}$-programs.

Example 8.3.5 *Consider, as an example, a sequence of terms $(M_n)_{n\in\mathbb{N}}$ defined inductively as follows:*

$$M_0 = \lambda x.\Omega; \qquad N_0 = \lambda x.\Omega;$$
$$M_{n+1} = \lambda y.(yM_n); \qquad N_{n+1} = \lambda y.(y(N_n \oplus^{\frac{1}{2}^{n+1}} \Omega)).$$

We define $\alpha_N := \prod_{1\leq i\leq N}(1-\frac{1}{2^N})$. Our goal is to show that the trace distance between M_N and N_N is smaller or equal to $(1-\alpha_N)$. To do that, we fix $N \in \mathbb{N}$, and we build a $(1-\alpha_N)$-bisimulation up-to closure by oriented contexts. We first define a sequence $(\beta^N)_{0\leq i\leq N}$ of non-negative real numbers in $[0,1]$, by: $\beta_N = 1$, and $\beta_i^N = \prod_{j=i+1}^{N}(1-\frac{1}{2^j})$. From there, we take as ε-bisimulation candidate the reflexive closure of:

$$R^N = \{(\{{M_N}^1\},\{{N_N}^1\})\} \cup \{(\{\widehat{M_i}^1\}, \beta_i \cdot \{\widehat{N_i}^1\}) \mid 0 \leq i \leq N\}$$
$$\cup \{(\{{M_i}^1\}, \beta_{i+1} \cdot \{\widehat{N_i \oplus^{\frac{1}{2^{i+1}}} \Omega}^1\}) \mid 0 \leq i \leq N-1\}.$$

*We can first see that R^N is indeed $(1-\alpha_N)$-bounded: indeed, for every $(s,t) \in R^N$, it holds that $|w(s) - w(t)| \leq (1 - \inf_{0\leq i\leq N} \beta_i^N) = (1-\alpha_N)$. Figure 8.7 allows us to see that moreover $R^N \rightarrowtail \mathbf{C}[R]^N$, which concludes the proof. Observe that it would have been much more complicated to find directly a $(1-\alpha_N)$-*bisimulation*, since it has to contain all the pairs $(\{VM_{N-1}{}^1\}, \{V(N_{N-1} \oplus_{\frac{1}{2^{N-1}}} \Omega)^1\})$ for every value V.*

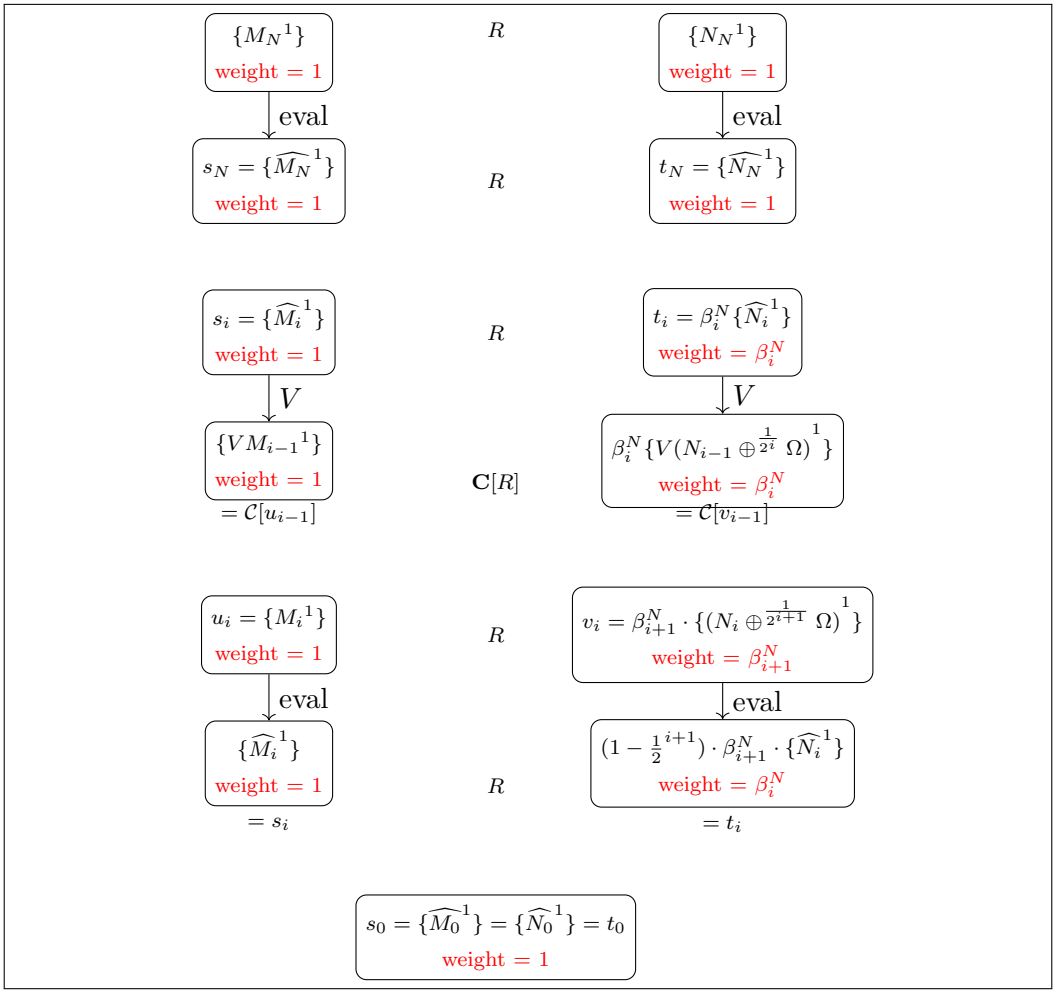


Figure 8.7: Fragment of the WLTS $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$ to compute the trace distance between M_N and N_N.

8.4 Trace Distance for $\Lambda^!_\oplus$

Recall that the bisimilarity distance on WLTSs $\mathscr{L}(\Lambda_\oplus{}^{\mathrm{cbn}})$ and $\mathscr{L}(\Lambda_\oplus{}^{\mathrm{cbv}})$ lead to *unsound* distances for respectively CBN $\Lambda_\oplus$ and CBV $\Lambda_\oplus$. Moreover, as we will see in the next chapter, it is also the case when we consider the bisimilarity distance on the LMC $\mathcal{M}^{\mathrm{cbv}}_{\Lambda_\oplus}$. So in order to account for the copying phenomenon, we need a significant change into the structure of the underlying transition systems. Our approach consists in asking a state of the transition system to represent *several* programs—instead of just one program, as we have done until now. This way, we become able to keep track of the evolution of

several copies of the same program. In this chapter, we build accordingly an *applicative WLTS for* $\Lambda^!_\oplus$ and thus define the *trace distance for* $\Lambda^!_\oplus$, using the bisimilarity distance on this WLTS. We then show that this trace distance is fully abstract with respect to the observational distance for $\Lambda^!_\oplus$.

The work in this section has been published in a joint paper with Ugo Dal Lago [26]. A long version of this paper can be found in [25]. It contains in particular the details of the non-expansiveness proof, which is similar—while more involved—to the non-expansiveness proof for the affine calculus $\Lambda^{\leq 1}_\oplus$, and that we will not recall in full details in the present thesis.

8.4.1 Some Useful Terminology and Notation

In this section, we will make heavy use of sequences of terms and types. It is thus convenient to introduce some terminology and notation about them.

A finite (ordered) sequence whose elements are $e_1, \ldots, e_n$ will be indicated as $\boldsymbol{e} = [e_1, \ldots, e_n]$, and called an *$n$-sequence*. Metavariables for sequences are boldface variations of the metavariables for their elements. Whenever $E = \{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$ and $i_1 < \ldots < i_m$, the sub-sequence $[e_{i_1}, \ldots, e_{i_m}]$ of an n-sequence $\boldsymbol{e}$ will be indicated as $\boldsymbol{e}_E$. If the above holds, E will be called an *n-set*. If $\boldsymbol{e}$ is an n-sequence, and $\mathcal{I}$ is a permutation on $\{1, \ldots, n\}$, we note $e_\mathcal{I}$ the n-sequence $[e_{\mathcal{I}(1)}, \ldots, e_{\mathcal{I}(n)}]$. We can turn an n-sequence into an $(n+1)$-sequence by adding an element at the end: this is the role of the semicolon operator. We denote by $[e^n]$ the n-sequence in which all components are equal to e. We denote by $\bar{\mathbf{0}}$ the unique 0-sequence—i.e. the empty sequence.

Whenever this does not cause ambiguity, notations like the ones above will be used in conjunction with syntactic constructions. For example, if $\boldsymbol{\sigma}$ is an n-sequence of types, then $!\boldsymbol{\sigma}$ stands for the sequence $[!\sigma_1, \ldots, !\sigma_n]$. As another example, if $\boldsymbol{\sigma}$ is an n-sequence of types and E is an n-set, then $\boldsymbol{x}_E : \boldsymbol{\sigma}_E$ stands for the environment assigning type σ_i to x_i for every $i \in E$. As a final example, if $\boldsymbol{M}$ is an n-sequence of terms and $\boldsymbol{\sigma}$ is an n-sequence of types, $\vdash \boldsymbol{M} : \boldsymbol{\sigma}$ holds iff $\vdash M_i : \sigma_i$ is provable for every $i \in \{1, \ldots, n\}$.

8.4.2 The WLTS $\mathscr{L}(\Lambda^!_\oplus)$.

Our first building block towards formalizing what the states of $\mathscr{L}(\Lambda^!_\oplus)$ are is the notion of a *tuple*: a *tuple element* consists in two sequences of programs, where the programs in the first sequence are non-evaluated programs meant to be duplicable, while programs in the other one are not allowed to be copied.

Definition 8.4.1 Tuples *are pairs of the form $K = (\boldsymbol{M}, \boldsymbol{N})$ where $\boldsymbol{M}$ and $\boldsymbol{N}$ are sequences of $\Lambda^!_\oplus$-programs. The set of all such tuples is indicated as $\mathbb{TUPLES}$. The first component of a tuple is called its* exponential part*, while the second one is called its* linear part*.*

We now equip tuple elements with types, in order to be able to control the type of each program contained in the tuple. Moreover, we design typing judgments for tuples which specify that the programs that appear at some fixed indexes in the linear part of the tuple must be $\Lambda^!_\oplus$-values. Observe that this idea is a generalization of what happens in $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$, where we have the class of programs, and the class of distinguished values. We first extend our typing judgments on $\Lambda^!_\oplus$-program: if M is a program, and $\sigma^\star = (\sigma, b) \in \mathscr{A} \times \{\bot, \top\}$, we write $\vdash M : \sigma^\star$ when $\vdash M : \sigma$ and moreover $((b = \top) \Rightarrow M \in \mathcal{V}_{\Lambda^!_\oplus})$.

Definition 8.4.2 *We define the set of* tuple types *as*

$$\mathbf{T} = \{(\boldsymbol{\sigma}, \boldsymbol{\tau}^\star) \mid \boldsymbol{\sigma} \text{ sequence over } \mathscr{A},\ \boldsymbol{\tau}^\star \text{ sequence over } \mathscr{A} \times \{\top, \bot\}\}.$$

We write $\vdash (\boldsymbol{M}, \boldsymbol{N}) : (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star)$ *if* $\vdash \boldsymbol{M} : \boldsymbol{\sigma}$, $\vdash \boldsymbol{N} : \boldsymbol{\tau}^\star$.

We say that $(\boldsymbol{\sigma}, \boldsymbol{\tau})$ is a (n, m)-*tuple type* if $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$ are, respectively, an n-sequence and an m-sequence. To make the notations more readable, if $(\sigma, b) \in \mathscr{A} \times \{\bot, \top\}$, we will sometimes denote it as σ^b.

Definition 8.4.3 *We call a pair* $(K, A) \in \mathbb{TUPLES} \times \mathbf{T}$ *such that* $\vdash K : A$ *an* explicitly typed tuple. *We denote by* $T_\vdash$ *the set of all explicitly typed tuples.*

In Example 8.4.1 below, we illustrate this notion of valid typing judgment for tuples.

Example 8.4.1 *We consider the program* $M := (\lambda!x.I) \oplus \Omega$. *Observe first—using the typing rules for* $\Lambda^!_\oplus$ *given in Chapter 7—that* $\vdash M : \sigma$ *with* $\sigma := !\alpha \to \beta \to \beta$. *We use this program* M *to build a tuple element: we take* $K = ([M, \Omega], [I])$. *We look now for a tuple type for* K*: we take* $A := ([\sigma, \alpha \to \alpha], [(\alpha \to \alpha)^\top])$. *Since* $\alpha \to \alpha$ *is a valid type for both* I *and* Ω, *and that moreover* I *is a value, we see that* $\vdash K : A$, *hence* (K, A) *is an instance of explicitly typed tuple.*

We now define the set of states we will use for the $\Lambda^!_\oplus$-applicative WLTS. Recall that the states of the WLTS $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$ were of two kinds: sub-distributions over programs, and sub-distributions over distinguished values. Here, we generalize this idea: the set of states is the disjoint union over all tuple types A, of sub-distribution over tuples of type A. We formalize this in Definition 8.4.4 below.

Definition 8.4.4 (States of the Applicative WLTS for $\Lambda^!_\oplus$) *We define the set of* tuple-states $\mathcal{S}_{\Lambda^!_\oplus}$ *as follows:*

$$\mathcal{S}_{\Lambda^!_\oplus} := \bigcup_{A \text{ tuple type}} \{(\mathscr{D}, A) \mid \mathscr{D} \in \Delta(\{K \mid \vdash K : A\})\}$$

Similarly to what we have done for the WLTS $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$ for the affine calculus, or the WLTS $\mathscr{L}({\Lambda_\oplus}^{\mathrm{cbn}})$ for $\Lambda_\oplus$, we associate a state to every valid typing judgment on $\Lambda^!_\oplus$-programs. First, to any program M, we associate a tuple element $\dot{M}$ as follows: we put M in the linear part, and we leave the exponential part empty.

Definition 8.4.5 *Let* M *be a* $\Lambda^!_\oplus$*-program, and* σ *a* $\Lambda^!_\oplus$*-type. We define* $\dot{M}$ *as the tuple element* $\dot{M} := (\bar{\mathbf{0}}, [M])$, *and* $\dot{\sigma}$ *as the tuple type* $\dot{\sigma} := (\bar{\mathbf{0}}, [\sigma^\bot])$. *If* $\kappa = (\vdash M : \sigma)$ *is a valid typing judgment in* $\Lambda^!_\oplus$, *we define the* state associated to κ *as* $s(\kappa) := (\{\dot{M}^1\}, \dot{\sigma}) \in \mathcal{S}_{\Lambda^!_\oplus}$.

We will also use the notation $s_\sigma(M)$ to mean $s(\vdash M : \sigma)$, when $\vdash M : \sigma$ is a valid typing judgment.

Example 8.4.2 *As in Example 8.4.1, we consider* $M := (\lambda!x.I) \oplus \Omega$, *and* $\sigma := !\alpha \to \beta \to \beta$. *Then the tuple state associated to the typing judgment for program* $(\vdash M : \sigma)$ *is as follows:*

$$s(\vdash M : \sigma) = (\{(\bar{\mathbf{0}}, [(\lambda!x.I) \oplus \Omega])^1\}, (\bar{\mathbf{0}}, \left[\sigma^\bot\right])).$$

Action Modalities for the environment

How do states in $\mathcal{S}_{\mathscr{M}^!_\oplus}$ interact with the environment? This is captured by the labels in $\mathcal{A}_{\mathscr{M}^!_\oplus}$, and the associated probability matrix. As the WLTSs $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$ and $\mathscr{L}(\Lambda_\oplus{}^{\mathrm{cbn}})$, $\mathscr{L}(\Lambda^!_\oplus)$ has applicative and evaluation actions. However, it has also two other kinds of actions, called *storing actions* and *Milner's actions*, that are designed to track more closely the copying phenomenon. All these actions are *type directed* in the sense that when we consider some state $s = (\mathscr{D}, A) \in \mathcal{S}_{\mathscr{M}^!_\oplus}$, the ability of the environment of performing an action a depends *only* on the type A, and similarly the type we will obtain *after* having done the action a depends also uniquely on A. Similarly, the actions respect the convex structure of the states, in the sense that the effect of a on $(\mathscr{D}, A)$ can be recovered—in the natural way—from the action of a on the $(\{K^1\}, A)$ for all $K \in \mathsf{S}(\mathscr{D})$. As a consequence, our approach to define the transition relation for the WLTS $\mathscr{L}(\Lambda^!_\oplus)$ is as follows: for each kind of action, we first specify the effect those actions have on types—i.e. for each action a of this kind, we give a relation $\xrightarrow{a} \subseteq \mathbf{T} \times \mathbf{T}$—and then we specify the effect they have on *atomic states* —i.e. we give a relation $\xrightarrow{a} \subseteq T_\vdash \times \mathcal{S}_{\Lambda^!_\oplus}$. From there, we will put back everything together, and we will define the full transition relation in Definition 8.4.18.

Evaluation Actions

We first define the counterpart of the eval action for $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$. The base principle is that only programs in the linear part of the tuple can be evaluated: this reflects the fact that in the operational semantics of $\Lambda^!_\oplus$, no reduction can take place inside !-boxes. Observe that a major difference with the WLTS $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$, or the applicative WLTSs for $\Lambda_\oplus$, is that a state can now contain possibly more than one non-evaluated program available to the environment. It means that the environment must choose to *which one* of those programs he wants to apply an evaluation action. It must be a program that has not been already evaluated, i.e. it must be typed by a $\tau^\perp$; after the evaluation, the tuple type will be updated to indicate that the program has been evaluated.

Definition 8.4.6 (Evaluation Actions on Types) *We define the* transition reduction at type level for evaluation actions: $\xrightarrow{eval^i} \subseteq \mathbf{T} \times \mathbf{T}$ *by:*

$$(\boldsymbol{\sigma}, \boldsymbol{\tau}^\star) \xrightarrow{eval^i} (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star{}_{\{1,\dots,i-1\}}; \tau_i^\top; \boldsymbol{\tau}^\star{}_{\{i+1,\dots,n\}}) \qquad \text{when } \tau_i^\star = \tau_i^\perp.$$

If A is in the domain of $\xrightarrow{eval^i}$, we say that $eval^i$ is an admissible action *on A.*

Definition 8.4.7 *Let $i \in \mathbb{N}$. We define $\xrightarrow{eval^i} \subseteq T_\vdash \times \mathcal{S}_{\Lambda^!_\oplus}$ —that we call* atomic transition relation for the action eval^i *—as follows: if $k = (K, A) \in T_\vdash$ such that $eval^i$ is an admissible action for A, we say that* eval^i is an admissible action for k*, and we define:*

$$k \xrightarrow{eval^i} (\sum_V [\![N_i]\!](V) \cdot \{(\boldsymbol{M}, \boldsymbol{N}_{\{1,\dots,i-1\}}; V; \boldsymbol{N}_{\{i+1,\dots,n\}})^1\}, B),$$

where the sequences $\boldsymbol{M}$, $\boldsymbol{N}$ are such that $K = (\boldsymbol{M}, \boldsymbol{N})$, and $A \xrightarrow{eval^i} B$.

Storing Actions

We introduce now the *storing actions*: it corresponds to the environment choosing a term of the form $!M$ from the linear part of the underlying tuple, unboxing it and transferring

the content to the exponential part of the tuple. Observe that for specifying such an action, it is sufficient to give the index corresponding to the position of the program that we want to unbox. We will denote $(?^i)$ the corresponding label, so we define:

$$\mathcal{A}_? := \{(?^i) \mid i \in \mathbb{N}\}.$$

Definition 8.4.8 (The Storing Action on Types) *Let $A = (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star)$ be a tuple type. We say that $(?^i)$ is an* admissible action at the type A *if $\exists \iota$ with $\tau_i = !\iota^\top$. If it is the case, we write $A \xrightarrow{(?^i)} B$ with $B = (\boldsymbol{\sigma}; \iota, \boldsymbol{\tau}^\star_{\{1,\dots,n\}\setminus\{i\}})$.*

Observe that whenever the action $(?^i)$ is admissible for some type for tuple A, it holds that for every tuple $K = (\boldsymbol{M}, \boldsymbol{N})$ of type A, the program N_i must be of the form $!L$. This remark allows us to define the storing action on such tuple K: we remove N_i of the linear part of the tuple, and we store L in the exponential part.

Definition 8.4.9 (The Storing Action on Tuple Element) *Let $i \in \mathbb{N}$. We define $\xrightarrow{(?^i)} \subseteq T_\vdash \times \mathcal{S}_{\Lambda^!_\oplus}$ —that we call* transition relation for the action $(?^i)$ *—as follows: if $(K, A) \in T_\vdash$ is such that $(?^i)$ is an admissible action at the type A, we say that $(?^i)$* is an admissible action for (K, A)*, and we define:*

$$(K, A) \xrightarrow{(?^i)} (\{(\boldsymbol{M}; L, \boldsymbol{N}_{\{1,\dots,n\}\setminus\{i\}})^1\}, B)$$

where B is the unique type with $A \xrightarrow{(?^i)} B$, the sequences $\boldsymbol{M}$, $\boldsymbol{N}$ are such that $K = (\boldsymbol{M}, \boldsymbol{N})$, and the program L is such that $N_i = !L$.

Please observe that this action is in fact deterministic: it means that the resulting tuple element is uniquely determined.

Example 8.4.3 *We consider the tuple element $K = [I], [I, !\Omega]$, and we note $\sigma = \alpha \to \alpha$. We can see that $\vdash K : A$ with $A = ([\sigma], [\sigma^\perp, !\sigma^\top]) \in \mathbf{T}$. Looking at Definition 8.4.8 and Definition 8.4.9, we see that the action $(?^1)$ is admissible at the type A, so the atomic transition relation $\xrightarrow{(?^1)}$ is admissible from (K, A), and:*

$$(K, A) \xrightarrow{(?^1)} (\{([I, \Omega], [I])^1\}, ([\sigma, \sigma], [\sigma^\perp])).$$

Milner's Actions

We now look at *Milner's actions*, that we named this way in reference to the so called Milner's law $!A \equiv A \otimes !A$ in Linear Logic. The action $(!^i)$ takes the i-th term in the exponential part of the tuple, makes a copy of it, evaluates it and put the result into the linear part. Observe that—as storing actions—a Milner's action is entirely specified by giving the index i identifying the program. We denote $(!^i)$ the corresponding label, and accordingly we define $\mathcal{A}_! := \{(!^i) \mid i \in \mathbb{N}\}$.

Definition 8.4.10 (Milner's action on Tuple Types) *Let $A = (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star)$ be a tuple type. We say that $(!^i)$ is an* admissible action at the type A *if the length of the exponential part $\boldsymbol{\sigma}$ is not smaller than i; when it is the case, we define $A \xrightarrow{(!^i)} (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star; \sigma_i^\top)$.*

Definition 8.4.11 (Milner's action on Tuple Elements) *Let $i \in \mathbb{N}$. We define $\xrightarrow{(!^i)} \subseteq T_\vdash \times \mathcal{S}_{\Lambda^!_\oplus}$ —that we call* transition relation for the action $(!^i)$ *—as follows: if $(K, A) \in T_\vdash$ is such that $(!^i)$ is an admissible action at the type A, we say that $(!^i)$ is an* admissible action for (K, A)*, and we define:*

$$(K, A) \xrightarrow{(!^i)} (\sum_V [\![M_i]\!](V) \cdot \{(\boldsymbol{M}; \boldsymbol{N}; V)^1\}, B)$$

where B is the unique tuple type with $A \xrightarrow{(!^i)} B$, and the sequences $\boldsymbol{M}$, $\boldsymbol{N}$ are such that $K = (\boldsymbol{M}, \boldsymbol{N})$.

Example 8.4.4 *We note $\sigma = \alpha \to \alpha$. We define respectively a tuple element K and a type for tuple A as:*

$$K = ([I \oplus \Omega], [I]); \qquad \text{and} \qquad A = ([\sigma], \left[\sigma^\top\right]).$$

We see that $\vdash K : A$ holds, and that the action $(!^1)$ is admissible at the indexed tuple type A, hence the action $(!^1)$ is admissible when starting from (K, A). We see that:

$$(K, A) \xrightarrow{(?^1)} (\frac{1}{2} \cdot \{([I \oplus \Omega], [I, I])^1\}, ([\sigma], \left[\sigma^\top, \sigma^\top\right])).$$

Applicative Actions

We look now at applicative actions, that play the same role as the applicative action on $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$: they model environments passing arguments to programs. Here, we have to adapt this idea to our tuple framework: indeed, we can see the tuple as a collection of programs available to the environment, which is free to choose *with which* of the programs to interact with by passing it an argument. A crucial point, however, is that when building this argument, the environment is allowed to use *other components* of the tuple. For this reason, we need to be able to express all open terms M that can be build by the environment using free variables that are designed to be replaced by components of the tuple.

We first look at a more general problem, which is to describe how we can transform a tuple into another tuple—in a uniform way: to do this, we define here a notion of *tuple context*, that we will use also later. Intuitively, a tuple context is simply a tuple whose components can possibly have free variables, taken in the set of special symbols $[\cdot]^!_1, \ldots, [\cdot]^!_n, \ldots, [\cdot]_1, \ldots, [\cdot]_n$—that are *not* regular variables, so cannot be used as bounded variables.

Definition 8.4.12 *A* tuple context *is a pair $(\boldsymbol{M}, \boldsymbol{N})$ where $\boldsymbol{M}$ and $\boldsymbol{N}$ are sequences of (not necessarily closed) typable terms, and their free variables are contained in the set $\{[\cdot]_i \mid i \in \mathbb{N}\} \cup \{[\cdot]^!_i \mid i \in \mathbb{N}\}$.*

If $\mathcal{C} = (\boldsymbol{M}, \boldsymbol{N})$ is a tuple context, and $B = (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star)$, $A = (\boldsymbol{\iota}, \boldsymbol{\upsilon}^\star)$ are tuple types, we say that $\mathcal{C}$ is a tuple context from A to B*, and we note $\mathcal{C} \in \mathbf{C}(B \to A)$ when:*

- $![\cdot]^! : !\boldsymbol{\sigma} \vdash \boldsymbol{M}_i : \boldsymbol{\iota}_i$;
- *We note $\boldsymbol{\tau}$ and $\boldsymbol{\upsilon}$ the underlying sequences over $\mathscr{A}$ from $\boldsymbol{\tau}^\star$ and $\boldsymbol{\upsilon}^\star$, and n and m their respective length. We ask that there exists a partition $\{E_1, \ldots E_m\}$ of $\{1, \ldots, n\}$ such that for every $j \in \{1, \ldots, m\}$, it holds that $![\cdot]^! : !\boldsymbol{\sigma}, [\cdot]_{E_j} : \boldsymbol{\tau}_{E_j} \vdash N_j : \upsilon_j$.*
- *for every j such that $\upsilon^\star_j$ is of the form $\upsilon^\top_j$, then either $N_j = [\cdot]_i$ with $\tau^\star_i = \tau^\top_i$, or N_j is of the form $\lambda z.N$, $\lambda !z.N$, or $!N$.*

We now formalize how we can *fill* a tuple context with a tuple element:

Definition 8.4.13 *Let A, B be two tuple types, and $\mathcal{C} = \boldsymbol{L}, \boldsymbol{K} \in \mathbf{C}(A \to B)$. Let $K = (\boldsymbol{M}, \boldsymbol{N})$ such that $\vdash K : A$. Then we define $\mathcal{C}[K] = (\boldsymbol{T}, \boldsymbol{U}) \in \mathbb{TUPLES}$ where:*

$$\forall i \in \{1, \dots, n\},\ T_i = L_i\{\boldsymbol{M}/[\cdot]^!\}$$
$$\forall i \in \{1, \dots, m\},\ U_i = K_i\{\boldsymbol{M}/[\cdot]^!\}\{\boldsymbol{N}/[\cdot]\},$$

and n, m are such that B is a (n, m)-tuple type.

Observe that when $\mathcal{C} \in \mathbf{C}(A \to B)$, and $\vdash K : A$, it indeed holds that $\vdash \mathcal{C}[K] : B$. Recall that our goal here is to build a value that will be passed as argument to one of the values of type $\sigma \to \tau$ contained in the linear part of the original tuple, in order to model the environment doing an applicative action—so we need to generate a tuple that consists only of a value in the linear part. More precisely, if we fix A a tuple type, and σ a $\Lambda^!_\oplus$-type that we take as target, we have to ask for a tuple context $\mathcal{C} \in \mathbf{C}(A \to (\bar{\mathbf{0}}, \lceil \sigma^\top \rceil))$: indeed it guarantees that $\mathcal{C}[K]$ will be of the form $\dot{V}$, with V some value of type σ, as soon as $\vdash K : A$. We also have to keep track of the programs in the linear part that are consumed when building the value V, hence if A is a tuple type, and E a set of indexes, we define a set $\mathbf{C}^{\mathcal{V}}(A \to \sigma \mid E)$ that corresponds to all the way of generating a value when having in input a tuple of type A, and such that the only non-duplicable programs used are the ones with index in E.

Definition 8.4.14 *Let $A = (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star)$ an tuple type. We denote by $\mathbf{C}^{\mathcal{V}}(A \to \sigma \mid E)$ the set of those $\Lambda^!_\oplus$-terms M—with possible free variables the $[\cdot]_i$ and $[\cdot]^!_i$—such that $(\bar{\mathbf{0}}, [M]) \in \mathbf{C}((\boldsymbol{\sigma}, \boldsymbol{\tau^\star}_E) \to (\bar{\mathbf{0}}, \lceil \sigma^\top \rceil))$.*

We now define the *applicative actions*: we need to specify which tuple type we take as starting point, the index i_0 of the non-duplicable program to which we want to pass a value as argument, and a $\Lambda^!_\oplus$-term that will be used to generate the argument.

Definition 8.4.15 *We call* applicative labels *the objects of the form (A, i, E, M) where:*

- *$A = (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star)$ is an tuple type, and there exists ι, υ such that $\tau_i^\star = (\iota \to \upsilon)^\top$;*
- *M is a $\Lambda^!_\oplus$-term such that $M \in \mathbf{C}^{\mathcal{V}}(A \to \iota \mid E)$, and $i \notin E$.*

We note $\mathcal{A}_@$ the set of all applicative labels.

We will also sometimes denote $@^i_A(E, M)$ for the label (A, i, E, M). An applicative action (A, i, E, M) is admissible only when starting from the type A. After having done the action, the resulting tuple type is as follows: τ_i must be replaced by υ, and all the programs in the linear part—hence usable only *once*—that have been used to build the argument must be removed.

Definition 8.4.16 (Applicative actions on Types) *Let $B = (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star)$ be a tuple type, and $a = (A, i, E, M)$ be an applicative label . We say that this label is an* admissible action for the tuple type B *when $A = B$. If it is the case, we write $B \xrightarrow{a} (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star{}_{\{1,\dots,n\}\setminus(E\cup\{i\})}; \upsilon^\top)$, with $A = (\boldsymbol{\sigma}, \boldsymbol{\tau}^\star)$, and υ the $\Lambda^!_\oplus$-type such that $\tau_i^\star = (\iota \to \upsilon)^\top$.*

Definition 8.4.17 (Applicative actions on Tuple elements) *Let $a = (A, i, E, M)$ be an applicative label. We define $\xrightarrow{a} \subseteq T_\vdash \times \mathcal{S}_{\Lambda^!_\oplus}$ as follows: for every $(K, A) \in T_\vdash$ such that a is admissible for A, we take:*

$$(K, A) \xrightarrow{a} (\sum_V [\![W_i(M[K])]\!](V) \cdot \{(\boldsymbol{M}, \boldsymbol{N}_{\{1,\dots,n\}\setminus(E\cup\{i\})}; V)^1\}, B).$$

where B is the unique type with $A \xrightarrow{a} B$, the sequences $\boldsymbol{M}$, $\boldsymbol{N}$ are such that $K = (\boldsymbol{M}, \boldsymbol{N})$.

Definition of the WLTS

We are now ready to define the applicative WLTS for $\Lambda^!_\oplus$. Our definition is based on the atomic transition relations $\xrightarrow{a}$ that we have defined for evaluation actions, storing actions, Milner's actions, and applicative actions respectively in Definitions 8.4.7, 8.4.9, 8.4.11 and 8.4.17. We also add actions indexed by tuple types, in order to be always able to distinguish between two explicitly typed tuples with distinct types.

Definition 8.4.18 *We define the* applicative WLTS for $\Lambda^!_\oplus$ *as* $\mathscr{L}(\Lambda^!_\oplus) = (\mathcal{S}_{\Lambda^!_\oplus}, \mathcal{L}_{\Lambda^!_\oplus}, \to, w)$ *where:*

- $\mathcal{S}_{\Lambda^!_\oplus}$ *is as defined in Definition 8.4.4;*
- $\mathcal{L}_{\Lambda^!_\oplus} := \{eval^i \mid i \in \mathbb{N}\} \cup \mathcal{A}_! \cup \mathcal{A}_? \cup \mathcal{A}_@ \cup \mathscr{A}_{\mathbb{TUPLES}}$*;*
- *The transition relation is defined as follows: if* $s = (\mathscr{D}, A)$*, then an action* a *is admissible for* s *if:*
  - *or* $a = A$*, and then* $s \xrightarrow{A} s$*.*
  - *or* $a \in \{eval^i \mid i \in \mathbb{N}\} \cup \mathcal{A}_! \cup \mathcal{A}_? \cup \mathcal{A}_@$*, and the action* a *is admissible at the type* A*. Then:*
$$s \xrightarrow{a} (\sum_{K \in \mathscr{D}} \mathscr{D}(K) \cdot \mathscr{E}_K, B),$$
where B *is defined by* $A \xrightarrow{a} B$*, and for every* $K \in \mathsf{S}(\mathscr{D})$*,* $\mathscr{D}_K$ *is defined by* $(K, A) \xrightarrow{a} (\mathscr{E}_K, B)$*.*
- $w(\mathscr{D}, A) = |\mathscr{D}|$*.*

Example 8.4.5 *We give in Figure 8.8 a fragment of* $\mathscr{L}(\Lambda^!_\oplus)$*, designed to illustrate our definitions. We consider programs of the form* $M_\varepsilon = !(\Omega \oplus^\varepsilon I)$*, for* $\varepsilon \in \mathbb{D}$*. We type those programs with the recursive type* $\sigma := \mu\alpha.(!\alpha \to !\alpha)$*: recall that it means that* $\sigma = !\sigma \to !\sigma$*. We look at the effect of* some *of the actions that can be triggered by the environment when starting from the state:*

$$s_{!\sigma}(M_\varepsilon) = (\{(\bar{\mathbf{0}}, [M_\varepsilon])^1\}, \bar{\mathbf{0}}, \left[!\sigma^\perp\right]).$$

8.4.3 The Trace Distance for $\Lambda^!_\oplus$

Our goal now is to use the WLTS $\mathscr{L}(\Lambda^!_\oplus)$ to build a distance on $\Lambda^!_\oplus$-programs, as we have done for the affine language $\Lambda^{\leq 1}_\oplus$ with $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$. We first look at which states of $\mathscr{L}(\Lambda^!_\oplus)$ are comparable: we have to find the equivalence classes for $R^1_{\mathscr{L}(\Lambda^!_\oplus)}$. As stated in Lemma 8.4.1 below, these equivalence classes are obtained by taking all states with the same underlying tuple type.

Lemma 8.4.1 *For every tuple type* A*, we define* $E_A := \{s \in \mathcal{S}_{\Lambda^!_\oplus} \mid \exists \mathscr{D}, s = (\mathscr{D}, A)\}$*. Then* $\mathscr{E}(\mathscr{L}(\Lambda^{\leq 1}_\oplus)) = \{E_A \mid A \in \mathbf{T}, E_A \neq \emptyset\}$*.*

Definition 8.4.19 *We define the* applicative trace distance *as the distance on* $\Lambda^!_\oplus$ *programs* $(\delta^{tr}_{\Lambda^!_\oplus})_\sigma : \boldsymbol{P}_\sigma \times \boldsymbol{P}_\sigma \to [0,1]$ *taken as:*

$$(\delta^{tr}_{\Lambda^!_\oplus})_\sigma(M, N) := \delta^{\mathscr{L}(\Lambda^!_\oplus)}_{E_{\dot\sigma}}(s_\sigma(M), s_\sigma(N)).$$

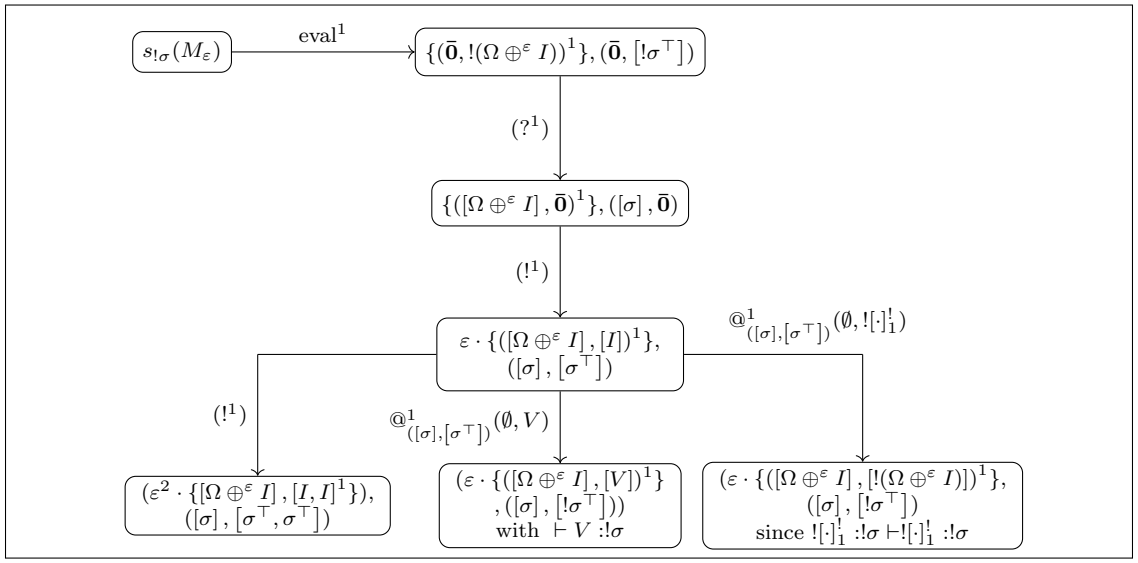


Figure 8.8: A Fragment of $\mathscr{L}(\Lambda^!_\oplus)$

Example 8.4.6 *Recall that the two $\Lambda^{\leq 1}_\oplus$-programs $(\Omega \oplus^{\varepsilon_1} I)$ and $(\Omega \oplus^{\varepsilon_2} I)$ are at observational distance $|\varepsilon_1 - \varepsilon_2|$ —as we can see easily for instance using the characterization by traces of the observational distance for $\Lambda^{\leq 1}_\oplus$. We want now to look at what happens when we are allowed to* duplicate *these programs, so we consider the family of $\Lambda^!_\oplus$-programs $M_\varepsilon := !(\Omega \oplus^\varepsilon I)$ for $\varepsilon \in \mathbb{D}$ that we introduced in Example 8.4.5. We fix any type τ, and define $\sigma = \tau \to \tau$; we can see that $\vdash M_\varepsilon : !\sigma$. In the present Example, we show—again by using a characterization by traces—that:*

$$(\delta^{tr}_{\Lambda^!_\oplus})_\sigma(M_{\varepsilon_1}, M_{\varepsilon_2}) \geq \sup_{n \in \mathbb{N}} |\varepsilon_1^n - \varepsilon_2^n|.$$

Indeed, observe that $s(\vdash M_\varepsilon : !\sigma) = (\{(\bar{\mathbf{0}}, [!(\Omega \oplus^\varepsilon I)])^1\}, (\bar{\mathbf{0}}, [!\sigma^\perp]))$. Let us now consider, for every $n \in \mathbb{N}$, the trace $\alpha_n = eval^1 \cdot (?^1) \cdot (!^1)^n$. For every $\varepsilon \in [0, 1]$, the execution of the trace α_n starting from $s_{M_\varepsilon} := s(\vdash M_\varepsilon : !\sigma)$ is as follows: $s_{M_\varepsilon} \overset{eval^1}{\to} t \overset{(?^1)}{\to} s_0 \overset{(!^1)}{\to} \ldots \overset{(!^1)}{\to} s_n$ with:

$$t = (\{(\bar{\mathbf{0}}, [!(\Omega \oplus^\varepsilon I)])^1\}, (\bar{\mathbf{0}}, \left[!\sigma^\top\right]))$$
$$s_i = (\varepsilon^i \cdot \{([\Omega \oplus^\varepsilon I], [I, \ldots, I])^1\}, ([\sigma], \left[\sigma^\top, \ldots, \sigma^\top\right]));$$

As a consequence, we see that $Prob(s_{M_\varepsilon} \downarrow)(\alpha_n) = \varepsilon^n$, which allows us to conclude that:

$$(\delta^{tr}_{\Lambda^!_\oplus})_{!\sigma}(M_{\varepsilon_1}, M_{\varepsilon_2}) \geq \sup_{n \in \mathbb{N}} |Prob(s_{M_{\varepsilon_1}} \downarrow)(\alpha_n) - Prob(s_{M_{\varepsilon_2}} \downarrow)(\alpha_n)| = \sup_{n \in \mathbb{N}} |\varepsilon_1^n - \varepsilon_2^n|.$$

We will show later—using an up-to technique combining contexts and convex closure—that the other inequality also holds.

Our first correctness criterion for a distance on programs is that it should be *observationally correct*, in the sense of Definition 8.1.3, i.e. that the distance should distinguish *at least as much* as the context $[\cdot]$, that simply executes the program it receives as input. As we show in Proposition 8.4.2 below, it is indeed the case for the trace distance on $\Lambda^!_\oplus$ that we have just defined.

Proposition 8.4.2 *The trace distance on $\Lambda^!_\oplus$-programs $\delta^{tr}_{\Lambda^!_\oplus}$ is observationally correct.*

Proof. Let σ be a type in $\mathscr{A}_{\Lambda^!_\oplus}$, and M, N two $\Lambda^!_\oplus$-programs at this type. Our goal is to show that $(\delta^{\mathrm{tr}}_{\Lambda^!_\oplus})_\sigma(M,N) \geq |\mathrm{Obs}_\sigma(M) - \mathrm{Obs}_\sigma(N)|$. As a first step, we unfold the definition of $(\delta^{\mathrm{tr}}_{\Lambda^!_\oplus})_\sigma$, and rewrite the observational correctness requirement using the trace distance on $\mathscr{L}(\Lambda^!_\oplus)$. It becomes:

$$\delta^{\mathscr{L}(\Lambda^!_\oplus)}_{E_{\dot\sigma}}(s_\sigma(M), s_\sigma(N)) \geq |\mathrm{Obs}_\sigma(M) - \mathrm{Obs}_\sigma(N)|.$$

We are going to use the characterization by traces of distance bisimilarity. Observe first that $w(s_\sigma(L)) = \mathrm{Obs}_\sigma(M)$ for every L with $\vdash L : \sigma$. Then we see that the empty trace ϵ is admissible for every state s, and besides $w(s) = \mathrm{Prob}(s\downarrow)(\alpha)$, hence we can conclude:

$$\begin{aligned}\delta^{\mathscr{L}(\Lambda^!_\oplus)}_{E_{\dot\sigma}}(s_\sigma(M), s_\sigma(N)) &\geq |\mathrm{Prob}(s_\sigma(M)\downarrow)(\epsilon) - \mathrm{Prob}(s_\sigma(N)\downarrow)(\epsilon)| \\ &= |w(s_\sigma(M)) - w(s_\sigma(N))| = |\mathrm{Obs}_\sigma(M) - \mathrm{Obs}_\sigma(N)|.\end{aligned}$$

□

Full Abstraction

The proofs of soundness and non-expansiveness for the trace distance on $\Lambda^!_\oplus$ follow the same path as the ones for the $\Lambda^{\leq 1}_\oplus$ trace distance—that we have presented in details in Section 8.3.1. We give here the main steps, but a complete proof can be found in [25]. We first equip the WLTS $\mathscr{L}(\Lambda^!_\oplus)$ with state-contexts—in the sense of Definition 8.3.3—since we want to be able to express non-expansiveness as a property of the WLTS $\mathscr{L}(\Lambda^!_\oplus)$. To do this, we use the tuple-contexts that we have build in Definition 8.4.12: recall that for every pair of tuple types A, B, we called $\mathbf{C}(A \to B)$ the set of tuple context with A as point of departure, and B as target. Here, we need to define sets of state-contexts $\mathbf{C}(E_1 \to E_2)$ for every pair E_1, E_2 of equivalence classes in $\mathscr{E}(\mathscr{L}(\Lambda^!_\oplus))$. Recall that we have shown in Lemma 8.3.2 that $\mathscr{E}(\mathscr{L}(\Lambda^!_\oplus)) = \{E_A \mid A \in \mathbf{T}\}$, where the E_A are all the states of the form $(\mathscr{D}, A) \in \mathcal{S}_{\Lambda^!_\oplus}$.

Definition 8.4.20 *We take $\mathbf{C}(E_A \to E_B) := \mathbf{C}(A \to B)$. If $s = (\mathscr{D}, A) \in E_A$, and $\mathcal{C} = (\boldsymbol{M}, \boldsymbol{N}) \in \mathbf{C}(A \to B)$, we denote by $\mathcal{C}[s]$ the state in E_B defined as:*

$$\mathcal{C}[s] := (\sum_K \mathscr{D}(K) \cdot \{\mathcal{C}[K]^1\}.$$

Any context for the programming language $\Lambda^!_\oplus$ can also be seen as a state-context: we state this formally in Remark 8.4.1 below.

Remark 8.4.1 *For every $\Lambda^!_\oplus$-program M of type σ, and $\mathcal{C}$ a $\Lambda^!_\oplus$-context such that $[\cdot] : \sigma \vdash \mathcal{C} : \tau$. Then $\mathcal{C}' = (\overline{\mathbf{0}}, [\mathcal{C}\{[\cdot]_1/[\cdot]\}])$ is a state-context in $\mathbf{C}(E_{\dot\sigma} \to E_{\dot\tau})$. By abuse of notation, we will again note $\mathcal{C}$ for the state-context $\mathcal{C}'$. Observe that with these notation, for every program M of type σ, $s_\tau(\mathcal{C}[M]) = \mathcal{C}[s_\sigma(M)]$.*

We first show that proving non-expansiveness for the WLTS $\mathscr{L}(\Lambda^!_\oplus)$ is a valid approach to show non-expansiveness of the trace distance on the programming language $\Lambda^!_\oplus$.

Proposition 8.4.3 *If the WLTS $\mathscr{L}(\Lambda^!_\oplus)$ equipped with state-contexts as specified in Definition 8.4.12 is non-expansive, then the trace distance on $\Lambda^!_\oplus$ is also non-expansive.*

Proof. Let M, N be two $\Lambda^!_\oplus$-programs of type σ, and $\mathcal{C}$ a $\Lambda^!_\oplus$-context such that $[\cdot] : \sigma \vdash \mathcal{C} : \tau$. To prove non-expansiveness of the trace distance, we need to show: $\delta^{\mathscr{L}(\Lambda^!_\oplus)}_{E_{\dot\tau}}(s_\tau(\mathcal{C}[M]), s_\tau(\mathcal{C}[N])) \leq \delta^{\mathscr{L}(\Lambda^!_\oplus)}_{E_{\dot\sigma}}(s_\sigma(M), s_\sigma(N))$. Using Remark 8.4.1, we are able to rewrite our objective as: $\delta^{\mathscr{L}(\Lambda^!_\oplus)}_{E_{\dot\tau}}(\mathcal{C}[s_\sigma(M)], \mathcal{C}[s_\sigma(N)]) \leq \delta^{\mathscr{L}(\Lambda^!_\oplus)}_{E_{\dot\sigma}}(s_\sigma(M), s_\sigma(N))$. We see that the latter statement holds as soon as the WLTS $\mathscr{L}(\Lambda^!_\oplus)$ is non-expansive, which concludes the proof. □

The path to show non-expansiveness for the WLTS $\mathscr{L}(\Lambda^!_\oplus)$ is broadly similar to the one we followed when showing non-expansiveness of the WLTS $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$. The first step consists in equipping $\mathscr{L}(\Lambda^!_\oplus)$ with a convex sum operator. As for $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$, our construction here is based on the fact that finite measure are a cone: if $s = (\mathscr{D}, A)$ and $t = (\mathscr{E}, A)$ are two states in E_A, we can take $\alpha \cdot s + \beta \cdot t := (\alpha \cdot \mathscr{D} + \beta \cdot \mathscr{E}, A)$. It means that we can consider the convex closure of any clustered relation on states, as defined in Definition 8.3.9. Moreover, we can see that the convex structure $\mathscr{L}(\Lambda^!_\oplus)$ is well-behaved, in the sense of Definition 8.3.12: as a consequence, using Proposition 8.3.9, we see that taking the convex closure is a valid up-to technique. However, there is a major difference with $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$, that essentially comes from the fact that $\Lambda^{\leq 1}_\oplus$ is an affine calculus, hence the operational semantics of a $\Lambda^!_\oplus$-program has always finite support. It is not the case here for $\Lambda^!_\oplus$, and as illustrated in Example 8.4.7 below, it implies that we cannot replicate the main component of the non-expansiveness proof for $\Lambda^{\leq 1}_\oplus$—the fact that as soon as R is a ε-bisimulation, it is also the case of $\overline{(\mathbf{C}[R])}^+$.

Example 8.4.7 *We fix $\sigma = \sigma^{cbn} \to \sigma^{cbn}$—where σ^{cbn} is the type for the encoding of CBN $\Lambda_\oplus$ into $\Lambda^!_\oplus$, that we defined in Chapter 7—, and we build a $\frac{1}{2}$-bisimulation on $\mathscr{L}(\Lambda^!_\oplus)$ as follow:*

$$R = \{(s_\sigma(\Omega), s_\sigma(I \oplus \Omega))\} \cup \{(\emptyset, A), (\frac{1}{2} \cdot \mathscr{D}, A) \mid (\mathscr{D}, A) \in \mathcal{S}_{\Lambda^!_\oplus}\}.$$

We are now going to build a particular context $\mathcal{C}$ in $\mathbf{C}(E_{\dot\sigma} \to \dot\sigma)$, such that it is not the case that $(\mathcal{C}[s_\sigma(\Omega)], \mathcal{C}[s_\sigma(I \oplus \Omega)]) \rightarrowtail \overline{(\mathbf{C}[R])}^+$. To do this, we are going to use the program $L \in \Lambda_\oplus$—that we build in Example 1.2.3 and analyzed in Example 1.2.4 of Chapter 1—such that its operational semantics in CBN $\Lambda_\oplus$ is $[\![L]\!] = \sum_{1 \leq n}^{\infty} \frac{1}{2^n} \cdot \{\underline{n}^1\}$, where $\underline{n}$ is the n-th Scott natural number. We look at the encoding of L in $\Lambda^!_\oplus$, and we denote again $\underline{n}$ for the encoding of $\underline{n}$ in $\Lambda^!_\oplus$.

We now consider the $\Lambda^!_\oplus$-context $\mathcal{C} = \lambda x.(\lambda y.[\cdot]x)L$, and we transform it into a tuple context by taking as usual $(\bar{\mathbf{0}}, [\mathcal{C}\{[\cdot]_1/[\cdot]\}])$. We denote again $\mathcal{C}$ this state-contexts, and we see that $\mathcal{C} \in \mathbf{C}(\dot\sigma \to \dot\sigma)$. We see that:

$$\mathcal{C}[s_\sigma(M)] = (\{(\bar{\mathbf{0}}, [\mathcal{C}[\Omega]])^1\}, \dot\sigma) \xrightarrow{eval^1} s = \sum_{n \in \mathbb{N}} \frac{1}{2^n} \cdot \{\bar{\mathbf{0}}, [\lambda y.(\Omega \underline{n})]^1\}, (\bar{\mathbf{0}}, \left[\sigma^\top\right])$$

$$\mathcal{C}[s_\sigma(N)] = (\{(\bar{\mathbf{0}}, [\mathcal{C}[I \oplus \Omega]])^1\}, \dot\sigma) \xrightarrow{eval^1} t = \sum_{n \in \mathbb{N}} \frac{1}{2^n} \cdot \{\bar{\mathbf{0}}, [\lambda y.((I \oplus \Omega)\underline{n})]^1\}, (\bar{\mathbf{0}}, \left[\sigma^\top\right])$$

But we can see now that $(s, t) \notin \overline{(\mathbf{C}[R])}^+$, hence $\overline{(\mathbf{C}[R])}^+$ is not a $\frac{1}{2}$-bisimulation.

This problem comes from the fact that the probabilistic execution tree of a $\Lambda^!_\oplus$-program can have infinite branches. To overcome this, we define the closure of a relation *up-to infinite convex sum*. We first need to add additional structure to a WLTS equipped with a convex structure, in order to be able to talk about infinite sum of states.

Definition 8.4.21 *We say that a WLTS $\mathscr{L}^w$ is equipped with a* countable convex sum operator*, if it is equipped with a convex structure, and that moreover for every $E \in \mathscr{E}(\mathscr{L}^w)$, the cone C_E is sequentially complete—in the sense of Definition 11.1.1 in Chapter 9.*

The completeness requirement on cones guarantee that, if $(s_n)_{n\in\mathbb{N}}$ is a sequence in E for some $E \in \mathscr{E}(\mathscr{L}^{\mathrm{w}})$, such that all the terms of the increasing sequence $t_N = \sum_{n\leq N} s_n$ are still in E, then the sequence $(t_N)_{N\in\mathbb{N}}$ has a supremum in E, that we denote by $\sum_{n\in\mathbb{N}} s_n$.

Definition 8.4.22 *Let $\mathscr{L}^w$ be a WLTS equipped with a countable convex sum convex operator. Let R be a clustered binary relation on states. We call* closure of R by infinite convex sum*, and we denote $\overline{(R)}^\star$ the clustered relation defined as: for every $E \in \mathscr{E}(\mathscr{L}(\Lambda^!_\oplus))$:*

$$\begin{array}{ll} \forall (s_n)_{n\in\mathbb{N}}, (t_n)_{n\in\mathbb{N}} \text{ sequences over } E,\ \forall (p_n)_{n\in\mathbb{N}} \text{ with } \sum p_i \leq 1, & \\ (\forall n \in \mathbb{N}, s_n\, R\, t_n) \quad \Rightarrow \quad (\sum_{n\in\mathbb{N}} p_n \cdot s_n)\, \overline{(R)}^\star\, (\sum_{n\in\mathbb{N}} p_n \cdot t_n). & \end{array}$$

The key of our non-expansiveness proof is that the operator $R \mapsto \overline{(\mathbf{C}[R])}^\star$ preserves ε-bisimulation on the WLTS $\mathscr{L}(\Lambda^!_\oplus)$. The proof of this result—that can be found in [25]—is similar to the one we did to show that $R \mapsto \overline{(\mathbf{C}[R])}^+$ preserves ε-bisimulation on $\mathscr{L}(\Lambda^{\leq 1}_\oplus)$, but more involved, since we need to deal with approximation semantics.

Proposition 8.4.4 *Let R be a ε-bisimulation on $\mathscr{L}(\Lambda^!_\oplus)$. Then $\overline{(\mathbf{C}[R])}^\star$ is also an ε-bisimulation.*

Since $\mathbf{C}[R] \subseteq \overline{(\mathbf{C}[R])}^\star$, Proposition 8.4.4 tells us that the WLTS $\mathscr{L}(\Lambda^!_\oplus)$ is non-expansive.

Theorem 8.4.5 *The WLTS $\mathscr{L}(\Lambda^!_\oplus)$, equipped with state-contexts as specified in Definition 8.4.12, is non-expansive.*

As a consequence, we obtain using Proposition 8.4.3 that the trace distance on $\Lambda^!_\oplus$ is non-expansive.

Theorem 8.4.6 (Non-Expansiveness) *The trace distance on $\Lambda^!_\oplus$ is non-expansive.*

Since we have shown that the trace distance on $\Lambda^!_\oplus$ is both observationally correct and non-expansive, we can now use Proposition 8.1.1, and concludes that it is sound with respect to observational distance. We can also prove completeness of the trace distance, by using the inductive characterization of the bisimilarity distance on the WLTS $\mathscr{L}(\Lambda^!_\oplus)$. Similarly to the $\Lambda^{\leq 1}_\oplus$ case, the proof—that can be found in [25]—is based on building contexts design to *emulate* traces. By combining completeness with soundness, we obtain full-abstraction as stated in Theorem 8.4.7 below.

Theorem 8.4.7 (Full Abstraction) *For every σ, for every program M, N of type σ, it holds that:*

$$(\delta^{tr}_{\Lambda^!_\oplus})_\sigma(M,N) = (\delta^{ctx}_{\Lambda^!_\oplus})_\sigma(M,N).$$

8.4.4 Up-to techniques on the applicative WLTS for $\Lambda^!_\oplus$

We also looked at up-to techniques on the applicative WLTS $\mathscr{L}(\Lambda^!_\oplus)$. A detailed account can be found in [25], we recall here only the main results.

Proposition 8.4.8 *Both $R \mapsto \overline{(R)}^\star$ and $R \mapsto \overline{(\mathbf{C}[R])}^\star$ are valid quantitative up-to techniques on $\mathscr{L}(\Lambda^!_\oplus)$.*

We can observe that these quantitative up-to technique are respectively the quantitative counterpart of the *up-to lifting technique* and *up-to lifting and contexts technique* that Vignudelli and Sangiorgi [107] develop for environmental bisimulation on $\Lambda_\oplus$.

Example 8.4.8 *Recall the programs $M_\varepsilon := !(\Omega \oplus^\varepsilon I)$, for $\varepsilon \in \mathbb{D}$, that we considered in Example 8.4.6, and we fix σ any type of the form $\tau \to \tau$. We have shown in Example 8.4.6—using the trace characterization of the bisimilarity distance on $\mathscr{L}(\Lambda^!_\oplus)$ a lower bound on the trace distance between M_{ε_1} and M_{ε_2} for any $\varepsilon_1, \varepsilon_2 \in \mathbb{D}$:*

$$(\delta^{tr}_{\Lambda^!_\oplus})_{!\sigma}(M_{\varepsilon_1}, M_{\varepsilon_2}) \geq \sup_{n\in\mathbb{N}} |\varepsilon_1^n - \varepsilon_2^n|.$$

Using our up-to techniques, we are now able to show that this lower bound is also an upper bound. In order to simplify the notations, we define for every $\varepsilon \in \mathbb{D}$, and $n \in \mathbb{N} \cup \{-1\}$, the states $s^\varepsilon_n \in \mathscr{L}^w_{\Lambda^!_\oplus}$ as:

$$s^\varepsilon_{-1} = (\{(\bar{\mathbf{0}}, !(\Omega \oplus^\varepsilon I)^1\}, (\bar{\mathbf{0}}, \left[!\sigma^\top\right]))$$
$$s^\varepsilon_n = (\varepsilon^n \cdot \{([(\Omega \oplus^\varepsilon I)], \bar{\mathbf{0}})^1\}, ([\sigma], \bar{\mathbf{0}})),$$

We fix $\varepsilon_1, \varepsilon_2 \in \mathbb{D}$, and we define the relation R as:

$$R = \{(s_{!\sigma}(M_{\varepsilon_1}), s_{!\sigma}(M_{\varepsilon_2}))\} \cup \{(s^{\varepsilon_1}_n, s^{\varepsilon_2}_n) \mid n \in \mathbb{N} \cup \{-1\}\}.$$

For any ε, We represent in Figure 8.9 the effect of all *admissible actions that start from the states $s_{!\sigma}(M_\varepsilon)$ and s^ε_n. We can see that R is a α-bisimulation up to $R \mapsto \overline{(\mathbf{C}[R])}^\star$, with $\alpha = \sup_{n\in\mathbb{N}} |\varepsilon_1^n - \varepsilon_2^n|$. Since, as stated in Proposition 8.4.8, the infinite convex closure of the composition by contexts is a valid up-to technique, we obtain that:*

$$(\delta^{tr}_{\Lambda^!_\oplus})_\sigma(M_{\varepsilon_1}, M_{\varepsilon_2}) = \sup_{n\in\mathbb{N}} |\varepsilon_1^n - \varepsilon_2^n|.$$

8.4.5 Trace Distance for $\Lambda_\oplus$ via encoding in $\Lambda^!_\oplus$

In [25], we also show that our results on the trace distance for $\Lambda^!_\oplus$ can be lifted back to ordinary probabilistic λ-calculi $\Lambda_\oplus$, both when call-by-name evaluation and call-by-value are considered. It is based on the embeddings of CBN and CBV $\Lambda_\oplus$ into $\Lambda^!_\oplus$, that we have presented in Section 7.3 of Chapter 7. More precisely, we defined $\mathscr{L}({\Lambda_\oplus}^{CBN})$ and $\mathscr{L}({\Lambda_\oplus}^{CBV})$ two *restrictions* of the WLTS $\mathscr{L}(\Lambda^!_\oplus)$ for respectively the CBN and the CBN operational semantics, by keeping only the states and action relevant in the setting of the embedding, and we showed that the trace distances obtained this way are fully abstract. It may be remarked that the WLTS obtained this way looks closely like the structure used by Vignudelli and Sangiorgi to define environmental bisimulation on CBN and CBV $\Lambda_\oplus$ in [107]. In this sense, what we obtain in this case can be seen as a quantitative extension of the environmental bisimulation studied by Vignudelli and Sangiorgi. We actually develop our trace distance for $\Lambda^!_\oplus$ roughly at the same time as Vignudelli and Sangiorgi worked on environmental bisimulation for probabilistic λ-calculi, and our common point of departure was the behavioral equivalences for $\Lambda_\oplus$ developed by Dal Lago, Sangiorgi and Alberti in [32]. Our objectives were however different, since our goal was to be able to talk about distances instead of equivalences, while they were looking for a notion of applicative equivalence compatible with imperative features as the ability for programs to store data.

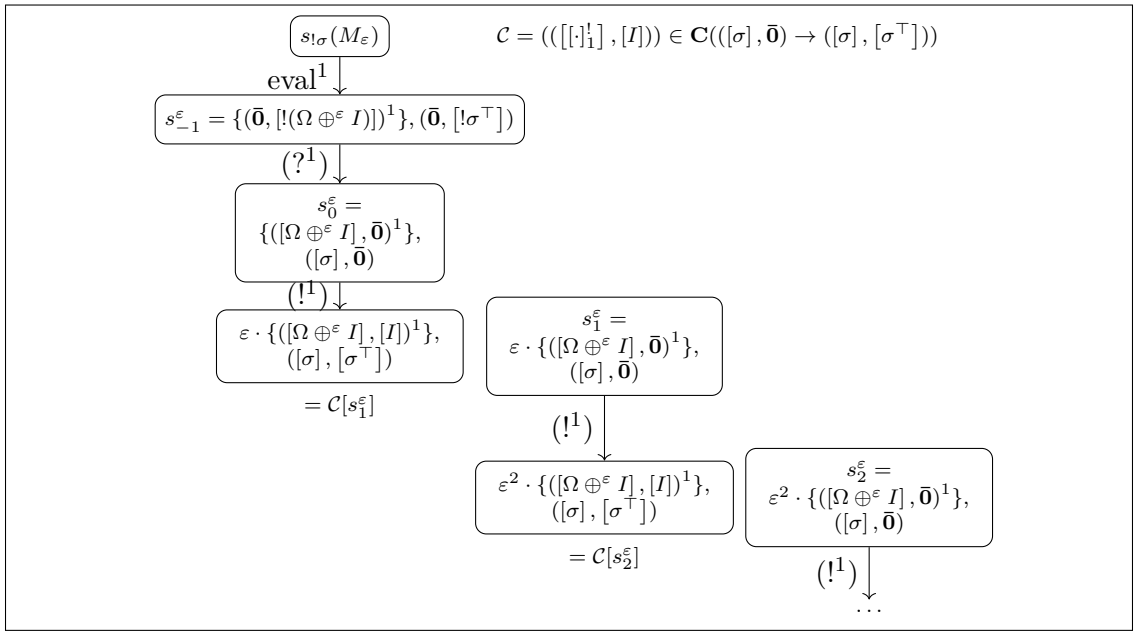


Figure 8.9: A Fragment of $\mathscr{L}(\Lambda^!_\oplus)$

It is remarkable that the answer to these two problems actually happened to be similar, i.e. to look at states that track the evolution of several programs—that correspond to the program we test, and its environment in environmental bisimulation—instead of just one program.

Part III

Denotational Semantics for Higher-Order Languages with Probabilities.

Chapter 9

Background: Two Denotational Models for Higher-Order Languages with Probability

The growing interest in enhancing programming languages with various probabilistic primitives led to a drive for adapting existing denotational semantics in order to make them able to interpret probabilistic behaviors. The first denotational semantics for an higher-order language with *probabilistic features* was put forward by Saheb-Djahromi in [102], but while overall his language is higher-order, the probability distributions are considered only on values of *ground* types, i.e. integers and booleans. To handle his language, Saheb-Djahromi adapts the non-deterministic powerdomain—considered before by Plotkin—into a *probabilistic powerdomain* for ground domains, i.e. the ones that interpret the integers and booleans types. Then, in their pioneering work [68, 67] Jones and Plotkin extend the probabilistic powerdomain to *all* types, and use it to give a denotational semantics for a higher-order language similar to $\mathsf{PCF}_{\oplus}$, resulting in a *computationally adequate* model. Their construction is based on Moggi's work on computational λ-calculus [86]—where he gives a denotational semantics to a λ-calculus with generic *computational effect* in any Cartesian closed category equipped with a strong monad—that they apply with a monad built from the probabilistic power-domain. This monadic approach has been continued by several works [10] [84] aiming to add a notion of *approximation* to Jones and Plotkin's model, with changes to both the monad and the underlying category. Another branch of the domain-theoretic approach based on the study of powerdomains for probability led to the introduction of *Kegelspitzen* by Keimel and Plotkin [95] that combines both the structure of a convex set and the structure of a directed-complete partial order; it has resulted in a denotational model of a variant of $\mathsf{PCF}_{\oplus}$ based on Kegelspitzen that has been built in [98]. A very different perspective on denotational semantics for higher-order probabilistic languages has been brought from the tradition of *games semantics*, where types are interpreted as *games*, and programs as *strategies* on these games. The first probabilistic model using games semantics is defined by Danos and Harmer [35], who introduced a *fully abstract* semantics for a probabilistic variant of *idealized algol*—i.e. an extension of $\mathsf{PCF}_{\oplus}$ with *ground state*. Recently, a model of $\mathsf{PCF}_{\oplus}$ itself, using *concurrent game semantics* has also been introduced and shown to be fully abstract [22]. In this thesis, we are actually mainly interested by yet another approach, that uses ideas coming from quantitative semantics and Linear Logic. It has been introduced by Girard in [55] for the purpose of defining probabilistic coherent spaces, designed as a generalization of *coherent spaces*, the objects he used in his model of Linear Logic. Danos and Ehrhard [34]

then used this notion to build a model of $\mathsf{PCF}_{\oplus}$, that was then shown to be fully abstract in [46]. Probabilistic coherent spaces were also used to build a fully abstract model of a probabilistic variant of Levy's Call-by-push-value language [116].

If instead of adding primitives for expressing *discrete* probabilities, we want to add primitives for *continuous probabilities* to a programming language, the problem of giving an adequate denotational semantics becomes even more complex. The first attempt in this direction was done by Kozen [70] on a first-order language with a while loop that was enriched with a random generator on real numbers. Kozen interpreted programs as stochastic Markov kernels between measurable spaces, obtaining a fully abstract denotational semantics. The category of Markov kernels and measurable spaces, however, is not Cartesian closed, which makes it impossible to use this category to build a model of an higher-order language. While there is at the present time no known fully abstract model for *higher-order* languages with continuous probabilities, a variety of models have been proposed over the past few years, and are still under investigation. Staton et al. developed [63] a denotational semantics for a continuous higher-order language that moreover handles Bayesian reasoning. Their model is based on *Quasi-Borel spaces*, that both allow to express continuous probability, and give rise to a Cartesian closed category. More recently, Ehrhard Pagani and Tasson [45] introduced a denotational model for $\mathsf{PCF}_{\text{sample}}$ based on the notion of *cones* and a *stability* condition on morphisms. The game semantics approach has been extended as well to the continuous case: Danos and Harmer model has been generalized by Ong and Vákár [89] into an adequate model of Idealized Algol enriched with continuous probabilities, while Paquet and Winskel [90] gave an adequate model for $\mathsf{PCF}_{\text{sample}}$—$\mathsf{PCF}$ enriched with continuous probabilities, presented in Chapter 1—using concurrent game semantics.

In this chapter, we are going to present the model of $\mathsf{PCF}_{\oplus}$ based on *probabilistic coherent spaces* introduced by Danos and Ehrhard [34], and the model for $\mathsf{PCF}_{\text{sample}}$ based on complete cones and stable functions introduced by Ehrhard, Pagani and Tasson [45]. In the first part of this chapter, we recall briefly some notions of category theory, that we then use to describe the underlying structure of the models. We then introduce **PCoh**, the category of probabilistic coherence spaces, and $\mathbf{Cstab_m}$, the category of measurable cones and measurable stable functions.

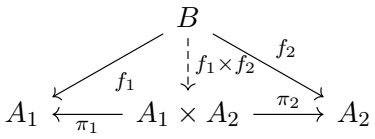
9.1 Categorical Semantics For Higher-Order Languages

In this section, we are going to present some tools from *category theory* that are designed to build denotational models for programming languages. Indeed, category theory allows us to describe which *structural properties* a concrete model should verify, in order to be a good candidate for soundness with respect to a given programming language. We assume the reader familiar with the basic definitions about category theory, for which we refer to [79]. A category is a collection of *objects* connected by arrows —called *morphisms*—that can be combined by a *composition* operator $\circ$. If $\mathbb{C}$ is a category, and A, B two object of $\mathbb{C}$, we denote by $\mathbb{C}(A, B)$ the collection of morphisms from A to B. Since we will usually consider in the following *small* categories, this collection will actually be a *set*, and accordingly we will call *hom-sets* the sets of morphisms of the form $\mathbb{C}(A, B)$. We denote by $\mathbb{C} \times \mathbb{D}$ the *product category* of two categories $\mathbb{C}$ and $\mathbb{D}$, as defined in Chapter 2 of [79] .

9.1.1 Cartesian Closed Categories

When interpreting a typed programming language in a category, types are interpreted as objects, while programs are interpreted as morphisms. It means that the category we consider should come with *functors* designed to express the type constructs. For instance, in the very simple setting of the *simply typed λ-calculus with pairs*, there are two type constructs: the binary product $\times$, and the arrow construct $\to$. The corresponding categorical structure is *Cartesian closeness*, that ensures that every objects A, B of such category have a product object $A \times B$, and an arrow object $A \Rightarrow B$. We recall here the definition of Cartesian Closed Category, since such categories are the natural place to build any model of higher-order computations.

Definition 9.1.1 (Product) *Let $\mathbb{C}$ be a category, and A_1, A_2 two objects of $\mathbb{C}$. A* product of A_1 and A_2 *is an object $(A_1 \times A_2)$ together with two morphisms $\pi_1 : A_1 \times A_2 \to A_1$, $\pi_2 : A_1 \times A_2 \to A_2$, that verifies the universal property represented below:*

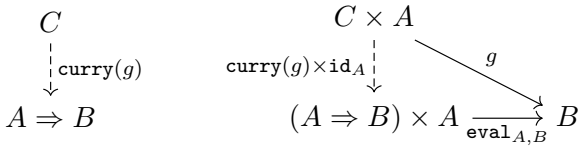


Whenever—as it is the case for Cartesian Closed categories, that we define below —binary product exists for every pair of objects, it actually defines a functor $\times : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ (see [79]). To interpret also the $\to$ type construct, we need our category to have moreover an *arrow object*. Accordingly, we ask for it to be *cartesian closed*, as defined below:

Definition 9.1.2 *A category $\mathbb{C}$ is a* Cartesian closed category (CCC) *if:*

- *it has a terminal object $\top$ —i.e. such that, for every object A of $\mathbb{C}$, there exists exactly one morphism from A to $\top$.*
- *Any two objects A, B have a product $A \times B$ in $\mathbb{C}$—in the sense of Definition 9.1.1.*
- *For any two objects A, B, there exists an object $(A \Rightarrow B)$, and a morphism $\texttt{eval}_{A,B} : (A \Rightarrow B) \times A \to B$ in $\mathbb{C}$ such that:*

$$\forall C \text{ object }, \forall g \in \mathbb{C}(C \times A, B),$$
$$\exists! \texttt{curry}(g) \in \mathbb{C}(C, A \Rightarrow B) \text{ such that } g = \texttt{eval}_{A,B} \circ (\texttt{curry}(g) \times \texttt{id}_A)$$



We say that the morphism $\texttt{curry}(g)$ is the currying *of g.*

In any Cartesian closed category, we can construct a model of simply typed λ-calculus: a typing judgments of the form $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \tau$ is interpreted as a morphism from the object $[\![\sigma_1]\!] \times \ldots \times [\![\sigma_n]\!]$ to the object $[\![\tau]\!]$. If we want to be able to interpret

not only the simply typed λ-calculus, but also more expressive higher-order languages, we need to add additional structures to some Cartesian closed category. We can for instance add coproducts, which led to models of λ-calculus with sum-types, and on the logical side to models of propositional intuitionistic logic. We could also want to interpret the untyped λ-calculus: in this case we have to ask for the presence of a *reflexive object* A, i.e. such that A is isomorphic to $A \Rightarrow A$; it comes from the fact that in an untyped setting, we cannot distinguish between first order function, second-order function, etc, as we do in the simply typed λ calculus. The first example of such Cartesian closed category with reflexive object to be looked at—the category **DCpo** of directed complete partial orders and Scott continuous functions—was built by Scott and Strachey [109, 111] in their pioneering work that gave rise to domain theory.

If our target language is PCF, we need to add to a model of simply typed λ-calculus the categorical structures needed to deal with ground types and with the fixpoint construction. Scott [110] introduced *the standard model of PCF* in the category **DCpo**, that was then shown by Plotkin [93] to be adequate, while not fully abstract. It has been then generalized into the class of *standard models of PCF* which are the cartesian closed category, with moreover a *natural number object*—the categorical structure encoding a recursive definition of natural numbers—and that is *ω-cpo-enriched*—in particular, it has for consequence that for every object A, B, the hom-set $\mathbb{C}(A, B)$ is a ω complete partial-order. A more general axiomatisation of sound models for PCF has been later formalized by Hyland and Ong [66] using the notion of *c-fix* category. A categorical axiomatisation for *adequate*—i.e. every non-terminating program is interpreted by the bottom element of hom-sets—models of PCF has also been proposed in [18].

9.1.2 Categorical Model of Linear Logic

Linear Logic (LL) has been introduced by Girard in [51]. We recall here only some basic facts about this logic; more intuitions and details may be found for instance in [54]. Linear Logic has three kind of connectives used to build formulas: the *multiplicative* connectives $\parr, \otimes, \mathbf{1}, \bot$, the *additive* ones $\&, \oplus, \top, 0$, and the *exponential* ones $!, ?$. An essential feature of LL is the presence of *two* disjunctions $\parr$ and $\oplus$, as well as *two* conjunctions $\&$ and $\otimes$. The formulas of LL can be interpreted in terms of *resources* available to some operator; for instance, we can interpret the two conjunctions as follows: $\mathsf{A} \otimes \mathsf{B}$ corresponds to the case where the operator may use simultaneously the resource A and B, while $\mathsf{A} \,\&\, \mathsf{B}$ means that the operator has to *choose* whether he wants to use the resource A or the resource B. The exponential construct $!\mathsf{A}$ means that the operator have unlimited access to the resource A.

The Classical Linear Logic (LL) formulas are generated as follows:

$$\begin{aligned}\mathsf{A} ::= p \;\mid\; p^{\bot} \;\mid\; \mathsf{A} \otimes \mathsf{A} \;\mid\; \mathsf{A} \oplus \mathsf{A} \;\mid\; \mathsf{A} \,\&\, \mathsf{A} \;\mid\; \mathsf{A} \parr \mathsf{A} \\ \mid\; \top \;\mid\; \bot \;\mid\; \mathbf{1} \;\mid\; 0 \\ \mid\; !A \;\mid\; ?A,\end{aligned}$$

where p is an element of a fixed set of *logical atoms*. Moreover, the dual operator $\bot$ is extended from logical atoms to *all formulas* using the following rules:

In its original presentation of LL, Girard defined inference rules using *one-sided* sequents, of the form $\vdash \gamma$, where γ is a list of formulas. The sequent $\vdash \mathsf{A}_1^{\bot}, \ldots, \mathsf{A}_n^{\bot}, \mathsf{B}_1, \ldots, \mathsf{B}_m$

$$(p)^\perp := p^\perp \qquad (p^\perp)^\perp := p$$
$$(\mathsf{A} \otimes \mathsf{B})^\perp := \mathsf{A}^\perp \parr \mathsf{B}^\perp \qquad (\mathsf{A} \parr \mathsf{B})^\perp := \mathsf{A}^\perp \otimes \mathsf{B}^\perp$$
$$(\mathsf{A} \oplus \mathsf{B})^\perp := \mathsf{A}^\perp \,\&\, \mathsf{B}^\perp \qquad (\mathsf{A} \,\&\, \mathsf{B})^\perp := \mathsf{A}^\perp \oplus \mathsf{B}^\perp$$
$$\mathbf{1}^\perp := \bot \quad \bot^\perp := \mathbf{1} \qquad 0^\perp := \top \quad \top^\perp := 0$$
$$(!\mathsf{A})^\perp := ?(\mathsf{A}^\perp) \qquad (?\mathsf{A})^\perp := !(\mathsf{A}^\perp)$$

Figure 9.1: Duality Operator in Linear Logic

$$\frac{\vdash \Gamma, \mathsf{A}, \mathsf{B}, \Delta}{\vdash \Gamma, \mathsf{B}, \mathsf{A}, \Delta}\ \text{exch} \qquad \frac{}{\vdash \mathsf{A}, \mathsf{A}^\perp}\ \text{init.seq.} \qquad \frac{\vdash \Gamma, \mathsf{A} \qquad \vdash \mathsf{A}^\perp, \Delta}{\vdash \Gamma, \Delta}\ \text{cut}$$

$$\frac{\vdash \Gamma, \mathsf{A} \qquad \vdash \Delta, \mathsf{B}}{\vdash \Gamma, \Delta, \mathsf{A} \otimes \mathsf{B}}\ \otimes \qquad \frac{\vdash \Gamma, \mathsf{A}, \mathsf{B}}{\vdash \Gamma, \mathsf{A} \parr \mathsf{B}}\ \parr \qquad \frac{}{\vdash \mathbf{1}}\ \mathbf{1} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\ \bot$$

$$\frac{\vdash \Gamma, \mathsf{A} \qquad \vdash \Gamma, \mathsf{B}}{\vdash \Gamma, \mathsf{A} \,\&\, \mathsf{B}}\ \& \qquad \frac{\vdash \Gamma, \mathsf{A}}{\vdash \Gamma, \mathsf{A} \oplus \mathsf{B}}\ \oplus L \qquad \frac{\vdash \Gamma, \mathsf{B}}{\vdash \Gamma, \mathsf{A} \oplus \mathsf{B}}\ \oplus R \qquad \frac{}{\vdash \Gamma, \top}\ \top$$

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?\mathsf{A}}\ \text{weakening} \qquad \frac{\vdash \Gamma, ?\mathsf{A}, ?\mathsf{A}}{\vdash \Gamma, ?\mathsf{A}}\ \text{contraction}$$

$$\frac{\vdash ?\Gamma, \mathsf{A}}{\vdash ?\Gamma, !\mathsf{A}}\ \text{promotion} \qquad \frac{\vdash \Gamma, \mathsf{A}}{\vdash \Gamma, ?\mathsf{A}}\ \text{dereliction}$$

Figure 9.2: Deduction Rules for Linear Logic

should be understood as follows: whenever all the formulas A_i are true, then at least one of the formula B_j holds. The LL inference rules are the ones presented in Figure 9.2. Observe that both the cut rule and the initial sequent rule use the duality operator defined in Figure 9.1. The resource policy is reflected in the deduction rules: for instance, the rule $\otimes$ may be read as follows: in order to know that $\vdash \Gamma^\perp, \mathsf{A} \otimes \mathsf{B}$—i.e. if all formulas in the list Γ hold, then $\mathsf{A} \otimes \mathsf{B}$ also holds—we have to split the list Γ into two disjoint parts, where the first one has the resources needed to build A, and the second one has the resources needed to build B. By contrast, the rule & may be understood as: in order to show $\vdash \Gamma^\perp, \mathsf{A} \,\&\, \mathsf{B}$, it is enough to prove that with the resources from Γ, we are able to construct separately both A and B. We can interpret the rules for exponentials in the same resource oriented framework. The weakening rule expresses the possibility of *discarding* a resource of type $!\mathsf{A}$. The contraction rule allows to transform a data of type $!\mathsf{A}$ into a pair consisting of two data of type $!\mathsf{A}$: it may be interpreted as observing that whenever we have an unlimited source of A, this situation is equivalent to having *two* unlimited sources of A available. The promotion rule should be interpreted as follows: if we are able to construct an element of type A using only resources of type $!\mathsf{B}$, then as soon as we have an unlimited access to resources of type B, we can also build as many resource of type A as we like ; it is because we can duplicate the procedure to produce a A res-source as many times as we want, and so we are actually able to build an unlimited number of data of type A. The dereliction rule should be interpreted as extracting a single resource of type A from an unlimited supply of resources of type A.

If we have a proof—i.e. a derivation tree built using the rules and axioms of Figure 9.2—for a particular sequent $\vdash \Gamma$, we are able to transform this proof into another proof of the same sequent $\vdash \Gamma$ that is *cut-free*—i.e. such that the rule Cut doesn't appear in its derivation tree; this transformation is called the *cut-elimination procedure*, and its definition may be found in [54]. There is also an intuitionistic variant of Linear Logic—ILL (Intuitionistic Linear Logic)—that was formalized in a subsequent paper by Girard and Lafont [52]. In ILL, the connective $⅋, ?, \bot$ are absent, and we add as connective the linear implication $\multimap$, that can be encoded in LL as $\mathsf{A} \multimap \mathsf{B} ::= \mathsf{A}^{\bot} ⅋ \mathsf{B}$. The rules for ILL may be found for instance in [54]. We will consider also in the following the fragments MLL—Multiplicative Linear Logic—and MILL—Multiplicative Intuitionistic Linear Logic, that consists in the restriction of respectively LL and ILL to formulas and proof trees that use only the multiplicative connectives.

The resource interpretation makes LL a natural framework for building a type system for higher-order programming languages that handle limited resources. LL, and relevant fragments of it, have been given this way a computational content in the spirit of the Curry-Howard correspondence; several linear λ-calculi have been proposed, including by Lafont [72], Abramsky [3], by Wadler [122], etc. The Surface λ-calculus we presented in Chapter 7, corresponds to ILL extended with recursive types. Recall that this language has non-terminating programs, meaning that the underlying logic has no cut-elimination procedure.

Categorical Semantics for Exponential Free Linear Logic

As illustrated before, when building a model for a typed programming language in some category, we usually interpret types as objects, and programs as morphisms. Similarly—once again in the spirit of the Curry-Howard's correspondence—a model of a logical system $\mathcal{L}$ is a category $\mathbb{C}$ equipped with an interpretation function $[\![\cdot]\!]$, that associates to every formula of $\mathcal{L}$ an *object* in $\mathbb{C}$, and to every proof in $\mathcal{L}$ a *morphism* in $\mathbb{C}$. We also ask for a correctness criterion on models: a model of a programming language is sound if the program interpretation is invariant by reduction. Similarly, we look at the equivalence on proofs defined as the reflexive and transitive closure of the cut-elimination procedure, plus some additional modularity constraints —we refer to [81] for a more precise definition of this equivalence relation—and we say that a model of LL should be sound with respect to this equivalence, i.e. that whenever two proofs are equivalent then their interpretation in the model coincide.

Definition 9.1.3 *If $\mathcal{L}$ is a logical system equipped with a cut-elimination procedure, a category $\mathbb{C}$ is a* sound model *of $\mathcal{L}$ if there exists an interpretation function $[\![\cdot]\!]$ such that whenever two proofs are equivalent with respect to the cut-elimination procedure, their interpretation by $[\![\cdot]\!]$ coincide.*

The next step consists in investigating the minimal requirements needed to guarantee that some concrete category is indeed a sound model for LL. We follow here the perspective developed by Melliès in [81]. We first look at less complex fragments of LL. First in order to have a model of MILL—multiplicative intuitionistic linear logic, i.e only the connectives $\otimes, \multimap, \mathbf{1}$ are present—, we need a *symmetric closed monoidal category (SMCC)*: it is a category equipped with a tensor product, and an object of morphisms, that allows to interpret respectively the connective $\otimes$ and $\multimap$, and with symmetry morphisms, that allows to implement the exchange rule. Below, we present the formal definition of SMCC in three steps: first we define what a *monoidal category* is, then we add symmetry requirement to it, and finally we give the definition of SMCC.

Definition 9.1.4 (Monoidal Category) *A monoidal category (MC) is a category* $\mathbb{C}$ *equipped with:*

- *A bifunctor* $\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$*, called the* tensor product*;*
- *An object* $\mathbf{1}$ *called the* unit object*;*
- *Three natural isomorphisms expressing respectively associativity of* $\otimes$*, and that* $\mathbf{1}$ *is a left and right neutral element:*

$$\begin{aligned} \mathtt{a}_{A,B,C} &: (A \otimes B) \otimes C \cong A \otimes (B \otimes C); \\ \mathtt{l}_A &: (\mathbf{1} \otimes A) \cong A; \\ \mathtt{r}_A &: (A \otimes \mathbf{1}) \cong A; \end{aligned}$$

such that the two diagrams below commute:

$$\begin{array}{ccc} (A \otimes \mathbf{1}) \otimes B & \xrightarrow{\mathtt{a}_{A,\mathbf{1},B}} & A \otimes (\mathbf{1} \otimes B) \\ & {\scriptstyle \mathtt{r}_A \otimes \mathtt{id}_B} \searrow \quad \swarrow {\scriptstyle \mathtt{id}_A \otimes \mathtt{l}_B} & \\ & A \otimes B & \end{array}$$

$$\begin{array}{ccc} ((A \otimes B) \otimes C) \otimes D & \xrightarrow{\mathtt{a}_{A\otimes B,C,D}} & (A \otimes B) \otimes (C \otimes D) \\ \downarrow {\scriptstyle \mathtt{a}_{A,B,C} \otimes \mathtt{id}_D} & & \\ (A \otimes (B \otimes C)) \otimes D & & \downarrow {\scriptstyle \mathtt{a}_{A,B,C\otimes D}} \\ \downarrow {\scriptstyle \mathtt{a}_{A,B\otimes C,D}} & & \\ A \otimes ((B \otimes C) \otimes D) & \xrightarrow{\mathtt{id}_A \otimes \mathtt{a}_{B,C,D}} & A \otimes (B \otimes (C \otimes D)) \end{array}$$

If we want moreover to be able to swap the objects A, B in the tensor product $A \otimes B$, we need to ask the monoidal category to be also *symmetric*, i.e. equipped with a natural isomorphism `swap` that performs the swapping.

Definition 9.1.5 (Symmetric Monoidal Categories) *A* symmetric monoidal category (SMC) *is a monoidal category* $(\mathbb{C}, \otimes, \mathbf{1})$ *equipped with a natural isomorphism*

$$\mathtt{swap}_{A,B} : A \otimes B \cong B \otimes A,$$

such that the three following diagrams commute:

$$\begin{array}{ccc} A \otimes \mathbf{1} & \xrightarrow{\mathtt{swap}_{A,\mathbf{1}}} & \mathbf{1} \otimes A \\ & {\scriptstyle \mathtt{r}_A} \searrow \quad \swarrow {\scriptstyle \mathtt{l}_A} & \\ & A & \end{array} \qquad \begin{array}{ccc} & B \otimes A & \\ {\scriptstyle \mathtt{swap}_{A,B}} \nearrow & & \searrow {\scriptstyle \mathtt{swap}_{B,A}} \\ A \otimes B & = & A \otimes B \\ & {\scriptstyle \mathtt{id}_{A\otimes B}} & \end{array}$$

$$\begin{array}{ccc} (A \otimes B) \otimes C & \xrightarrow{\mathtt{swap}_{A,B} \otimes \mathtt{id}_C} & (B \otimes A) \otimes C \\ \downarrow {\scriptstyle \mathtt{a}_{A,B,C}} & & \downarrow {\scriptstyle \mathtt{a}_{B,A,C}} \\ A \otimes (B \otimes C) & & B \otimes (A \otimes C) \\ \downarrow {\scriptstyle \mathtt{swap}_{A,B\otimes C}} & & \downarrow {\scriptstyle \mathtt{id}_B \otimes \mathtt{swap}_{A,C}} \\ (B \otimes C) \otimes A & \xrightarrow{\mathtt{a}_{B,C,A}} & B \otimes (C \otimes A) \end{array}$$

Every cartesian category—i.e. equipped with a terminal object and finite products—is also a symmetric monoidal category: indeed, we can define natural isomorphisms for associativity and elimination of the neutral element as $\mathtt{a}$, $\mathtt{l}$, $\mathtt{r}$ by combining the projections π_1, π_2.

To model the linear arrow of ILL, we need to add a closed structure to a SMC. The approach is similar to the way we add a closed structure to a cartesian category in Definition 9.1.2, thus obtaining a cartesian closed category.

Definition 9.1.6 (Symetric Monoidal Closed Category) *A* symmetric monoidal closed category (SMCC) $\mathbb{C}$ *is a SMC equipped, for every object* A, B *with an object* $(A \multimap B)$ *and a morphism* $\mathtt{eval}_{A,B} : (A \multimap B) \otimes A \to B$ *such that:*

$$\forall C \text{ object }, \forall g \in \mathbb{C}(C \otimes A, B),$$
$$\exists! \mathtt{curry}(g) \in \mathbb{C}(C, A \multimap B) \text{ such that } g = \mathtt{eval}_{A,B} \circ (\mathtt{curry}(g) \otimes \mathtt{id}_A).$$

We want now to add to a categorical model of MILL—i.e. a SMCC—the categorical structure required in order to obtain a categorical model for MLL. Observe that the LL multiplicative connective $\parr$ not present in ILL may be expressed by using the duality operator and the multiplicative connectives from ILL. It means that as soon as we have an appropriate duality mechanism in our category, we can reconstruct a denotation for multiplicative LL from a denotation for multiplicative ILL. We can use this observation to define what is a categorical model of *multiplicative LL*: it is a $\star$-autonomous category, i.e. a SMCC with a dualizing object. The notion of $\star$-autonomous category was introduced by Barr [11].

Definition 9.1.7 ($\star$-autonomous category) *A* $\star$*-autonomous category is a symmetric monoidal closed category* $(\mathbb{C}, \otimes, \mathbf{1}, \multimap)$ *endowed with a* dual object $\perp$ *such that the morphism*

$$\mathtt{d}_A : A \to (A \multimap \perp) \multimap \perp$$

is an isomorphism —where $\mathtt{d}_A$ *is defined as the transpose of the evaluation map* $\mathtt{eval}_{A,\perp} : (A \multimap \perp) \otimes A \to \perp$.

The definition of a $\star$-autonomous category may also be understood in a logical way: indeed asking for $\mathtt{d}$ to be an *isomorphism* correspond to adding the law of *excluded middle* to a model of ILL.

When we want to add the additive connectives $\&, \oplus$, we need to add Cartesian product to our model: accordingly, a model of exponential-free LL is a $\star$-autonomous category with finite products.

Categorical Models for Linear Logic

Difficulties arise, however, when we want to add to a $\star$-autonomous category a structure interpreting the exponential connective !. Our account here follows again [81], to which we refer the reader for a more extensive presentation. A slightly different viewpoint may be also found in [37]. We can see first that ! should be a comonad, i.e. a functor $! : \mathbb{C} \to \mathbb{C}$ with two natural transformation $\eta : ! \to Id_{\mathbb{C}}$ and $\mu : ! \to !^2$ that moreover verify additional commutative diagrams (that the reader may find in [79]). From this co-monadic structure, we obtain morphisms for the *dereliction* and *digging* : for every object $A \in \mathbb{L}$, we have morphisms $\mathrm{der}_A : !A \to A$, and $\mathrm{digg}_A : !A \to !!A$. However, a co-monad ! does *not* necessarily give rise to a sound model of LL. From there, several distinct propositions to

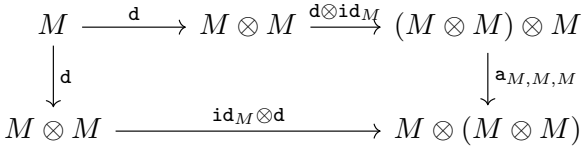
strengthen the interaction of ! with the structure of the $\star$-autonomous category had been put forward, and it's only recently that they were discovered to be all particular instances of *linear-non-linear* models, a categorical axiomatisation introduced by Benton [12].

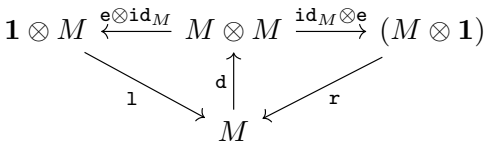
We present below two of the earlier axiomatisations of sound LL models, in the sense of Definition 9.1.3 : the new-Seely Model and the Lafont Model. The notion of Lafont's model was introduced by Lafont in [73], while the new-Seely model is due to Seely [112] and Bierman [15].

Lafont Models of Linear Logic

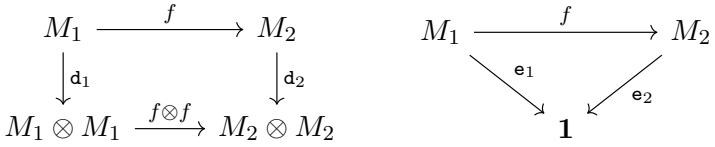
Lafont's model is based on the notion of *commutative comonoid*, that we recall below.

Definition 9.1.8 *Let $(\mathbb{C}, \otimes, \mathbf{1})$ be a monoidal category. Then a* co-monoid *consists of a triple $(M, \mathtt{d}, \mathtt{e})$, where M is an object of $\mathbb{C}$, and $\mathtt{d} : M \to M \otimes M$ —called* co-multiplication*—and $\mathtt{e} : M \to \mathbf{1}$ —called the* co-unit*—are two morphisms in $\mathbb{C}$ such that the two diagrams below hold:*

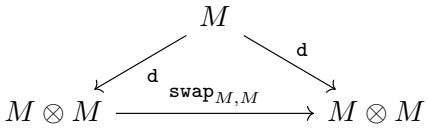




If $(M_1, \mathtt{d}_1, \mathtt{e}_1)$ and $(M_2, \mathtt{d}_2, \mathtt{e}_2)$ are two comonoids, a morphism of comonoid from M_1 to M_2*, denoted $f : (M_1, \mathtt{d}_1, \mathtt{e}_1) \to (M_2, \mathtt{d}_2, \mathtt{e}_2)$, is a morphism $f \in \mathbb{C}(M_1, M_2)$ that respects the comonoid structure, i.e. such that the two diagrams below commute:*



If moreover $(\mathbb{C}, \otimes, \mathbf{1})$ is a commutative monoidal category, a comonoid $(M, \mathtt{d}, \mathtt{e})$ is commutative *if the following diagram holds:*



The next step to present Lafont construction is the notion of *free* commutative comonoid generated by some object A.

Definition 9.1.9 *Let* $(\mathbb{C}, \otimes, \mathbf{1})$ *a commutative monoidal category, and* A *an object of* $\mathbb{C}$. *A comonoid* $(M, \mathtt{d}, \mathtt{e})$ *is* the free commutative comonoid over A *if there exists a morphism* $\epsilon_A : M \to A$, *such that moreover the following universal property holds:*

$$\begin{aligned} &\forall (M', \mathtt{d}', \mathtt{e}')\ \textit{commutative comonoid}\ \forall f : M' \to A \\ &\qquad \exists! f^\dagger \textit{comonoid morphism} (M', \mathtt{d}', \mathtt{e}') \to (M, \mathtt{d}, \mathtt{e}) \\ &\qquad \textit{such that the diagram below commutes in}\ \mathbb{C} : \end{aligned}$$

$$\begin{array}{ccc} M & \xrightarrow{\epsilon_A} & A \\ {\scriptstyle f^\dagger}\uparrow & \nearrow_{f} & \\ M' & & \end{array}$$

The universal property of Definition 9.1.9 implies that when it exists, the free comutative comonoid over A is unique up to (unique) isomorphism. Another formulation of this universal property is that the category of commutative comonoids over A has M as terminal object. By definition, the structure of an exponential modality turns an object A into a commutative comonoid $!A$, with the comultiplication given by contraction and the neutral element given by weakening:

$$1 \xleftarrow{\mathtt{weak}_{!A}} !A \xrightarrow{\mathtt{contr}_{!A}} !A \otimes !A.$$

The converse does not hold in general, since commutative comonoids may lack a comonad structure (dereliction, digging, and the action on morphisms). However, Lafont proves that in case the commutative comonoid is the *free* one then its universal property allows one to canonically construct the missing structure, in such a way that we obtain a sound model of LL.

Definition 9.1.10 (Lafont Model) *A* $\star$*-autonomous category* $\mathbb{C}$ *is a* Lafont model *if:*

1. *it has finite products and,*
2. *for every object* A, *there exists a commutative comonoid* $(!_f A, \mathtt{weak}_{!_f A}, \mathtt{contr}_{!_f A})$ *which is the free commutative comonoid generated by* A.

For every A, we denote by $\mathrm{der}_{\mathrm{f}A}$ the morphism in $\mathbb{C}(!_f A, A)$ obtained by the universal property stated in Definition 9.1.9. It is this morphism that implements the dereliction rule of LL.

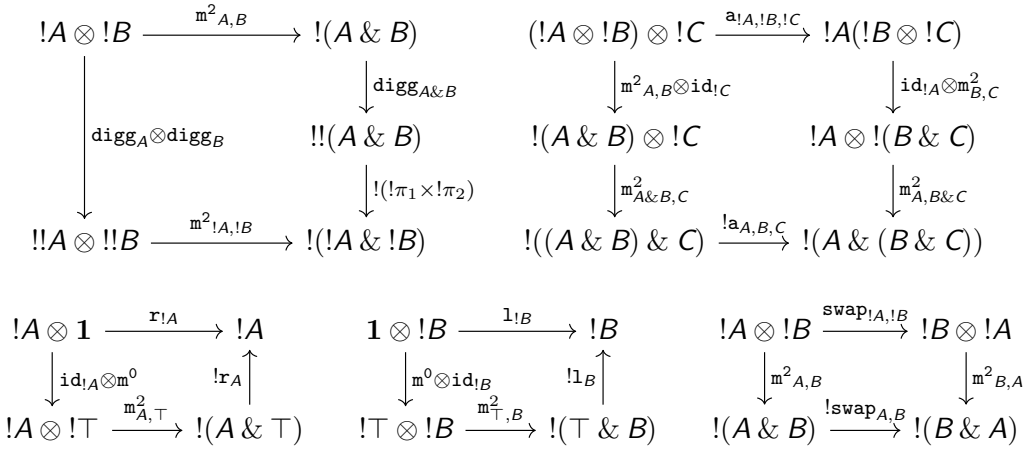
New-Seely Models of Linear Logic

Another axiomatisation was proposed by Seely—and then corrected by Bierman. It may be noted that until now, every known concrete sound models of LL is a new-Seely category.

Definition 9.1.11 (New-Seely Category) *A* new-Seely *category consists of:*

- *A* $\star$*-autonomous category* $\mathbb{C}$ *with finite products*
- *A comonad* $(!, \mathtt{der}, \mathtt{digg})$ *to model the modality*
- *Two natural isomorphisms* $\mathtt{m}^0 : \mathbf{1} \cong !\top$ *and* $\mathtt{m}^2 : !A \otimes !B \cong !(A \,\&\, B)$.

such that the diagrams below commute:



The definition of new-Seely category may also be expressed in a more elegant way by using the notion of *monoidal adjunction*; this viewpoint is developed in [81].

Every model of Linear Logic may be seen as a model of Intuitionistic Logic, i.e. we can construct a Cartesian closed category from any sound model of Linear Logic. The Seely axiomatisation of LL models had the explicit purpose of guaranteeing that the co-Kleisli category associated to ! should be cartesian closed, thus giving a model of Intuitionistic Logic, or equivalently of simply typed λ-calculus. We recall below the co-Kleisli construction.

Definition 9.1.12 (Co-Kleisli Category) *Let $\mathbb{C}$ be a category, and (T, η, μ) a co-monad on $\mathbb{C}$. The co-Kleisli category $\mathbb{C}_T$ associated to T is defined as:*

- *the objects of $\mathbb{C}_T$ are the same as those of $\mathbb{C}$;*
- *for every A, B objects of $\mathbb{C}_T$, $\mathbb{C}_T(A, B) = \mathbb{C}(TA, B)$.*

When the category $\mathbb{C}$ is a new-Seely category, the co-Kleisli category $\mathbb{C}_!$ becomes Cartesian closed, and consequently a model of simply typed λ-calculus.

Proposition 9.1.1 *If $\mathbb{C}$ is a new-Seely category, then the category $\mathbb{C}_!$ is Cartesian closed.*

Example 9.1.1 (Coh) *The first model of LL was introduced by Girard using the notion of coherence space [51]. A coherence space $A = (|A|, \frown_A)$ is a pair of a set $|A|$, the* web*, and a symmetric reflexive relation $\frown_A$, the* coherence*.*

The exponential modality of the original model is given by the finite cliques functor, $|!A| = \{x \subseteq_f |A| \mid \forall a, a' \in x, a \frown_A a'\}$, leading to a new-Seely model. By contrast, the free exponential modality $|!_f A| = \{\mu \in \mathcal{M}_f(|A|) \mid \forall a, a' \in \mathsf{S}(\mu), a \frown_A a'\}$ is given by the finite multi-*cliques functor, as shown in [120]. The morphism $(\mathtt{der}_{!A})^\dagger$ factoring $!A$ through $!_f A$ is given by the support relation: $(\mathtt{der}_{!A})^\dagger = \{(\mathsf{S}(\mu), \mu)\ ;\ \mu \in |!_f A|\}$.*

On the non-uniqueness of exponential structure.

The additive and multiplicative connectives of LL are entirely determined by the logical rules that we have presented in Figure 9.2: indeed, if we add another tensor product $\otimes'$ to the logical system, then for every formulas A, B, it holds that $A \otimes B$ and $A \otimes' B$ are isomorphic. This phenomenon is reflected in the categorical semantics of LL: given a category, the multiplicative-additive structures (if exist) are unequivocally characterized by

universal properties as soon as a notion of multilinear map is given. It is notably different for the exponential: if we add another exponential connective $!'$ into the linear system, we are not able, using the linear logic rules, to show that $!$ and $!'$ are equivalent. This has also consequences on the semantics side: it means that several unrelated exponential notions can coexist in a model of linear logic. An example of such situation may be found when looking at the category **Coh**, the model of Linear Logic introduced by Girard, that we have already considered in Example 9.1.1. As mentioned in Example 9.1.1, we can define two *distinct* exponentials in the **Coh** model: the *finite cliques functor* and the *finite multi-cliques functor*. Such a situation also arises in the category **Rel** of sets and relations, when the standard exponential is given by the multiset functor, but where a wide family of exponential modalities has been introduced using infinite multiplicities [21].

The choice of an exponential structure decides how the model is going to implement the *erasure* and *copy* operations when interpreting a programming language. It may have deep consequences on the structure of programs interpretations: for instance choosing the finite cliques functor lead to a *finite* model of simply typed λ-calculus with finite ground types—e.g. booleans. Such a choice of exponential structure may also depend on which computational properties we want the denotational semantics to capture: for instance, in **Rel** the exponential from [21] leads to a *non-sensible* —meaning that non-terminating terms do not have necessarily the same denotational semantics—model of untyped λ-calculus, while every model of this language based on the standard exponential is sensible. A well-chosen exponential modality may also allow to model a more expressive language in the category: for instance, the **Rel** modalities introduced in [21] were used in [20] to give a denotational semantics to a λ-calculus with co-effects.

In this setting, Lafont's models—presented in Definition 9.1.10—offer a *canonical* criterion to choose an exponential modality. Indeed, while for every exponential modality, $!A$ is a commutative comonoid generated by A with respect to $\otimes$, there is at most one *free* commutative comonoid.

Moreover, Lafont's way of defining the exponential structure expresses a direct connection with the tensor product of the underlying symmetric monoidal category. The universal property associated to the free exponential allows to express every exponential modality using the free one, thus giving some *genericity* to the Lafont construction. More precisely, if $!$ is any exponential modality in the same symmetric monoidal category, the fact that $\mathrm{der}^{\dagger}$ is a morphism of comonoids means that we have the diagrams presented in Figure 9.3:

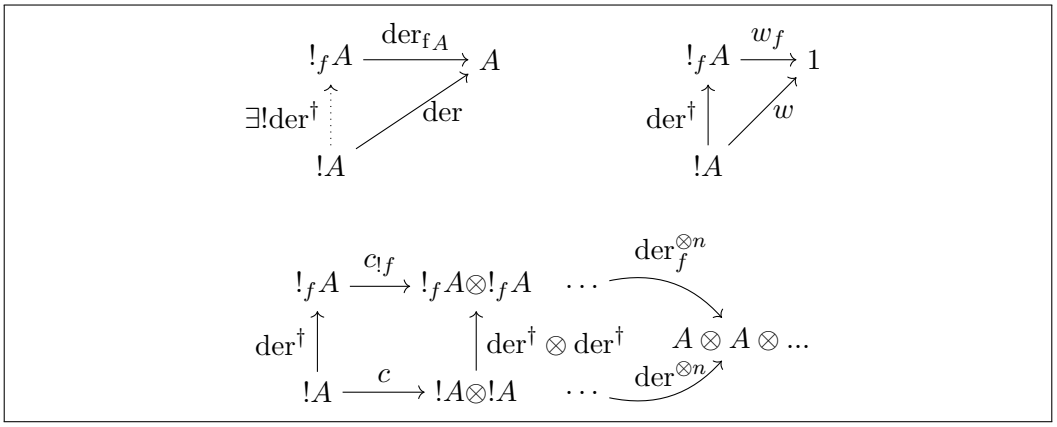


Figure 9.3: Expressing any exponential modality ! using the free exponential modality $!_f$.

9.2 PCoh: The Category of Probabilistic Coherent Spaces

We present now the category **PCoh** of probabilistic coherence spaces, and its structure as a model of Linear Logic, as investigated by Danos and Ehrhard in [34]. Linear Logic formulas are interpreted by *probabilistic coherence spaces (PCS)*. A PCS consists of both a *web*, and a set of *quantitative cliques* that are vectors over this web, and that moreover must verify some conditions designed to guarantee that the set of quantitative cliques is well-formed.

We use the following notations: $\mathbb{R}_+$ is the set of non-negative real numbers. When talking about vectors over a countable set $|\mathcal{A}|$, e_a denotes the *base vector* for $a \in |\mathcal{A}|$: $(e_a)_b := \delta_{a,b}$, with δ the Kronecker delta; and for any vector $v \in \mathbb{R}^{|\mathcal{A}|}$, we denote v_a its a component. If X is a set of vectors over $|\mathcal{A}|$, we call *dual of* X the set $X^\perp = \{y \mid \forall x \in X, \langle x, y\rangle \leq 1\}$, where the scalar product for $x, y \in \mathbb{R}_+^{|\mathcal{A}|}$ is taken as $\langle x, y\rangle = \sum_{a \in |\mathcal{A}|} x_a y_a \in \mathbb{R}_+ \cup \{\infty\}$.

Definition 9.2.1 ([55, 34]) *A* probabilistic coherence space *is a pair* $\mathcal{A} = (|\mathcal{A}|, \mathrm{P}(\mathcal{A}))$ *where* $|\mathcal{A}|$ *is a countable set called the* web *of* $\mathcal{A}$ *and* $\mathrm{P}(\mathcal{A})$ *is a subset of* $(\mathbb{R}_+)^{|\mathcal{A}|}$ *called the* cliques *of* $\mathcal{A}$*, such that the following conditions hold:*

closeness: $\mathrm{P}(\mathcal{A})^{\perp\perp} = \mathrm{P}(\mathcal{A})$,

boundedness: $\forall a \in |\mathcal{A}|$, $\exists \mu > 0$, $\forall v \in \mathrm{P}(\mathcal{A})$, $v_a \leq \mu$,

completeness: $\forall a \in |\mathcal{A}|$, $\exists \lambda > 0$, $\lambda e_a \in \mathrm{P}(\mathcal{A})$,

where the dual of a PCS $\mathcal{A}$ *is defined by* $\mathcal{A}^\perp := (|\mathcal{A}|, \mathrm{P}(\mathcal{A})^\perp)$.

A pair $(|\mathcal{A}|, \mathrm{P}(\mathcal{A}))$ *that verifies the boundedness and completeness conditions—but that may not be closed—is called a* pre-PCS.

The boundedness condition means that for every a in the web, we can look at the supremum of the valuation of any clique on this particular element of the web. Since we will need later to talk about those least upper bound, we introduce the following notation now.

Notation 9.2.1 *For any PCS* $\mathcal{A}$*, we will call* maximum coefficient of a in $\mathcal{A}$ *the positive real* $(\sup \mathcal{A})_a := \sup_{x \in \mathrm{P}(\mathcal{A})} x_a$.

We may observe that the family of maximum coefficients for all a *do not* completely characterize the PCS: it can be seen for instance by looking at the PCSs represented respectively in Example 9.2.1 and Example 9.2.6.

PCSs are a probabilistic generalization of coherent spaces: instead of coherence spaces cliques—sets of vertices contained in the web—we are now interested into *quantitative* cliques: a PCS clique do not associate a boolean value at each vertex anymore, but a weight that is an element in $\mathbb{R}_+$. The bi-duality condition enforced in Definition 9.2.1 is also similar to the one used by Girard on coherence spaces, in order to distinguish the family of set vertices that are indeed the cliques of some underlying graph structure.

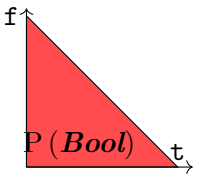
The duality operator gives us a standard way to construct PCS from some generating elements: indeed if we take any pre-PCS X, it holds that $X^{\perp\perp}$ is a PCS. We also endows the set of cliques $\mathrm{P}(\mathcal{A})$ with a *partial order*: we take the pointwise order on vectors, i.e. $u \leq v$ when for all $a \in |\mathcal{A}|$, $u_a \leq v_a$. With this partial order, we can express in a more geometrical way the requirements enforced in Definition 9.2.1. The bi-duality operation can indeed be interpreted as taking the closure of the space of cliques under Scott-closure and convexity. It is expressed by the following proposition, proved in [55] by a standard application of the Hahn-Banach Theorem.

Proposition 9.2.2 ([55]) *Let I be a countable set and $S \subseteq \mathbb{R}_+[I]$. The pair (I, S) is a probabilistic coherence space iff:*

1. *S is bounded and complete (see Def. 9.2.1);*
2. *S is* Scott closed*, i.e. $\forall u \leq v \in S$, $u \in S$, and $\forall D \subseteq S$ directed, it holds that $\sup D \in S$;*
3. *S is* convex*, i.e. $\forall u, v \in S$, $\lambda \in [0,1]$, $\lambda u + (1-\lambda)v \in S$.*

The convexity inherent to PCSs makes **PCoh** particularly suitable for modeling (discrete) probabilistic computations: the barycentric sum $\kappa v + \rho w$ (for $\kappa, \rho \in [0,1]$, $\kappa + \rho \leq 1$, v, w vectors) expresses a computation which returns v with probability κ, w with probability ρ, and the remainder $1 - (\kappa + \rho)$ is the probability that the computation diverges. We illustrate this idea by looking at the PCSs modeling data, i.e. corresponding to ground types.

Example 9.2.1 (The PCS Bool.)



To model the boolean type, we take as web the pure*—i.e. non-probabilistic—data of this type, i.e. $|\mathbf{Bool}| = \{\mathtt{t}, \mathtt{f}\}$. The cliques are then generated as $\mathrm{P}(\mathbf{Bool}) = \{(1,0),(0,1)\}^{\perp\perp}$, meaning that we take the convexity closure and Scott closure of the program that returns **true** with probability 1, and the one that returns **false** with probability 1. What we obtain is all the sub-probability distributions over booleans, i.e $\mathrm{P}(\mathbf{Bool}) = \{(p,q)\ ;\ p+q \leq 1\}$.*

The way we defined the PCS **Bool** may be extended to other kinds of ground type data. We give below the PCS modeling the type N of natural numbers in $\mathsf{PCF}_\oplus$: the cliques are all sub-distributions over natural numbers.

Example 9.2.2 (PCS of Natural Numbers) *We define the PCS $\mathrm{N}^{\mathbf{PCoh}}$ by taking $|\mathrm{N}|^{\mathbf{PCoh}} = \mathbb{N}$, and $\mathrm{P}\left(\mathrm{N}^{\mathbf{PCoh}}\right) = \{u \in \mathbb{R}_+^{\mathbb{N}} \mid \sum_{n\in\mathbb{N}} u_n \leq 1\}$.*

We look now at the morphisms of the category **PCoh**: the morphisms from a PCS $\mathcal{A}$ to a PCS $\mathcal{B}$ are the Scott-continuous and linear functions mapping the vector set $\mathrm{P}(\mathcal{A})$ into the vector set $\mathrm{P}(\mathcal{B})$. These linear maps are described as (usually infinite-dimensional) matrices. The connective $\multimap$ is interpreted in such a way that we have immediately the correspondence with **PCoh**-morphisms: the cliques over $\mathcal{A} \multimap \mathcal{B}$ are exactly the morphism from $\mathcal{A}$ to $\mathcal{B}$.

Definition 9.2.2 (PCS morphisms) *Let $\mathcal{A}$, $\mathcal{B}$ two PCSs.*

- *A PCS morphism from $\mathcal{A}$ to $\mathcal{B}$ is a matrix $f \in (\mathbb{R}_+)^{|\mathcal{A}|\times|\mathcal{B}|}$ s.t.:*

$$\forall v \in \mathrm{P}(\mathcal{A}), f\,v \in \mathrm{P}(\mathcal{B}), \tag{9.1}$$

  where $f\,v$ is the usual matricial product: $\forall b \in |\mathcal{B}|$, $(f\,v)_b := \sum_{a\in|\mathcal{A}|} f_{a,b}v_a$.

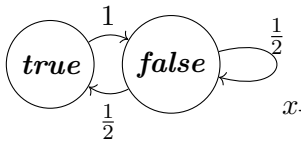
- *The object of linear morphisms $\mathcal{A} \multimap \mathcal{B}$ is defined as $|\mathcal{A} \multimap \mathcal{B}| := |\mathcal{A}| \times |\mathcal{B}|$, and $\mathrm{P}(\mathcal{A} \multimap \mathcal{B}) := \mathbf{PCoh}(\mathcal{A}, \mathcal{B})$.*

We will often use Lemma 9.2.3 below, allowing us to infer condition (9.1) for all $v \in \mathrm{P}(\mathcal{A})$, just by testing it on a set G of generators of $\mathrm{P}(\mathcal{A})$.

Lemma 9.2.3 *Let $\mathcal{A}, \mathcal{B}$ be two probabilistic coherence spaces and f be a matrix in $\mathbb{R}_+^{|\mathcal{A}|\times|\mathcal{B}|}$. Let $G \subseteq \mathrm{P}(\mathcal{A})$ such that: (i) $G^{\perp\perp} = \mathrm{P}(\mathcal{A})$, and (ii) $\forall v \in G$, $f\,v \in \mathbb{R}_+^{|\mathcal{B}|}$,then: $f(G)^{\perp\perp} = f(\mathrm{P}(\mathcal{A}))^{\perp\perp}$.*

We illustrate Definition 9.2.3 by looking at the morphisms from **Bool** to itself. Using Lemma 9.2.3, we see that it is those matrices that send both $e_{\mathtt{t}}$—the pure state **true**—and $e_{\mathtt{f}}$—the pure state **false**—into the cliques space of **Bool**. We see that these morphisms correspond exactly to Markov chains on two states.

Example 9.2.3 (Morphisms in Bool $\multimap$ Bool.) *The element of $\mathrm{P}(\boldsymbol{Bool} \multimap \boldsymbol{Bool})$ are the matrices $x \in \mathbb{R}_+^{\{\mathtt{t},\mathtt{f}\}\times\{\mathtt{t},\mathtt{f}\}}$ such that $x_{\mathtt{t},\mathtt{t}} + x_{\mathtt{t},\mathtt{f}} \leq 1$, and also $x_{\mathtt{f},\mathtt{t}} + x_{\mathtt{f},\mathtt{f}} \leq 1$. Those matrices are the one that represent Markov Chains on two states, as illustrated on the figure below.*



$x \in \mathrm{P}(\boldsymbol{Bool} \multimap \boldsymbol{Bool})$ *defined as:*

$x_{\mathtt{t},\mathtt{t}} = 0 \quad x_{\mathtt{t},\mathtt{f}} = 1 \quad x_{\mathtt{f},\mathtt{t}} = \frac{1}{2} \quad x_{\mathtt{f},\mathtt{f}} = \frac{1}{2}$.

We define now the category **PCoh**: we need to specify its objects, its hom-sets, and how the composition and identity for morphisms are defined.

Definition 9.2.3 *The category* **PCoh** *has as* objects *PCSs, and PCSs morphisms as morphisms.*

- *The* identity $\mathrm{id}^{\mathcal{A}}$ *on $\mathcal{A}$ is defined as the diagonal matrix given by $(\mathrm{id}^{\mathcal{A}})_{a,a'} = \delta_{a,a'}$.*
- Composition *of morphisms is matrix multiplication: $(g \circ f)_{a,c} = \sum_{b\in|\mathcal{B}|} f_{a,b}g_{b,c}$, where $f \in \mathbf{PCoh}(\mathcal{A}, \mathcal{B})$, $g \in \mathbf{PCoh}(\mathcal{B}, \mathcal{C})$, and $a \in |\mathcal{A}|$, $c \in |\mathcal{C}|$.*

The above sum $\sum_{b\in|\mathcal{B}|} f_{a,b}g_{b,c}$ converges in $\mathbb{R}_+$, because f, g verify condition (9.1).

9.2.1 PCoh as Model of Exponential-Free Linear Logic

We are now going to present the structure of **PCoh** as a model of linear logic, that have been built by Danos and Ehrhard in [34]. First, we give the $\star$-autonomous structure of **PCoh**, and then we define the comonad that have been introduced in [34] in order to make **PCoh** a new-Seely category. Our presentation here follows the same structure as the one done in [28].

We first look at the monoidal structure of **PCoh**: we present here the bifunctor $\otimes : \mathbf{PCoh} \times \mathbf{PCoh} \to \mathbf{PCoh}$ that interprets the tensor product.

Definition 9.2.4 (The bifunctor $\otimes$.) *We define a bifunctor $\otimes : \mathbf{PCoh} \times \mathbf{PCoh} \to \mathbf{PCoh}$ as follows:*

- *The action of $\otimes$ on objects is specified by: $|\mathcal{A} \otimes \mathcal{B}| := |\mathcal{A}| \times |\mathcal{B}|$, and $\mathrm{P}(\mathcal{A} \otimes \mathcal{B}) := \{v \otimes w \mid v \in \mathrm{P}(\mathcal{A}), w \in \mathrm{P}(\mathcal{B})\}^{\perp\perp}$, where $(x \otimes y)_{(a,b)} = x_a y_b$, for $a \in |\mathcal{A}|$ and $b \in |\mathcal{B}|$.*
- *The action of $\otimes$ on morphisms $u \in \mathbf{PCoh}(\mathcal{A}, \mathcal{B})$ and $v \in \mathbf{PCoh}(\mathcal{A}', \mathcal{B}')$ is defined as $(u \otimes v)_{(a,a'),(b,b')} := u_{a,b} v_{a',b'}$, for $(a, a') \in |\mathcal{A} \otimes \mathcal{A}'|$, $(b, b') \in |\mathcal{B} \otimes \mathcal{B}'|$.*
- *The symmetry $\mathtt{swap} \in \mathbf{PCoh}(\mathcal{A} \otimes \mathcal{B}, \mathcal{B} \otimes \mathcal{A})$ is given by $\mathtt{swap}_{(a,b),(b',a')} := \delta_{a,a'}\delta_{b,b'}$. The other natural isomorphisms (associativity, neutrality) are given similarly.*

Example 9.2.4 (The PCS Bool $\otimes$ Bool.) *Looking at the definition of $\otimes$, we see that:*

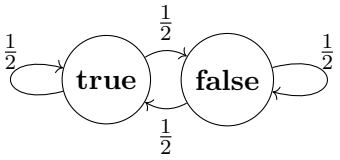
$$|\boldsymbol{Bool} \otimes \boldsymbol{Bool}| = \{(\mathtt{t},\mathtt{t}), (\mathtt{t},\mathtt{f}), (\mathtt{f},\mathtt{t}), (\mathtt{f},\mathtt{f})\}$$
$$\mathrm{P}(\boldsymbol{Bool} \otimes \boldsymbol{Bool}) = \{x \in \mathbb{R}_+^{|\boldsymbol{Bool} \otimes \boldsymbol{Bool}|} \mid x_{\mathtt{t},\mathtt{t}} + x_{\mathtt{t},\mathtt{f}} + x_{\mathtt{f},\mathtt{t}} + x_{\mathtt{f},\mathtt{f}} \leq 1\}.$$

Observe that the cliques of $\boldsymbol{Bool} \otimes \boldsymbol{Bool}$ correspond to all sub-distributions over pair of booleans—it is because the tensor product distributes on the coproduct due to the monoidal closure. Consider the program $M = \langle N, N\rangle$, where $N = \boldsymbol{true} \oplus \boldsymbol{false}$. We see that we can take as denotation: $[\![M]\!]_{(b_1,b_2)} = \frac{1}{4}$ for every $b_1, b_2 \in \{\mathtt{t},\mathtt{f}\}$. It corresponds to the fact that the probability of obtaining (b_1, b_2) after evaluating M is equal to the probability of obtaining first b_1 and then b_2 when we evaluate twice N.

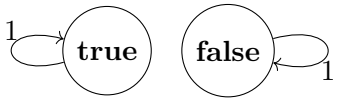
Example 9.2.5 (Actions of $\otimes$ on morphisms from Bool to Bool.) *We illustrate here the action of $\otimes$ on morphisms, by considering $u, v \in \mathbf{PCoh}(\boldsymbol{Bool}, \boldsymbol{Bool})$ corresponding to the Markov chains in Figure 9.4a and 9.4b respectively. The resulting morphism $u \otimes v$ may be interpreted as follows: it is the Markov chain that has for states the pair of booleans, and whose transition function is defined as follows: from a state (b_1, b_2), it applies independently the transition function of the Markov chain encoded by u to the boolean on the left, and the one encoded by v to the boolean on the right. This morphism $u \otimes v$ is represented in Figure 9.4c.*

The unit of $\otimes$ is given by the singleton web PCS $\mathbf{1} := (\{\star\}, [0,1]^{\{\star\}})$, which is also equal to $\mathcal{A}^{\otimes 0}$ for any $\mathcal{A}$. Observe that, accordingly to the rules of Linear Logic, we can express the object of linear morphisms in **PCoh** using only the bifunctor $\otimes$ and the dual construction on PCSs: indeed, it holds that $\mathcal{A} \multimap \mathcal{B} = (\mathcal{A}^\perp \otimes \mathcal{B})^\perp$. With the bifunctor $\otimes$, and the linear implication $\multimap$, we have in effect equipped the category **PCoh** with a symmetric monoidal closed structure, i.e. a model of multiplicative ILL.

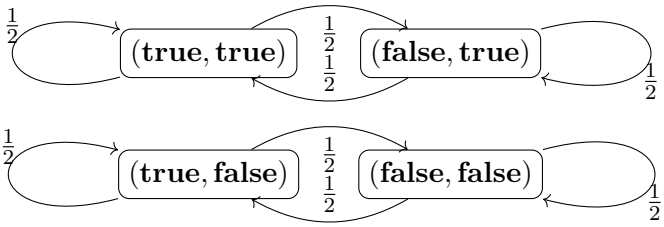
Proposition 9.2.4 (from [34]) *The category* **PCoh** *is symmetric monoidal closed.*



(a) $u \in \mathbf{PCoh}(\mathbf{Bool}, \mathbf{Bool})$



(b) $v \in \mathbf{PCoh}(\mathbf{Bool}, \mathbf{Bool})$



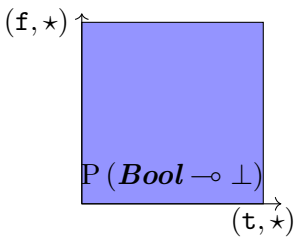
(c) $u \otimes v \in \mathbf{PCoh}(\mathbf{Bool} \otimes \mathbf{Bool}, \mathbf{Bool} \otimes \mathbf{Bool})$

Figure 9.4: Example of tensor product of morphisms in **PCoh**.

To obtain a model of multiplicative LL —i.e Linear Logic with only the multiplicative connective $\otimes, \parr, \mathbf{1}$—we need to define now a dualizing object $\bot$ in the category **PCoh**, that makes it $\star$-autonomous: we take $\bot := \mathbf{1}^\bot$. Moreover, we can see that $\mathbf{1}$ is preserved by dualization, i.e. $\bot = \mathbf{1}$.

Proposition 9.2.5 (from [34]) *The category* **PCoh** *is $\star$-autonomous.*

Example 9.2.6



We illustrate here the use of the dualizing object by considering the PCS $(\boldsymbol{Bool} \multimap \bot)$. *Looking at the definition of the linear implication $\multimap$, we see that* $|\boldsymbol{Bool} \multimap \bot| = \{(\mathtt{t}, \star), (\mathtt{f}, \star)\}$ *and* $\mathrm{P}(\boldsymbol{Bool} \multimap \bot) = \{x \in \mathbb{R}_+^{|\boldsymbol{Bool} \multimap \bot|} \mid x_{(\mathtt{t},\star)} \leq 1 \wedge x_{\mathtt{f},\star} \leq 1\}$. *Observe that this PCS is isomorphic to* $\boldsymbol{Bool}^\bot$.

Recall that to have a model of exponential-free LL, we need to build *finite products* on top of the $\star$-autonomous structure of **PCoh**. It will allow to interpret the additive connective &, and by duality also the additive connective $\oplus$.

Proposition 9.2.6 (from [34]) PCoh *admits Cartesian products of any countable family $(\mathcal{A}_i)_{i \in I}$ of PCSs, defined as*

$$|\&_{i \in I} \mathcal{A}_i| := \biguplus_{i \in I} |\mathcal{A}_i| = \bigcup_{i \in I} (\{i\} \times |\mathcal{A}_i|),$$
$$\mathrm{P}(\&_{i \in I} \mathcal{A}_i) := \left\{ v \in \mathbb{R}_+^{|\&_{i \in I} \mathcal{A}_i|}; \forall i \in I, \pi_i\, v \in \mathrm{P}(\mathcal{A}_i) \right\}$$

where $\pi_i\, v$ is the vector in $(\mathbb{R}_+)^{|\mathcal{A}_i|}$ denoting the i-th component of v, i.e. $\pi_i\, v_a := v_{(i,a)}$.

Notice that the empty product yields the terminal object $\top$ defined as $|\top| = \emptyset$ and $\mathrm{P}(\top) = \{\mathbf{0}\}$. We may write the binary product by $\mathcal{A}_1 \,\&\, \mathcal{A}_2$. If $u \in \mathrm{P}(\mathcal{A})$ and $v \in \mathrm{P}(\mathcal{B})$, we note $\langle u, v \rangle$ the *cartesian product of the vector u and v* defined as: for $a \in \mathrm{P}(\mathcal{A})$,

$\langle u, v\rangle_{(1,a)} = u_a$, and for $b \in \mathrm{P}(\mathcal{B})$, $\langle u, v\rangle_{(2,b)} = v_b$. When $\mathrm{P}(\mathcal{A})$ and $\mathrm{P}(\mathcal{B})$ are disjoint, we will also omit the indexes. Observe that any $v \in \mathrm{P}(\mathcal{A}_1 \,\&\, \mathcal{A}_2)$ is equal to the pair $(\langle \pi_1 v, \pi_2 v\rangle) \in \mathrm{P}(\mathcal{A}_1) \times \mathrm{P}(\mathcal{A}_2)$ of its components.

Example 9.2.7 *We illustrate the product construct of Proposition 9.2.6 by looking at the PCS* $\boldsymbol{Bool} \,\&\, \boldsymbol{Bool}$. *By definition,* $\mathrm{P}(\boldsymbol{Bool} \,\&\, \boldsymbol{Bool})$ *consists of those elements* x *such that both* $\pi_1 x$ *and* $\pi_2 x$ *are in* $\mathrm{P}(\boldsymbol{Bool})$. *For instance, it contains the following element:*

$$x_{(1,\mathtt{t})} = \frac{1}{2} \quad x_{(1,\mathtt{t})} = \frac{1}{2} \quad \textit{and} \quad x_{(2,\mathtt{t})} = 1 \quad x_{(2,\mathtt{f})} = 0.$$

We can see that elements in $\mathrm{P}(\boldsymbol{Bool} \,\&\, \boldsymbol{Bool})$ *encode two independent distribution over the booleans, in contrast with elements in* $\mathrm{P}(\boldsymbol{Bool} \otimes \boldsymbol{Bool})$, *that encode one distribution over the set of pairs of booleans.*

Observe that using the product construct &, we can easily constructs PCSs that cannot be interpreted as the set of sub-distributions on some underlying set: it is for instance the case of the PCS $\mathbf{Bool} \,\&\, \mathbf{Bool}$ of Example 9.2.7.

9.2.2 PCoh as a new-Seely category, hence a model of LL.

We recall here briefly the exponential modality given by Danos and Ehrhard in [34], that makes **PCoh** a *new-Seely* category and as a consequence gives it the structure of a model of linear logic. We call this exponential modality the *entire exponential modality*, denoted by $!_e$. The use of the adjective *entire* is motivated by the fact that the morphisms in the Kleisli category associated to $!_e$ should be seen as entire functions from $\mathrm{P}(\mathcal{A})$ to $\mathrm{P}(\mathcal{B})$, as we will highlight in this section. We refer to [34] for details.

Definition 9.2.5 *We define a functor* $!_e : \mathbf{PCoh} \to \mathbf{PCoh}$.

- *Its action on objects is taken as:*

$$|!_e\mathcal{A}| := \mathcal{M}_f(|\mathcal{A}|), \qquad \mathrm{P}(!_e\mathcal{A}) := \{v^{!_e} \mid v \in \mathrm{P}(\mathcal{A})\}^{\perp\perp}, \tag{9.2}$$

  where $v^{!_e}$ *is the* promotion *of* v, *i.e. the vector of* $(\mathbb{R}_+)^{\mathcal{M}_f(|\mathcal{A}|)}$ *defined as* $v^{!_e}_\mu := \prod_{a \in \mathsf{S}(\mu)} v_a^{\mu(a)}$, *for any* $\mu \in \mathcal{M}_f(|\mathcal{A}|)$.

- *The action of* $!_e$ *on a morphism* $f \in \mathbf{PCoh}(\mathcal{A}, \mathcal{B})$ *is defined, for any* $\mu \in |!_e\mathcal{A}|$ *and* $\nu = [b_1, \ldots, b_n] \in |!_e\mathcal{B}|$, *as:*

$$(!_e f)_{\mu,\nu} := \sum_{\substack{(a_1,\ldots,a_n) \text{ s.t.} \\ [a_1,\ldots,a_n]=\mu}} \prod_{i=1}^{n} f_{a_i,b_i} \tag{9.3}$$

Notice that the above sum varies on the set of *different* enumerations of μ. If μ is a multiset $[a, \ldots, a]$ of support a singleton, then the sum has only one term, while in case of multisets with no repetitions, the sum has $n!$ terms. Remark also that the definition is independent from the chosen enumeration of ν.

Example 9.2.8 *We look to the effect of* $!_e$ *on the morphisms from* $\boldsymbol{Bool}$ *to itself. Let us consider the morphism* $g \in \mathbf{PCoh}(\boldsymbol{Bool}, \boldsymbol{Bool})$ *defined at Example 9.2.3.* $!_e g$ *becomes then a morphism from* $!_e\boldsymbol{Bool}$ *to itself. First, we see that* $!_e g_{\mu,\nu} = 0$ *whenever the cardinality*

of μ and ν are not equal. We look then at what happens when we consider multiset ν, μ of cardinality 1:

$$(!_e g)_{[\mathtt{t}],[\mathtt{t}]} = g_{(\mathtt{t},\mathtt{t})} \quad (!_e g)_{[\mathtt{t}],[\mathtt{f}]} = g_{(\mathtt{t},\mathtt{f})} \quad (!_e g)_{[\mathtt{f}],[\mathtt{t}]} = g_{(\mathtt{f},\mathtt{t})} \quad (!_e g)_{[\mathtt{f}],[\mathtt{f}]} = g_{(\mathtt{f},\mathtt{f})}$$

So we can see that the projections of $!_e g$ on the set of multiset of cardinality 1 are essentially the linear morphism g. If we consider now the multisets of cardinality 2, we see that for instance:

$$(!_e g)_{[\mathtt{t}^2],[\mathtt{t},\mathtt{f}]} = g_{(\mathtt{t},\mathtt{t})} g_{(\mathtt{t},\mathtt{f})} \quad (!_e g)_{[\mathtt{f}^2],[\mathtt{t},\mathtt{f}]} = g_{(\mathtt{f},\mathtt{t})} g_{(\mathtt{f},\mathtt{f})}$$
$$(!_e g)_{[\mathtt{t},\mathtt{f}],[\mathtt{t},\mathtt{f}]} = g_{(\mathtt{t},\mathtt{t})} g_{(\mathtt{f},\mathtt{f})} + g_{(\mathtt{t},\mathtt{f})} g_{(\mathtt{f},\mathtt{t})}$$

We see that the coefficients are obtained by looking at all the possible ways to choose how to send elements of the multisets μ into elements of the multiset ν, and then taking into account the cost of sending them using the Markov transition g.

Example 9.2.9 *Equation (9.3) introduces arbitrary large scalars, moving us away from the intuitive setting of distributions and stochastic matrices (see Examples 9.2.1, 9.2.3). For an example, consider the morphism $f \in \mathbf{PCoh}(\boldsymbol{Bool}, \mathbf{1})$ defined by $f_{\boldsymbol{true},*} = f_{\boldsymbol{false},*} = 1$. Remark that for any $n, m \in \mathrm{N}$, we have: $!_e f_{[\boldsymbol{true}^n, \boldsymbol{false}^m],[\star^{n+m}]} = \frac{(n+m)!}{n!m!}$, which is the number of different enumerations of the multiset $[\boldsymbol{true}^n, \boldsymbol{false}^m]$ with n (resp. m) occurrences of **true** (resp. **false**). This shows why, in the definition of a PCS, coefficients have to be in the whole of $\mathbb{R}_+$ and cannot be restricted to $[0, 1]$.*

Denotations of $\mathsf{PCF}_\oplus$ programs are going to be morphisms in the co-Kleisli category associated with the comonad $!_e$, i.e. elements of $\mathbf{PCoh}(!_e\mathcal{A}, \mathcal{B})$. Recall that the morphisms in $\mathbf{PCoh}(\mathcal{A}, \mathcal{B})$ are the linear functions from the set of $\mathcal{A}$ cliques to the set of $\mathcal{B}$ cliques. Formally, morphisms in $\mathbf{PCoh}(!_e\mathcal{A}, \mathcal{B})$ are matrices in $\mathbb{R}_+^{\mathcal{M}_f(|\mathcal{A}|)\times|\mathcal{B}|}$. We can however represent them also as functions from $\mathrm{P}(\mathcal{A})$ into $\mathrm{P}(\mathcal{B})$, as highlighted below.

Definition 9.2.6 *We call* functional interpretation *of a matrix $f \in \mathbb{R}_+^{\mathcal{M}_f(|\mathcal{A}|)\times|\mathcal{B}|}$ the function:*

$$\widetilde{f} : \mathrm{P}(\mathcal{A}) \to (\mathbb{R}_+ \cup \{\infty\})^{|\mathcal{B}|}$$
$$x \mapsto f\, x^{!_e}$$

Lemma 9.2.7 (from [34]) *The functional interpretation entirely characterizes the morphisms in $\mathbf{PCoh}(!_e\mathcal{A}, \mathcal{B})$, in the following sense:*

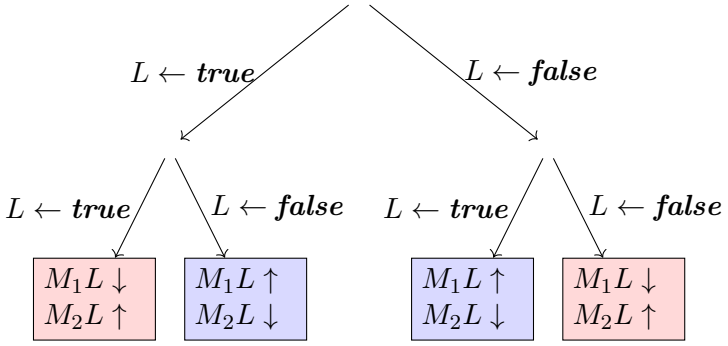
- *For any $f \in \mathbb{R}_+^{\mathcal{M}_f(|\mathcal{A}|)\times|\mathcal{B}|}$, f is a morphism in $\mathbf{PCoh}(!_e\mathcal{A}, \mathcal{B})$, if and only if the image of the functional interpretation of f is contained in $\mathrm{P}(\mathcal{B})$.*
- *Two morphisms $f, g \in \mathbf{PCoh}(!_e\mathcal{A}, \mathcal{B})$ are equal if and only if their functional interpretation $\widetilde{f}$ and $\widetilde{g}$ coincide.*

Observe that the functional interpretation of morphisms in $\mathbf{PCoh}(!_e\mathcal{A}, \mathcal{B})$ are the power series with non-negative coefficients from $\mathcal{A}$ cliques into $\mathcal{B}$ cliques. We highlight this below on examples for the **Bool** type.

Example 9.2.10 *We consider the denotation of the following two $\mathsf{PCF}_\oplus$ programs—for a CBN operational semantics.*

$$M_1 = \lambda x.\, [\mathsf{if}\, x\, \mathsf{then}\, (\mathsf{if}\, x\, \mathsf{then}\, I\, \mathsf{else}\, \Omega)\, \mathsf{else}\, (\mathsf{if}\, x\, \mathsf{then}\, \Omega\, \mathsf{else}\, I)]\,;$$
$$M_2 = \lambda x.\, [\mathsf{if}\, x\, \mathsf{then}\, (\mathsf{if}\, x\, \mathsf{then}\, \Omega\, \mathsf{else}\, I)\, \mathsf{else}\, (\mathsf{if}\, x\, \mathsf{then}\, I\, \mathsf{else}\, \Omega)]\,.$$

These programs take as argument a probability distribution on booleans. At execution time, they both sample two times their argument. $M_1 L$ terminates if and only if the two sampling of L coincide, while $M_2 L$ terminates if and only if the two sampling of L do not *coincide. We represent this idea below by looking at the execution tree:*



The morphism f^1, f^2 in $\mathbf{PCoh}(!\textbf{\textit{Bool}}, \mathbf{1})$ interpreting respectively the program M_1 and N_2 are as follows:

$$f^1_{\mu,\star} = \begin{cases} 1 & \text{if } \mu = [\mathtt{t},\mathtt{t}] \text{ or } \mu = [\mathtt{f},\mathtt{f}]; \\ 0 & \text{otherwise.} \end{cases} \qquad f^2_{\mu,\star} = \begin{cases} 2 & \text{if } \mu = [\mathtt{t},\mathtt{f}]; \\ 0 & \text{otherwise.} \end{cases}$$

Observe that the coefficient $f^i_{\mu,\star}$ above corresponds to the number of paths in the execution tree such that the multiset consisting of all sample results coincide with μ, and moreover the path end up in a terminating state for $M_i L$. In particular, we see that we can obtain coefficients that are greater that 1, as soon as we consider types build using $!_e$. Those scalars may become arbitrarily large when we construct programs that sample arbitrarily many times from their argument. We can now look at the functional interpretation of f: it it the function from $\mathrm{P}(\textbf{\textit{Bool}})$ into $\mathrm{P}(\mathbf{1})$ defined as:

$$\begin{aligned} \widetilde{f^1}(x) &= f^1\, x^{!_e} = x^{!_e}_{[\mathtt{t},\mathtt{t}]} + x^{!_e}_{[\mathtt{f},\mathtt{f}]} \\ &= x^2_{\mathtt{t}} + x^2_{\mathtt{f}} \end{aligned} \qquad \begin{aligned} \widetilde{f^2}(x) &= f^2\, x^{!_e} = 2 \cdot x^{!_e}_{[\mathtt{t},\mathtt{f}]} \\ &= 2 \cdot x_{\mathtt{t}} \cdot x_{\mathtt{f}} \end{aligned}$$

We can see that both $\widetilde{f^1}$ and $\widetilde{f^2}$ are polynomials of order two with two indeterminates, that moreover have all their coefficients non-negative.

Example 9.2.11 *We present here an example of morphism in $\mathbf{PCoh}(!_e\textbf{\textit{Bool}}, \mathbf{1})$ that is not a polynomial, but is still an* analytic *function, i.e. a power series. Let us consider the following program of $\mathsf{PCF}_\oplus$, build using the fixpoint construction—see the syntax of $\mathsf{PCF}_\oplus$ in Definition 1.3.1 of Chapter 1:*

$$N = \mathit{fix}\, x^{B \to 1} \lambda y.^B (\mathsf{if}\ y\ \mathsf{then}\ I\ \mathsf{else}\ xy)$$

*This program's behavior is as follows: it takes an argument of boolean type, and samples it repeatedly until it gets the value **true**. In $\mathbf{PCoh}$, the program N is interpreted as the morphism $g \in \mathbf{PCoh}(!_e\textbf{\textit{Bool}}, \mathbf{1})$ defined as:*

$$g_{\mu,b} = \begin{cases} 1 & \text{if } \exists n,\ \mu = [\mathtt{f}^n, \mathtt{t}]; \\ 0 & \text{otherwise.} \end{cases}$$

Looking now at the functional interpretation of g, we see that $\widetilde{g}$ is a power series which has as indeterminates $x_{\mathtt{t}}$ and $x_{\mathtt{f}}$:

$$\widetilde{g}(x) = \sum_{n \in \mathbb{N}} x^n_{\mathtt{f}} x_{\mathtt{t}}$$

The functorial promotion is equipped with a structure of comonad. The counit (or *dereliction*) $\mathtt{der}_{\mathcal{A}} \in \mathbf{PCoh}(!_e\mathcal{A}, \mathcal{A})$ is defined as $(\mathtt{der}_{\mathcal{A}})_{\mu,a} = \delta_{\mu,[a]}$. The comultiplication (or *digging*), denoted as $\mathtt{digg}_{\mathcal{A}} \in \mathbf{PCoh}(!_e\mathcal{A}, !_e!_e\mathcal{A})$, is given by $(\mathtt{digg}_{\mathcal{A}})_{\mu,M} = \delta_{\mu,\biguplus M}$, where $\biguplus M$ is the multiset in $|!_e\mathcal{A}|$ obtained as the multiset union of the multisets in $M \in |!_e!_e\mathcal{A}|$.

Proposition 9.2.8 (from [34]) PCoh *with the co-monad* $!_e$ *is a new-Seely category, hence a model of Linear Logic.*

In particular, contraction $\mathtt{contr}_{\mathcal{A}} \in \mathbf{PCoh}(!_e\mathcal{A}, !_e\mathcal{A} \otimes !_e\mathcal{A})$ and weakening $\mathtt{weak}_{\mathcal{A}} \in \mathbf{PCoh}(!_e\mathcal{A}, \mathbf{1})$ are given by $(\mathtt{contr}_{\mathcal{A}})_{\mu,(\mu',\mu'')} = \delta_{\mu,\mu' \uplus \mu''}$, and $(\mathtt{weak}_{\mathcal{A}})_{\mu,\star} = \delta_{\mu,[]}$.

9.2.3 $\mathbf{PCoh}_!$: a model for higher-order languages with discrete probability.

In [34], Danos and Ehrhard define a denotational semantics in $\mathbf{PCoh}_!$ for both $\Lambda_\oplus$ and $\mathsf{PCF}_\oplus$. We first look at how the convexity inherent to the PCS structure allows to encode the probabilistic choice: the interpretation of a term $M \oplus N$ is taken as the morphism $\frac{1}{2}[\![M]\!] + \frac{1}{2}[\![N]\!]$, where $[\![M]\!]$ and $[\![N]\!]$ are respectively the denotational interpretation of the terms M and N . Danos and Ehrhard then need to build in $\mathbf{PCoh}_!$ the categorical structure allowing to build models of untyped λ-calculus and PCF: as we have already mentioned, it is respectively the existence of a reflexive object, and the existence of a natural numbers object and cpo-enrichment. The reflexive object is obtained in $\mathbf{PCoh}_!$ as the fixpoint of the operation $\mathcal{A} \mapsto (!(\,\&_{n\in\mathbb{N}}\,\mathcal{A}))^\perp$. The natural numbers object is the PCS $\mathbb{N}^{\mathbf{PCoh}}$ that we have presented in Example 9.2.2, while the partial order on morphisms is taken as the componentwise order on matrices—i.e if $u, v \in \mathbf{PCoh}_!(\mathcal{A}, \mathcal{B})$, $u \leq v$ if for all $\mu \in \mathcal{M}_f(|\mathcal{B}|)$, and $b \in |\mathcal{B}|$, it holds that $u_{\mu,b} \leq v_{\mu,b}$. Observe that this order is stronger that the extensional order on the functional interpretations of morphisms—where the extensional order is taken as: $\widetilde{u} \leq^{\text{ext}} \widetilde{v}$ if for every $x \in \mathrm{P}(\mathcal{A})$, $\widetilde{u}(x) \leq \widetilde{v}(x)$, with $\leq$ the componentwise order on $|\mathcal{A}|$.

Building on these elements, Ehrhard and Danos define formally an interpretation of every $\mathsf{PCF}_\oplus$ term in $\mathbf{PCoh}_!$. The resulting denotational model has been shown to be fully abstract by Ehrhard, Pagani, and Tasson in [46].

9.3 $\mathbf{Cstab_m}$: The Category of Measurable cones and measurable stable functions.

We give here a brief overview of the Cartesian closed category $\mathbf{Cstab_m}$ that was introduced by Ehrhard, Pagani, and Tasson in [45]. It has been designed to provide a model of higher-order computations with *continuous* probabilities. To illustrate this, the authors of [45] built in $\mathbf{Cstab_m}$ a computationally adequate model of the language $\mathsf{PCF}_{\text{sample}}$—the higher-order language with continuous probabilities presented in Chapter 1. The construction of the category $\mathbf{Cstab_m}$ in [45] is done in two steps: the authors first built the category **Cstab** of abstract cones and so-called *stable* functions between cones, and then defined the category $\mathbf{Cstab_m}$ as a refinment of **Cstab** with additional measurability constraints. We present here the two categories **Cstab** and $\mathbf{Cstab_m}$ following the presentation in [45]. All the results we state in this section have also been extracted from [45].

9.3.1 Cones

The use of a notion of cones in denotational semantics to deal with probabilistic behavior goes back to Kozen [70]. We take here the same definition of cone as in [45].

Definition 9.3.1 *A cone C is a $\mathbb{R}_+$-semimodule given together with an $\mathbb{R}_+$ valued function $\|\cdot\|_C$ called* norm of C*, and verifying:*

$$\begin{array}{ll} \big(x+y=x+y'\big) \Rightarrow y=y' & \|\alpha x\|_C = \alpha\|x\|_C \\ \|x+x'\|_C \leq \|x\|_C + \|x'\|_C & \|x\|_C = 0 \Rightarrow x = 0 \\ \|x\|_C \leq \|x+x'\|_C & \end{array}$$

The most immediate example of cone is the non-negative real half-line, when we take as norm the identity. Another example is the positive quadrant in a 2-dimensional plane, endowed with the euclidean norm. In a way, the notion of cone is the generalization of the idea of a space where all elements are *non-negative*. This analogy gives us a generic way to define a pre-order, using the $+$ of the cone structure.

Definition 9.3.2 *Let be C a cone. Then we define a partial order $\preceq_C$ on C by: $x \preceq_C y$ if there exists $z \in C$, with $y = x + z$.*

If $x \preceq_C y$, we will denote by $y - x$ the—unique—element z such that $y = x + z$. We define $\mathscr{B}C$ as the set of elements in C of norm smaller or equal to 1. We will sometimes call it the *unit ball* of C. Moreover, we will also be interested in the *open unit ball* $\mathscr{B}^\circ C$, defined as the set of elements of C of norm smaller than 1. In [45], the authors restrict themselves to cones verifying a completeness criterion: it allows them to define the denotation of the recursion operator in $\mathsf{PCF}_{\text{sample}}$, thus enforcing the existence of fixpoints.

Definition 9.3.3 *A cone C is said to be* sequentially complete *if any non-decreasing sequence $(x_n)_{n\in\mathbb{N}}$ of elements of $\mathscr{B}C$ has a least upper bound $\sup_{n\in\mathbb{N}} x_n \in \mathscr{B}C$.*

We illustrate Definition 11.1.1 by giving the sequentially complete cone used in [45] as the denotational semantics of the base type R in $\mathsf{PCF}_{\text{sample}}$.

Example 9.3.1 *We take Meas(X) as the set of* finite *measures over a measurable space X and we take $\|\mu\|_{Meas(X)} = \mu(X)$ as the underlying norm. It holds that Meas(X) is a sequentially complete cone, and moreover its unit ball is exactly the set of sub-probability measures.*

The cone Meas$(\mathbb{R})$ —where $\mathbb{R}$ is endowed with the Borel σ-field as done in Example 1.4.1 in Chapter 1—allows to model continuous data, as shown in [45]: indeed we can interpret in it the term $\underline{r}$ of PCF_{sample} for every $r \in \mathbb{R}$: we take its denotational semantics as δ_r, the Dirac measure with respect to r *defined by taking $\delta_r(U) = 1$ if $r \in U$, and $\delta_r(U) = 0$ otherwise.*

Example 9.3.2 *We look now at another example of sequentially complete cone. We fix X a measurable space, and μ a measure on X. We take as cone the set of measurable function $X \to \mathbb{R}_+$ for some measurable space X, with the norm given as $\|f\| = \int_A f d\mu$. Lebesgue Monotone Convergence Theorem—see [100]—shows that this cone is sequentially complete.*

Definition 9.3.4 *The product cone is defined as $\prod_{i\in I} C_i = \{(x_i)_{i\in I} \mid \forall i \in I, x_i \in C_i\}$, and $\|x\|_{\prod_{i\in I} C_i} = \sup_{i\in I} \|x_i\|_{C_i}$. We denote by $C_1 \times C_2$ the binary product $\prod_{i\in\{1,2\}} C_i$.*

The object of the category **Cstab** are going to be sequentially complete cones. We need now to look at the morphisms between such cones. The relevant part of the cone for interpreting programs is actually its unit ball: as a consequence, our morphisms from a cone C into a cone D are going to be a particular class of functions $\mathscr{B}C \to D$.

Definition 9.3.5 *Let C, D be cones. We call* cone-functions from C to D *the functions $\mathscr{B}C \to D$.*

- *We say that a cone function f from C to D is* bounded, *if there exists $\alpha > 0$ such that $f(\mathscr{B}C) \subseteq \alpha \cdot \mathscr{B}D = \{\alpha \cdot x \mid x \in \mathscr{B}D\}$. We say that f is* 1*-bounded if it holds with $\alpha = 1$.*
- *We say that a bounded cone function is* sequentially Scott-continuous *if it is non-decreasing, and it commutes with the least upper bound of non-decreasing sequences.*

It is standard, in *order-based* denotational models, to ask morphisms to be Scott-continuous. We can first observe that the addition and the scalar operation—seen as cones functions—of a sequentially complete cone are always Scott-continuous.

Lemma 9.3.1 *Let C be a sequentially complete cone. The addition $+ : \mathscr{B}C \times C \to C$ and the scalar multiplication $\cdot : [0,1] \times \mathscr{B}C \to C$ are sequentially Scott-continuous.*

However, Ehrhard Pagani and Tasson remarked that if we take as morphisms all Scott-continuous and 1-bounded cones functions, the resulting category is *not* Cartesian closed. We sum up briefly the problem here; it essentially comes from the fact that the cone order on Scott-continuous functions is much stronger than the extensional order. First, we will consider the denotation of Abramsky's *parallel convergence tester*—it takes two programs, and terminates if at least one of them terminates; see Chapter 5 for a more detailed computational description.

Definition 9.3.6 (Parallel Convergence Tester for Cones) *We denote as $\mathbf{1}$ the sequentially complete cones $\mathbb{R}_+$, with the absolute value $|\cdot|$ as norm. We consider the cone function $T_\parallel : \mathscr{B}\mathbf{1} \times \mathscr{B}\mathbf{1} \to \mathscr{B}\mathbf{1}$, defined as:*

$$T_\parallel(x,y) = x + y - x \cdot y.$$

The cone function $T_\parallel$ is 1-bounded, non-decreasing, and also Scott-continuous—it can be seen by observing that it is continuous in the sense of real analysis.

Suppose now that the category of sequentially complete cones, and Scott-continuous 1-bounded cones functions is Cartesian closed. It means that we can consider the currying of $T_\parallel$, that is a morphism $T_\parallel^{\mathrm{curr}} : \mathbf{1} \to (\mathbf{1} \Rightarrow \mathbf{1})$. Since it is a morphism, it has to be non-decreasing, and as a consequence $T_\parallel^{\mathrm{curr}}(0) \leq T_\parallel^{\mathrm{curr}}(1)$. It's here that the fact that we take an order on cones much stronger than the extensional order intervenes: indeed, what the previous inequality means is that there exists a 1-bounded non-decreasing Scott-continuous function $g : \mathscr{B}\mathbf{1} \to \mathbf{1}$ such that $T_\parallel^{\mathrm{curr}}(0) + g = T_\parallel^{\mathrm{curr}}(1)$. However, we can see that:

$$T_\parallel^{\mathrm{curr}}(0,\cdot) : \mathscr{B}\mathbf{1} \to \mathbf{1} \qquad\qquad T_\parallel^{\mathrm{curr}}(1,\cdot) : \mathscr{B}\mathbf{1} \to \mathbf{1}$$
$$z \mapsto z \qquad\qquad z \mapsto 1,$$

and as a consequence we should have $g = 1 - z$. We reach a contradiction here, because it means that g isn't non-increasing: it proves that the category we considered cannot be Cartesian closed.

To obtain a Cartesian closed category, Ehrhard Pagani and Tasson ask for an additional condition, i.e. the stability condition presented in the next section. Another possibility would be to relax the requirement of Scott-continuity, by asking them to be Scott-continuous for the extensional order, instead of the cone order. It is similar to what is done for Kegelspitzen, for instance in [95]. However, sticking with the cone order and adding the stability condition also allows to put more constraints on the morphisms.

9.3.2 Cstab: The Category of cones and stable functions.

Stable functions on cones are a generalization of well-known *absolutely monotonic functions* in real analysis: they are those functions $f : [0, \infty) \to \mathbb{R}_+$ which are infinitely differentiable, and such that moreover all their derivatives are non-negative. The relevance of such functions in real analysis comes from a result due to Bernstein: every absolutely monotonic function coincides with a power series. Moreover, it is possible to characterize absolutely monotonic functions without explicitly asking for them to be differentiable: it is exactly those functions such that all the so-called *higher-order differences*—which are quantities that can be define for any real function—are non-negative. (see [124], chapter 4). The definition of *pre-stable functions* in [45] generalizes this characterization.

First, we want to be able to talk about those $\vec{u} = (u_1, \ldots, u_n)$, such that $\|x + \sum u_i\|_C \leq 1$ for a fixed $x \in \mathscr{B}C$, and $n \in \mathbb{N}$. To that end, we introduce a cone C^n_x whose unit ball is exactly such elements. It is an adaptation of the definition given in [45] for the case where $n = 1$, and we show in the same way that it is indeed a cone.

Definition 9.3.7 (Local Cone) *Let be C a cone, $n \in \mathbb{N}$, and $x \in \mathscr{B}^\circ C$. We call* n-local cone at x, *and we denote C^n_x the cone C^n endowed with the following norm:*

$$\|(u_1, \ldots, u_n)\|_{C^n_x} = \inf \left\{ \frac{1}{r} \mid x + r \cdot \sum_{1 \leq i \leq n} u_i \in \mathscr{B}C \wedge r > 0 \right\}.$$

For $n \in \mathbb{N}$, we use $\mathcal{P}_+(n)$ (respectively $\mathcal{P}_-(n)$) for the set of all subsets I of $\{1, \ldots, n\}$ such that $n - \mathrm{card}(I)$ is even (respectively odd). We are now ready to introduce *higher-order differences*. Since we have only explicit addition, not subtraction, we define separately the positive part Δ^n_+ and the negative part Δ^n_- of those differences: For $f : \mathscr{B}C \to D$, $x \in \mathscr{B}C$, $\vec{u} \in \mathscr{B}C^n_x$, and $\epsilon \in \{-, +\}$, we define:

$$\Delta^n_\epsilon(f)(x \mid \vec{u}) = \sum_{I \in \mathcal{P}_\epsilon(n)} f(x + \sum_{i \in I} u_i)$$

Definition 9.3.8 *We say that a cone function $f : \mathscr{B}C \to D$ is* pre-stable *if, for every $n \in \mathbb{N}$, for every $x \in \mathscr{B}C$, $\vec{u} \in \mathscr{B}C^n_x$, it holds that:*

$$\Delta^n_-(f)(x \mid \vec{u}) \leq \Delta^n_+(f)(x \mid \vec{u}).$$

If f is pre-stable, we will set $\Delta^n f(x \mid \vec{u}) = \Delta^n_+ f(x \mid \vec{u}) - \Delta^n_- f(x \mid \vec{u})$. Observe that the quantity $\Delta^n f(x \mid \vec{u})$ is actually symmetric in $\vec{u}$, i.e. stable under permutations of the coordinates of $\vec{u}$.

Definition 9.3.9 *A cone function $f : \mathscr{B}C \to D$ is called a* stable function from C to D *if it is bounded, sequentially Scott-continuous, and pre-stable.*

Definition 9.3.10 *The category* **Cstab** *has as objects the sequentially complete cones, and as morphisms from C to D the stable functions f from C to D such that $f(\mathscr{B}C) \subseteq \mathscr{B}D$.*

It was shown in [45] that it is possible to endow **Cstab** with a Cartesian closed structure.

The function cone $C \Rightarrow D$ is the set of all stable functions, with $\|f\|_{C\Rightarrow D} = \sup_{x\in\mathscr{B}C} \|f(x)\|_D$. It was also shown in [45] that these cones are indeed sequentially complete, and that the lub in $C \Rightarrow D$ is computed pointwise. We will use also the cone of pre-stable functions from C to D, which is also sequentially complete.

9.3.3 Adding Measurability Requirements

In order to obtain a model of $\mathsf{PCF}_{\text{sample}}$ based on stable functions, the authors of [45] imposed *measurability requirements* to the category **Cstab**. The necessity of taking into account the measurability appears when looking at the denotational semantics of the $\mathsf{let}\, x \,=\, M \,\mathsf{in}\, N$. First, recall the typing rule for the $\mathsf{let}\cdot \,=\, \cdot\, \mathsf{in}$ construct:

$$\frac{\Gamma, x : R \vdash M : R \qquad \Gamma \vdash N : R}{\Gamma \vdash \mathsf{let}\, x \,=\, N \,\mathsf{in}\, M : R}$$

The interpretation of those typing judgments—in the case where Γ is the empty typing context—in the denotational model should be as follows:

$$[\![x : R \vdash M : R]\!] \,:\, \mathrm{Meas}(\mathbb{R}) \to \mathrm{Meas}(\mathbb{R})$$
$$[\![\vdash N : R]\!] \,:\, \mathrm{Meas}(\mathbb{R})$$

The computational behavior of the term $\mathsf{let}\, x \,=\, N \,\mathsf{in}\, M$ is as follows: first it samples from the distribution over reals encoded by M, and then it executes the program N with replacing the variable x by the result of the sampling. To express this behavior mathematically, we would like $[\![\vdash \mathsf{let}\, x \,=\, N \,\mathsf{in}\, M : R]\!]$ to be the measure over $\mathbb{R}$ expressed as follows:

$$[\![\vdash \mathsf{let}\, x \,=\, N \,\mathsf{in}\, M : R]\!] \,:\, \mathrm{Borels}(\mathbb{R}) \to \mathbb{R}_+$$
$$U \mapsto \int_{r\in\mathbb{R}} g_U(r) d\mu$$

where the measure μ is obtained as $\mu = [\![\Gamma \vdash N : R]\!]$, and the function $g_U : \mathbb{R} \to \mathbb{R}_+$ is defined as $g_U(r) = [\![\vdash \mathsf{let}\, x \,=\, N \,\mathsf{in}\, M \,:\, R]\!](\{r^1\})(U)$. However, to be able to write this integral, we need to be sure that the function g_U is *measurable*. It is for this purpose that Ehrhard Pagani and Tasson impose additional measurability constraint at all types, thus refining the category **Cstab** into another category $\mathbf{Cstab_m}$.The objects of $\mathbf{Cstab_m}$ are going to be complete cones, endowed with a family of *measurability tests*.

If C is a complete cone, we denote by C' the set of linear and Scott-continuous functions $C \to \mathbb{R}_+$.

Definition 9.3.11 *A* measurable cone *(MC) is a pair consisting of a cone C, and a collection of* measurability tests *$(\mathcal{M}^n(C))_{n\in\mathbb{N}}$), where for every n, $\mathcal{M}^n(C) \subseteq C'^{\mathbb{R}^n}$, such that:*

- *for every $n \in \mathbb{N}$, $0 \in \mathcal{M}^n(C)$;*
- *for every $n, p \in \mathbb{N}$, if $l \in \mathcal{M}^n(C)$, and $h : \mathbb{R}^p \to \mathbb{R}^n$ is a measurable function, then $l \circ h \in \mathcal{M}^p(C)$;*

- *for any $l \in \mathcal{M}^n(C)$, and $x \in C$, the function $u \in \mathbb{R}^n \mapsto l(u)(x) \in \mathbb{R}$ is measurable.*

Example 9.3.3 (from [45]) *Let X be a measurable space. We endow the cone of finite measures $Meas(X)$ with the family $\mathcal{M}(X)$ of measurable tests defined as:*

$$\mathcal{M}^n(X) = \{\epsilon_U \mid U \in \Sigma_X\} \quad \textit{where} \quad \epsilon_U(\vec{r})(\mu) = \mu(U),$$

where Σ_X is the set of all measurable subsets of X. Observe that in this case, the measurable tests are those functions that takes a measure—and a parameter $\vec{r}$—and return the value of the measure on some measurable set U. In the following, we will denote $\overline{Meas(X)}$ the measurable cone $(Meas(X), (\mathcal{M}^n(X))_{n\in\mathbb{N}})$.

We define now *measurable paths*, which are meant to be the *admissible* ways to send $\mathbb{R}^n$ into a MC C.

Definition 9.3.12 (Measurable Paths) *Let be $(C, (\mathcal{M}^n(C))_{n\in\mathbb{N}})$ a measurable cone. A* measurable path of arity n at C *is a function $\gamma : \mathbb{R}^n \to C$, such that $\gamma(\mathbb{R}^n)$ is bounded in C, and for every $k \in \mathbb{N}$, for every $l \in \mathcal{M}^k(C)$, the function $(\vec{r}, \vec{s}) \in \mathbb{R}^{k+n} \mapsto l(\vec{r})(\gamma(\vec{s})) \in \mathbb{R}_+$ is a measurable function.*

We denote $\text{Paths}^n(C)$ the set of measurable paths of arity n for the MC C . When a measurable path γ verifies $\gamma(\mathbb{R}^n) \subseteq \mathscr{B}C$, we say it is *unitary*. Using measurable paths, the authors of [45] add *measurability requirements* to their definition of stable functions.

Definition 9.3.13 *Let be C, D two MCs. A stable function $f : \mathscr{B}C \to D$ is* measurable *if for all unitary $\gamma \in Paths^n(C)$, $f \circ \gamma \in Paths^n(D)$.*

The category $\mathbf{Cstab_m}$ is therefore the category whose objects are MCs, and whose morphisms are measurable stable functions between MCs. In [45], the Cartesian closed structure of $\mathbf{Cstab_m}$ is derived from that of **Cstab** by endowing its exponentials and products with the measurability tests presented in Figure 9.5.

$$\mathcal{M}^n(\prod_{i\in I} \overline{C}_i) = \{\bigoplus_{i\in I} l_i \mid \forall i \in I, l_i \in \mathcal{M}^n(\overline{C}_i)\} \quad \text{with } I \text{ finite set.}$$

$$\mathcal{M}^n(\overline{C} \Rightarrow_m \overline{D}) = \{\gamma \rhd m \mid \gamma \in \text{Paths}^n(\overline{C}), m \in \mathcal{M}^n(\overline{D})\},$$

$$\text{with } (\textstyle\bigoplus_{i\in I} l_i(\vec{r}))((x_i)_{i\in I}) = \sum_{i\in I} l_i(\vec{r})(x_i) \in \mathbb{R}_+;$$
$$\text{and } (\gamma \rhd m)(\vec{r})(f) = m(\vec{r})(f(\gamma(\vec{r}))).$$

Figure 9.5: Cartesian Closed structure of $\mathbf{Cstab_m}$.

Chapter 10

! is the free comonoid in PCoh.

As we developed at some length in Chapter 9, distinct exponential structures may coexist in a model of Linear Logic. The axiomatisation proposed by Lafont, where the exponential modality $!A$ is the free comonoid generated by A, leads to a genericity criterion, since there is *at most* one free exponential modality in a model, and moreover it allows—up to some point, see the considerations in Section 9.1.2—to factorize every other exponential structure. The free exponential modality is well-known in both the coherence space model of LL **Coh**—see Example 9.1.1 in Chapter 9—and the relational model of LL **Rel**, being given respectively by the multi-clique and the multi-set functors. The goal of this chapter is to prove the existence of and to study the free exponential modality of the category **PCoh** of probabilistic coherence spaces. The only known exponential modality of **PCoh** is the entire exponential modality $!_e$, introduced by Danos and Ehrhard in [34], that we have presented in Chapter 9, and which is based on the notion of entire function. It is then natural to ask whether $!_f$ exists in **PCoh** and if it is the case, how it relates with $!_e$. In this chapter, we answer the first question positively and prove that $!_f$ and $!_e$ are the same modality, in spite of their different presentations. This result was published in a joint work with Ehrhard, Pagani and Tasson in [28].

Our main tool is a recipe given by Melliès, Tabareau and Tasson [82] for constructing $!_f$ out of a model of the multiplicative additive fragment of LL—i.e. a $\star$-autonomous category with finite products, see Chapter 9. The idea is to adapt the well-known formula defining the symmetric algebra generated by a vector space, the latter being the free commutative monoid. More precisely, the recipe gives (under suitable conditions) $!_f\mathcal{A}$ as the limit of a sequence of approximates $\mathcal{A}^{\leq n}$ which correspond to the equalizers of the tensor symmetries of a suitable space (see Section 10.1). At a first sight, $\mathcal{A}^{\leq n}$ moves us far away from the entire exponential $!_e\mathcal{A}$ of [34], in fact the coefficients appearing in $\mathcal{A}^{\leq n}$ are greater than those of $!_e\mathcal{A}$, so that one is tempted to suppose that $!_f\mathcal{A}$ is a space much bigger than $!_e\mathcal{A}$, exactly as it is the case for standard coherence spaces where the images under the finite multi-clique functor strictly contain those under the finite clique functor (Example 9.1.1). For any n, the coefficients of $\mathcal{A}^{\leq n}$ are larger than those of $!_e\mathcal{A}$, but as $n \to \infty$, these coefficients tend to be exactly those of $!_e\mathcal{A}$ (see Section 10.3).

10.1 Melliès, Tasson and Tabareau's formula

Melliès et al. give a recipe for constructing free commutative comonoids in [82], adapting the well-known formula defining the symmetric algebra generated by a vector space in the setting of LL. This adaptation is non-trivial mainly because the vector space construction uses biproducts, while products and coproducts are in general distinct in LL models, and

in fact this is the case for **PCoh**. The idea of [82] is to define $!_f A$ as the projective limit of the sequence of its "approximates" $A^{\leq 0}, A^{\leq 1}, A^{\leq 2}, \ldots$, where an approximate $A^{\leq n}$ is the equalizer of the $n!$ tensor symmetries over $(A \,\&\, 1)^{\otimes n}$. Intuitively, the object $A^{\leq n}$ describes the behavior of data of type $!_f A$ when duplicated *at most* n times. This behavior is given by the equalizers of the tensor symmetries because the model does not distinguish between the evaluation order of the n copies (in categorical terms, we are considering *commutative* comonoids). We start from $A \,\&\, 1$ instead of A because, in order to have a sequence, we need that each $A^{\leq n}$ in some sense encompasses its predecessors $A^{\leq 0}, A^{\leq 1}, \ldots, A^{\leq n-1}$. The exact meaning of "*to encompass*" is given by a family of morphisms $\mathtt{p}_{n,n-1} \in \mathbb{C}(A^{\leq n}, A^{\leq n-1})$ generated by the right projection of the product $A \,\&\, 1$ (see Definition 10.1.9). In standard coherence spaces, this turns out to be the simple fact that the set of cliques of $A^{\leq n+1}$ *contains* the cliques of $A^{\leq n}$. In contrast, this intuition is misleading for **PCoh** (Example 10.2.5), making our construction considerably subtler.

10.1.1 The tensor product of n objects

This section deals with the formal definition of the categorical object $A^{\otimes n}$, as well as the $n!$ symmetries over it, in the setting of any symmetric monoidal category. The formal construction is a bit involved, but we deemed necessary to include this presentation here, because symmetries are the basic building blocks of the categorical construction done by Melliès, Tabareau and Tasson, on which we rely heavily for this work. However, as it will be highlighted in Example 10.1.7, the concrete symmetry morphisms in **PCoh** are actually quite simple, and can be easily inferred from the definition of the $\otimes$ bifunctor—Definition 9.2.4 from Chapter 9.

To handle the tensor product of n objects in a monoidal category $\mathbb{C}$, we would like to consider an object of the form $A_1 \otimes (A_2 \otimes (\ldots))$, but we have to be careful since there are different possible ways to do the bracketing. In [79], Mac Lane encodes syntactically those bracketing as $\otimes$*-words*.

Definition 10.1.1 *The set of $\otimes$-words is generated by the following grammar:*

$$w ::= \mathbf{1} \;\big|\; \_ \;\big|\; (w \otimes w)$$

We associate inductively a length *to all $\otimes$-words, by taking length$(\mathbf{1}) = 0$, length$(_) = 1$ and length$(w \otimes w') =$ length$(w) +$ length(w').*

For any $\otimes$-word of length n, and any MC $\mathbb{C}$, we define a functor $\widetilde{w}^{\mathbb{C}} : \mathbb{C}^n \to \mathbb{C}$, as follows: when $A_1, \ldots, A_n$ are objects of $\mathbb{C}$, $\widetilde{w}^{\mathbb{C}}(A_1, \ldots, A_n)$ is defined inductively on the length n of w as:

$$\widetilde{\mathbf{1}}^{\mathbb{C}}() = \mathbf{1}_{\mathbb{C}}; \qquad \widetilde{(\_)}^{\mathbb{C}}(A) = A;$$
$$\widetilde{(w \otimes w')}^{\mathbb{C}}(A_1, \cdot, A_n, A_{n+1}, \cdot, A_{n+m}) = \widetilde{w}^{\mathbb{C}}(A_1, \cdot, A_n) \otimes_{\mathbb{C}} \widetilde{w'}^{\mathbb{C}}(A_{n+1}, \cdot, A_{n+m})$$
when w of length n, w' of length m.

Example 10.1.1 *We illustrate here the action of $\otimes$-words on any MC $\mathbb{C}$, by looking at $\otimes$-words of length 4. We consider the two different bracketings given by the $\otimes$-words: $w = (\_ \otimes \_) \otimes (\_ \otimes \_)$ and $w' = \mathbf{1} \otimes (\_ \otimes (\_ \otimes (\_ \otimes \_)))$. Whenever $A_1, \ldots, A_4$ are four objects of $\mathbb{C}$, the functors $\widetilde{w}^{\mathbb{C}}$ and $\widetilde{w'}^{\mathbb{C}}$ act on these objects as follows:*

$$\widetilde{w}^{\mathbb{C}}(A_1, \ldots, A_4) = (A_1 \otimes_{\mathbb{C}} A_2) \otimes_{\mathbb{C}} (A_3 \otimes_{\mathbb{C}} A_4)$$
$$\widetilde{w'}^{\mathbb{C}}(A_1, \ldots, A_4) = \mathbf{1}_{\mathbb{C}} \otimes_{\mathbb{C}} (A_1 \otimes_{\mathbb{C}} (A_2 \otimes_{\mathbb{C}} (A_3 \otimes_{\mathbb{C}} A_4))).$$

Notation 10.1.1 *We illustrate here how the functors $\widetilde{w}$ allow to formalize the different ways of bracketing a tuple of elements. We place ourselves in the symmetric monoidal category* **Set***, when we take the monoidal product as $\times$—the cartesian product on sets—and the set with a unique element $\{\star\}$ as $\mathbf{1}_{\mathbf{Set}}$. If $a_1, \dots, a_n$ are n elements of some set X, we note $w_{\mathbf{Set}}(a_1, \dots, a_n)$ for the unique element in the set $\widetilde{w}(\{a_1\}, \dots, \{a_n\})$. With this notation, we see that if we consider for instance the $\otimes$-word $w = (\_ \otimes \_) \otimes (\_ \otimes \_)$, $w_{\mathbf{Set}}(a_1, a_2, a_3, a_4)$ is the tuple $((a_1, a_2), (a_3, a_4))$, while if we take $w' = \mathbf{1} \otimes (\_ \otimes (\_ \otimes (\_ \otimes \_)))$, we obtain $w'_{\mathbf{Set}}(a_1, a_2, a_3, a_4) = (\star, (a_1, (a_2, (a_3, a_4))))$.*

Notation 10.1.2 *Recall that when we presented the bifunctor $\otimes$ for* **PCoh***—in Definition 9.2.4 of Chapter 9—we also defined a tensorial product $\otimes$ on vectors with non-negative real coefficients: if $x \in \mathbb{R}_+^X$, $y \in \mathbb{R}_+^Y$, $x \otimes y$ is the vector in $\mathbb{R}_+^{X \times Y}$ defined as $(x \otimes y)_{(v,w)} = x_v \cdot y_w$. Below, we formalize the n-tensor product of such vectors by using the functors $\widetilde{w}$ from Definition 10.1.1. We will indeed need to talk about n-tensor products of vectors when considering the n-tensor product in the category* **PCoh***.*

We define the category **Vect** *as follows: the objects are the pair (A, x) such that A is a countable set, and $x \in \mathbb{R}_+^A$, and the only morphisms in the category are the identity morphisms. We endow* **Vect** *with a symmetric monoidal structure: the action of $\otimes$ on objects is as follows $(A, x) \otimes (B, y) = (A \times B, x \otimes y)$, where $(x \otimes y)_{a,b} = x_a \cdot y_b$ for every $a \in A, b \in B$, accordingly to what we did in Definition 9.2.4. Since the only morphisms are the identity morphisms, we define $\otimes$ on morphisms as $\mathrm{id}_{A,x} \otimes \mathrm{id}_{B,y} = \mathrm{id}_{(A,x)\otimes(B,y)}$. We take as $\mathbf{1}_{\mathbf{Vect}}$ the pair $(\star, 1)$. If $x \in \mathbb{R}_+^A$, we will also note simply x for the object (A, x) in* **Vect***. To illustrate the notation, we see that if $w = (\_ \otimes \_) \otimes \_$, then $\widetilde{w}_{\mathbf{Vect}}(\frac{1}{2}e_{\mathtt{t}} + \frac{1}{2}e_{\mathtt{f}}, e_{\mathtt{t}}, e_{\mathtt{f}}) = \frac{1}{2}e_{((\mathtt{t},\mathtt{t}),\mathtt{f})} + \frac{1}{2}e_{((\mathtt{f},\mathtt{t}),\mathtt{f})} \in \mathbb{R}_+^{\{\mathtt{t},\mathtt{f}\}^3}$.*

We look now at the unitary restriction of the functors $\widetilde{w}$, i.e. when they are seen as functors $\mathbb{C} \to \mathbb{C}$ instead of $\mathbb{C}^n \to \mathbb{C}$. Indeed recall that our goal in this part is to investigate the n-th tensor product of a single object A.

Definition 10.1.2 (Power Tensor Functors) *Let be $\mathbb{C}$ a MC. If w is an $\otimes$-word of length n, we note $\overline{w}^{\mathbb{C}} : \mathbb{C} \to \mathbb{C}$ the functor defined as $\overline{w}^{\mathbb{C}}(A) = \widetilde{w}^{\mathbb{C}}(A, \dots, A)$, and $\overline{w}^{\mathbb{C}}(f) = \widetilde{w}^{\mathbb{C}}(f, \dots, f)$.*

For every $n \in \mathbb{N}$, we define a particular $\otimes$-word w^n: it is the $\otimes$ word of length n that corresponds to the precise bracketing $(((\_ \otimes \_) \otimes \_) \dots)$, i.e. the rightmost bracketing. We call n power of A the object $A^{\otimes n} := \overline{w^n}^{\mathbb{C}}(A)$.

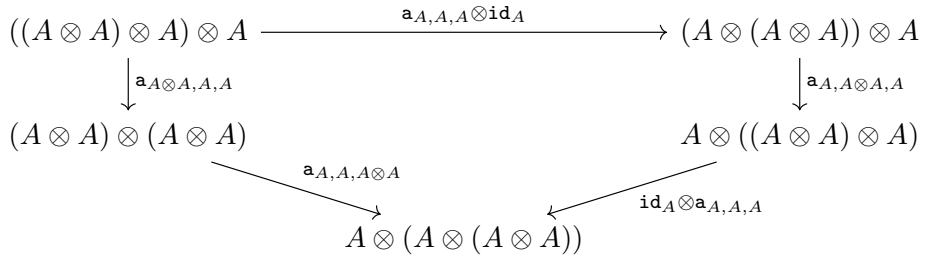
Observe that the bracketing is chosen in such a way that $A^{\otimes n+1} = A^{\otimes n} \otimes A$. We illustrate her the n-th tensor power when applied to the category **PCoh**: recall indeed that we are primarily interested in applying Melliès, Tabareau and Tasson recipe to this concrete SMC.

Example 10.1.2 *Let be $n \in \mathbb{N}$, and $\mathcal{A}$ be a PCS. Recall the definition of the bifunctor $\otimes$ in* **PCoh***—given in Definition 9.2.4. We see that $\mathcal{A}^{\otimes n}$ is the PCS described as follows:*

$$\left|\mathcal{A}^{\otimes n}\right| = \{\widetilde{w}^{\mathbf{Set}}(a_1, \dots, a_n) \mid \forall i, a_i \in |\mathcal{A}|\}$$
$$\mathrm{P}\left(\mathcal{A}^{\otimes n}\right) = \{\widetilde{w}^{\mathbf{Vect}}(x_1, \dots, x_n) \mid \forall i,\, x_i \in \mathrm{P}\left(\mathcal{A}\right)\}^{\perp\perp}$$

The next step is to look at how we can go from one bracketing to another, when using only the morphisms induced by the monoidal structure of $\mathbb{C}$, i.e. $\mathtt{a}, \mathtt{l}, \mathtt{r}$ and their inverses. Observe that, as we illustrate in Example 10.1.3 below, if w_1 and w_2 are two different $\otimes$-words, then there may actually be several different ways to construct such morphism from $\overline{w_1}(A)$ to $\overline{w_2}(A)$.

Example 10.1.3 *Let $\mathbb{C}$ be a monoidal category. We consider the two $\otimes$-words $w_1 = (((\_\otimes\_)\otimes\_)\otimes\_)$, and $w_2 = (\_\otimes(\_\otimes(\_\otimes\_)))$. The two paths in the diagram below represent two different ways to go from the bracketing specified by w_1 to the one specified by w_2.*



From the coherence diagrams for a symmetric monoidal category—that we recalled in Definition 9.1.4 from Chapter 9—it it possible to show that the diagram above commutes. As a consequence these two different ways to go from w_1 to w_2 are equivalent.

As illustrated in Example 10.1.1, the morphisms induced by the monoidal structure may be combined in different ways to go from one bracketing to another. Mac Lane shows in [79] a *coherence theorem*, that states that all these combinations lead actually to the *same* morphism in the category $\mathbb{C}$. We first define syntactic expressions to handle the construction of those morphisms. We say that a *formal SMC morphism* is a syntactic element generated by the following grammar.

$$e ::= (e \otimes e) \;\mid\; e^{-1} \;\mid\; \mathtt{id}_w \;\mid\; \mathtt{l}_w \;\mid\; \mathtt{r}_w \;\mid\; \mathtt{a}_{w_1,w_2,w_3} \;\mid\; e \circ e,$$

where w, w_1, w_2, w_3 are $\otimes$-words. To an expression e, we can associate in a natural way both a *domain* $\mathrm{D}(e)$ and a *co-domain* $\mathrm{C}(e)$, that are $\otimes$-words. Observe that we cannot in fact do this for all expressions, but for those which are *valid*, i.e. such that $\circ$ occurs only between expressions such that the domain of the first and the codomain of the second coincide.

Definition 10.1.3 *We say that a formal morphism e is* valid *if it is possible to associate to it a domain $Dom(e)$ and a codomain $C(e)$ under the inductive procedure of Figure 10.1.*

	$\mathrm{D}(\mathtt{id}_w) = w$	$\mathrm{C}(\mathtt{id}_w) = w$
	$\mathrm{D}(\mathtt{l}_w) = \mathbf{1} \otimes w$	$\mathrm{C}(\mathtt{l}_w) = w$
	$\mathrm{D}(\mathtt{r}_w) = w \otimes \mathbf{1}$	$\mathrm{C}(\mathtt{r}_w) = w$
	$\mathrm{D}(\mathtt{a}_{w_1,w_2,w_3}) = (w_1 \otimes w_2) \otimes w_3$	$\mathrm{C}(\mathtt{a}_{w_1,w_2,w_3}) = w_1 \otimes (w_2 \otimes w_3)$
	$\mathrm{D}(e \otimes e') = (\mathrm{D}(e) \otimes \mathrm{D}(e'))$	$\mathrm{C}(e \otimes e') = (\mathrm{C}(e) \otimes \mathrm{C}(e'))$
	$\mathrm{D}(e^{-1}) = \mathrm{C}(e)$	$\mathrm{C}(e^{-1}) = \mathrm{D}(e)$
if $\mathrm{C}(e') = \mathrm{D}(e)$	$\mathrm{D}(e \circ e') = \mathrm{D}(e')$	$\mathrm{C}(e \circ e') = \mathrm{C}(e)$

Figure 10.1: Domain and codomain for formal morphisms

For any valid formal MC morphism e, and for any category $\mathbb{C}$, we can define the *interpretation of* e as a natural transformation in $\mathbb{C}$ from the functors $\overline{\mathrm{D}(e)}^{\mathbb{C}}$ to $\overline{\mathrm{C}(e)}^{\mathbb{C}}$. It is obtained simply by replacing the syntactic construct $\mathtt{id}, \otimes, \ldots$ in the formal morphism by the concrete morphism of same name in the category $\mathbb{C}$.

Definition 10.1.4 *Let $\mathbb{C}$ be a MC. The interpretation of a valid formal MC morphism e in $\mathbb{C}$ with w_1 as domain and w_2 as co-domain is the natural transformation $[e]^{\mathbb{C}} : \overline{w_1}^{\mathbb{C}} \to \overline{w_2}^{\mathbb{C}}$ formally defined in Figure 10.2.*

$$
\begin{array}{ll}
[\mathtt{id}_w]^{\mathbb{C}}{}_A = \mathtt{id}^{\mathbb{C}}_{\overline{w}^{\mathbb{C}}(A)} & \\
[\mathtt{l}_w]^{\mathbb{C}}{}_A = \mathtt{l}^{\mathbb{C}}_{\overline{w}^{\mathbb{C}}(A)} & [\mathtt{r}_w]^{\mathbb{C}}{}_A = \mathtt{r}^{\mathbb{C}}_{\overline{w}^{\mathbb{C}}(A)} \\
[\mathtt{a}_{w_1,w_2,w_3}]^{\mathbb{C}}{}_A = \mathtt{a}^{\mathbb{C}}_{\overline{w_1}^{\mathbb{C}}(A),\overline{w_2}^{\mathbb{C}}(A),\overline{w_3}^{\mathbb{C}}(A)} & [e \otimes e']^{\mathbb{C}} = [e]^{\mathbb{C}} \otimes [e']^{\mathbb{C}} \\
[e^{-1}]^{\mathbb{C}} = ([e]^{\mathbb{C}})^{-1} & [e \circ e']^{\mathbb{C}} = [e]^{\mathbb{C}} \circ [e']^{\mathbb{C}}
\end{array}
$$

Figure 10.2: Interpretation for valid formal morphisms

Example 10.1.4 *We illustrate here how we interpret a formal morphism in some monoidal category $\mathbb{C}$. We consider the formal morphism*

$$e = (\mathtt{id}_{\_\otimes\_} \otimes \mathtt{l}^{-1}_{\_}) \circ \mathtt{a}^{-1}_{\_,\_,\_}$$

It is indeed a valid morphism, since we are able to attribute a domain and a codomain to e, by following the procedure in Figure 10.1. We represent graphically this procedure below, by splitting e into its basic components, and exhibiting the domain and codomain for each of them.

$$\_ \otimes (\_ \otimes \_) \xrightarrow{\mathtt{a}^{-1}_{\_,\_,\_}} (\_ \otimes \_) \otimes \_ \xrightarrow{\mathtt{id}_{\_\otimes\_} \otimes \mathtt{l}^{-1}} (\_ \otimes \_) \otimes (\mathbf{1} \otimes \_)$$

It means that $Dom(e) = \_ \otimes (\_ \otimes \_)$, while $C(e) = (\_ \otimes \_) \otimes (\mathbf{1} \otimes \_)$. Since e is a valid formal morphism, it has an interpretation in every MC $\mathbb{C}$: for every object A in $\mathbb{C}$, $[e]^{\mathbb{C}}_A$ is the $\mathbb{C}$ morphism in $A \otimes (A \otimes A) \to (A \otimes A) \otimes (\mathbf{1}_{\mathbb{C}} \otimes A)$ built as follows:

$$A \otimes (A \otimes A) \xrightarrow{\mathtt{a}^{-1}_{A,A,A}} (A \otimes A) \otimes A \xrightarrow{\mathtt{id}_{A\otimes A} \otimes \mathtt{l}^{-1}_A} (A \otimes A) \otimes (\mathbf{1} \otimes A)$$

As stated in Mac Lane's Coherence Theorem below, if we take any two expression e and e' with the same domain and codomain, then their interpretation in any MC $\mathbb{C}$ coincide. Moreover, there always exists such an interpretation between two $\otimes$ words of same length.

Proposition 10.1.3 (Mac Lane's Coherence Theorem) *Let $\mathbb{C}$ be a monoidal category, and w_1, w_2 be two $\otimes$-words of length n. Then for every object A, there exists a unique (natural) isomorphism $\overline{w_1}^{\mathbb{C}}(A) \to \overline{w_2}^{\mathbb{C}}(A)$ which is the interpretation of a valid SMC formal morphism.*

As a consequence of the coherence theorem, every formal morphism—i.e. resulting from the interpretation of a formal morphism with w^n as both domain and co-domain—that we can draw from $A^{\otimes n}$ to itself coincide with $\mathtt{id}_{A^{\otimes n}}$.

When moreover the category $\mathbb{C}$ is a *symmetric* monoidal category, we can also use the symmetry `swap` on the object $A^{\otimes n}$, and we obtain a new class of formal endomorphisms on $A^{\otimes n}$.

Definition 10.1.5 *The* SMC formal morphisms *are the syntactic expression generated by the grammar below:*

$$e ::= e \otimes e \;\Big|\; e^{-1} \;\Big|\; \mathtt{id}_w \;\Big|\; \mathtt{l}_w \;\Big|\; \mathtt{r}_w \;\Big|\; \mathtt{a}_{w_1,w_2,w_3} \;\Big|\; e \circ e \;\Big|\; \mathtt{swap}_{w_1,w_2},$$

where w, w_1, w_2, w_3 are $\otimes$ words.

We now extend the definition of valid formal morphisms, domain and codomain to this new class of formal morphisms. We need to extend both the definition of domain and codomain from Figure 10.1, and the definition of interpretations from Figure 10.2 to the SMC formal morphisms, i.e. to specify how we deal with the formal morphism $\mathtt{swap}_{w_1,w_2}$.

$$\mathrm{D}(\mathtt{swap}_{w_1,w_2}) = (w_1 \otimes w_2) \qquad \mathrm{C}(\mathtt{swap}_{w_1,w_2}) = (w_2 \otimes w_1)$$

$$[\mathtt{swap}_{w_1,w_2}]^{\mathbb{C}}_{A} = \mathtt{swap}^{\mathbb{C}}_{\overline{w_1}^{\mathbb{C}}(A),\overline{w_2}^{\mathbb{C}}(A)}$$

Observe that it is not necessarily true—i.e. false in the non trivial case—that the interpretations of all SMC formal morphisms with w^n as both domain and codomain coincide with the identity on $A^{\otimes n}$. This phenomenon is highlighted in Example 10.1.5 below.

Example 10.1.5 *Let $\mathbb{C}$ be any SMC. Observe that we can construct for instance a formal morphism f with $w^3 = (\_ \otimes \_) \otimes \_$ as both domain and codomain, as follows:*

$$\begin{array}{ccccc}
(_ \otimes _) \otimes _ & \xrightarrow{\mathtt{swap}_{_,_} \otimes \mathtt{id}} & (_ \otimes _) \otimes _ & \xrightarrow{\mathtt{a}_{_,_,_}} & _ \otimes (_ \otimes _) \\
 & & & & \downarrow{\scriptstyle \mathtt{id}__ \otimes \mathtt{swap}_{_,_}} \\
(_ \otimes _) \otimes _ & & \xleftarrow[\mathtt{a}^{-1}_{_,_,_}]{} & & _ \otimes (_ \otimes _)
\end{array}$$

It does not hold in general—and indeed it does not holds in **PCoh***, that the interpretation of f and the interpretation of $\mathtt{id}_\_$ coincide on every $A^{\otimes 3}$.*

What actually happens is that a formal morphism is not characterized just by its domain and co-domain anymore, but also by its *underlying permutation*, that keep track of all the transposition done while applying the morphism, and that can be defined inductively on the structure of formal morphisms.

Definition 10.1.6 (Underlying Permutation) *For every SMC formal morphisms e, with domain and co-domain of length n, we define its underlying permutation as the element $p(e) \in \mathfrak{S}_n$ defined inductively on the structure of e as specified in Figure 10.3.*

Example 10.1.6 *We see that the* underlying permutation *of the formal SMC morphism f defined in Example 10.1.5 is $\sigma = (2,3) \circ (1,2) \in \mathfrak{S}_3$.*

Mac Lane showed for SMC a similar coherence theorem as the one for MC: essentially it states that every formal diagrams generated by the structural SMC morphisms commute as soon as they have the same underlying permutation, and that moreover such a diagram exist for every permutation.

$$p(\mathtt{id}_w) = \mathbf{1}_{\mathfrak{S}_n}; \qquad p(\mathtt{l}_w) = \mathbf{1}_{\mathfrak{S}_n}; \qquad p(\mathtt{r}_w) = \mathbf{1}_{\mathfrak{S}_n}; \qquad \text{with } n = |w|$$

$$\begin{aligned} &p(\mathtt{a}_{w_1,w_2,w_3}) = \mathbf{1}_{\mathfrak{S}_n}; && \text{with } n = |w_1| + |w_2| + |w_3| \\ &p(e \otimes e') = p(e)^{\,n} \mid^{m} p(e') && \text{with } n = |\mathrm{D}(e)|;\ m = |\mathrm{D}(e')| \\ &p(e^{-1}) = p(e)^{-1} && p(e \circ e') = p(e) \circ p(e') \end{aligned}$$

$$p(\mathtt{swap}_{w_1,w_2}) = j \in \{1,\dots,n+m\} \mapsto (j-n \text{ if } j > n;\ j+m \text{ if } j \leq n) \quad \text{with } n = |w_1|, m = |w_n|$$

Figure 10.3: Underlying permutation for SMC formal morphisms

Proposition 10.1.4 (Mac Lane's Coherence Theorem for SMC) *Let be $\mathbb{C}$ a SMC, and $n \in \mathbb{N}$. Then for every permutation σ, there exists a unique natural isomorphism $A^{\otimes n} \to A^{\otimes n}$ which is the interpretation of a formal morphism with w^n as domain and codomain, and underlying permutation σ. We will call* symmetry specified by σ over $A^{\otimes n}$ *this isomorphism, and we will denote it $\mathtt{s}^\sigma_A : A^{\otimes n} \to A^{\otimes n}$.*

Example 10.1.7 *Let $\mathcal{A}$ be a PCS. Recall the expression of $\mathcal{A}^{\otimes n}$ from Example 10.1.2. For every $n \in \mathbb{N}$, and $\sigma \in \mathfrak{S}_n$, we can see that $\mathtt{s}^\sigma_{\mathcal{A}} \in \mathbf{PCoh}(\mathcal{A}^{\otimes n}, \mathcal{A}^{\otimes n})$ is the matrix defined as follows:*

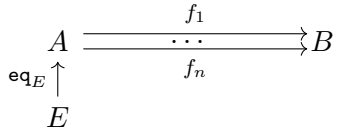
$$(\mathtt{s}^\sigma_{\mathcal{A}})_{w^n(a_1,\dots,a_n),w^n(a'_1,\dots,a'_n)} = \delta_{a'_1,a_{\sigma(1)}} \dots \delta_{a'_n,a_{\sigma(n)}}.$$

In the following, we will use sequences of the form $\vec{a} = a_1,\dots,a_n$ to represent the elements $w^n(a_1,\dots,a_n) \in |\mathcal{A}^{\otimes n}|$. We will also use $x_1 \otimes \dots \otimes x_n$ to represent $w^n(x_1,\dots,x_n)$, when the x_i are vectors with non-negative real coefficients.

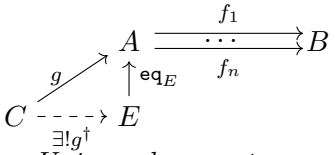
10.1.2 The Equalizer of symmetries

We present now the first building block of Melliès, Tabareau and Tasson's construction: it consists in building *the exponential approximations* $A^{\leq n}$, for every order $n \in \mathbb{N}$. These $A^{\leq n}$ are taken as—when it exists—the *equalizer* of the $n!$ symmetries from $(A \,\&\, \mathbf{1})^{\otimes n}$. Here, we recall the definition of equalizers in category theory, as well as Melliès, Tabareau, Tasson's definition of the exponential approximations.

Definition 10.1.7 (Equalizer) *Let $\mathbb{C}$ be some category, and A, B two objects of $\mathbb{C}$. Let $f_1,\dots,f_n$ a family of morphisms in $\mathbb{C}(A,B)$. An* equalizer *of the $(f_i)_{1\leq i\leq n}$ is a pair $(E, \mathtt{eq}_E)$, where E is an object in $\mathbb{C}$, and $\mathtt{eq}_E \in \mathbb{C}(E,A)$, such that:*



$$\forall i,j,\ (f_i \circ \mathtt{eq}_E = f_j \circ \mathtt{eq}_E)$$



Universal property:

$$\forall i,j\ f_i \circ g = f_j \circ \mathtt{eq}_E \Rightarrow \exists! g^\dagger,\ g = \mathtt{eq}_E \circ g^\dagger$$

Observe that there is no guarantee on the existence of such equalizer. However, as soon as it exists, the universal property implies that the equalizer object is unique up to (unique) isomorphism. Melliès, Tabareau and Tasson's construction does not only ask for the existence of some equalizers, but also ask for those to *commute with the tensor product*, in the sense of Definition 10.1.8 below.

Definition 10.1.8 (Commutation of an equalizer with $\otimes$) *Let $\mathbb{C}$ be a monoidal category, A_1, A_2 two objects of $\mathbb{C}$, and $f_1, \dots, f_n$ a family of morphisms in $\mathbb{C}(A_1, A_2)$. We suppose that the equalizer of the $(f_i)_{1\leq i\leq n}$ exists, and we denote it by $(E, \mathtt{eq}_E)$. We say that this equalizer* commutes with the tensor product*, if for every object B in $\mathbb{C}$, $(E \otimes B, \mathtt{eq}_E \otimes \mathtt{id}_B)$ is an equalizer for the family of morphisms $(f_i \otimes \mathtt{id}_B : A_1 \otimes B \to A_2 \otimes B)_{1\leq i\leq n}$.*

Looking at Definition 10.1.8 we can see that asking for an equalizer $(E, \mathtt{eq})$ for the $(f_i)_{1\leq i\leq n}$ to additionally commutes with the tensor product actually boils down to require the following additional universal property:

$$\begin{array}{ccccc} E \otimes B & \xrightarrow{\mathtt{eq}_E \otimes \mathtt{id}_B} & A_1 \otimes B & \overset{\longrightarrow}{\underset{(f_i \otimes \mathtt{id}_B)_{i\in\mathbb{N}}}{\longrightarrow}} & A_2 \otimes B \\ \exists! g^\dagger \uparrow & \nearrow_{g} & & & \\ C & & & & \end{array} \tag{10.1}$$

Melliès, Tabareau and Tasson's construction asks for equalizers to exist for very specific families of morphisms: the symmetry morphisms over objects build as $(A \,\&\, \mathbf{1})^{\otimes n}$. We consider from now on $\mathbb{C}$ to be a SMC with finite products. Recall that the symmetry morphisms $\mathtt{s}^\sigma_A$ are the one formalized in Proposition 10.1.4. Once such equalizers are obtained, Melliès, Tabareau and Tasson use them to define the *exponential approximations to all orders*, that will be used in the next step of the construction to build the free exponential structure.

Definition 10.1.9 (Exponential Approximations) *Let $\mathbb{C}$ be a SMC with finite products, such that for every object A, and every $n \in \mathbb{N}$, the equalizer of the family of morphisms $\mathtt{s}^\sigma_{A\&\mathbf{1}} : (A \,\&\, \mathbf{1})^{\otimes n} \to (A \,\&\, \mathbf{1})^{\otimes n}$ exists. Then we denote by $(A^{\leq n}, \mathtt{eq}_{A^{\leq n}})$ this equalizer, and we say that the object $A^{\leq n}$ is the* exponential approximation of the object A at order n.

We sum up graphically below the requirements for the $A^{\leq n}$—they are inferred from the relevant instantiation of Definition 10.1.7.

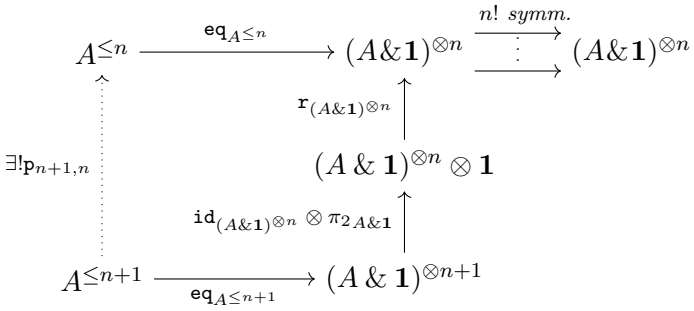
> Universal property of $(A^{\leq n}, \mathrm{eq})$: $\forall C, \forall f \in \mathbb{C}(C, (A \otimes 1)^{\otimes n})$ invariant under $\otimes$ symm., $\exists! f^\dagger \in \mathbb{C}(C, A^{\leq n})$ making the adjacent diagram commute.
>
> $$\begin{array}{ccccc} A^{\leq n} & \overset{\mathtt{eq}_{A^{\leq n}}}{\rightarrow} & (A\&\mathbf{1})^{\otimes n} & \overset{\rightarrow}{\underset{n!\text{ symm.}}{\rightarrow}} & (A\&\mathbf{1})^{\otimes n} \\ \exists! f^\dagger \uparrow & \nearrow_{f} & & & \\ C & & & & \end{array}$$

The exponential approximation $A^{\leq n}$ should be understood as follows: it contains the resources of type A that can be duplicated as most n times. If we go on with this analogy, we see that whenever a resource of type A may be duplicated at most $n+1$ times, we can discard the information about its $n+1$-th occurrence, and obtain a resource that may be duplicated at most n times. Melliès, Tabareau and Tasson formalize this in their categorical framework by looking at the canonical *projection morphisms* $A^{\leq n+1} \to A^{\leq n}$, that arise from the universal property for exponential approximations as described below.

Definition 10.1.10 *Let $\mathbb{C}$ be a SMC with products, and $n \in \mathbb{N}$ such that both $A^{\leq n}$ and $A^{\leq n+1}$ are defined. We define the* projection morphism at order n*, that we note $\mathtt{p}_{n+1,n}$ as the morphism in $\mathbb{C}(A^{\leq n+1}, A^{\leq n})$ obtained by applying the universal property of the equalizer $(A^{\leq n}, \mathtt{eq}_{A^{\leq n}})$ to:*

- *the object $C = A^{\leq n+1}$*
- *the morphism $f : C \to A^{\leq n}$ taken as $f = \mathtt{r}_{(A\&\mathbf{1})^{\otimes n}} \circ (\mathtt{id} \otimes \pi_2) \circ \mathtt{eq}_{A^{\leq n+1}}$, with π_2 denoting the right projection of $A \,\&\, \mathbf{1}$.*

We represent graphically the definition of $\mathtt{p}_{n+1,n}$ below:



10.1.3 The free Commutative Comonoid as limit of the exponential approximations.

To complete their construction, Melliès, Tabareau and Tasson assume the existence of the exponential approximations to all orders, and additionally ask for the existence of a *categorical limit* to the diagram formed by the projection morphisms between them. More precisely, the idea is to look at the diagram $\Delta^{\leq}$ below, that sums up the action of the projection morphisms on the $A^{\leq n}$ and to take $!_f A$ as the *limit cone*, when it exists, of this diagram $\Delta^{\leq}$—what it precisely meant for such a diagram to have a limit will be stated later in Definition 10.1.12:

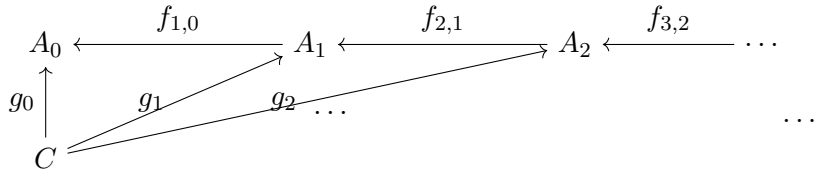
$$\Delta^{\leq} : \qquad A^{\leq 0} \xleftarrow{\mathtt{p}_{1,0}} A^{\leq 1} \xleftarrow{\mathtt{p}_{2,1}} A^{\leq 2} \xleftarrow{\mathtt{p}_{3,2}} A^{\leq 3} \xleftarrow{\mathtt{p}_{4,3}} \cdots$$

The next definitions deal with the formal presentation of limit cones for diagrams that are *chain-like*—i.e. that have the same shape as $\Delta^{\leq}$. There is in fact also a generic categorical way to define the limit object of an arbitrary diagram, that can be found for instance in [79], but for this work we need to consider only chain-like diagram. We actually split our definition in two parts: in Definition 10.1.11 below, we define *projective cones* for chain-like diagram, and then we take the limit cone as the *most general* projective cone, in a sense formalized by a universal property that we give in Definition 10.1.12.

Definition 10.1.11 (Δ-cone) *Let $\mathbb{C}$ be a category, and Δ a* chain-like diagram*, i.e. that consists of a family of objects $(A_i)_{i\in\mathbb{N}}$ and a family of morphisms $(f_{i+1,i} : A_{i+1} \to A_i)_{i\in\mathbb{N}}$:*

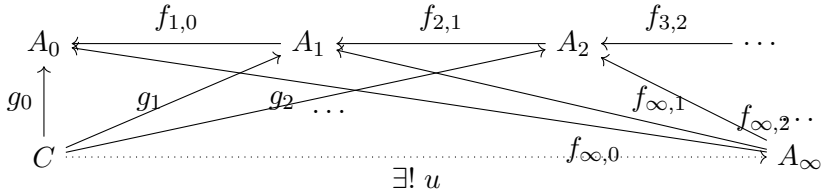
$$\Delta : \qquad A_1 \xleftarrow{f_{1,0}} A_2 \xleftarrow{f_{2,1}} A_3 \xleftarrow{f_{3,2}} A_4 \xleftarrow{f_{4,3}} \cdots$$

A projective cone based on Δ *is an object* C*, and a family of morphisms* $(g_i : C \to A_i)_{i\in\mathbb{N}}$ *such that* $\forall i \in \mathbb{N}$, $(g_i = f_{i+1,i} \circ g_{i+1})$ *i.e. the following diagram commute:*



A chain-like diagram Δ admits a limit when one of its projective cones is more general than all the others. We formalize this idea by a universal property in Definition 10.1.12 below.

Definition 10.1.12 *Let* $\mathbb{C}$ *be a category, and* $\Delta = ((A_i)_{i\in\mathbb{N}}, f_{i+1,i} : A_{i+1} \to A_i)$ *a chain-like diagram. A* limit cone for the diagram Δ *is a projective cone based on* Δ $(A_\infty, (f_{\infty,i} : A_\infty \to A_i))$ *such that: for every projective cone based on* Δ $(C, (g_i : C \to A_i)_{i\in\mathbb{N}})$ $\exists! u : C \to A_\infty$ *such that* $\forall i \in \mathbb{N}$, $(g_i = f_{\infty,i} \circ u)$*. We represent this requirement graphically below:*



We say that moreover the limit cone $(A_\infty, (f_{\infty,i} : A_\infty \to A_i)_{i\in\mathbb{N}})$ *of the diagram* Δ commutes with tensor products*, if the projective cone* $(A_\infty \otimes B, (f_{\infty,i} \otimes \mathtt{id}_B : A_\infty \otimes B \to A_i \otimes B)_{i\in\mathbb{N}})$ *is the limit cone of the diagram* $\Delta \otimes B$ *taken as:*

$$\Delta \otimes B : \qquad A_0 \otimes B \xleftarrow{f_{1,0} \otimes \mathtt{id}_B} A_1 \otimes B \xleftarrow{f_{2,1} \otimes \mathtt{id}_B} A_2 \otimes B \xleftarrow{f_{3,2} \otimes \mathtt{id}_B} \cdots$$

Observe that the universal property specified in Definition 10.1.12 above guarantees that as soon as the limit of Δ exists, it is unique—as usual, up to unique isomorphism. The result of Melliès, Tabareau and Tasson in [82] said that: assuming the existence of all the $A^{\leq i}$, and moreover of a limit cone for $\Delta^{\leq}$, as well as some additional requirements with respect to the commutation of these structures with $\otimes$, it holds that the limit cone of $\Delta^{\leq}$ is exactly the *free commutative comonoid*. It means that when we start from a category $\mathbb{C}$ which is already a model of exponential-free LL—i.e. $\star$-autonomous, see Chapter 9—we obtain this way a Lafont model—see Section 9.1.2—and consequently a sound model of LL.

We state formally this result in Proposition 10.1.5 below.

Proposition 10.1.5 (Melliès, Tabareau Tasson [82]) *Let* A *be an object of a symmetric monoidal category with finite products* $\mathbb{C}$*. The free commutative comonoid generated by* A *is the limit cone* $!_f A$ *of the diagram* $\Delta^{\leq}$*, provided that:*

1. $\forall n \in \mathbb{N}$*, the equalizer* $A^{\leq n}$ *exists and commutes with tensor products;*
2. *the limit cone* $(!_f A, (\mathtt{p}_{\infty,i} :!_f A \to A^{\leq i})_{i\in\mathbb{N}})$ *of the diagram* $\Delta^{\leq}$ *exists and commutes with tensor products.*

We sum up graphically condition 2 of Proposition 10.1.5, in Figure 10.4.

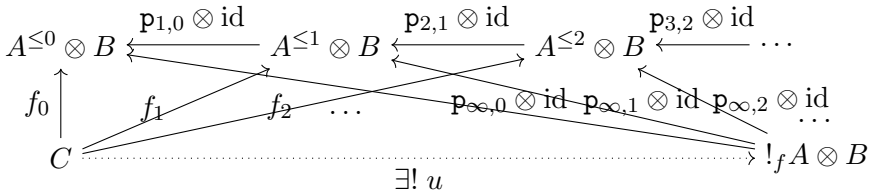


Figure 10.4: Diagram defining the commutation of the limit $!_f A$ with $\otimes$: for every objects B, C and family of morphisms $(f_i)_{i\in\mathrm{N}}$ as represented in the diagram, there exists a unique $u \in \mathbb{C}(C, !_f A \otimes B)$ making the diagram commute.

10.2 The free exponential structure in PCoh.

Our goal now is to apply Melliès, Tabareau and Tasson's construction in order to build the free commutative comonoid in **PCoh**. In that end, we need to check that **PCoh** verifies the conditions 1 and 2 of Proposition 10.1.5. We introduce now some notations to handle sequences and multiset over a PCS web, that we will need in the following technical development

Notation 10.2.1 *We denote by $\mathcal{M}_n(X)$ the subset of $\mathcal{M}_f(X)$ consisting of the finite multisets over X of length* exactly *n, and $\mathcal{M}_{\leq n}(X) = \bigcup_{k\leq n} \mathcal{M}_k(X)$ the set of those of length not greater than n. If $\vec{a}$ is a sequence of n elements in $|X|$, we denote by $\widetilde{\vec{a}}$ the underlying multiset in $\mathcal{M}_n(X)$. With this notation, if we have any multiset $\mu \in \mathcal{M}_f(X)$, we call* enumerations of μ *all sequences $\vec{a}$ such that their underlying multiset $\widetilde{\vec{a}}$ coincides with μ. We denote* $\mathrm{Enum}(\mu)$ *the set of all enumerations of μ. We call* multinomial coefficient of μ, *and denote $m(\mu)$ the number of the enumeration of μ, i.e. the cardinality of* $\mathrm{Enum}(\mu)$.

10.2.1 Construction of the equalizer of $n!$ symmetries.

The next lemmas and definitions deal with the proof that condition 1 indeed holds in the category **PCoh**; we will be able to state this result in Proposition 10.2.6. More precisely, starting from any PCS $\mathcal{A}$, we build here the equalizer of the symmetries over $\mathcal{A}^{\otimes n}$—that we will note $\mathcal{A}^{=n}$—and we show that it commutes with $\otimes$. From there, we will be able to build the exponential approximations, since by Definition 10.1.9, $\mathcal{A}^{\leq n} = (\mathcal{A} \,\&\, \mathbf{1})^{\otimes n}$.

We first discuss our construction informally: we want to build a PCS $\mathcal{A}^{=n}$ such that the diagram below commutes:

$$\mathcal{A}^{=n} \xrightarrow[\mathtt{eq}_{\mathcal{A}^{=n}}]{} \mathcal{A}^{\otimes n} \underset{\mathtt{s}^{\sigma,n}_{\mathcal{A}}}{\overset{\mathtt{s}^{\sigma,n}_{\mathcal{A}}}{\rightrightarrows}} \mathcal{A}^{\otimes n}$$

and we want to do so in the most general possible way, in the aim that $\mathcal{A}^{=n}$ also meets the requirements of the universal properties of Definition 10.1.7. Our approach is as follows: we take the cliques in $\mathrm{P}(\mathcal{A}^{\otimes n})$ as a starting point, and we seek to *erase* from them any information that is not invariant under symmetries. This erasure operation can be expressed internally in the category **PCoh**, by the morphism $\mathtt{j}^n_{\mathcal{A}} : \mathcal{A}^{\otimes n} \to \mathcal{A}^{\otimes n}$ formed as the barycentric sum of the actions of $\mathfrak{S}_n$ over $\mathcal{A}$. Formally, for $n \in \mathbb{N}$, and $\mathcal{A}$ a PCS, we define $\mathtt{j}^n_{\mathcal{A}} \in \mathbf{PCoh}(\mathcal{A}^{\otimes n}, \mathcal{A}^{\otimes n})$ as:

$$\mathtt{j}^n_{\mathcal{A}} := \frac{1}{n!} \sum_{\sigma\in\mathfrak{S}_n} \mathtt{s}^{\sigma,n}_{\mathcal{A}}.$$

Observe that this construction is made possible by the convex structure on **PCoh**-morphisms highlighted in Proposition 9.2.2. Looking at the concrete description —done in Example 10.1.7— of the symmetry morphisms $\mathbf{s}^{\sigma,n}_{\mathcal{A}}$ in **PCoh**, we also give in Lemma 10.2.2 below a more direct characterization of this morphism, as a matrix in $\mathbb{R}_+^{|\mathcal{A}^{\otimes n}|\times|\mathcal{A}^{\otimes n}|}$.

Lemma 10.2.2 *Let $\mathcal{A}$ be a PCS, and $n \in \mathbb{N}$. Then for every $\vec{a}, \vec{a}' \in |\mathcal{A}^{\otimes n}|$:*

$$(\mathtt{j}^n_{\mathcal{A}})_{\vec{a},\vec{a}'} = \delta_{\widetilde{\vec{a}},\widetilde{\vec{a}'}} \frac{1}{m(\widetilde{\vec{a}})},$$

where $\widetilde{\vec{a}}$ and $m(\widetilde{\vec{a}})$ have meaning as specified in Notation 10.2.1.

Proof. To see why Lemma 10.2.2 holds, it is enough to see that:

$$(\mathtt{j}^n_{\mathcal{A}})_{\vec{a},\vec{a}'} = \frac{\#\left\{\sigma \in \mathfrak{S}_n \;;\; \forall i \leq n, \vec{a}_{\sigma(i)} = \vec{a}'_i\right\}}{n!} = \delta_{\widetilde{\vec{a}},\widetilde{\vec{a}'}} \frac{\prod_{a \in \mathtt{S}(\widetilde{\vec{a}})} (\widetilde{\vec{a}})(a)!}{n!}.$$

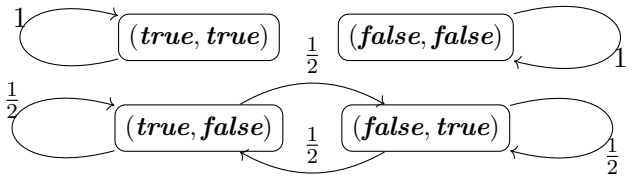
□

We can think about this endomorphism as a *jammer*, in the sense that it chooses randomly a permutation, and then applies it to its input. We illustrate this point in Example 10.2.1 below, where we consider pairs of booleans.

Example 10.2.1 (Jammer morphism on $\mathbf{Bool}^{\otimes 2}$) *There are only two elements in the permutation group $\mathfrak{S}_2$: the identity and the exchange map. As a consequence, we see that:*

$$\mathtt{j}^2_{\boldsymbol{Bool}} = \frac{1}{2}\mathtt{id}_{\boldsymbol{Bool}\otimes\boldsymbol{Bool}} + \frac{1}{2}\mathtt{swap}_{\boldsymbol{Bool},\boldsymbol{Bool}} : \boldsymbol{Bool}^{\otimes 2} \to \boldsymbol{Bool}^{\otimes 2}$$

Thus $\mathtt{j}^2_{\boldsymbol{Bool}}$ is the matrix in $\mathbb{R}_+^{\{\mathtt{t},\mathtt{f}\}\times\{\mathtt{t},\mathtt{f}\}}$ defined as: $(\mathtt{j}^2_{\boldsymbol{Bool}})_{(a_1,a_2),(a'_1,a'_2)} = \frac{1}{2}\delta_{a_1,a'_1}\delta_{a_2,a'_2} + \frac{1}{2}\delta_{a_1,a'_2}\delta_{a_2,a'_1}$, i.e. represents the following Markov chain:



Observe that as desired, the morphism $\mathtt{j}^2_{\boldsymbol{Bool}}$ erases *asymmetric information: when we apply the morphism $\mathtt{j}^2_{\boldsymbol{Bool}}$ to some arbitrary distribution $\mathscr{D}$ over pairs of booleans, the probabilities $\mathscr{D}(\boldsymbol{true},\boldsymbol{false})$ and $\mathscr{D}(\boldsymbol{false},\boldsymbol{true})$ are* uniformly reallocated *between these two states. We illustrate this by looking for instance at $u = \frac{1}{2}\cdot e_{(\mathtt{t},\mathtt{t})} + \frac{1}{2}\cdot e_{(\mathtt{t},\mathtt{f})}$ in $\mathrm{P}(\boldsymbol{Bool}\otimes\boldsymbol{Bool})$. Then we see that indeed $(\mathtt{j}^2_{\boldsymbol{Bool}}u) \in \mathrm{P}(\boldsymbol{Bool}\otimes\boldsymbol{Bool})$ is as follows:*

$$\mathtt{j}^2_{\boldsymbol{Bool}}u = \frac{1}{2}\cdot e_{(\mathtt{t},\mathtt{t})} + \frac{1}{4}\cdot e_{(\mathtt{t},\mathtt{f})} + \frac{1}{4}\cdot e_{(\mathtt{f},\mathtt{t})}.$$

The observation done in the example above holds more generally: indeed for any PCS $\mathcal{A}$, $\mathtt{j}^n_{\mathcal{A}}$ acts on $\mathcal{A}^{\otimes n}$ by reallocating the coefficients in such a way as to erase the information about the orders of the elements in a n-length sequence. Accordingly the morphism $\mathtt{j}^n_{\mathcal{A}}$ is *invariant by all symmetries*, in the following sense: for every $\sigma \in \mathfrak{S}_n$, it holds that

$\mathtt{s}^{\sigma,n}_{\mathcal{A}} \circ \mathtt{j}^n_{\mathcal{A}} = \mathtt{j}^n_{\mathcal{A}}$—we prove this by looking at the definition of $\mathtt{j}^n_{\mathcal{A}}$, since $\tau \in \mathfrak{S}_n \mapsto \sigma \circ \tau \in \mathfrak{S}_n$ is a bijection over $\mathfrak{S}_n$.

It is tempting, as a consequence, to choose as cliques for $\mathcal{A}^{=n}$ the set $\mathtt{j}^n_{\mathcal{A}}(\mathrm{P}(\mathcal{A}^{\otimes n})) := \{\mathtt{j}^n_{\mathcal{A}}\, x \mid x \in \mathrm{P}(\mathcal{A}^{\otimes n})\}$. We have however to be careful, since $(|\mathcal{A}^{\otimes n}|, \mathtt{j}^n_{\mathcal{A}}(\mathrm{P}(\mathcal{A}^{\otimes n})))$ is *not* a PCS: indeed it is not even downward closed for the pre-PCS order—see Section 9.2 from Chapter 9—as we illustrate on the example below,.

Example 10.2.2 *We look at the same vector $u \in \boldsymbol{Bool}^{\otimes 2}$ as the one we defined in Example 10.2.1. We see that for instance $\frac{1}{4} \cdot e_{(\mathtt{t},\mathtt{f})} \leq_{|\mathcal{A}^{\otimes n}|} \mathtt{j}^2_{\boldsymbol{Bool}} u$, but we can see that $\frac{1}{4} \cdot e_{(\mathtt{t},\mathtt{f})} \notin \mathtt{j}^2_{\boldsymbol{Bool}}(\mathrm{P}(\boldsymbol{Bool}^{\otimes 2}))$, since indeed this vector is not invariant under symmetry morphisms—for instance, it is not invariant by the symmetry morphism that swaps the states $(\mathtt{t},\mathtt{f})$ and $(\mathtt{f},\mathtt{t})$.*

To overcome this problem, we choose to build a PCS $(|\mathcal{A}^{=n}|, \mathrm{P}(\mathcal{A}^{=n}))$ whose set of cliques $\mathrm{P}(\mathcal{A}^{=n})$ is isomorphic to $\mathtt{j}^n_{\mathcal{A}}(\mathrm{P}(\mathcal{A}^{\otimes n}))$, through a *change of basis*: we see by Lemma 10.2.2 that every vector of the form $(\mathtt{j}^n_{\mathcal{A}}\, x)$ may be expressed as:

$$\mathtt{j}^n_{\mathcal{A}}\, x = \sum_{\mu \in \mathcal{M}_n(|\mathcal{A}|)} \left(\frac{1}{m(\mu)} \cdot \sum_{\vec{a} \in \mathrm{Enum}(\mu)} x_{\vec{a}} \right) \cdot e'_\mu,$$

where for any multiset $\mu \in \mathcal{M}_n(|\mathcal{A}|)$ we have taken $e'_\mu = \sum_{\vec{a} \in \mathrm{Enum}(\mu)} e_{\vec{a}}$. It means that $\mathcal{M}_n(|\mathcal{A}|)$ can be seen as the *canonical base* of the image set $\mathtt{j}^n_{\mathcal{A}}(\mathrm{P}(\mathcal{A}^{\otimes n}))$.To make this change of basis formal, we construct now a matrix $\mathtt{jm}^n_{\mathcal{A}} \in \mathbb{R}_+^{|\mathcal{A}^{\otimes n}| \times \mathcal{M}_n(|\mathcal{A}|)}$, such that:

$$\forall x \in \mathrm{P}\left(\mathcal{A}^{\otimes n}\right),\ \mathtt{j}^n_{\mathcal{A}}\, x = \sum_{\mu \in \mathcal{M}_n(|\mathcal{A}|)} (\mathtt{jm}^n_{\mathcal{A}}\, x)_\mu \cdot e'_\mu$$

Definition 10.2.1 *Let $\mathcal{A}$ be a PCS, and $n \in \mathbb{N}$. We define the* modified jammer *matrix $\mathtt{jm}^n_{\mathcal{A}} \in \mathbb{R}_+^{|\mathcal{A}^{\otimes n}| \times \mathcal{M}_n(|\mathcal{A}|)}$ as:*

$$(\mathtt{jm}^n_{\mathcal{A}})_{\vec{a},\mu} := \delta_{\mu, \underset{\sim}{\vec{a}}} \cdot \frac{1}{m(\mu)}, \qquad \text{for } \mu \in \mathcal{M}_n(|\mathcal{A}|),\ \vec{a} \in \left|\mathcal{A}^{\otimes n}\right|$$

Observe that $(\mathtt{jm}^n_{\mathcal{A}})_{\vec{a},\mu} = (\mathtt{j}^n_{\mathcal{A}})_{\vec{a},\vec{a}'}$ for $\vec{a}'$ any enumeration of μ. Essentially $\mathtt{j}^n_{\mathcal{A}}\, x \in \mathbb{R}_+^{|\mathcal{A}^{\otimes n}|}$ and $\mathtt{jm}^n_{\mathcal{A}}\, x \in \mathbb{R}_+^{\mathcal{M}_n(|\mathcal{A}|)}$ are the same vector, but expressed in a different canonical base. It is relevant, because the canonical base generated by multisets lead to a presentation of $\mathtt{jm}^n_{\mathcal{A}}(\mathcal{A}^{\otimes n})$ as the cliques of a PCS—that we make explicit below—while it was not the case with the base generated by sequences.

Definition 10.2.2 *Let $\mathcal{A}$ be a PCS, and $n \in \mathbb{N}$. We define the PCS $\mathcal{A}^{=n}$ by:*

$$|\mathcal{A}^{=n}| = \mathcal{M}_n(|\mathcal{A}|) \qquad \mathrm{P}(\mathcal{A}^{=n}) = \{\mathtt{jm}^n_{\mathcal{A}}\, u \mid u \in \mathrm{P}\left(\mathcal{A}^{\otimes n}\right)\}.$$

We see that the space $\mathcal{A}^{=n}$ is indeed a PCS by using Proposition 9.2.2. Our goal now is to check that it is indeed the equalizer of $n!$ symmetry. First, we define a morphism $\mathtt{eq}_{\mathcal{A}^{=n}} : \mathcal{A}^{=n} \to \mathcal{A}^{\otimes n}$, designed to meet the requirements of Definition 10.1.7. We define it as the matrix $(\mathtt{eq}_{\mathcal{A}^{=n}})_{\mu,\vec{a}} := \delta_{\mu, \underset{\sim}{\vec{a}}}$. Observe that it essentially *reverses* the change of basis, since indeed:

$$(\mathtt{eq}_{\mathcal{A}^{=n}} \circ \mathtt{jm}^n_{\mathcal{A}})\, x = \mathtt{j}^n_{\mathcal{A}}\, x\,. \tag{10.2}$$

We need first to check that $\mathtt{eq}_{\mathcal{A}^{=n}}$ is indeed a morphism in $\mathbf{PCoh}(\mathcal{A}^{=n}, \mathcal{A}^{\otimes n})$. It comes as a consequence of Equation 10.2, as can be seen in the proof of Lemma 10.2.3 below:

Lemma 10.2.3 *Let $\mathcal{A}$ be a PCS, and $n \in \mathbb{N}$. Then $\mathtt{eq}_{\mathcal{A}^{=n}}$ is a morphism in $\mathbf{PCoh}(\mathcal{A}^{=n}, \mathcal{A}^{\otimes n})$.*

Proof. We need to show that for every $x \in \mathrm{P}(\mathcal{A}^{=n})$, it holds that $(\mathtt{eq}_{\mathcal{A}^{=n}})\, x \in \mathrm{P}(\mathcal{A}^{\otimes n})$. Looking at the definition of the PCS $\mathcal{A}^{=n}$, we see that our goal can be reformulated as:

$$\forall y \in \mathrm{P}\left(\mathcal{A}^{\otimes n}\right),\ (\mathtt{eq}_{\mathcal{A}^{=n}} \circ \mathtt{jm}^n_{\mathcal{A}})\, y \in \mathrm{P}\left(\mathcal{A}^{\otimes n}\right) \tag{10.3}$$

From Equation (10.2), we can go one step further: (10.3) can actually be deduced as soon as we know: $\forall y \in \mathrm{P}(\mathcal{A}^{\otimes n}),\ (\mathtt{j}^n_{\mathcal{A}})\, y \in \mathrm{P}(\mathcal{A}^{\otimes n})$. Since $\mathtt{j}^n_{\mathcal{A}}$ is a morphism in $\mathbf{PCoh}(\mathcal{A}^{\otimes n}, \mathcal{A}^{\otimes n})$, we see that the latter claim holds, which ends the proof. □

Using equation (10.2) we also see that every vector of the form $\mathtt{eq}_{\mathcal{A}^{=n}}\, x$ is invariant by all symmetry morphisms, as stated in Lemma 10.2.4 below.

Lemma 10.2.4 *Let $\mathcal{A}$ be a PCS, and $n \in \mathbb{N}$. Then for every $\sigma \in \mathfrak{S}_n$, it holds that $(\mathtt{eq}_{\mathcal{A}^{=n}} \circ \mathtt{s}^{\sigma,n}_{\mathcal{A}} = \mathtt{eq}_{\mathcal{A}^{=n}})$, i.e.:*

$$\mathcal{A}^{=n} \xrightarrow[\mathtt{eq}_{\mathcal{A}^{=n}}]{} \mathcal{A}^{\otimes n} \overset{\mathtt{s}^{\sigma,n}_{\mathcal{A}}}{\underset{\cdots}{\rightrightarrows}} \mathcal{A}^{\otimes n}$$

Proof. We have to show that for every $\sigma \in \mathfrak{S}_n$, and $x \in \mathrm{P}(\mathcal{A}^{=n})$, it holds that $(\mathtt{eq}_{\mathcal{A}^{=n}} \circ \mathtt{s}^{\sigma,n}_{\mathcal{A}})\, x = \mathtt{eq}_{\mathcal{A}^{=n}}\, x$. Let $\sigma \in \mathfrak{S}_n$, and $x \in \mathrm{P}(\mathcal{A}^{=n})$. Let $y \in \mathcal{A}^{\otimes y}$, such that $x = \mathtt{s}_{\mathtt{n}}\, y$; from Equation 10.2, we see that: $\mathtt{eq}_{\mathcal{A}^{=n}}\, x = \mathtt{j}^n_{\mathcal{A}}\, y$. Recall that $\mathtt{j}^n_{\mathcal{A}}$ is invariant by symmetry, it means that $(\mathtt{j}^n_{\mathcal{A}} \circ \mathtt{s}^{\sigma,n}_{\mathcal{A}})\, y = \mathtt{s}^{\sigma,n}_{\mathcal{A}}\, y$, and it ends the proof. □

Our goal now is to check that the universal property for equalizers—as stated in Definition 10.1.7—holds for $(\mathcal{A}^{=n}, \mathtt{eq}_{\mathcal{A}^{=n}})$. Toward this goal, we first show in Lemma 10.2.5 below, an alternative characterization of $\mathcal{A}^{=n}$, focusing on a set of *generators* that can be computed directly from $\mathcal{A}$.

Lemma 10.2.5 *For every PCS $\mathcal{A}$, and $n \in \mathbb{N}$, $\mathrm{P}(\mathcal{A}^{=n}) = \{\mathtt{jm}^n_{\mathcal{A}}(\otimes^n_{i=1} u_i)\ ;\ u_i \in \mathrm{P}(\mathcal{A})\}^{\perp\perp}$.*

Proof. We denote by $(\mathcal{A}^{=n})^\star$ the pre-PCS defined as $|(\mathcal{A}^{=n})^\star| = \mathcal{M}_n(|\mathcal{A}|)$, and $\mathrm{P}((\mathcal{A}^{=n})^\star) = \{\mathtt{jm}^n_{\mathcal{A}}(\otimes^n_{i=1} u_i)\ ;\ u_i \in \mathrm{P}(\mathcal{A})\}^{\perp\perp}$. First, we can observe that $(\mathcal{A}^{=n})^\star$ is a PCS, since it is generated by bi-duality, and that the boundedness conditions on coefficients also hold. The equivalence between $(\mathcal{A}^{=n})^\star$ and $\mathcal{A}^{=n}$ is then obtained by Lemma 9.2.3 from Chapter 9, taking $\mathcal{A} = \mathcal{A}^{\otimes n}$, $G = \{\otimes^n_{i=1} u_i\ ;\ u_i \in \mathrm{P}(\mathcal{A})\}$, $\mathcal{B} = \mathcal{A}^{=n}$ and $f = \mathtt{jm}^n_{\mathcal{A}}$. Then $(\mathcal{A}^{=n})^\star$ is equal to $f(G)^{\perp\perp} = f(\mathrm{P}(\mathcal{A}^{\otimes n}))^{\perp\perp}$ which turns out to be the bipolar of $\mathcal{B}$ by Lemma 9.2.3. We conclude since we have already shown $\mathcal{A}^{=n}$ to be a PCS, and as thus $\mathcal{A}^{=n} = (\mathcal{A}^{=n})^{\perp\perp}$. □

We are now ready to check that $(\mathcal{A}^{=n}, \mathtt{eq}_{\mathcal{A}^{=n}})$ is indeed the equalizer of the $n!$ symmetry over $\mathcal{A}^{\otimes n}$. Recall that it is the first step in order to apply Melliès, Tabareau and Tasson recipe to **PCoh**, as presented in Proposition 10.1.5.

Proposition 10.2.6 *Let $\mathcal{A}$ be a PCS and $n \in \mathrm{N}$. The object $\mathcal{A}^{=n}$ together with the morphism $\mathtt{eq}_{\mathcal{A}^{=n}} \in \mathbf{PCoh}(\mathcal{A}^{=n}, \mathcal{A}^{\otimes n})$, is the equalizer of the $n!$ symmetries of the n-fold tensor $\mathcal{A}^{\otimes n}$. Moreover, these equalizers commute with the tensor product, in the sense of Definition 10.1.8.*

Proof. We only prove the commutation with $\otimes$, as the universal property of the equalizer is a direct consequence of the commutation, taking $\mathcal{B} = \mathbf{1}$. Let $\mathcal{C}$ be a PCS and $f \in \mathbf{PCoh}(\mathcal{C}, \mathcal{A}^{\otimes n} \otimes \mathcal{B})$ be a morphism such that $(\mathtt{s}_{\mathcal{A}}^{\sigma,n} \otimes \mathtt{id}_{\mathcal{B}}) \circ f = f$ for any $\sigma \in \mathfrak{S}_n$. Then, define $f^\dagger \in \mathbf{PCoh}(\mathcal{C}, \mathcal{A}^n \otimes \mathcal{B})$ as follows: for every $c \in |\mathcal{C}|$, $b \in |\mathcal{B}|$, $\mu \in |\mathcal{A}^{=n}|$, $f^\dagger_{c,(\mu,b)} = f_{c,(\vec{a},b)}$, where $\vec{a}$ is any enumeration of μ (no matter which one because f is invariant under the tensor symmetries). The fact that $f^\dagger$ is the unique morphism commuting the diagram of Equation (10.1) is a trivial calculation. We have then to prove that it is indeed a morphism in $\mathbf{PCoh}(\mathcal{C}, \mathcal{A}^{=n} \otimes \mathcal{B})$, that means that, for any $v \in \mathrm{P}(\mathcal{C})$, $f^\dagger v \in \mathrm{P}(\mathcal{A}^{=n} \otimes \mathcal{B})$.

Consider $w \in \mathrm{P}(\mathcal{A}^{=n} \otimes \mathcal{B})^\perp$ and let us prove that $\langle f^\dagger v, w\rangle \leq 1$. This will allow us to conclude $f^\dagger v \in \mathrm{P}(\mathcal{A}^{=n} \otimes \mathcal{B})$. Our proof strategy is as follows: we build a vector $\overline{w}$ derived from w, such that:

$$\overline{w} \in \mathrm{P}\left(\mathcal{A}^{\otimes n} \otimes \mathcal{B}\right)^\perp \tag{10.4}$$

$$\langle f^\dagger v, w\rangle = \langle f v, \overline{w}\rangle \tag{10.5}$$

Define $\overline{w} \in \mathbb{R}_+^{\left|\mathcal{A}^{\otimes n} \otimes \mathcal{B}\right|}$ as $\overline{w}_{(\vec{a},b)} = \frac{1}{m(\vec{a})} w_{\sim_{\vec{a}},b}$. We first show that Equation (10.4) holds. In fact, for any $u \in \mathrm{P}(\mathcal{A}^{\otimes n})$, $z \in \mathrm{P}(\mathcal{B})$:

$$\begin{aligned}\langle u \otimes z, \overline{w}\rangle &= \sum_{\vec{a},b} u_{\vec{a}} z_b \overline{w}_{(\vec{a},b)} = \sum_{\vec{a},b} u_{\vec{a}} z_b \frac{1}{m(\vec{a})} w_{\sim_{\vec{a}},b} = \sum_{\mu,b} z_b w_{\mu,b} \sum_{\vec{a} \in \mathrm{Enum}(\mu)} \frac{1}{m(\mu)} u_{\vec{a}} \\ &= \sum_{\mu,b} z_b w_{\mu,b} (\mathtt{jm}^n_{\mathcal{A}}\, u)_\mu = \langle (\mathtt{jm}^n_{\mathcal{A}}\, u) \otimes z, w\rangle\ .\end{aligned}$$

From there, we can use the fact that by definition of $\mathcal{A}^{=n}$, $(\mathtt{jm}^n_{\mathcal{A}}\, u) \in \mathrm{P}(\mathcal{A}^{=n})$. Since $z \in \mathrm{P}(\mathcal{B})$, it means that $(\mathtt{jm}^n_{\mathcal{A}}\, u) \otimes z \in \mathrm{P}(\mathcal{A}^{\otimes n} \otimes \mathcal{B})$. Recall moreover that we have taken $w \in \mathrm{P}(\mathcal{A}^{=n} \otimes \mathcal{B})^\perp$; it allows us to conclude that $\langle u \otimes z, \overline{w}\rangle \leq 1$. Since the elements $u \otimes z$, with $u \in \mathrm{P}(\mathcal{A}^{\otimes n})$, $z \in \mathrm{P}(\mathcal{B})$ are generators of $\mathcal{A}^{\otimes n} \otimes \mathcal{B}$ —see the definition of the bifunctor $\otimes$ in **PCoh** stated in Chapter 9—it means that indeed $\overline{w} \in \mathrm{P}\left((\mathcal{A}^{\otimes n} \otimes \mathcal{B})^\perp\right)$.

We show now Equation 10.5: $\langle f^\dagger v, w\rangle = \langle f v, \overline{w}\rangle$. In fact,

$$\langle f^\dagger v, w\rangle = \sum_{\mu,b} (f^\dagger v)_{\mu,b} w_{\mu,b} = \sum_{\vec{a},b} \frac{(f v)_{\vec{a},b} w_{\sim_{\vec{a}},b}}{m(\vec{a})} = \sum_{\vec{a},b} (f v)_{\vec{a},b} \overline{w}_{\vec{a},b} = \langle f v, \overline{w}\rangle\ .$$

Now from Equations (10.4) and (10.5) we can obtain the result, because by hypothesis $f v \in \mathrm{P}(\mathcal{A}^{\otimes n} \otimes \mathcal{B})$, and thus $\langle f^\dagger v, w\rangle = \langle f v, \overline{w}\rangle \leq 1$. □

Remark 10.2.1 *We have tried to present above a way to understand where our construction for $\mathcal{A}^{=n}$ comes from, by starting from the jammer morphism $\mathtt{j}^n_{\mathcal{A}}$. It should be reminded, however, that the notion of equalizer is only defined* up to (unique) isomorphism, *which means that the choice of the matrix $\mathtt{jm}^n_{\mathcal{A}}$ is somewhat arbitrary. Our construction has the advantage to make the equalizer morphism $\mathtt{eq}_{\mathcal{A}^{=n}}$ as simple as possible, i.e. a web morphism, in the sense defined by Danos and Ehrhard: a morphism entirely specified by a function between the PCSs webs.*

Example 10.2.3 *Let us illustrate the construction of $\mathcal{A}^{=n}$ in the case where $\mathcal{A}$ is the boolean space **Bool**. It is trivial to check that **Bool**$^{=0}$ is isomorphic to $\mathbf{1}$ and **Bool**$^{=1}$ to **Bool**. Concerning **Bool**$^{=2}$, we have:*

$$\left|\textbf{\textit{Bool}}^{=2}\right| = \{[\mathtt{t},\mathtt{f}], [\mathtt{t},\mathtt{t}], [\mathtt{f},\mathtt{f}]\}.$$

By definition, $\mathrm{P}\left(\boldsymbol{Bool}^{=2}\right) = \{\mathtt{jm}^2_{\boldsymbol{Bool}}\, u \mid u \in \boldsymbol{Bool}^{\otimes 2}\}$. *Looking now at the definition of* $\mathtt{jm}^2_{\boldsymbol{Bool}}$ *we see that the action of* $\mathtt{jm}^2_{\boldsymbol{Bool}}$ *on any clique* $u \in \mathrm{P}\left(\boldsymbol{Bool}^{\otimes 2}\right)$ *is as follows:*

$$\mathtt{jm}^2_{\boldsymbol{Bool}}\, u = u_{(\mathtt{t},\mathtt{t})} e_{[\mathtt{t},\mathtt{t}]} + u_{(\mathtt{f},\mathtt{f})} e_{[\mathtt{f},\mathtt{f}]} + (\frac{1}{2} \cdot u_{(\mathtt{t},\mathtt{f})} + \frac{1}{2} \cdot u_{(\mathtt{f},\mathtt{t})}) \cdot e_{[\mathtt{t},\mathtt{f}]}.$$

We want now to obtain a more direct characterization—i.e. one that does not use $\mathrm{P}\left(\boldsymbol{Bool}\right)$ *explicitly. Recall that Example 9.2.4 computes* $\mathrm{P}\left(\boldsymbol{Bool}^{\otimes 2}\right)$ *as the set of sub-probability distributions over boolean pairs. As a consequence every element of the form* $(\mathtt{jm}^2_{\boldsymbol{Bool}}\, u)$ *must verify the following constraint:*

$$2 \cdot (\mathtt{jm}^2_{\boldsymbol{Bool}}\, u)_{[\mathtt{t},\mathtt{f}]} + (\mathtt{jm}^2_{\boldsymbol{Bool}}\, u)_{[\mathtt{t},\mathtt{t}]} + (\mathtt{jm}^2_{\boldsymbol{Bool}}\, u)_{[\mathtt{f},\mathtt{f}]} = u_{(\mathtt{t},\mathtt{f})} + u_{(\mathtt{f},\mathtt{t})} + u_{(\mathtt{t},\mathtt{t})} + u_{(\mathtt{f},\mathtt{f})} \leq 1$$

This constraint means that $\mathrm{P}\left(\boldsymbol{Bool}^{=2}\right) \subseteq X$, *where* X *is the vector set defined as* $X := \{w \in \mathbb{R}_+^{\mathcal{M}_f(2)|\boldsymbol{Bool}|} \;;\; w_{[\mathtt{t},\mathtt{f}]} + w_{[\mathtt{f},\mathtt{f}]} + 2w_{[\mathtt{t},\mathtt{f}]} \leq 1\}$. *We want now to show that the reverse inclusion also holds. Recall that since* $\boldsymbol{Bool}^{=2}$ *is a PCS, it coincides with its bi-dual, and consequently for every* $x \in \mathbb{R}_+^{\mathcal{M}_2(|\boldsymbol{Bool}|)}$*:*

$$\left(\forall y \in \mathrm{P}\left((\boldsymbol{Bool}^{=2})^{\perp}\right), \langle x, y\rangle \leq 1\right) \quad \Rightarrow \quad x \in \mathrm{P}\left(\boldsymbol{Bool}^{=2}\right) \tag{10.6}$$

So let $x \in X$, *and* $y \in \mathrm{P}\left((\boldsymbol{Bool}^{=2})^{\perp}\right)$. *To show that* $\langle x, y\rangle \leq 1$, *we first enforce upper bounds on the coefficients of* y. *Since* y *has to be dual with every element of the form* $\mathtt{jm}^2_{\boldsymbol{Bool}}(u_1 \otimes u_2)$ *with* $u_i \in \mathrm{P}\left(\boldsymbol{Bool}\right)$, *we obtain in particular the following inequalities:*

$$1 \geq \langle y, \mathtt{jm}^2_{\boldsymbol{Bool}}(e_{\mathtt{t}} \otimes e_{\mathtt{t}})\rangle = y_{[\mathtt{t},\mathtt{t}]} \qquad 1 \geq \langle y, \mathtt{jm}^2_{\boldsymbol{Bool}}(e_{\mathtt{f}} \otimes e_{\mathtt{f}})\rangle = y_{[\mathtt{f},\mathtt{f}]}$$

$$1 \geq \langle y, \mathtt{jm}^2_{\boldsymbol{Bool}}(e_{\mathtt{t}} \otimes e_{\mathtt{f}})\rangle = \frac{1}{2} \cdot y_{[\mathtt{t},\mathtt{f}]}.$$

From those inequalities we are now able to show that:

$$\begin{aligned}\langle x, y\rangle &= x_{[\mathtt{t},\mathtt{t}]} \cdot y_{[\mathtt{t},\mathtt{t}]} + x_{[\mathtt{f},\mathtt{f}]} \cdot y_{[\mathtt{f},\mathtt{f}]} + x_{[\mathtt{t},\mathtt{f}]} \cdot y_{[\mathtt{t},\mathtt{f}]} \\ &\leq x_{[\mathtt{t},\mathtt{t}]} + x_{[\mathtt{f},\mathtt{f}]} + 2 \cdot x_{[\mathtt{t},\mathtt{f}]} \cdot y_{[\mathtt{t},\mathtt{f}]} \\ &\leq 1 \qquad \textit{since } x \in X.\end{aligned}$$

And using Equation 10.6 we can now conclude that $\mathrm{P}\left(\boldsymbol{Bool}^{=2}\right) = \{w \in \mathbb{R}_+^{\mathcal{M}_2(|\boldsymbol{Bool}|)} \;;\; w_{[\mathtt{t},\mathtt{f}]} + w_{[\mathtt{f},\mathtt{f}]} + 2w_{[\mathtt{t},\mathtt{f}]} \leq 1\}$.

10.2.2 The approximations $\mathcal{A}^{\leq n}$

The existence of equalizers for the symmetry morphisms over $\mathcal{A}^{=n}$ for any PCS $\mathcal{A}$ —that we established in Proposition 10.2.6 —implies that we can now define in **PCoh** the *exponential approximations*, that consist of the family of PCSs $\mathcal{A}^{\leq n} := (\mathcal{A} \,\&\, \mathbf{1})^{=n}$. Here, we look at those PCSs more extensively, thus obtaining a direct characterization of $\mathcal{A}^{\leq n}$ by means of an operator $\langle u_1, \dots u_n\rangle$ over the vectors in $\mathrm{P}\left(\mathcal{A}\right)$, crucial for the next step. This characterization is presented in Lemma 10.2.7 below, where we use the following notation: $\mathcal{M}_{\leq n}(X)$ is the set of all multisets over X of length smaller or equal to n, and $f : X \hookrightarrow Y$ to denote an *injective* function from X to Y.

Lemma 10.2.7 (Caracterisation of the $\mathcal{A}^{\leq n}$) *The PCS* $\mathcal{A}^{\leq n}$ *can be presented as follows:*

$$\left|\mathcal{A}^{\leq n}\right| = \mathcal{M}_{\leq n}(|\mathcal{A}|), \qquad \mathrm{P}\left(\mathcal{A}^{\leq n}\right) = \{\langle u_1, \dots, u_n\rangle \;;\; \forall i \leq n, u_i \in \mathrm{P}\left(\mathcal{A}\right)\}^{\perp\perp}$$

where, for any $[a_1, \dots, a_k] \in \left|\mathcal{A}^{\leq n}\right|$,

$$\langle u_1, \dots, u_n \rangle_{[a_1,\dots,a_k]} = \frac{1}{n!} \sum_{\sigma \in \mathfrak{S}_n} \prod_{i=1}^{k} (u_{\sigma(i)})_{a_i} \tag{10.7}$$

$$= \frac{(n-k)!}{n!} \sum_{f:\{1,\dots,k\} \hookrightarrow \{1,\dots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i}. \tag{10.8}$$

Proof. In the proof, we note $\mathcal{A}^{\leq n}$ for the new presentation proposed in the statement, and $(\mathcal{A} \,\&\, \mathbf{1})^{=n}$ for the one coming from the definition of symmetries equalizers in Definition 10.2.2. First, notice that there is a bijection between $\left|\mathcal{A}^{\leq n}\right|$ and $\mathcal{M}_n(|\mathcal{A}| \uplus \{\star\}) = |(\mathcal{A} \,\&\, \mathbf{1})^{=n}|$ obtained just by adding the necessary number of $\star$'s to a multiset in $\left|\mathcal{A}^{\leq n}\right|$. Second, the definitions of $\mathrm{P}\left(\mathcal{A}^{\leq n}\right)$ and of $\langle u_1, \dots, u_n \rangle$ follow by remarking that the latter is a notation for $\mathtt{jm}^n_{\mathcal{A}}\left((\langle u_1, e_\star \rangle) \otimes \cdots \otimes (\langle u_n, e_\star \rangle)\right)$, with $\mathtt{jm}^n_{\mathcal{A}}$ the morphism $(\mathcal{A} \,\&\, \mathbf{1})^{\otimes n} \to (\mathcal{A} \,\&\, \mathbf{1})^{\leq n}$ defined in Definition 10.2.1, and $\langle u, v \rangle$ the cartesian product between vectors defined in Section 9.2.1 of Chapter 9—i.e. when defining the cartesian product in **PCoh**. The result is then a consequence of Lemma 9.2.3: indeed recall from the definition of $\otimes$ that $G = \{v_1 \otimes \ldots \otimes v_n \mid v_i \in \mathrm{P}\,(\mathcal{A} \,\&\, \mathbf{1})\}$ is a set of generators for $(\mathcal{A} \,\&\, \mathbf{1})^{\otimes n}$. Moreover, observe that for every $v \in \mathrm{P}\,(\mathcal{A} \,\&\, \mathbf{1})$ it holds that $v \leq_{|\mathcal{A}\&\mathbf{1}|} \langle \pi_1 v, e_\star \rangle$. As a consequence, if we denote $G' = \{(\langle u_1, e_\star \rangle) \otimes \cdots \otimes (\langle u_n, e_\star \rangle) \mid u \in \mathrm{P}\,(\mathcal{A})\}$, we see that $G' \subseteq G$, and that every element in G is dominated by an element in G', and since the bi-duality operator is downward closing, it implies that $G'^{\perp\perp} = G^{\perp\perp}$. As a consequence, G' also is a set of generators for $\mathcal{A} \,\&\, \mathbf{1}$, and we can conclude using Lemma 9.2.3, that tells us that $\mathrm{P}\,((\mathcal{A} \,\&\, \mathbf{1})^{=n}) = \mathtt{jm}^n_{\mathcal{A}}\,(G')^{\perp\perp}$. □

Example 10.2.4 *We have no simple characterization of* $\boldsymbol{Bool}^{\leq 2}$*, but Lemma 10.2.7 helps us in computing its generators. For example, we have:*

$$\langle e_{\mathtt{t}}, e_{\mathtt{f}} \rangle = e_{[\cdot]} + \frac{1}{2} e_{[\mathtt{t},\mathtt{f}]} + \frac{1}{2} e_{[\mathtt{t}]} + \frac{1}{2} e_{[\mathtt{f}]}$$
$$\langle e_{\mathtt{t}}, e_{\mathtt{t}} \rangle = e_{[\cdot]} + e_{[\mathtt{t}]} + e_{[\mathtt{t},\mathtt{t}]},$$

One can observe more generally that $(\sup \boldsymbol{Bool}^{\leq n})_\mu = 1$ *for any multiset* $\mu \in \mathcal{M}_{\leq n}(\{\mathtt{t}, \mathtt{f}\})$ *which is* uniform *(i.e. of which support is at most a singleton), while* $(\sup \boldsymbol{Bool}^{\leq n})_\mu < 1$ *for* μ non-uniform.

Henceforth, we will consider $\mathcal{A}^{\leq n}$ as presented in Lemma 10.2.7.

10.2.3 The limit $!_f\mathcal{A}$

The quest for a limit $!_f\mathcal{A}$ of the family $(\mathcal{A}^{\leq n})_{n \in \mathbb{N}}$ requires studying the relations between exponential approximations of different degree. This is done starting from the following notions of injection and projection. Given $\mathcal{A}, \mathcal{B}$ s.t. $|\mathcal{A}| \subseteq |\mathcal{B}|$, we define the matrices *injection* $\iota_{\mathcal{A},\mathcal{B}} \in \mathbb{R}_+^{|\mathcal{A}| \times |\mathcal{B}|}$ and *projection* $\rho_{\mathcal{B},\mathcal{A}} \in \mathbb{R}_+^{|\mathcal{B}| \times |\mathcal{A}|}$ as follows:

$$(\iota_{\mathcal{A},\mathcal{B}})_{a,b} = (\rho_{\mathcal{B},\mathcal{A}})_{b,a} = \delta_{a,b}. \tag{10.9}$$

The injection $\iota_{\mathcal{A},\mathcal{B}}$ maps a vector $u \in \mathbb{R}_+^{|\mathcal{A}|}$ to the vector $(\langle u, \vec{0} \rangle) \in \mathbb{R}_+^{|\mathcal{B}|}$ associating the directions in $|\mathcal{B}| \setminus |\mathcal{A}|$ with zero—here $\times$ denotes the cartesian product between vectors as defined in Section 9.2.1 of Chapter 9, and $\vec{0}$ the zero vector. The projection $\rho_{\mathcal{B},\mathcal{A}}$ maps

a vector $(\langle u, v\rangle) \in \mathbb{R}_+^{|\mathcal{B}|}$ to its restriction u to the directions within $|\mathcal{A}|$. In order to have these matrices as morphisms in **PCoh**, we have to prove that $(\langle u, \vec{0}\rangle) \in \mathrm{P}(\mathcal{B})$ whenever $u \in \mathrm{P}(\mathcal{A})$ (resp. $u \in \mathrm{P}(\mathcal{A})$ whenever $(\langle u, v\rangle) \in \mathrm{P}(\mathcal{B})$).

The projection matrix $\rho_{\mathcal{A}^{\leq n+1},\mathcal{A}^{\leq n}}$ of $\mathcal{A}^{\leq n+1}$ into $\mathcal{A}^{\leq n}$ actually coincide with the categorically defined projection morphisms $\mathtt{p}_{n+1,n}$ from Definition 10.1.10. As a corollary, we get the following lemma.

Lemma 10.2.8 *For any $m \geq n$ and $\mathcal{A}$, we have: $\rho_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq n}} \in \mathbf{PCoh}(\mathcal{A}^{\leq m}, \mathcal{A}^{\leq n})$ and $\iota_{\mathcal{A}^{\leq n\perp},\mathcal{A}^{\leq m\perp}} \in \mathbf{PCoh}(\mathcal{A}^{\leq n\perp}, \mathcal{A}^{\leq m\perp})$.*

Proof. Let $n \in \mathbb{N}$. Recall that $\mathtt{p}_{n+1,n}$ is defined as the unique morphism such that the following morphisms $\mathcal{A}^{\leq n+1} \to (\mathcal{A} \,\&\, \mathbf{1})^{\otimes n}$ coincide:

$$\mathtt{r}_{(\mathcal{A}\&\mathbf{1})^{\otimes n}} \circ (\mathtt{id} \otimes \pi_2) \circ \mathtt{eq}_{\mathcal{A}^{\leq n+1}} = \mathtt{eq}_{\mathcal{A}^{\leq n}} \circ \mathtt{p}_{n+1,n} \tag{10.10}$$

When we unfold Equation (10.10) by looking at matrix coefficients—recall the expressions of $\mathtt{r}$, π_2 in **PCoh**, stated in Section 9.2.1—we obtain that for every $\mu \in \mathcal{M}_{\leq n+1}(|\mathcal{A}|)$, and $\vec{a} \in |\mathcal{A} \,\&\, \mathbf{1}|$, $\sum_{\nu \in \mathcal{M}_{\leq n}(|\mathcal{A}|)} (\mathtt{p}_{n+1,n})_{\nu,\vec{a}}^{\sim} = \delta_{\mu,\vec{a}}^{\sim}$. We see that these constraints entirely characterize the matrix $\mathtt{p}_{n+1,n}$, and more precisely that $\mathtt{p}_{n+1,n}$ must be exactly $\rho_{\mathcal{A}^{\leq n+1},\mathcal{A}^{\leq n}}$. As a consequence, $\rho_{\mathcal{A}^{\leq n+1},\mathcal{A}^{\leq n}} \in \mathbf{PCoh}(\mathcal{A}^{\leq n+1}, \mathcal{A}^{\leq n})$. Since for every m, n with $m > n$, $\rho_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq n}} = \rho_{\mathcal{A}^{\leq n+1},\mathcal{A}^{\leq n}} \circ \ldots \circ \rho_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq m-1}}$, it also holds that $\rho_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq n}}$ is a morphism in $\mathbf{PCoh}(\mathcal{A}^{\leq m}, \mathcal{A}^{\leq n})$. Moreover, we can also see that $\iota_{\mathcal{A}^{\leq n\perp},\mathcal{A}^{\leq m\perp}}$ is indeed a morphism $\mathcal{A}^{\leq n\perp} \to \mathcal{A}^{\leq m\perp}$, since:

$$\begin{aligned}(\forall x \in \mathrm{P}\left(\mathcal{A}^{\leq m}\right), \rho_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq n}} \in \mathrm{P}\left(\mathcal{A}^{\leq n}\right)) \\ \Rightarrow (\forall y \in \mathrm{P}\left(\mathcal{A}^{\leq n}\right)^{\perp}, \iota_{\mathcal{A}^{\leq n\perp},\mathcal{A}^{\leq m\perp}} \in \mathrm{P}\left(\mathcal{A}^{\leq m}\right)^{\perp})\end{aligned}$$

□

The interesting point is that the dual version of Lemma 10.2.8 does not hold: in general, the injection of $\mathcal{A}^{\leq n}$ into $\mathcal{A}^{\leq n+1}$ (resp. projection of $(\mathcal{A}^{\leq n+1})^{\perp}$ into $(\mathcal{A}^{\leq n})^{\perp}$) is *not* a morphism of **PCoh**. We illustrate this using the boolean PCS **Bool** in Example 10.2.5 below.

Example 10.2.5 *In Example 10.2.4, we discussed $\langle e_{\mathtt{t}}, e_{\mathtt{f}}\rangle \in \mathrm{P}\left(\boldsymbol{Bool}^{\leq 2}\right)$. Let us prove now that $\iota_{\boldsymbol{Bool}^{\leq 2},\boldsymbol{Bool}^{\leq 3}} \langle e_{\mathtt{t}}, e_{\mathtt{f}}\rangle \notin \mathrm{P}\left(\boldsymbol{Bool}^{\leq 3}\right)$. It is a consequence of the following facts:*

$$\begin{aligned}(\iota_{\boldsymbol{Bool}^{\leq 2},\boldsymbol{Bool}^{\leq 3}} \langle e_{\mathtt{t}}, e_{\mathtt{f}}\rangle)_{[\mathtt{t},\mathtt{f}]} &= \langle e_{\mathtt{t}}, e_{\mathtt{f}}\rangle_{[\mathtt{t},\mathtt{f}]} = \frac{1}{2} \\ (\sup \mathrm{P}\left(\boldsymbol{Bool}^{\leq 3}\right))_{[\mathtt{t},\mathtt{f}]} &= \frac{1}{3}.\end{aligned}$$

The latter claim is because:

$$\mathrm{P}\left(\boldsymbol{Bool}^{\leq 3}\right) = \{\langle e_{\mathtt{t}}, e_{\mathtt{t}}, e_{\mathtt{t}}\rangle, \langle e_{\mathtt{t}}, e_{\mathtt{t}}, e_{\mathtt{f}}\rangle, \langle e_{\mathtt{t}}, e_{\mathtt{f}}, e_{\mathtt{f}}\rangle, \langle e_{\mathtt{f}}, e_{\mathtt{f}}, e_{\mathtt{f}}\rangle\}^{\perp\perp}$$

and the maximal value of these generators on $[\mathtt{t}, \mathtt{f}]$ is $\frac{1}{3}$. Since the bi-duality operator consists in the convex, downward and Scott-closure, it means that no element in $\mathrm{P}\left(\boldsymbol{Bool}^{\leq 3}\right)$ may have a coefficient on $[\mathtt{t}, \mathtt{f}]$ greater that $\frac{1}{3}$.

One can however, for every $N \geq n$, add a correction factor to the injection matrix $\iota_{\mathcal{A},n,N}$ in order to obtain a new matrix $\iota^{cor}_{\mathcal{A},n,N}$ that embeds $\mathcal{A}^{\leq n}$ into $\mathcal{A}^{\leq N}$, i.e. that is both invertible, and a morphism $\mathcal{A}^{\leq n} \to \mathcal{A}^{\leq N}$. We define the *amended injection matrix* ι^{cor} as follows:

$$(\iota^{cor}_{\mathcal{A},n,N})_{\mu,\nu} := \begin{cases} \frac{(N-k)!q^k n!}{N!(n-k)!} & \text{if } \mu = \nu \text{ and } \#\mu = k \leq n \\ & \text{and } q = \lfloor \frac{N}{n} \rfloor \text{ and } r = N \bmod n; \\ 0 & \text{otherwise.} \end{cases} \tag{10.11}$$

Lemma 10.2.9 *For every PCS $\mathcal{A}$, and for any $n, N \in \mathbb{N}$ with $n < N$, the amended injection matrix $\iota^{cor}_{\mathcal{A},n,N}$ is a morphism in $\mathbf{PCoh}(\mathcal{A}^{\leq n}, \mathcal{A}^{\leq N})$ mapping, in particular, $\langle u_1, \dots, u_n \rangle \in \mathrm{P}\left(\mathcal{A}^{\leq n}\right)$ to $\langle u_1^q, \dots, u_n^q, \vec{0}^r \rangle \in \mathrm{P}\left(\mathcal{A}^{\leq N}\right)$, where q is the quotient $\lfloor \frac{N}{n} \rfloor$ and r the remainder $N \bmod n$ of the euclidean division $\frac{N}{n}$. Moreover, u_i^q is a notation for $\underbrace{u_i, \dots, u_i}_{q \text{ times}}$ (and similarly for $\vec{0}^r$).*

Proof. One has just to prove the last part of the statement, the rest follows by Lemma 9.2.3 because the vectors of the form $\langle u_1, \dots, u_n \rangle$ are a set of generators for $\mathrm{P}\left(\mathcal{A}^{\leq n}\right)$ (Lemma 10.2.7). We have, for any multiset $\mu = [a_1, \dots, a_k]$,

$$\begin{aligned} (\iota^{cor}_{\mathcal{A},n,N} \langle u_1, \dots, u_n \rangle)_\mu &= \left(\frac{(N-k)!q^k n!}{N!(n-k)!} \right) \frac{(n-k)!}{n!} \sum_{f:\{1,\dots,k\} \hookrightarrow \{1,\dots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i} \\ &= \frac{(N-k)!}{N!} q^k \sum_{f:\{1,\dots,k\} \hookrightarrow \{1,\dots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i} \\ &= \frac{(N-k)!}{N!} \sum_{f:\{1,\dots,k\} \hookrightarrow \{1,\dots,nq\}} \prod_{i=1}^{k} (u_{(\lfloor f(i)/q \rfloor)+1})_{a_i} \\ &= \langle u_1^q, \dots, u_n^q, \mathbf{0}^r \rangle_\mu. \end{aligned}$$

where we use $\hookrightarrow$ to denote injective functions and where from line 2 to 3, we use the count q^k to enlarge the codomain of the injections f indexing the sum. □

At this point, we can do the following crucial observation: if we fix $n \in \mathbb{N}$, and we look at the morphism $\iota^{cor}_{\mathcal{A},n,N} : \mathcal{A}^{\leq n} \to \mathcal{A}^{\leq N}$ when N tends towards $+\infty$, we can see that the corresponding matrix coefficients also admit a limit. We express that formally in Equations (10.12) and (10.13) below. If we note $N = nq + r$, it holds that:

$$\lim_{N \to \infty} \frac{(N-k)!q^k n!}{N!(n-k)!} = \frac{n!}{n^k (n-k)!}, \tag{10.12}$$

$$(\lim_{N \to \infty} \iota^{cor}_{\mathcal{A},n,N} \langle u_1, \dots, u_n \rangle)_\mu = \frac{1}{n^k} \sum_{f:\{1,\dots,k\} \hookrightarrow \{1,\dots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i}. \tag{10.13}$$

This observation allows us to take the following approach to construct the limit object: the contribution of each $\mathcal{A}^{\leq n}$ to the limit object $!_f\mathcal{A}$ should be the limit of its contribution to the $\mathcal{A}^{\leq N}$, when N is greater than n and tends toward $+\infty$. We formalize this intuition in Definition 10.2.3 below, and then proceed to show that the object we build this way is indeed the free exponential comonoid.

Definition 10.2.3 *Given $\mathcal{A}$, we define $!_f\mathcal{A}$ as:*

$$|!_f\mathcal{A}| = \mathcal{M}_f(|\mathcal{A}|), \qquad \mathrm{P}(!_f\mathcal{A}) = \{\langle\!\langle u_1, \dots, u_n\rangle\!\rangle \ ; \ n \in \mathrm{N}, \forall i \leq n, u_i \in \mathrm{P}(\mathcal{A})\}^{\perp\perp},$$

where: $\langle\!\langle u_1, \dots, u_n\rangle\!\rangle_{[a_1,\dots,a_k]} = \frac{1}{n^k} \sum_{f:\{1,\dots,k\}\hookrightarrow\{1,\dots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i}$.

Notice that whenever $k > n$, $\langle\!\langle u_1, \dots, u_n\rangle\!\rangle_{[a_1,\dots,a_k]} = 0$.

Proposition 10.2.10 *Let $\mathcal{A}$ be a PCS. The object $!_f\mathcal{A}$ together with the family of morphisms $\rho_{!_f\mathcal{A},\mathcal{A}^{\leq n}} \in \mathbf{PCoh}(!_f\mathcal{A}, \mathcal{A}^{\leq n})$ for $n \in \mathrm{N}$, constitutes the limit of the chain:*

$$\mathbf{1} \overset{\mathtt{p}^{1,0}}{\leftarrow} \mathcal{A}^{\leq 1} \overset{\mathtt{p}^{2,1}}{\leftarrow} \mathcal{A}^{\leq 2} \overset{\mathtt{p}^{3,2}}{\leftarrow} \dots$$

Moreover, this limit commutes with the tensor product (Figure 10.4).

Proof. First, we prove that $\rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}}$ is a correct morphism mapping $\mathrm{P}(!_f\mathcal{A})$ into $\mathrm{P}\left(\mathcal{A}^{\leq m}\right)$, for any m. By Lemma 9.2.3 it is enough to check that any $\langle\!\langle u_1, \dots, u_n\rangle\!\rangle$ is mapped to the PCS $\mathrm{P}\left(\mathcal{A}^{\leq m}\right)$ by $\rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}}$. As $\mathrm{P}\left(\mathcal{A}^{\leq m}\right) = \mathrm{P}\left(\mathcal{A}^{\leq m}\right)^{\perp\perp}$, it is equivalent to show that for any $n \in \mathrm{N}$, $u_i \in \mathrm{P}(\mathcal{A})$, and $w \in \mathrm{P}\left(\mathcal{A}^{\leq m}\right)^{\perp}$ we have:

$$\langle \rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}} \langle\!\langle u_1, \dots, u_n\rangle\!\rangle, w\rangle \leq 1 \tag{10.14}$$

Lemma 10.2.8 and 10.2.9 give us, $\forall N \geq m, n$, resp.: $\iota_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq N}} w \in \mathrm{P}\left(\mathcal{A}^{\leq N}\right)^{\perp}$ and $\iota^{cor}_{\mathcal{A},n,N} \langle u_1, \dots, u_n\rangle \in \mathrm{P}\left(\mathcal{A}^{\leq N}\right)$. Thus the inner product between the two vectors is bounded by 1. Consider then the limit of this product for $N \to \infty$:

$$\begin{aligned}
1 \geq & \lim_{N\to\infty} \langle \iota^{cor}_{\mathcal{A},n,N} \langle u_1, \dots, u_n\rangle, \iota_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq N}} w\rangle \\
= & \langle \langle\!\langle u_1, \dots, u_n\rangle\!\rangle, \iota_{\mathcal{A}^{\leq m},!_f\mathcal{A}} w\rangle && \text{(Eq. (10.13) and Def. 10.2.3)} \\
= & \langle \rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}} \langle\!\langle u_1, \dots, u_n\rangle\!\rangle, (\rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}} \circ \iota_{\mathcal{A}^{\leq m},!_f\mathcal{A}}) w\rangle \\
= & \langle \rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}} \langle\!\langle u_1, \dots, u_n\rangle\!\rangle, w\rangle
\end{aligned}$$

Line 2 gives line 3 using of the definition of ι and π in (10.9). Now we prove that $!_f\mathcal{A}$ together with its projections $\rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}}$ is indeed a limit cone. As for Proposition 10.2.6, we prove straight the commutation with the $\otimes$, as the first part of the statement is a consequence of this latter, taking $\mathcal{B} = \mathbf{1}$.

Take a PCS $\mathcal{C}$ and an $\mathbb{N}$-indexed family of morphisms $f_n \in \mathbf{PCoh}(\mathcal{C}, \mathcal{A}^{\leq n} \otimes \mathcal{B})$ commuting with the chain $\mathcal{B} \overset{\mathtt{p}^{1,0}}{\leftarrow} \mathcal{A}^{\leq 1} \otimes \mathcal{B} \overset{\mathtt{p}^{2,1}}{\leftarrow} \mathcal{A}^{\leq 2} \otimes \mathcal{B} \overset{\mathtt{p}^{3,2}}{\leftarrow} \dots$. We should define a unique $f^\dagger$ s.t. Figure 10.4 commutes. The matrix $f^\dagger$ is defined as:

$$f^\dagger_{c,(\mu,b)} = (f_{\#\mu})_{c,(\mu,b)}.$$

The fact that $f^\dagger$ is the unique one such that Figure 10.4 commutes is an easy calculation. We should then prove that it is a morphism in $\mathbf{PCoh}(\mathcal{C}, !_f\mathcal{A} \otimes \mathcal{B})$, i.e. for every $v \in \mathrm{P}(\mathcal{C})$, $f^\dagger v \in \mathrm{P}(!_f\mathcal{A} \otimes \mathcal{B})$. Since $\mathrm{P}(!_f\mathcal{A} \otimes \mathcal{B}) = \mathrm{P}(!_f\mathcal{A} \otimes \mathcal{B})^{\perp\perp}$, it is equivalent to prove that: $\forall w \in \mathrm{P}(!_f\mathcal{A} \otimes \mathcal{B})^{\perp}$, $\langle f^\dagger v, w\rangle \leq 1$. For any n, define $w \downarrow_n \in \mathbb{R}_+^{|\mathcal{A}^{\leq n} \otimes \mathcal{B}|}$ as:

$$(w \downarrow_n)_{(\mu,b)} = \begin{cases} \frac{n!}{n^k(n-k)!} w_{(\mu,b)} & \text{if } \#\mu = k \leq n, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that, for any $\langle u_1, \dots, u_n \rangle \in \mathrm{P}\left(\mathcal{A}^{\leq n}\right)$ and $z \in \mathrm{P}\left(\mathcal{B}\right)$, we have the inequality:

$$\langle \langle u_1, \dots, u_n \rangle \otimes z, w \downarrow_n \rangle = \langle \langle\!\langle u_1, \dots, u_n \rangle\!\rangle \otimes z, w \rangle \leq 1.$$

We conclude that $w \downarrow_n \in \mathrm{P}\left(\mathcal{A}^{\leq n} \otimes \mathcal{B}\right)^{\perp}$, for any n. Then we have:

$$\begin{aligned} 1 \geq \lim_{n\to\infty} \langle f_n\, v, w \downarrow_n \rangle &= \lim_{n\to\infty} \langle (\rho_{!_f\mathcal{A},\mathcal{A}^{\leq n}} \circ f^{\dagger})\, v, w \downarrow_n \rangle \qquad (\text{def } f^{\dagger}) \\ &= \langle f^{\dagger}\, v, w \rangle \qquad \text{since } \lim_{n\to\infty} \frac{n!}{n^k (n-k)!} = 1 \end{aligned}$$

□

Propositions 10.1.5, 10.2.6 and 10.2.10 give the last corollary.

Corollary 10.2.1 *For any PCS $\mathcal{A}$, the PCS $!_f\mathcal{A}$ yields the free commutative comonoid generated by $\mathcal{A}$.*

The free comonad $!_f\mathcal{A}$ yields to a Lafont model—see Definition 9.1.10 in Chapter 9—of linear logic which seems different from the new-Seely one developed by Danos and Ehrhard —that we've presented in Section 9.2.2 of Chapter 9 —because the definition of $!_f\mathcal{A}$ seems apparently different from the entire exponential modality $!_e\mathcal{A}$ (Definition 9.2.5). In fact, the two spaces are the same, as shown in the following section.

10.3 The free and entire exponential modalities are the same.

How do the exponential approximations $\mathcal{A}^{\leq n}$ of $!_f\mathcal{A}$ relate with $!_e\mathcal{A}$? Let us consider the PCS $\mathcal{A} = \mathbf{Bool}$, and compare the maximal coefficients of these spaces on $[\mathtt{t}, \mathtt{f}]$. It is easy to check that:

$$(\sup !_e\mathbf{Bool})_{[\mathtt{t},\mathtt{f}]} = (\frac{e_{\mathtt{t}} + e_{\mathtt{f}}}{2})^{!}_{[\mathtt{t},\mathtt{f}]} = \frac{1}{4}.$$

On the other hand, if we look at the value of those same maximal coefficients in the exponential approximations $\mathbf{Bool}^{\leq n}$, we obtain:

$$(\sup \mathbf{Bool}^{\leq n})_{[\mathtt{t},\mathtt{f}]} = (\langle e_{\mathtt{t}}^{\lfloor \frac{n}{2} \rfloor}, e_{\mathtt{f}}^{\lceil \frac{n}{2} \rceil} \rangle_{[\mathtt{t},\mathtt{f}]} = (\langle e_{\mathtt{t}}^{\lceil \frac{n}{2} \rceil}, e_{\mathtt{f}}^{\lfloor \frac{n}{2} \rfloor} \rangle_{[\mathtt{t},\mathtt{f}]},$$

whose values are, for $n = 2, 3, 4, \dots$ (using Equation (10.8)):

$$\frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{3}{10}, \frac{3}{10}, \frac{2}{7}, \dots, \frac{1}{n(n-1)} \lfloor \frac{n}{2} \rfloor \lceil \frac{n}{2} \rceil \dots \quad \overset{n\to\infty}{\to} \frac{1}{4}$$

This remark can be generalized, showing that the exponential approximations $\mathcal{A}^{\leq n}$ are actually approaching to $!_e\mathcal{A}$ from above, giving that their limit is equal to $!_e\mathcal{A}$.

Proposition 10.3.1 *For any PCS $\mathcal{A}$, we have $!_f A = !_e A$.*

Proof. The two spaces have the same web, we prove that $\mathrm{P}\left(!_f A\right) = \mathrm{P}\left(!_e A\right)$. We first show that $\mathrm{P}\left(!_f A\right) \subseteq \mathrm{P}\left(!_e A\right)$. Take any $\langle\!\langle u_1, \dots, u_n \rangle\!\rangle \in \mathrm{P}\left(!_f A\right)$, we have $\langle\!\langle u_1, \dots, u_n \rangle\!\rangle \in \mathrm{P}\left(!_e A\right)$, because $\langle\!\langle u_1, \dots, u_n \rangle\!\rangle \leq (\frac{1}{n} \sum_i u_i)^{!} \in \mathrm{P}\left(!_e A\right)$.

Conversely, let u^n denotes $u, \dots, u$ repeated n times in

$$\langle\!\langle u^n \rangle\!\rangle_{[a_1,\dots,a_k]} = \frac{n!}{n^k (n-k)!} \prod_{i=1}^{k} u_{a_i} = \frac{n!}{n^k (n-k)!} u^{!}_{[a_1,\dots,a_k]}.$$

As for $k < n$, $\frac{n!}{n^k (n-k)!}$ is an increasing sequence converging to 1, we get that $\forall u \in \mathrm{P}\left(A\right)$, $\sup_n \langle\!\langle u^n \rangle\!\rangle = u^{!}$. Now, $u^{!} \in \mathrm{P}\left(!_f\mathcal{A}\right)$ since it is Scott-closed. Since $u^{!}$ for $u \in \mathrm{P}\left(\mathcal{A}\right)$ are generating $\mathrm{P}\left(!_e\mathcal{A}\right)$, we conclude that $\mathrm{P}\left(!_e A\right) \subseteq \mathrm{P}\left(!_f A\right)$. □

Chapter 11

PCoh is a full subcategory of $\mathbf{Cstab_m}$

The category $\mathbf{Cstab_m}$ was introduced as a model for $\mathsf{PCF}_{\text{sample}}$—the higher-order language with continuous probabilities that we presented in Section 9.3 from Chapter 9 —by Ehrhard, Pagani and Tasson in [45], i.e. they built in $\mathbf{Cstab_m}$ a sound and adequate interpretation of $\mathsf{PCF}_{\text{sample}}$. It is not known, however, if this model is also *fully abstract*. We *do not* solve this question here, but the work we present now was designed in the hope that it could be a first step in this direction. Our perspective is as follows: while we have until now no full abstraction result for a model of an higher-order language with *continuous probabilities*, it is different in the case of *discrete* probabilities: for instance **PCoh** is a fully abstract model of $\mathsf{PCF}_{\oplus}$, and there are also game models that have been shown to be fully abstract for diverse higher-order language with probabilities—see Chapter 9 for a more precise account. What we do here is to look at $\mathbf{Cstab_m}$ from the perspective of discrete probabilities: our result consists in establishing that the discrete model induced by $\mathbf{Cstab_m}$ is exactly the $\mathsf{PCF}_{\oplus}$ model in **PCoh**. It enables us to reformulate the full-abstraction problem as follows: is it possible to generalize the full-abstraction proof for **PCoh** in a more general—i.e. continuous—setting ?

To make more precise what we mean when we talk about $\mathbf{Cstab_m}$ as a model of discrete higher-order computation, let us first consider a *continuous* higher-order language with an *explicit discrete fragment*. We define a language $\mathsf{PCF}_{\oplus,\mathtt{sample}}$ that has all syntactic constructs of both $\mathsf{PCF}_{\oplus}$ and $\mathsf{PCF}_{\mathtt{sample}}$, with both a real type R, and a natural number type N as base types. We want this language to be able to *convert* natural data types into real data type, and reversely, in the following sense: first, we endow this language with a *conversion operator* `real`, that turns every distribution over $\mathbb{N}$ into a distribution over $\mathbb{R}$, without information loss, and consequently is associated with a typing rule as follows:

$$\frac{\Gamma \vdash M : N}{\Gamma \vdash \mathtt{real}(M) : R}$$

This operator is designed to enable the continuous constructs to act on the discrete fragment, by giving a way to see any distribution on $\mathbb{N}$ as a distribution on $\mathbb{R}$. The language $\mathsf{PCF}_{\oplus,\mathtt{sample}}$ is designed to talk about *approximating by discrete tests* the programs in $\mathsf{PCF}_{\mathtt{sample}}$: for instance if we consider a $\mathsf{PCF}_{\mathtt{sample}}$ program $M : R \to R$, we can approximate its behavior by considering all programs $\lambda x \cdot KM(Lx) : N \to N$, where L and K are $\mathsf{PCF}_{\oplus,\mathtt{sample}}$ programs of type respectively $N \to R$, and $R \to N$. Observe that the program K may be built using for instance the order operators—see Section 1.4.2 of Chapter 1—while the existence of such programs L is guarantee by our $\mathtt{real}(\cdot)$ construct.

We can extend in a natural way the denotational semantics of $\mathsf{PCF}_{\texttt{sample}}$ given in [45] to $\mathsf{PCF}_{\oplus,\texttt{sample}}$: in the same way that the denotational semantics of R is taken as the set of all finite measures on $\mathbb{R}$, we take the denotational semantics of N as the set $\mathrm{Meas}(\mathbb{N})$ of all finite measures over $\mathbb{N}$. We take as denotational semantics of the operator `real` the function: $[\![\texttt{real}]\!]_{\mathbf{Cstab_m}} : \mu \in \mathrm{Meas}(\mathbb{N}) \mapsto (A \in \Sigma_{\mathbb{R}} \mapsto \sum_{n \in \mathbb{N} \cap A} \mu(n)) \in \mathrm{Meas}(\mathbb{R})$. We will see later that this function is indeed a morphism in $\mathbf{Cstab_m}(\mathrm{Meas}(\mathbb{N}), \mathrm{Meas}(\mathbb{R}))$. What we would like to know is: what is the structure of the sub-category of $\mathbf{Cstab_m}$ given by the *discrete types* of $\mathsf{PCF}_{\oplus,\texttt{sample}}$, i.e generated inductively by $[\![N]\!]_{\mathbf{Cstab_m}}$, $\Rightarrow$, $\times$?

The starting point of our work is the connection highlighted in [45] between PCSs and complete cones: every PCSs can be seen as a complete cone, in such a way that the denotational semantics of N in $\mathbf{PCoh_!}$ becomes the set of finite measures over $\mathbb{N}$. We formalize this connection by a functor $F^m : \mathbf{PCoh_!} \to \mathbf{Cstab_m}$. However, to be able to use $\mathbf{PCoh_!}$ to obtain information about the discrete types sub-category of $\mathbf{Cstab_m}$, we need to know whether this connection is preserved at higher-order types: does the $\Rightarrow$ construct in $\mathbf{Cstab_m}$ make some wild functions not representable in $\mathbf{PCoh_!}$ to appear, e.g. not analytic? In this paper, we show that this is not the case, meaning that the functor F^m is full and faithful, and cartesian closed. Since $\mathbf{PCoh_!}$ is a fully abstract model of $\mathsf{PCF}_{\oplus}$, it means that the discrete fragment of $\mathsf{PCF}_{\oplus,\texttt{sample}}$ is fully abstract in $\mathbf{Cstab_m}$. More generally, it tells us also that for any $\mathsf{PCF}_{\texttt{sample}}$ program M, regardless of the continuous computations it does, every approximation of M by discrete tests has for denotation in $\mathbf{Cstab_m}$ a power series: hence we are able to gather information on the behavior of M by looking only at power series, that are much easier to handle that general functions between cones.

It was noted in [45] that there is a natural way to see any probabilistic coherent space as an object of $\mathbf{Cstab}$. In this work, we show that this connection leads to a full and faithful functor $\mathcal{F}$ from $\mathbf{PCoh_!}$—the Kleisli category of $\mathbf{PCoh}$—into $\mathbf{Cstab}$. We do that by showing that every stable function between probabilistic coherent spaces can be seen as a power series, using McMillan's extension [80] to an abstract setting of Bernstein's theorem for absolutely monotonic functions. We then show that this functor $\mathcal{F}$ is cartesian closed, i.e. respects the cartesian closed structure of $\mathbf{PCoh_!}$. In the last part, we turn $\mathcal{F}$ into a functor $\mathcal{F}^m : \mathbf{PCoh_!} \to \mathbf{Cstab_m}$, and we show that $\mathcal{F}^m$ too is cartesian closed. To sum up, the contribution of this paper is the construction of a cartesian closed full embedding from $\mathbf{PCoh_!}$ into $\mathbf{Cstab_m}$. Since $\mathbf{PCoh_!}$ is known to be a fully abstract denotational model of $\mathsf{PCF}_{\oplus}$, an immediate corollary is that $\mathbf{Cstab_m}$ too is a fully abstract model of $\mathsf{PCF}_{\oplus}$.

11.1 A generalization of Bernstein's theorem for pre-stable functions

In [80], McMillan generalized Bernstein's Theorem on absolutely monotonic function from real analysis to general domains with partitions systems. Here, we present its result in the more restricted setting of pre-stable functions on directed-complete lattice cones. Its approach consists in first defining an analogue of derivatives for pre-stable functions, and then showing that pre-stable functions can be written as the infinite sum generated by an analogue of Taylor expansion on $\mathscr{B}^\circ C$. We give here the main steps of the construction directly on cones, and highlight some properties of the Taylor series which are true for directed-complete lattice cones, but not in the general framework McMillan considered.

11.1.1 Directed-complete Lattice Cones

We first define the restricted set of cones that we consider; it is designed to enable us to use McMillan's results while also containing all cones fenrated by PCSs.

Definition 11.1.1 *A cone C is said to be:*

- directed complete *if for any directed subset D of $\mathscr{B}C$, D has a least upper bound* $\sup D \in \mathscr{B}C$.
- a lattice cone *if any two elements x, y of C have a least upper bound $x \vee y$.*

Observe that a directed-complete cone is always sequentially complete.

Lemma 11.1.1 *Let be C a lattice cone. Then it holds that:*

- *Any two element x, y of C have a greatest lower bound $x \wedge y$.*
- *Decomposition Property: if $z \leq x + y$, there there exists $z_1, z_2 \in C$ such that $z = z_1 + z_2$, and $z_1 \leq x$, and $z_2 \leq y$.*

Proof. Recall that, if $a \geq b$, we denote by $a - b$ the element c such that $a = b + c$.

- We consider $z = x + y - (x \vee y)$, and we show that z is indeed the greatest lower bound of x and y.
- We take $z_2 = (x \vee z) - x$, and $z_1 = z - z_1$. First, we see that $z_2 \leq (x + y) - x$, and so $z_2 \leq y$. Moreover, $z_1 = x - ((x \vee z) - z) \leq x$.

□

11.1.2 Derivatives of a pre-stable function

We are now going, following McMillan [80], to construct derivatives for pre-stable functions on directed complete cones. This construction is based on the use of a notion of *partition*: a *partition* of $x \in \mathscr{B}C$ is a multiset $\pi = [u_1, \ldots, u_n] \in \mathcal{M}_f(C)$ such that $x = \sum_{1 \leq i \leq n} u_i$. We write $\pi \sim x$ when the multiset π is a partition of x. We will denote by $+$ the usual union on multiset: $[y_1, \ldots, y_n] + [z_1, \ldots, z_m] = [y_1, \ldots, y_n, z_1, \ldots, z_m]$. We call $\mathrm{Parts}(x)$ the set of partitions of x.

Definition 11.1.2 (Refinement Preorder) *If π_1, π_2 are in Parts(x), we says that $\pi_1 \leq \pi_2$ if $\pi_1 = [u_1, \ldots, u_n]$, and $\pi_2 = \alpha_1 + \ldots + \alpha_n$ with each of the α_i a partition of u_i.*

Observe that when π_1 and π_2 are partition of x, $\pi_2 \leq \pi_1$ means that π_1 is a more *finely grained* decomposition of x. If $\vec{u}$ is an n-tuple in $\mathscr{B}C$, we extend the refinement order to $\mathrm{Parts}(\vec{u}) = \mathrm{Parts}(u_1) \times \ldots \times \mathrm{Parts}(u_n)$.

Lemma 11.1.2 *Let C be a lattice cone, and $x \in C$. Then Parts(x) is a directed set.*

Proof. We are going to use the following notion: we say that two non-zero elements x and y of C are *orthogonal*, and we note $x \perp y$, if $x \wedge y = 0$. Let be $\pi_1, \pi_2 \in \mathrm{Parts}(x)$. We first show that it cannot exist $z \in \pi_1$ which is orthogonal to all the element of π_2. Indeed, suppose that it is the case: we take $y_1, \ldots, y_n$ such that $\pi_2 = [y_1, \ldots, y_n]$. Then by hypothesis, $z \leq x = \sum_{1 \leq i \leq n} y_i$. We can now use the decomposition property from

Lemma 11.1.1. It means that $z = \sum_{1 \leq i \leq n} z_i$, with $z_i \leq y_i$. But since for all i, $z \perp y_i$, it folds that $z_i = 0$ for all i, and so $z = 0$, and we have a contradiction.

Now, we are going to present a procedure to construct a partition π of x with $\pi \leq \pi_1$, and $\pi \leq \pi_2$. We can suppose that all elements of π_1 and π_2 are non-zero. We start form $\pi = []$, $\theta_1 = \pi_1$, $\theta_2 = \pi_2$, and $w = x$, $v = 0$. Through the procedure, we guarantee:

- $\theta_1, \theta_2 \in \mathrm{Parts}(w)$, $\pi \in \mathrm{Parts}(v)$, and $w + v = x$;
- all the elements of Θ_1 and Θ_2 are non-zero;
- $\pi + \Theta_1 \leq \pi_1$, and $\pi + \Theta_2 \leq \pi_2$ (for the refinment order).

Then at each step of the procedure, if θ_1 is non empty, we do the following: let $\theta_1 = [a_1, \ldots, a_n]$, and $\theta_2 = [b_1, \ldots, b_m]$. Then we know that there is a j, such that a_1 and b_j are not orthogonal. We modify the variables as follows:

$$\begin{aligned}
\pi &= \pi + [a_1 \wedge b_j] \\
v &= v + a_1 \wedge b_j \\
\theta_1 &= \begin{cases} [a_1 - a_1 \wedge b_j, a_2, \ldots a_n] \text{ if } a_1 \wedge b_j \neq a_1 \\ [a_2, \ldots, a_n] \text{ otherwise.} \end{cases} \\
\theta_2 &= \begin{cases} [b_1, \ldots, b_{j-1}, b_j - a_1 \wedge b_j, b_{j+1}, \ldots, b_m] \text{ if } a_1 \wedge b_j \neq b_j \\ [b_1, \ldots, b_{j-1}, b_{j+1}, \ldots, b_m] \text{ otherwise.} \end{cases} \\
x &= x - a_1 \wedge b_j
\end{aligned}$$

At every step of the procedure presented above, the quantity:

$$\mathrm{card}((i,j) \mid \text{ not } (a_i \perp b_j))$$

decreases. Indeed:

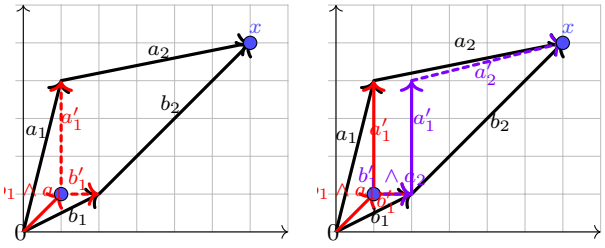
- or we remove either a_1 of Θ_1, or b_j of Θ_2, and then the statement above holds.
- or we replace a_1 by $(a_1 - a_1 \wedge b_j)$, and b_j by $(b_j - a_1 \wedge b_j)$. Then we see that $(a_1 - a_1 \wedge b_j) \perp (a_1 - a_1 \wedge b_j)$. Moreover, the pairs that were orthogonal before are still orthogonal: indeed for every z with $z \perp a_1$ it holds that $z \perp a_1 - a_1 \wedge b_j$, and the same for b_j.

As a consequence, the procedure will terminates. It means that we reach a state where Θ_1 is empty, and all the invariants presented above hold. Then we see that $\pi \in \mathrm{Parts}(x)$, and $\pi \leq \pi_1, \pi_2$.

We are going to illustrate the procedure above on a very basic example. We consider the cone consisting of the positive quadrant of $\mathbb{R}^2$, endowed by the order defined as: $x \leq y$ if $x_1 \leq y_1$, and $x_2 \leq y_2$. We take two partitions of a vector $x \in \mathbb{R}^2$: $\pi_1 = [b_1, b_2]$, and $\pi_2 = [a_1, a_2]$, where a_1, a_2, b_1, b_2 are taken as pictured in Figure 11.1a. We are going to apply our procedure in order to obtain a refinment of both π_1 and π_2. At the beginning, we have $\Theta_1 = \pi_1$, $\Theta_2 = \pi_2$, $w = x$, $v = 0$.

- The first step is represented in Figure 11.1a. Observe that the procedure is actually non-deterministic: we may choose any (a, b) with $a \in \pi_1$, $b \in \pi_2$, and a and b not orthogonal. Here, we choose to start from (b_1, a_1). We take $v = a_1 \wedge b_1$ (and we represent it by a red vector in Figure 11.1a): it is going to be the first element of our new partition π. Accordingly, we take $\pi = [v]$. We know update the partition Θ_1 and Θ_2 into partitions of $w = x - v$: Θ_2 becomes $[a_1', a_2]$, and Θ_1 becomes $[b'1, b_2]$ where $a_1' = a_1 - b_1 \wedge a_1$ and $b_1' = b_1 - a_1 \wedge b_1$ are represented also in red in Figure 11.1a.

- The second step is represented in Figure 11.1b. Observe that now a'_1 and b'_1 are orthogonal, so we have to choose another pair. Here, we choose (b'_1, a_2). As before, we add to π the glb of b'_1 and a_2: we obtain $\pi = [a_1 \wedge b_1, b'_1 \wedge a_2]$. Observe that now (as can be seen on Figure 11.1b, $b'_1 \leq a_2$, and so $b'_1 \wedge a_2 = b'_1$. So when we update the partition Θ_1 and Θ_2, we take: $\Theta_2 = [b_2]$, and $\Theta_1 = [a'1, a'_2]$ where $a'_2 = a_2 - b'1$ is represented in purple in Figure 11.1b.
- By doing again two steps of the procedure, we see that the final partition π is $[a_1 \wedge b_1, b'_1, a'_1, a'_2]$. We can see by looking at Figure 11.1b that it is indeed a refinment of both π_1 and π_2..



(a) First step of the Procedure

(b) Second Step of the Procedure

Figure 11.1: Illustration of the Proof of Lemma11.1.2

□

Observe that, as a consequence, the refinement preorder turns also Parts($\vec{u}$) into a directed set.

Definition 11.1.3 (from [80]) *Let C be a lattice cone, D a cone, and let $f : \mathscr{B}C \to D$ be a pre-stable function. Then for every $x \in \mathscr{B}C$, and $\vec{u} = (u_1, \ldots, u_n) \in \mathscr{B}C^n_x$, we define $\Phi^{f,n}_{x,\vec{u}} : Parts(\vec{u}) \to D$ as:*

$$\Phi^{f,n}_{x,\vec{u}}(\pi_1, \ldots \pi_n) = \sum_{y_1 \in \pi_1} \cdots \sum_{y_n \in \pi_n} \Delta^n f(x \mid y_1, \ldots, y_n).$$

It holds (see [80] for more details) that $\Phi^{f,n}_{x,\vec{u}}$ is a non-increasing function whenever f is pre-stable (it is shown in Lemma 3.2 of [80] by looking at the definition of higher-order differences). Since Parts($\vec{u}$) is a directed set (by Lemma 11.1.2), $\Phi^{f,n}_{x,\vec{u}}$ has a greatest lower bound whenever D is a directed-complete cone.

Definition 11.1.4 (from [80]) *Let be C a lattice cone, D a directed-complete lattice cone, and $f : \mathscr{B}C \to D$ a pre-stable function. Let be $\vec{u} \in \mathscr{B}C^n_x$. Then the* derivative of f in x at rank n towards the direction $\vec{u}$ *is the function* $\mathbf{D}^n f(x \mid \cdot) : \mathscr{B}C^n_x \to D$ *defined as*

$$\mathbf{D}^n f(x \mid \vec{u}) = \inf_{\vec{\pi} \in Parts(\vec{u})} \Phi^f_{x,\vec{u}}(\vec{\pi}).$$

In order to highlight the link with differentiation in real analysis, we illustrate Definition 11.1.4 on the basic case where $f : \mathbb{R}_+ \to \mathbb{R}_+$,

Example 11.1.1 *We take C and D as the positive real half-line, and $x \in [0,1)$. Let be h such that $x + h \leq 1$. Then:*

$$\mathbf{D}^1 f(x \mid h) = \inf_{\pi \text{ with } \pi \sim h} \sum_{y \in \pi} f(x+y) - f(x).$$

Since f is pre-stable hence an absolutely monotonic function on reals, f is convex, and moreover differentiable (see [124]). From there, by considering a particular family of partitions, we can show that $\mathbf{D}^1 f(x \mid h) = h \cdot f'(x)$.

Proof. *First, let π be any partition of y. Since f is differentiable and convex, it holds that:*

$$\forall z,\ f(x+z) - f(x) \geq f'(x) \cdot z.$$

As a consequence, we see that for any partition π of h, it holds that $\sum_{y \in \pi} f(x+y) - f(x) \geq f'(x) \cdot h$, and it implies that $\mathbf{D}^1 f(x \mid h) \geq f'(z) \cdot h$. To show the reverse inequality, it is enough to consider the particular family of partition $\pi_n = [\frac{h}{n}, \ldots, \frac{h}{n}]$ of h: we see that

$$\begin{aligned} \sum_{y \in \pi_n} f(x+y) - f(x) &= n \cdot f(x + \frac{h}{n}) - f(x) \\ &= h \cdot \frac{f(x + \frac{h}{n}) - f(x)}{\frac{h}{n}} \to_{n \to \infty} h \cdot f'(x). \end{aligned}$$

□

Lemma 11.1.3 (from [80]) *Let C be a lattice cone, D a directed complete cone, f a pre-stable function from C to D. Let be $x \in \mathscr{B}^\circ C$. Then $\mathbf{D}^n f(x \mid \cdot)$ is a symmetric function $\mathscr{B}(C_x^n) \to D$ such that moreover:*

- $0 \leq \mathbf{D}^n f(x \mid \vec{u}) \leq \Delta^n f(x \mid \vec{u})$;
- *both $\vec{u} \mapsto \mathbf{D}^n f(x \mid \vec{u})$ and $\vec{u} \mapsto \Delta^n f(x \mid \vec{u}) - \mathbf{D}^n f(x \mid \vec{u})$ are pre-stable functions from C_x^n to D.*

Proof. The proof is given in Lemma 3.31 in [80]. It comes almost directly from Definition 11.1.4. □

We have seen in Example 11.1.1 that our so-called derivatives of pre-stable functions play the same role as the differential of a differentiable function, which are actually *linear* operators $df_x^n : \mathbb{R}^n \to \mathbb{R}$. While the abstract domains considered in [80] do not have to be $\mathbb{R}_+$ semi-modules, so have no notion of linearity, we are able to show in our case that the $\mathbf{D}^n f$ are linear in the sense of Lemma 11.1.4 below.

Lemma 11.1.4 *Let C, D be two directed complete lattice cones, $x \in \mathscr{B}^\circ C$.*

- *Let $f : \mathscr{B}C \to D$ be a pre-stable function. Then $\mathbf{D}^n f(x \mid \cdot) : \mathscr{B}(C_x^n) \to D$ is n-linear, in the sense that:*

$$\begin{aligned} \mathbf{D}^n f(x \mid u_1, \ldots, \lambda \cdot v + w, \ldots u_n) = \lambda \cdot \mathbf{D}^n f(x \mid u_1, \ldots, v, \ldots u_n) \\ + \mathbf{D}^n f(x \mid u_1, \ldots, w, \ldots u_n). \end{aligned}$$

- *For any $\vec{u} \in \mathscr{B}(C_x^n)$, the function $f \in \mathbf{Cstab}(C, D) \mapsto \mathbf{D}^n f(x \mid \vec{u}) \in D$ is linear and directed Scott-continuous.*

Proof. We are going to use the following auxiliary lemma:

Lemma 11.1.5 (from [80]) *Let C and D be two directed cones, and $f : C \to D$ linear and non-decreasing, such that moreover for all subset F of C directed for the reverse order, $f(\inf F) = \inf f(F)$. Then f is directed Scott-continuous.*

Proof. Let be E a directed subset of C. We define $F = \{\sup E - x \mid x \in E\}$. Since E is directed, F is directed for the reverse order, and as a consequence:$\inf f(F) = f(\inf F)$. But we see that $\inf F = 0$. Therefore, since f is linear, $f(\inf F) = 0$. As a consequence (and again by linearity of f): $f(\sup E) - \sup f(E) = \inf f(F) = 0$. □

We are now going to show Lemma 11.1.4.

- We first show that $\mathbf{D}^n f(x \mid \cdot) : \mathscr{B}(C_x^n) \to D$ is n-linear. The additivity is given by Lemma 3.72 of [80]. The commutation with scalar multiplication is not proved on this form in [80] because they have a more general notion of a system of partition. We first show that the result holds when λ is a rational number. To do that, we use the fact that $\pi = [\frac{x}{n}, \dots, \frac{x}{n}]$ is always a partition of x. Then, let $\lambda \in \mathbb{R}_+$ and $\vec{u} = (u_1, \dots, u_n)$ such that both $\vec{u}$ and $\vec{v} = (u_1, \dots, \lambda u_i, \dots, u_n)$ are in $\mathscr{B}C_x^n$. Let be $\overline{r} = (r_m)_{m\in\mathbb{N}}, \overline{q} = (q_m)_{m\in\mathbb{N}}$ two sequences of rational number such that $\overline{r}$ tends to λ by below, and $\overline{q}$ tends to λ by above. We see that:

$$\mathbf{D}^n f(x \mid \vec{v}) = 2 \cdot \mathbf{D}^n f(x \mid u_1, \dots, \frac{\lambda}{2} \cdot u_i, \dots u_n).$$

  We take N such that for every $m \geq N$, $q_m \leq 2 \cdot \lambda$: since $\mathbf{D}^n f(x \mid \cdot)$ is non-decreasing, we see that:

$$\begin{aligned}
&\mathbf{D}^n f(x \mid u_1, \dots, \frac{r_m}{2} \cdot u_i, \dots, u_n) \\
&\quad \leq \mathbf{D}^n f(x \mid u_1, \dots, \frac{\lambda}{2} \cdot u_i, \dots, u_n) \\
&\quad \leq \mathbf{D}^n f(x \mid u_1, \dots, \frac{q_m}{2} \cdot u_i, \dots, u_n)
\end{aligned}$$

  Applying now the linearity for rational numbers, we see that for every $m \geq N$:

$$\begin{aligned}
&r_m \cdot \mathbf{D}^n f(x \mid u_1, \dots, \frac{1}{2} \cdot u_i, \dots, u_n) \\
&\quad \leq \mathbf{D}^n f(x \mid u_1, \dots, \frac{\lambda}{2} \cdot u_i, \dots, u_n) \\
&\quad \leq q_m \cdot \mathbf{D}^n f(x \mid u_1, \dots, \frac{1}{2} \cdot u_i, \dots, u_n)
\end{aligned}$$

  As a consequence:

$$\begin{aligned}
&\sup_{m\geq N} r_m \cdot \mathbf{D}^n f(x \mid u_1, \dots, \frac{1}{2} \cdot u_i, \dots, u_n) \\
&\quad \leq \mathbf{D}^n f(x \mid u_1, \dots, \frac{\lambda}{2} \cdot u_i, \dots, u_n) \\
&\quad \leq \inf_{m\in N} q_m \mathbf{D}^n f(x \mid u_1, \dots, \frac{1}{2} \cdot u_i, \dots, u_n)
\end{aligned}$$

and by Scott-continuity of $\cdot$, it tells us that $\mathbf{D}^n f(x \mid u_1, \ldots, \frac{\lambda}{2}\cdot u_i, \ldots, u_n) = \lambda \mathbf{D}^n f(x \mid u_1, \ldots, \frac{1}{2} \cdot u_i, \ldots, u_n)$. We can now conclude: recall that $\mathbf{D}^n f(x \mid \vec{v}) = 2 \cdot \mathbf{D}^n f(x \mid u_1, \ldots, \frac{\lambda}{2} \cdot u_i, \ldots u_n)$. Therefore:

$$\begin{aligned}\mathbf{D}^n f(x \mid \vec{v}) &= 2 \cdot \lambda \cdot \mathbf{D}^n f(x \mid u_1, \ldots, \frac{1}{2} \cdot u_i, \ldots, u_n) \\ &= \lambda \cdot \mathbf{D}^n f(x \mid \vec{u}) \quad \text{since } \frac{1}{2} \in \mathbb{Q}.\end{aligned}$$

- We show now that $f \in \mathbf{Cstab}(C, D) \mapsto \mathbf{D}^n f(x \mid \vec{u}) \in D$ is linear and Scott-continuous. It is immediate that it is linear, since every one of the $f \mapsto \Delta^n f(x \mid u)$ is. We are now going to use 11.1.5 to show the Scott-continuity: it tells us that we have only to check that for every $E \subseteq C \Rightarrow_m D$ directed for the reverse order, $\mathbf{D}^n f(\inf E \mid \vec{u}) = \inf \mathbf{D}^n f(E \mid \vec{u})$. Observe that:

$$\begin{aligned}\mathbf{D}^n(\inf E)(x \mid \vec{u}) &= \inf_{\vec{\pi} \in \mathrm{Parts}(\vec{u})} \sum_{y_1 \in \pi_1} \cdots \sum_{y_n \in \pi_n} \Delta^n(\inf E)(x \mid y_1, \ldots, y_n) \\ &= \inf_{\vec{\pi} \in \mathrm{Parts}(\vec{u})} \sum_{y_1 \in \pi_1} \cdots \sum_{y_n \in \pi_n} \inf_{f \in E} \{\Delta^n f(x \mid y_1, \ldots, y_n)\} \\ &= \inf_{\vec{\pi} \in \mathrm{Parts}(\vec{u})} \inf_{f \in E} \{ \sum_{y_1 \in \pi_1} \cdots \sum_{y_n \in \pi_n} \Delta^n f(x \mid y_1, \ldots, y_n)\} \\ &= \inf_{f \in E} \mathbf{D}^n f(x \mid \vec{u}) \text{ since the infs can be exchanged.}\end{aligned}$$

□

The linearity of the derivatives means that for every $x \in \mathscr{B}^\circ C$, we can extend $\mathbf{D}^n f(x \mid \cdot)$ to a function $C_n^x \to D$. We will use implicitly this extension in the following, especially in Definition 11.1.5.

11.1.3 Taylor Series for pre-stable functions

We have seen above that the $\mathbf{D}^n f$ are a notion of differential for pre-stable functions. Following further this idea, McMillan defined an analogue to the Taylor expansion. We give here a slightly different, but equivalent, formulation, taking advantage of the linearity of the derivatives for cones (the original formulation, as well as the proof of equivalence, are detailed in the long version). In all this section C and D are going to be directed complete lattice cones, and $f : \mathscr{B}C \to D$ a pre-stable function.

Definition 11.1.5 *Let be $x \in \mathscr{B}^\circ C$. We call* Taylor partial sum of f in x at the rank N *the function $Tf^N(x \mid \cdot) : \mathscr{B}C_x^1 \to D$ defined as:*

$$Tf^N(x \mid y) = f(x) + \sum_{k=1}^{N} \frac{1}{k!} \mathbf{D}^k f(x \mid y, \ldots, y).$$

The next step consists in establishing that the $T^N f$ are actually a non-increasing bounded sequence in the cone of pre-stable functions from C to D, which will allow to define the *Taylor series of f*, as the supremum of the $T^N f$.

To that end, we are first going to establish an alternative characterization of the Taylor series, which is the one used in [80], in the framework of abstract domains. It consists in substituting each of the $\mathbf{D}^n f(x \mid y, \ldots, y)$ with its expression given by Lemma 11.1.6 below. The validity of Lemma 11.1.4, and thus the equivalence of the two definitions, depends on the fact we work with directed-complete cones.

Lemma 11.1.6 (Alternative Caracterisation of Derivatives) *Let $x \in \mathscr{B}^\circ C, y \in \mathscr{B}C^1_x$, and $k \in \mathbb{N}$. Then it holds that $\mathbf{D}^k f(x \mid y, \dots, y)$ is equal to:*

$$\sup_{\pi=[u_1,\dots,u_n]\in Parts(y)} \sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,n]\!]} \mathbf{D}^k f(x \mid u_{\sigma(1)}, \dots u_{\sigma(n)})$$

Proof. We first introduce the following notation: if $\pi = (u_1, \dots, u_n)$ is a partition of x, and $\sigma : [\![1,k]\!] \hookrightarrow [\![1,n]\!]$ an injective function, we denote $\sigma(\pi) = (u_{\sigma(1),\dots,u_\sigma(k)})$. We denote by $A = \sup_{\pi\in\mathrm{Parts}(y)} \sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,n]\!]} \mathbf{D}^k f(x \mid \sigma(\pi))$. We show separately the two inequalities.

- We first show that $A \geq \mathbf{D}^k f(x \mid y, \dots, y)$. For every $n \in \mathbb{N}$, it holds that $\pi = (\frac{1}{n} \cdot y, \dots, \frac{1}{n} \cdot y)$ is a partition of y. Therefore for every $n \in \mathbb{N}$:

$$\begin{aligned} A &\geq \sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,n]\!]} \mathbf{D}^k f(x \mid \frac{y}{n}, \dots, \frac{y}{n}) \\ &= \frac{n!}{(n-k)!} \mathbf{D}^k f(x \mid \frac{y}{n}, \dots, \frac{y}{n}) \\ &= \frac{n!}{(n-k)! \cdot n^k} \mathbf{D}^k f(x \mid y, \dots, y) \end{aligned}$$

  The sequence $\frac{n!}{(n-k)!\cdot n^k}$ tends to 1 when n tends to infinity (see in the long version). By Scott-continuity, it means that $A \geq \mathbf{D}^k f(x \mid y, \dots, y)$.

- Let us show now that $A \leq \mathbf{D}^k f(x \mid y, \dots, y)$. Let be $\pi = (u_1, \dots, u_n) \in \mathrm{Parts}(y)$. Then:

$$\begin{aligned} &\sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,n]\!]} \mathbf{D}^k f(x \mid \sigma(\pi)) \\ &\leq \sum_{i_1\in\{1,n\}} \dots \sum_{i_k\in\{1,\dots,n\}} \mathbf{D}^k f(x \mid u_{i_1}, \dots u_{i_k}) \\ &= \mathbf{D}^k f(x \mid y, \dots, y) \text{ by } n\text{-linearity of } \mathbf{D}^k f(x \mid \cdot) \end{aligned}$$

  Since $A = \sup_{\pi\in\mathrm{Parts}(y)} \sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,n]\!]} \mathbf{D}^k f(x \mid \sigma(\pi))$, we see that $A \leq \mathbf{D}^k f(x \mid y, \dots, y)$, which ends the proof.

□

With this characterization, [80] shows that the sequence of functions $(x \in \mathscr{B}C^1_y \mapsto Tf^n(x \mid y))$ is bounded by $(x \in \mathscr{B}C^1_y \to f(x+y))$ in the cone of pre-stable functions from C^1_x to D.

Lemma 11.1.7 *Let be y is in $\mathscr{B}^\circ C$, and x in $\mathscr{B}C^1_y$. Then $\forall N \in \mathbb{N}$, $Tf^N(x \mid y) \leq f(x+y)$, and the function $(x \in \mathscr{B}C^1_y \mapsto f(x+y) - Tf^N(x \mid y))$ is pre-stable.*

Proof. Let be $x \in \mathscr{B}^\circ C$ and $y \in \mathscr{B}C_x$. We are able to express $f(x+y)$ by using $f(x)$ and finite differences on any partition of y: indeed, for every partition π of y, it holds that:

$$f(x+y) = f(x) + \sum_{1\leq k\leq n} \frac{1}{k!} \sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,n]\!]} \Delta^k f(x, \sigma(\pi)) \tag{11.1}$$

Proof. It is an algebraic calculation, done in [80]. We give here the proof for $n = 2$. Let $\pi = [y_1, y_2]$ a partition of y. Then we see that:

$$\begin{aligned} f(x) + \sum_{1\leq k\leq n} \frac{1}{k!} \sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,n]\!]} \Delta^k f(x, \sigma(\pi)) \\ = f(x) + \Delta^1 f(x, y_1) + \Delta^1 f(x, y_2) \\ + \frac{1}{2} \cdot (\Delta^2 f(x, y_1, y_2) + \Delta^2 f(x, y_2, y_1)) \\ = f(x) + (f(x + y_1) - f(x)) + (f(x + y_2) - f(x)) \\ + (f(x + y_1 + y_2) - f(x + y_1) - f(x + y_2) + f(x)) \\ = f(x + y_1 + y_2) = f(x + y). \end{aligned}$$

□

Moreover we are also able to express the derivatives of f at x towards the direction y also using the partitions of y (it is the sense of Lemma 11.1.6). Accordingly:

$$\begin{aligned} Tf^N(x \mid y) = f(x) + \sum_{k=1}^{N} \frac{1}{k!} \mathbf{D}^k f(x \mid y, \dots, y) \\ = f(x) + \sum_{k=1}^{N} \frac{1}{k!} \sup_{\pi\in \mathrm{Parts}(y)} \sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,n]\!]} \mathbf{D}^k f(x \mid \sigma(\pi)) \end{aligned}$$

Using Lemma 11.1.3, we see that it implies:

$$Tf^N(x \mid y) \leq f(x) + \sum_{k=1}^{N} \frac{1}{k!} \sup_{\pi\in \mathrm{Parts}(y)} \sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,\#(\pi)]\!]} \Delta^k f(x \mid \sigma(\pi))$$

We can now use the Scott continuity of $+$ and $\cdot$, and we obtain:

$$Tf^N(x \mid y) \leq \sup_{\pi\in \mathrm{Parts}(y)} f(x) + \sum_{k=1}^{N} \frac{1}{k!} \sum_{\sigma:[\![1,k]\!]\hookrightarrow[\![1,\#(\pi)]\!]} \Delta^k f(x \mid \sigma(\pi))$$

We can now conclude using (11.1):

$$Tf^N(x \mid y) \leq \sup_{\pi\in \mathrm{Parts}(y)} f(x + y) \leq f(x + y)$$

The proof of the pre-stability of the function can be found in [80]. It is based on the fact that each one of the above inequality can be seen as an inequality in the cone of pre-stable functions. □

Since we have shown that the partial sum of the Taylor series of f was a bounded non-decreasing sequence in the sequentially complete cone of pre-stable functions from C^1_x to D, we can now define the *Taylor series of* f as its supremum.

Definition 11.1.6 *We define* $Tf(x \mid \cdot) : \mathscr{B}C^1_x \to D$ the Taylor series of f in x *and* $Rf(x \mid \cdot) : \mathscr{B}C^1_x \to D$ the Remainder of f in x *as:*

$$\begin{aligned} Tf(x \mid y) = \sup_{N\in\mathbb{N}} Tf^N(x \mid y) \\ Rf(x \mid y) = f(x + y) - Tf(x \mid y). \end{aligned}$$

11.1.4 Extended Bernstein's theorem

Our goal from here is to show that for any $x \in \mathscr{B}^\circ C$, $Rf(0 \mid x) = 0$. We recall here the main steps of the proof of [80]. It is based on two technical lemmas, that analyze more precisely the behavior of the remainder of f. The first one is actually a summary of several technical results shown separately in [80].

Lemma 11.1.8 *Let be $x \in \mathscr{B}^\circ C$. Then it holds that both:*

$$\begin{aligned} & Rf_x : y \in \mathscr{B}C^1_x \mapsto Rf(x \mid y) \in D \\ \text{and} \quad & Rf^y : x \in \mathscr{B}C^1_y \mapsto Rf(x \mid y) \in D \end{aligned}$$

are pre-stable functions. Moreover $Rf_x(0) = 0$, and for every $x \in \mathscr{B}C^1_y$, it holds that $T(Rf^y)(0 \mid x) = 0$.

Proof. We give here only sketches of the proofs. The detailed proof can be found in [80].

- For Rf^y, it is a consequence of the fact that both $f^y : (x \in \mathscr{B}C^1_y \mapsto f(x+y)$ and $Tf^y : x \in \mathscr{B}C^1_y \to Tf(x \mid y)$ are pre-stable functions, with $Tf^y \leq f^y$ in the cone of pre-stable functions, and $Rf^y = f^y - Tf^y$.
- The pre-stability of Rf_x is stated in Theorem 4.1. of [80]. It is based on a previous technical lemma shown in [80], which says it is sufficient for a function to be pre-stable, to have all its differences *in 0* to be non-negative. Then the idea is to fix x, and to consider for every $N \in \mathbb{N}$, the function $g_N : y \in \mathscr{B}C^1_x \mapsto f(x+y) - Tf^N(x \mid y)$. It is then possible to show that for any $n \in \mathbb{N}$, and $\vec{u} \in \mathscr{B}C^n_y$, $\Delta^n g_N(0 \mid \vec{u}) = \Delta^n f(x \mid \vec{u}) - \Delta^n(Tf^N_x)(0 \mid \vec{u})$, with $Tf^N_x : y \mapsto Tf^N(x \mid y)$. By a computation on the $\Delta^n(Tf^N_x)(0 \mid \vec{u})$, we see that the $\Delta^n g_N(0 \mid \vec{u})$ are non-negative. Then, we conclude using the fact that $Rf_x(y) = \inf_{N \in \mathbb{N}} Tf^N_x(y)$.
- The fact that $Rf_x(0) = 0$ is a direct consequence of the n-linearity of the map $\vec{u} \mapsto \mathbf{D}^n f(x \mid \vec{u})$ for $n \geq 1$.
- The fact that $T(Rf^y)(0 \mid x) = 0$ is shown in [80] in Lemma 5.26. It is based on the fact that the Scott-continuity of $f \mapsto \mathbf{D}^n f(x \mid \vec{u})$ allows us to show that if we take $g(y) = \mathbf{D}^n f(x \mid \vec{u})$, then $\mathbf{D}^k g(x \mid \vec{v}) = \mathbf{D}^{n+k} f(y_0 \mid \vec{u}, \vec{v})$, and from there to compute the Taylor series of Rf^y.

□

The second technical lemma gives us a way to decompose $Rf(x \mid y)$ into smaller pieces. It is stated in Theorem 5.3 in [80].

Lemma 11.1.9 *Let be x, y such that $x + y \in \mathscr{B}C$. Then $Rf(0 \mid x + y) \leq Rf(y \mid x) + Rf(x \mid y)$, and furthermore $Rf(0 \mid x+y) \geq Rf(x \mid y)$, and $Rf(0 \mid x+y) \geq Rf(y \mid x)$, and moreover all the inequality are in the cone of pre-stable functions.*

Proof. We give here a brief sketch of the proof of the first statement. More details can be found in [80]. We introduce the function $Rf^{+x} : y \in \mathscr{B}C^1_x \mapsto Rf(0 \mid x + y)$. The proof is based on the fact that it is possible to establish (see [80]):

$$T(Rf^{+x})(0 \mid y) = T(Rf^x)(0 \mid y) \tag{11.2}$$

$$\text{and } R(Rf^{+x})(0 \mid y) = R(Rf_x)(0 \mid y) \tag{11.3}$$

As a consequence, we can write:

$$\begin{aligned}
&Rf(x \mid y) + Rf(y \mid x) = Rf_x(y) + Rf^x(y) \\
&= T(Rf_x)(0 \mid y) + R(Rf_x)(0 \mid y) + T(Rf^x)(0 \mid y) + R(R^x_f)(0 \mid y) \\
&= T(Rf_x)(0 \mid y) + R(Rf^{+x})(0 \mid y) + T(Rf^{+x})(0 \mid y) + R(R^x_f)(0 \mid y) \\
&\quad \text{by (11.2) and (11.3)} \\
&= Rf^{+x}(y) + T(Rf_x)(0 \mid y) + R(R^x_f)(0 \mid y) \\
&\geq Rf^{+x}(y) = Rf(0 \mid x + y),
\end{aligned}$$

and we see that we have also shown that the difference is pre-stable. The other two statement are shown in a similar way. □

We use Lemma 11.1.9 to show the a more involved upper bound on $Rf(0 \mid x)$.

Lemma 11.1.10 *Let be $x \in \mathscr{B}C$, and $\pi = [x_1, \ldots, x_n]$ a partition of x, such that for every $x_i \in \pi$, $x + x_i \in \mathscr{B}C$. Then $Rf(0 \mid x) \leq \sum_{1 \leq i \leq n} \inf_{\pi_i \mid \pi_i \sim x_i} \sum_{z \in \pi_i} Rf(x \mid z)$.*

Proof. For every $x \in \mathscr{B}C$, we denote $g_x : y \in \mathscr{B}C^1_x \mapsto f(x + y)$. From the definitions of the $\mathbf{D}^n$, we see that it holds that $Rg_x(0 \mid y) = Rf(x \mid y)$.

Let be $\pi_1, \ldots, \pi_n$ such that π_i is a partition of x_i over J. Then $\pi_1 + \ldots + \pi_n$ is a partition of x. Lemma 11.1.9 applied several times , combined with the fact that $Rg_x(0 \mid y) = Rf(x \mid y)$, tells us that:

$$Rf(0 \mid x_0) \leq \sum_{z \in \pi_1 + \ldots + \pi_n} Rf(z' \mid z),$$

where $z' = \sum_{u \in \pi_1 + \ldots + \pi_n \mid u \neq z} u$. Moreover, we know that Rf^z is pre-stable (by lemma 11.1.8). Since, for every $z \in \pi_1 + \ldots + \pi_n$, $z' \leq x$ (it is immediate, since $\pi_1 + \ldots + \pi_n$ is a partition of x), it folds that $Rf(z' \mid z) = Rf^z(z') \leq Rf^z(x) = Rf(x \mid z)$. As a direct consequence, we see that $Rf(0 \mid x_0) \leq \sum_i \sum_{z \in \pi_i} Rf(x \mid z)$, which leads to the result. □

We are now ready to show the main result of this section.

Proposition 11.1.11 (Extended Bernstein's Theorem) *Let be C, D directed-complete lattice cones, and $f : \mathscr{B}C \to D$ a pre-stable function. Then for every $x \in \mathscr{B}^\circ C$, it holds that $f(x) = Tf(0 \mid x)$.*

Proof. Let be $x \in \mathscr{B}C$. First, we consider the partition $\pi = [\frac{x}{N}, \ldots, \frac{x}{N}]$ of x, with N taken such as $x + \frac{x}{N} \in \mathscr{B}C$. We know that such an N exists since x is in the open unit ball $\mathscr{B}^\circ C$. We use Lemma 11.1.10 on $Rf(0 \mid x)$, and the partition π, and it tells us that:

$$Rf(0 \mid x) \leq \sum_{1 \leq j \leq N} \inf_{\pi = (u_1, \ldots, u_n) \in \mathrm{Parts}(\frac{x}{N})} \sum_{1 \leq i \leq n} Rf(x \mid u_i). \tag{11.4}$$

Observe that the above expression is valid, since for every u_i in a partition π of $\frac{x}{N}$, $x + u_i \in \mathscr{B}C$. We know, by Lemma 11.1.8 that $Rf(x \mid 0) = 0$. Therefore, we can rewrite (11.4) as:

$$Rf(0 \mid x) \leq \sum_{1 \leq j \leq N} \inf_{\pi = (u_1, \ldots, u_n) \in \mathrm{Parts}(\frac{x}{N})} \sum_{1 \leq i \leq n} Rf(x \mid u_i) - Rf(x \mid 0). \tag{11.5}$$

Moreover, we are able to express the right part of (11.5) by the finite differences of the pre-stable function Rf_x; indeed for each i,

$$\Delta^1 Rf_x(0, u_i) = Rf(x \mid u_i) - Rf(x \mid 0). \tag{11.6}$$

By the definition of derivatives (see Definition 11.1.4), we see that

$$\mathbf{D}^1 Rf_x(0 \mid \frac{x}{N}) = \inf_{\pi \in \mathrm{Parts}(\frac{x}{N})} \sum_{v \in \pi} \Delta^1 Rf_x(0, v) \tag{11.7}$$

We see now that combining (11.5), (11.6) and (11.7) leads us to $Rf(0 \mid x) \leq \sum_{1 \leq j \leq N} \mathbf{D}^1 Rf_x(0 \mid \frac{x}{N})$. Moreover, we know that for every $y \in \mathscr{B}C^1_x$, $\mathbf{D}^1 Rf_x(0 \mid y) \leq T(Rf_x)(0 \mid y)$. Hence by using again Lemma 11.1.8, which says that $T(Rf_x)(0 \mid \frac{x}{N}) = 0$, it holds that $Rf(0 \mid x) = 0$. □

11.2 Cstab is a conservative extension of PCoh!

Probabilistic coherence spaces (PCSs) were introduced by Ehrhard and Danos in [34] as a model of higher-order probabilistic computation. We presented them in Chapter 9. In this section, we highlight an embedding from PCSs into cones. As highlighted in Example 4.4 from [45], we can associate in a generic way a cone to any PCS. The idea is that we consider the extension of the space of cliques by all uniform scaling by positive reals. We formalize this idea in Definition 11.2.1 below.

Definition 11.2.1 *Let be $\mathcal{X}$ a PCS. We define a cone $C_{\mathcal{X}}$ as the $\mathbb{R}_+$ semi-module $\{\alpha \cdot x$ s.t. $\alpha \geq 0, x \in \mathrm{P}(\mathcal{X})\}$ where the $+$ is the usual addition on vectors. We endow it with $\| \cdot \|_{C_{\mathcal{X}}}$ defined by:*

$$\|x\|_{C_{\mathcal{X}}} = \sup_{y \in \mathrm{P}(\mathcal{X}^\perp)} \langle x, y \rangle = \inf\{\frac{1}{r} \mid r \cdot x \in \mathrm{P}(\mathcal{X})\}.$$

It is easily seen that it is indeed a cone—observe that the proof uses the so-called technical conditions from Definition 9.2.1. Moreover, we can see that $\mathscr{B}C_{\mathcal{X}}$ consists exactly in the set $\mathrm{P}(\mathcal{X})$ of cliques of $\mathcal{X}$. Looking at the cone order $\preceq_{C_{\mathcal{X}}}$, as defined in Definition 9.3.2, we see that it coincides on $\mathrm{P}(\mathcal{X})$ with the pointwise order in $\mathbb{R}_+|\mathcal{X}|$. It is relevant since we know already from [34] that $\mathrm{P}(\mathcal{X})$ is a bounded-complete and ω-continuous cpo with respect to this pointwise order.

Lemma 11.2.1 *For every PCS $\mathcal{X}$, it holds that $C_{\mathcal{X}}$ is a directed-complete lattice cone.*

Proof. To show that $C_{\mathcal{X}}$ is directed complete, we use the fact that $\mathrm{P}(\mathcal{X})$ is a complete partial order. To show that it is a lattice, we see that $x \vee y$ can be defined as: $(x \vee y)_a = \max x_a, y_a \, \forall a \in |\mathcal{X}|$. □

11.2.1 A fully faithful functor $\mathcal{F} : \mathbf{PCoh}_! \to \mathbf{Cstab}_\mathbf{m}$.

Recall that Definition 11.2.1 gave a way to see a PCS as a cone. Moreover, as stated in Proposition 11.2.2 below, a morphism in **PCoh**! can also be seen as a stable function, in the sense that $\mathcal{E}^{\mathcal{X},\mathcal{Y}} \subseteq \mathbf{Cstab}(C_{\mathcal{X}}, C_{\mathcal{Y}})$.

Proposition 11.2.2 *Let be $f \in \mathbf{PCoh}_!(\mathcal{X}, \mathcal{Y})$. Then $\widetilde{f}$ is a stable function from $C_{\mathcal{X}}$ to $C_{\mathcal{Y}}$.*

Proof. We know from [34] that $\widetilde{f} : \mathrm{P}(\mathcal{X}) \to \mathrm{P}(\mathcal{Y})$ is sequentially Scott-continuous with respect to the orders $\preceq_{C_\mathcal{X}}$, $\preceq_{C_\mathcal{Y}}$. Moreover $\widetilde{f}$ is pre-stable: it comes from the fact that $\widetilde{f}$ can be written as a power series with all its coefficients non-negative. Finally, we have to show that $\widetilde{f}(\mathscr{B}C_\mathcal{X}) \subseteq \mathscr{B}C_\mathcal{Y})$. Since $\mathscr{B}C_\mathcal{X} = \mathrm{P}(\mathcal{X})$, $\mathscr{B}C_\mathcal{X} = \mathrm{P}(\mathcal{X})$, and moreover f is a morphism in $\mathbf{PCoh}_!(\mathcal{X}, \mathcal{Y})$, we see that the result holds. □

Thus we can define a functor $\mathcal{F} : \mathbf{PCoh}_! \to \mathbf{Cstab}$, by taking $\mathcal{F}\mathcal{X} = C_\mathcal{X}$, and $\mathcal{F}f = \widetilde{f}$. Our goal now is to show that $\mathcal{F}$ is full and faithful, which will make $\mathbf{PCoh}_!$ a full subcategory of **Cstab**. As mentioned before, it was shown in [34] that $\widetilde{\cdot}$ is a bijection from $\mathbf{PCoh}_!(\mathcal{X}, \mathcal{Y})$ to $\mathcal{E}^{\mathcal{X},\mathcal{Y}}$. It tells us directly that $\mathcal{F}$ is indeed *faithful*. In the remainder of this section, we are going to show that $\mathcal{F}$ is actually also *full*, hence makes **Cstab** a conservative extension of $\mathbf{PCoh}_!$.

In the following, we fix $\mathcal{X}$ and $\mathcal{Y}$ two PCSs, and $g \in \mathbf{Cstab}(\mathcal{F}\mathcal{X}, \mathcal{F}\mathcal{Y})$. Our goal is to show that there exists $f \in \mathbf{PCoh}_!(\mathcal{X}, \mathcal{Y})$ such that $\widetilde{f} = g$. First, recall that we have shown in Lemma 11.2.1 that for every PCS $\mathcal{Z}$, the cone $\mathcal{F}\mathcal{Z}$ is a directed complete lattice cone. It means that all results in Section 11.1 can be used here: in particular, g has higher-order derivatives $D^n g$, which makes Definition 11.2.2 below valid.

Definition 11.2.2 *We define $f \in \mathbb{R}_+\mathcal{M}_f(|\mathcal{X}|) \times |\mathcal{Y}|$ by taking:*

$$f_{[a_1,\ldots,a_k],b} = \frac{\alpha_{[a_1,\ldots,a_k]}}{k!}\left(\mathbf{D}^k g(0 \mid e_{a_1}, \ldots e_{a_k})\right)_b \in \mathbb{R}^+.$$

where $\alpha_\mu = \#\{(c_1, \ldots, c_k) \in |\mathcal{X}|^k \ \text{ with } \mu = [c_1, \ldots, c_k]\}$.

We have to show now that $f \in \mathbf{PCoh}_!(\mathcal{X}, \mathcal{Y})$, and that $\widetilde{f}$ coincides with g on $\mathrm{P}(\mathcal{X})$. The key observation here is that we have actually built f in such a way that it is going to coincide with $Tg(0 \mid \cdot)$—the Taylor series of g defined in Definition 11.1.6. We first show it for the elements of $\mathrm{P}(\mathcal{X})$ with *finite support*, by using finite additivity of the $\mathbf{D}^k g(0 \mid \cdot)$.

Lemma 11.2.3 *Let be $x \in \mathrm{P}(\mathcal{X})$, such that $Supp(x) = \{a \in \mathrm{P}(\mathcal{X}) \mid x_a > 0\}$ is finite. Then it holds that $f \cdot x^!$ is finite, and moreover $f \cdot x^! = Tg(0 \mid x)$.*

Proof. Let $A = \{a_1, \ldots, a_m\} \subseteq |\mathcal{X}|$ be the set Supp(x). For any $b \in |\mathcal{Y}|$, we can deduce from the definition of f that:

$$(f \cdot x^!)_b = \sum_{k=0}^{\infty} \sum_{\mu=[c_1,\ldots,c_k] \in \mathcal{M}_n(k)A} \frac{\alpha_\mu}{k!} \cdot \mathbf{D}^k g(0 \mid e_{c_1}, \ldots e_{c_k})_b \cdot x^\mu$$

Looking at the definition of α_μ, we see that this implies:

$$(f \cdot x^!)_b = \sum_{k=0}^{\infty} \sum_{(c_1,\ldots,c_k) \in A^k} \frac{1}{k!} \mathbf{D}^k g(0 \mid e_{c_1}, \ldots e_{c_k})_b \cdot \prod_{i=1}^{k} x_{c_i} \tag{11.8}$$

By Lemma 11.1.4, we know that $\mathbf{D}^k g(0 \mid \cdot)$ is k-linear. As a consequence, and since $x = \sum_{i=1}^m x_{c_i} \cdot e_{c_i}$ and that moreover A is *finite*, we see that (11.8) implies the result:

$$(f \cdot x^!)_b = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{D}^k g(0 \mid x, \ldots x)_b = Tg(0 \mid x).$$

□

We are now going to apply the generalized Bernstein's theorem, as stated in Proposition 11.1.11, to the stable function g from $\mathcal{F}\mathcal{X}$ to $\mathcal{F}\mathcal{Y}$. It tells us that:

$$\forall x \in \mathscr{B}^\circ C_{\mathcal{X}}, \quad g(x) = Tg(0 \mid x). \tag{11.9}$$

Combining (11.9) with Lemma 11.2.3, we obtain that:

$$\forall x \in \mathscr{B}^\circ C_{\mathcal{X}} \text{ with } \mathrm{Supp}(x) \text{ is finite, } f \cdot x^! = g(x). \tag{11.10}$$

We can now use (11.10) to show that $\widetilde{f}$ and g coincide on $\mathrm{P}(\mathcal{X})$: the key point is that the subset of elements in $\mathrm{P}(\mathcal{X})$ of norm smaller than 1 and finite support is dense, and that moreover g is Scott-continuous.

Lemma 11.2.4 *$\forall x \in \mathrm{P}(\mathcal{X})$, $f \cdot x^! = g(x)$, and moreover $f \in \mathbf{PCoh}_!(\mathcal{X}, \mathcal{Y})$.*

Proof. Let be $x \in \mathrm{P}(\mathcal{X})$. It is easy to see that there exists a non-decreasing sequence y_n, with $x = \sup_{n\in\mathbb{N}} y_n$, and for every n, $\|y_n\|_{C_{\mathcal{X}}} < 1$, and y_n has finite support. For every y_n, we can use (11.10), and we see that $g(y_n) = f \cdot y_n^!$. Since g is a morphism in **Cstab**, g is sequentially Scott-continuous, hence:

$$g(x) = \sup_{n\in\mathbb{N}} g(y_n). \tag{11.11}$$

Moreover, we know from [34] that both $x \mapsto x^!$ and $x \mapsto u \cdot x$ are Scott continuous. It means that:

$$f \cdot x^! = \sup_{n\in\mathbb{N}} f \cdot y_n^!. \tag{11.12}$$

Combining (11.11) and (11.12), we obtain $f \cdot x^! = g(x)$. Since $g(\mathscr{B}C_{\mathcal{X}}) \subseteq \mathscr{B}C_{\mathcal{Y}}$, it implies also that $\widetilde{f}(\mathrm{P}(\mathcal{X})) \subseteq \mathrm{P}(\mathcal{Y})$. Thus by Lemma 9.2.7 $f \in \mathbf{PCoh}_!(\mathcal{X}, \mathcal{Y})$. □

Since we have indeed be able to show in Lemma 11.2.4 that for any fixed stable function g in $\mathbf{Cstab}(\mathcal{F}\mathcal{X}, \mathcal{F}\mathcal{Y})$, there exists a $f \in \mathbf{PCoh}_!(\mathcal{X}, \mathcal{Y})$ such that $\mathcal{F}f = g$, we have indeed shown that $\mathcal{F}$ is full.

11.2.2 $\mathcal{F}$ preserves the cartesian structure.

We want now to give more guarantee on the functor $\mathcal{F}$: we want to show that it is a *cartesian closed functor*, meaning that it embeds the cartesian closed category $\mathbf{PCoh}_!$ into the cartesian closed category **Cstab** in such a way that:

- $\mathcal{F}$ preserves the product: for every family $(\mathcal{X}_i)_{i\in I}$ of PCSs, $\mathcal{F}(\prod_{i\in I}^{\mathbf{PCoh}_!} \mathcal{X}_i)$ is isomorphic to $\prod_{i\in I}^{\mathbf{Cstab_m}} \mathcal{F}\mathcal{X}_i$;
- $\mathcal{F}$ preserves function spaces: for every $\mathcal{X}, \mathcal{Y}$ PCSs, $\mathcal{F}(\mathcal{X} \Rightarrow \mathcal{Y})$ is isomorphic to $\mathcal{F}\mathcal{X} \Rightarrow \mathcal{F}\mathcal{Y}$.

Lemma 11.2.5 *$\mathcal{F}$ preserves cartesian products.*

Proof. We fix a family $\mathscr{F} = (\mathcal{X}_i)_{i\in I}$ of PCSs. In order to construct an isomorphism, we have a canonical candidate, given by:

$$\Psi^{\mathscr{F}} = \langle \mathcal{F}(\pi_i) \mid i \in I \rangle \in \mathbf{Cstab}(\mathcal{F}(\prod_{i\in I}^{\mathbf{PCoh}_!} \mathcal{X}_i), \prod_{i\in I}^{\mathbf{Cstab_m}} \mathcal{F}\mathcal{X}_i).$$

Let us now see that $\Psi^{\mathscr{F}}$ is an isomorphism. Looking at the definition of cartesian product in $\mathbf{PCoh}_!$ defined in Section 9.2 of Chapter 9, and the one of cartesian product in **Cstab**, defined in Section 9.3.2, we see that for every $x \in \mathscr{B}\mathcal{F}(\prod_{i\in I}^{\mathbf{PCoh}_!} \mathcal{X}_i)$:

$$\Psi^{\mathscr{F}}(x) = (y_i)_{i\in I} \quad \text{where} \quad \forall i \in I, \forall a \in |\mathcal{X}_i|\,,\ (y_i)_a = x_{(i,a)}.$$

We want now to show that $\Psi^{\mathscr{F}}$ has an inverse. The only candidate is $\Theta^{\mathscr{F}} : y \in \mathscr{B}(\prod_{i\in I}^{\mathbf{Cstab}} \mathcal{F}\mathcal{X}_i) \mapsto \Theta(y) \in (\mathcal{F}(\prod_{i\in I}^{\mathbf{PCoh}_!} \mathcal{X}_i))$, defined by: $\forall i \in I, a \in |\mathcal{X}_i|\,, \Theta(y)_{i,a} = (y_i)_a$. We see immediately that $\Theta^{\mathscr{F}}$ is linear, hence pre-stable, and that moreover it is Scott-continuous. Besides, it is also non-expansive since$\forall y \in \mathscr{B}D$, $\|\Theta^{\mathscr{F}}(y)\|_{\mathcal{F}(\prod_{i\in I}^{\mathbf{PCoh}_!} \mathcal{X}_i)} = \|y\|_{\prod_{i\in I}^{\mathbf{Cstab}} \mathcal{F}\mathcal{X}_i}$. We show now the non-expansiveness of $\Theta^{\mathscr{F}}$:

$$\begin{aligned}\|\Theta^{\mathscr{F}}(y)\|_C &= \inf\{\frac{1}{r} \mid r \cdot \Theta^{\mathscr{F}}(y) \in \mathrm{P}\left(\prod\right)_{i\in I}^{\mathbf{PCoh}_!} \mathcal{X}_i\} \\ &= \inf\{\frac{1}{r} \mid \forall i \in I, r \cdot y_i \in \mathrm{P}\,(\mathcal{X})_i\} = \sup_{i\in I} \|y_i\|_{\mathcal{F}\mathcal{X}_i} = \|y\|_{\prod_{i\in I}^{\mathbf{Cstab}} \mathcal{F}\mathcal{X}_i}.\end{aligned}$$

Thus $\Theta^{\mathscr{F}}$ is a morphism in **Cstab**. □

Lemma 11.2.6 *$\mathcal{F}$ preserves function spaces.*

Proof. Let $\mathcal{X}, \mathcal{Y}$ two PCSs. As previously, there is a canonical candidate for the isomorphism: we define $\Upsilon^{\mathcal{X},\mathcal{Y}}$ as the currying in **Cstab** of the morphism:

$$\mathcal{F}(\mathcal{X} \Rightarrow \mathcal{Y}) \times \mathcal{F}\mathcal{X} \xrightarrow{\Theta^{\mathcal{X}\Rightarrow\mathcal{Y},\mathcal{X}}} \mathcal{F}(\mathcal{X} \Rightarrow \mathcal{Y} \times \mathcal{X}) \xrightarrow{\mathcal{F}(\mathrm{eval}_{\mathcal{X},\mathcal{Y}})} \mathcal{F}\mathcal{Y},$$

where $\Theta^{\mathcal{X}\Rightarrow\mathcal{Y},\mathcal{X}}$ is as defined in the proof of Lemma 11.2.5 above.

Unfolding the definition, we see that actually: $\Upsilon^{\mathcal{X},\mathcal{Y}} : f \in \mathscr{B}\mathcal{F}(\mathcal{X} \Rightarrow \mathcal{Y}) \mapsto \widetilde{f} \in (\mathcal{F}\mathcal{X} \Rightarrow \mathcal{F}\mathcal{Y})$. Since we have shown that $\mathcal{F}$ is full and faithful, we can consider $\Xi^{\mathcal{X},\mathcal{Y}}$ the inverse function of $\Upsilon^{\mathcal{X},\mathcal{Y}}$. Recall from the proof of the fullness of $\mathcal{F}$ in Section 11.2.1 that for every $\mu = [a_1, \ldots, a_k] \in \mathcal{M}_f(|\mathcal{X}|)$, and $b \in |\mathcal{Y}|$:

$$\Xi^{\mathcal{X},\mathcal{Y}}(f)_{\mu,b} = \frac{\alpha_{[a_1,\ldots,a_k]}}{k!}\left(\mathbf{D}^k f(0 \mid e_{a_1}, \ldots e_{a_k})\right)_b.$$

Recall from Lemma 11.1.4 that for any $\vec{u} \in \mathscr{B}(C_x^k)$, the function $f \in \mathbf{Cstab}(\mathcal{F}\mathcal{X}, \mathcal{F}\mathcal{Y}) \mapsto \mathbf{D}^k f(x \mid \vec{u}) \in \mathcal{F}\mathcal{Y}$ is linear and Scott-continuous. As a consequence, $\Xi^{\mathcal{X},\mathcal{Y}}$ too is linear and Scott-continuous.

To know that $\Xi^{\mathcal{X},\mathcal{Y}}$ is stable, we have still to show that it is bounded: we are actually going to show that it preserves the norm. Indeed, for every $f \in \mathscr{B}(\mathcal{F}\mathcal{X} \Rightarrow \mathcal{F}\mathcal{Y})$, we see using the definition of the norm on a cone obtained from a PCS (see Definition 11.2.1), that:

$$\|\Xi^{\mathcal{X},\mathcal{Y}}(f)\|_{\mathcal{F}(\mathcal{X}\Rightarrow\mathcal{Y})} = \inf\{\frac{1}{r} \mid r \cdot \Xi^{\mathcal{X},\mathcal{Y}}(f) \in \mathrm{P}\,(()\,\mathcal{X} \Rightarrow \mathcal{Y})\} \tag{11.13}$$

It was shown in [34] that:

$$r \cdot \Xi^{\mathcal{X},\mathcal{Y}}(f) \in \mathrm{P}\,(()\,\mathcal{X} \Rightarrow \mathcal{Y}) \Leftrightarrow \forall x \in \mathrm{P}\,(\mathcal{X})\,,\ (r \cdot \Xi^{\mathcal{X},\mathcal{Y}}(f)) \cdot x^! \in \mathrm{P}\,(\mathcal{Y})\,. \tag{11.14}$$

We see that $(r \cdot \Xi^{\mathcal{X},\mathcal{Y}}(f)) \cdot x^! = r \cdot f(x)$ since $\Xi^{\mathcal{X},\mathcal{Y}}$ has been defined as the inverse of $\Upsilon^{\mathcal{X},\mathcal{Y}}$. It means that we can rewrite (11.14) as:

$$r \cdot \Xi^{\mathcal{X},\mathcal{Y}}(f) \in \mathrm{P}\,(()\,\mathcal{X} \Rightarrow \mathcal{Y}) \Leftrightarrow \forall x \in \mathrm{P}\,(\mathcal{X})\,,\ r \cdot f(x) \in \mathrm{P}\,(\mathcal{Y})\,. \tag{11.15}$$

Since for every PCS $\mathcal{Z}$, it holds that $\mathrm{P}(\mathcal{Z}) = \mathscr{B}\mathcal{F}\mathcal{Z}$, we can now use (11.15) to rewrite (11.13) as:

$$\|\Xi^{\mathcal{X},\mathcal{Y}}(f)\|_{\mathcal{F}(\mathcal{X}\Rightarrow\mathcal{Y})} = \inf\{\frac{1}{r} \mid \forall x \in \mathscr{B}\mathcal{F}\mathcal{X}, r \cdot f(x) \in \mathscr{B}\mathcal{F}\mathcal{Y}\} \tag{11.16}$$

Looking now at the definition of the norm in the cone $\mathcal{F}\mathcal{X} \Rightarrow \mathcal{F}\mathcal{Y}$, we can complete the proof using (11.16) and the homogeneity of the norm. Indeed:

$$\begin{aligned}\|\Xi^{\mathcal{X},\mathcal{Y}}(f)\|_{\mathcal{F}(\mathcal{X}\Rightarrow\mathcal{Y})} &= \inf\{\frac{1}{r} \mid \|r \cdot f\|_{\mathcal{F}\mathcal{X}\Rightarrow\mathcal{F}\mathcal{Y}} \leq 1\}\\ &= \inf\{\frac{1}{r} \mid r \cdot \|f\|_{\mathcal{F}\mathcal{X}\Rightarrow\mathcal{F}\mathcal{Y}} \leq 1\}\\ &= \|f\|_{\mathcal{F}\mathcal{X}\Rightarrow\mathcal{F}\mathcal{Y}}\end{aligned}$$

□

As a direct consequence of Lemma 11.2.5 and Lemma 11.2.6, we can state the following theorem:

Theorem 11.2.7 *$\mathcal{F}$ is full and faithful, and it respects the cartesian closed structures.*

11.3 $\mathbf{PCoh_!}$ is a full subcategory of $\mathbf{Cstab_m}$

We want now to convert the functor $\mathcal{F} : \mathbf{PCoh_!} \to \mathbf{Cstab}$ into a functor $\mathcal{F}^m : \mathbf{PCoh_!} \to \mathbf{Cstab_m}$. To build $\mathcal{F}^m$, we are going to endow each $\mathcal{F}\mathcal{X}$ with measurability tests, in such a way that $\mathcal{F}(f)$ will be a *measurable* stable function for any morphism $f \in \mathbf{PCoh_!}$.

Observe that this requirement does not determine uniquely the choice of measurability tests. For instance, it would be verified if we choose $\{0\}$ as measurability tests for every $\mathcal{F}\mathcal{X}$. However, as explained in the introduction of the present chapter, we want also $\mathcal{F}^m(\mathbb{N}^{\mathbf{PCoh}})$ to be isomorphic to $[\![\mathbf{Cstab_m}]\!]_{\mathbf{Cstab_m}}$: we would like to be able to inject any discrete distribution on $\mathbb{N}$ into a distribution on $\mathbb{R}$. A natural way to ensure this is to use the discrete structure of the web to give the following definition of the MC arising from a PCS.

Definition 11.3.1 *For any $\mathcal{X} \in \mathbf{PCoh}$, we define $C^m_{\mathcal{X}}$ as the measurable cone $C_{\mathcal{X}}$ endowed with the family $\mathcal{M}^n(\mathcal{X})_{n\in\mathbb{N}}$ of measurability tests defined as $\mathcal{M}^n(\mathcal{X}) = \{\epsilon_a \mid a \in |\mathcal{X}|\}$, where $\epsilon_a(\vec{r}, x) = x_a$.*

We see that the ϵ_a are indeed linear (i.e commuting with linear combinations), and moreover Scott-continuous: hence they are indeed element of $C'_{\mathcal{X}}$. It is easy to verify that the other conditions are verified, and so $C^m_{\mathcal{X}}$ is indeed a MC.

Lemma 11.3.1 *Let be $\mathcal{X}$ a PCS. Then $\mathrm{Paths}^n(C^m_{\mathcal{X}})$ is the set of those $\gamma : \mathbb{R}^n \to C_{\mathcal{X}}$ such that:*

- $\exists\lambda \in \mathbb{R}, \gamma(\mathbb{R}^n) \subseteq \lambda\mathscr{B}C_{\mathcal{X}}$
- $\forall a \in |\mathcal{X}|$, $\gamma_a : \vec{r} \in \mathbb{R}^n \mapsto \gamma(\vec{r})_a \in \mathbb{R}_+$ *is measurable.*

Two MCs with the same underlying cone, but different measurability tests may be isomorphic in **Cstab**: it is enough for them to have the same *measurable paths*. It is what happens in the example below, where we consider $\mathcal{F}^m\mathbb{N}^{\mathbf{PCoh}}$ and Meas($\mathbb{N}$). It is actually also what happens at higher-order types, as we will explain in Section 11.4.

Example 11.3.1 *Recall from the definition of $\mathcal{F}^m$ that the underlying cones of $\mathcal{F}^m\mathbb{N}^{\mathbf{PCoh}}$ and Meas($\mathbb{N}$) are the same. However they have not the same measurable tests:*

$$\mathcal{M}^n(\mathcal{F}^m\mathbb{N}^{\mathbf{PCoh}}) = \{\epsilon_n \mid n \in \mathbb{N}\}; \quad \mathcal{M}^n(Meas(\mathbb{N})) = \{\epsilon_U \mid U \subseteq \mathbb{N}\}.$$

We consider the identity function $id : x \in \mathrm{P}(\mathbb{N})^{\mathbf{PCoh}} \mapsto (A \mapsto \sum_{n\in A} x_n) \in Meas(\mathbb{N})$. Let $\gamma \in Paths^n(\mathcal{F}^m\mathbb{N}^{\mathbf{PCoh}})$. We want to show that for every $U \subseteq \mathbb{N}$:

$$\vec{u} \in \mathbb{R}^n \mapsto \epsilon_U(id(\gamma(\vec{u})) \in \mathbb{R}_+ \text{ is measurable.}$$

We see that $\epsilon_U(id(\gamma(\vec{u})) = \sum_{n\in U} \epsilon_n(\gamma(\vec{u}))$. Since $\gamma \in Paths^n(\mathcal{F}^m\mathbb{N}^{\mathbf{PCoh}})$ it holds that for every $n \in \mathbb{N}$, the function $(\vec{u} \in \mathbb{R}^n \mapsto \epsilon_n(\gamma(\vec{u})) \in \mathbb{R}_+)$ is measurable. Since the class of measurable functions are closed by finite sum and pointwise limit, we see that $(\vec{u} \in \mathbb{R}^n \mapsto \epsilon_U(id(\gamma(\vec{u})) \in \mathbb{R}_+)$ is measurable too, and therefore id is a morphism in $\mathbf{Cstab_m}$. We show in the same way that its inverse is measurable too, and we have the result.

Lemma 11.3.2 *Let $\mathcal{X}, \mathcal{Y}$ be two PCSs, and $f \in \mathbf{PCoh}_!(\mathcal{X}, \mathcal{Y})$. Then $\mathcal{F}f$ is measurable from $C^m_{\mathcal{X}}$ into $C^m_{\mathcal{Y}}$.*

Proof. We have to show that $\mathcal{F}f$ preserves measurable paths. Let $\gamma \in \mathrm{Paths}^n(C^m_{\mathcal{X}})$: our goal is to show that $f \circ \gamma \in \mathrm{Paths}^n(C^m_{\mathcal{Y}})$. Recall that Lemma 11.3.1 gives us a characterization of $\mathrm{Paths}^n(C^m_{\mathcal{Y}})$. Since γ and $\mathcal{F}f$ are bounded, we see immediately that $\mathcal{F}f \circ \gamma$ is bounded. Let b be in $|\mathcal{Y}|$. We see that:

$$(\mathcal{F}f \circ \gamma)_b(\vec{r}) = \sum_{\mu \in \mathcal{M}_f(|\mathcal{X}|)} f_{\mu,b} \cdot \prod_{a \in \mathrm{Supp}(\mu)} \gamma_a(\vec{r})^{\mu(a)}.$$

Since $\gamma \in \mathrm{Paths}^n(C^m_{\mathcal{X}})$, it holds that $\gamma_a : \mathbb{R}^n \to \mathbb{R}_+$ is measurable for all $a \in |\mathcal{X}|$. We conclude by using the fact that the class of measurable functions $\mathbb{R}^n \to \mathbb{R}_+$ is closed under multiplication, finite sums and limit of non-decreasing sequences: it tells us that $\vec{r} \in \mathbb{R}^n \mapsto (\mathcal{F}f \circ \gamma)_b(\vec{r}) \in \mathbb{R}_+$ is measurable, and the result folds. □

Theorem 11.3.3 *The functor $\mathcal{F}^m : \mathbf{PCoh}_! \to \mathbf{Cstab_m}$ defined as $\mathcal{F}^m\mathcal{X} = C^m_{\mathcal{X}}$, and $\mathcal{F}^m f = \mathcal{F}f$, is full and faithful.*

Proof. It is a direct consequence of the fact that $\mathcal{F}$ is full and faithful, coincide with $\mathcal{F}^m$ on morphisms, and moreover $\mathbf{Cstab_m}(\mathcal{F}^m\mathcal{X}, \mathcal{F}^m\mathcal{Y}) \subseteq \mathbf{Cstab}(\mathcal{F}\mathcal{X}, \mathcal{F}\mathcal{Y})$. □

11.4 $\mathcal{F}^m$ is cartesian closed.

We want now to show that just as $\mathcal{F}$, $\mathcal{F}^m$ is cartesian closed. Since the forgetful functor between $\mathbf{Cstab_m}$ and $\mathbf{Cstab}$ is cartesian closed, we see that we have only to show that the $\mathbf{Cstab}$-isomorphisms used in the proofs Lemmas 11.2.5 and 11.2.6 are also morphisms in $\mathbf{Cstab_m}$. Observe that it is immediate that the $\Psi^{\mathscr{F}}$ and $\Upsilon^{\mathcal{X},\mathcal{Y}}$ are also morphisms in $\mathbf{Cstab_m}$: indeed we have defined them canonically by using the structural morphisms linked to the cartesian structure of $\mathbf{Cstab}$, which are the same in $\mathbf{Cstab_m}$ (see [45]).

Lemma 11.4.1 *For all $\mathscr{F} = (\mathcal{X}_i)_{i\in I}$ a finite family of PCSs,*

$$\Theta^{\mathscr{F}} \in \mathbf{Cstab_m}(\prod_{i\in I}^{\mathbf{Cstab_m}} \mathcal{F}^m\mathcal{X}_i, \mathcal{F}^m(\prod_{i\in I}^{\mathbf{PCoh}_!} \mathcal{X}_i)).$$

Proof. Since we already know that $\Theta^{\mathscr{F}}$ is a morphism in **Cstab**, we have only to show that it is measurable, i.e. that it preserves measurable paths. Let γ be in $\mathrm{Paths}^n(\prod_{i\in I}^{\mathbf{Cstab_m}} \mathcal{F}^m\mathcal{X}_i)$. We have to show that $\Theta^{\mathscr{F}} \circ \gamma$ is a measurable path for $\mathcal{F}^m(\prod_{i\in I}^{\mathbf{PCoh_!}} \mathcal{X}_i)$. Recall that Lemma 11.3.1 gives us a characterization of such paths. Since both $\Theta^{\mathscr{F}}$ and γ are bounded, it is immediate that $\Theta^{\mathscr{F}} \circ \gamma$ is bounded too. Now we show that for all $(i, a_i) \in \left|\prod_{i\in I}^{\mathbf{PCoh_!}} \mathcal{X}_i\right|$, $(\Theta^{\mathscr{F}} \circ \gamma)_{(i,a_i)}$ is measurable.

By looking at the definition of $\Theta^{\mathscr{F}}$, we see that $(\Theta^{\mathscr{F}} \circ \gamma)_{(i,a_i)}(\vec{r}) = (\gamma(\vec{r})_i)_a$. We see now that we can construct a measurability test $m \in \mathcal{M}^0(\prod_{i\in I}^{\mathbf{Cstab_m}} \mathcal{F}^m\mathcal{X}_i)$ such that $(\gamma(\vec{r})_i)_a = m(\cdot)(\gamma(\vec{r}))$: it is enough to take $m = \oplus_{j\in I} l_j$, with $l_j = 0$ if $j \neq i$, and $l_i = \epsilon_{a_i}$. Since γ is a measurability tests, it means that $\vec{r} \in \mathbb{R}^n \mapsto m(\cdot)(\gamma(\vec{r})) \in \mathbb{R}_+$ is measurable, and the result folds. □

Lemma 11.4.2 *For all $\mathcal{X}$, $\mathcal{Y}$ PCSs, $\Xi^{\mathcal{X},\mathcal{Y}}$ is in $\mathbf{Cstab_m}(\mathcal{F}^m\mathcal{X} \Rightarrow \mathcal{F}^m\mathcal{Y}, \mathcal{F}(\mathcal{X} \Rightarrow \mathcal{Y}))$.*

Proof. We have to show that $\Xi^{\mathcal{X},\mathcal{Y}}$ preserves measurable paths. Let γ be in $\mathrm{Paths}^n(\mathcal{F}^m\mathcal{X} \Rightarrow \mathcal{F}^m\mathcal{Y})$. We fix $\mu \in \mathcal{M}_f(|\mathcal{X}|)$, and $b \in |\mathcal{Y}|$. Our goal is to show that $(\Xi^{\mathcal{X},\mathcal{Y}} \circ \gamma)_{\mu,b} : \mathbb{R}^n \to \mathbb{R}_+$ is measurable. Since γ is a measurable path, we know that for every $p \in \mathbb{N}$, and $l \in \mathrm{Paths}^p(\mathcal{F}^m\mathcal{X})$, $\epsilon_b \rhd l$ is a measurability test on $\mathcal{F}^m\mathcal{X} \Rightarrow \mathcal{F}^m\mathcal{Y}$, and therefore:

$$(\vec{r}, \vec{u}) \in \mathbb{R}^{p+n} \mapsto (\epsilon_b \rhd l(\vec{r}))(\gamma(\vec{u})) \in \mathbb{R}_+ \text{ is measurable.} \tag{11.17}$$

We are going to apply (11.17) to a particular measurable path on $\mathcal{F}^m\mathcal{X}$. Let p be the cardinality of $\mathrm{Supp}(\mu)$, and $\{a_1, \ldots, a_p\} = \mathrm{Supp}(\mu)$. We define $l^\mu : \mathbb{R}^p \to \mathcal{F}^m\mathcal{X}$ as:

$$l^\mu : \vec{r} \in \mathbb{R}^p \mapsto \begin{cases} \sum_{1\leq i\leq m} r_i \cdot e_{a_i} \text{ if } r_i \geq 0 \forall i \text{ and } \sum_{1\leq i\leq m} r_i \leq 1; \\ 0 \text{ otherwise.} \end{cases}$$

We see that $l^\mu(\mathbb{R}^p)$ is bounded in $\mathcal{F}^m\mathcal{X}$, and moreover for every $a \in |\mathcal{X}|$, the function $\vec{r} \in \mathbb{R}^p \mapsto l^\mu(\vec{r})_a \in \mathbb{R}^+$ is measurable. Using the characterization of $\mathrm{Paths}^p(\mathcal{F}^m\mathcal{X})$ in Lemma 11.3.1, we see that l^μ is in $\mathrm{Paths}^p(\mathcal{F}^m\mathcal{X})$. Thus we can apply (11.17) with $l = l^\mu$. Observe that

$$(\epsilon_b \rhd l^\mu(\vec{r}))(\gamma(\vec{u})) = (\gamma(\vec{u})(l^\mu(\vec{r})))_b \,.$$

Therefore (11.17) tells us that $\phi^{\mu,b} : \mathbb{R}^{p+n} \to \mathbb{R}_+$ is measurable, with $\phi^{\mu,b}$ defined as $\phi^{\mu,b} : (\vec{r}, \vec{u}) \in \mathbb{R}^{p+n} \mapsto \gamma(\vec{u})(l^\mu(\vec{r}))_b \in \mathbb{R}_+$. We define $J \subseteq \mathbb{R}^p$ as $[0, \frac{1}{p}[^p$. We are going to look at the restriction of the function $\phi^{\mu,b}$ to $J \times \mathbb{R}^n$: indeed we are going to show that $\phi^{\mu,b}$ has partial derivatives on that interval. We define $\psi^{\mu,b} : J \times \mathbb{R}^n \to \mathbb{R}_+$ as the restriction of $\phi^{\mu,b}$ to $J \times \mathbb{R}^n$. Since $\phi^{\mu,b}$ is a measurable function, and $J \times \mathbb{R}^n$ a measurable subset of $\mathbb{R}^{p+n}$, $\psi^{\mu,b}$ also is measurable.

Lemma 11.4.3 below (which is shown in the long version) is key: it says that we can recover the coefficients of the power series $\psi^{\mu,b}$ by looking at its partial derivatives. We will then show that we can do it in a *measurable* way.

Lemma 11.4.3 *For every multiset $\nu \in \mathcal{M}_f(\{1, \ldots, p\})$, there exists an interval K of the form $[0, c]^p$ such that the partial derivative $\partial^\nu \psi^{\mu,b} = \frac{\partial(\psi^{\mu,b}_{|K\times\mathbb{R}^n})^{card(\nu)}}{\partial {r_1}^{\nu(1)} \ldots \partial {r_p}^{\nu(p)}} : K \times \mathbb{R}^n \to \mathbb{R}_+$ exists, and moreover:*

$$\partial^\nu \psi^{\mu,b}(\vec{0}, \vec{u}) = \Xi^{\mathcal{X},\mathcal{Y}}(\gamma(\vec{u}))_{\nu,b} \cdot \prod_{1\leq i\leq p} \nu(i)!$$

Proof. Since $l^\mu(\vec{r}) = \sum_{1\leq i\leq p} r_i \cdot e_i$ for $\vec{r} \in J$, we see that:

$$\phi^{\mu,b}(\vec{r},\vec{u}) = \sum_{\nu\in\mathcal{M}_f(|\mathcal{X}|)} \Xi^{\mathcal{X},\mathcal{Y}}(\gamma(\vec{u}))_{\nu,b} \cdot \vec{r}^{\nu} \in \mathbb{R}_+.$$

For a fixed $\vec{s}$, we can see it as a generalization of entire series in real analysis. There are well-known results about the differentiation of such series: for instance, a uniformly convergent entire series is differentiable on its (open) domain of convergence. Here, we are going to show the counterpart of some properties on entire series, on what we call *multisets series of p real variables*: those are the series of the form

$$S(\vec{r}) = \sum_{\nu\in\mathcal{M}_f(1,\dots,p)} a_\nu \cdot \vec{r}^{\nu} \quad \text{where} \quad \vec{r} \in \mathbb{R}^p.$$

First, we observe that for each $\vec{r}$, we can look at $S(\vec{r})$ as an infinite sum over natural numbers:

$$S(\vec{r}) = \sum_{n\in\mathbb{N}} \Big(\sum_{\mu\in\mathcal{M}_f(1,\dots,m)|\mathrm{card}(\mu)=n} a_\mu \cdot \vec{r}^{\mu} \Big).$$

We recall here a classical result of real analysis on power series, that we will use in the following.

Lemma 11.4.4 *[Derivation of a series] Let $f_n : I \to \mathbb{R}$ be a sequence of functions from a bounded interval I. We suppose that $f(x) = \sum_{n\in\mathbb{N}} f_n(x)$ is convergent for every $x \in I$, and moreover for each $n \in \mathbb{N}$, f_n is derivable and $\sum_{n\in\mathbb{N}} f_n'$ is uniformly convergent on I. Then f is derivable, and moreover $f' = \sum_{n\in\mathbb{N}} f_n'$.*

Lemma 11.4.5 *Let $p \in \mathbb{N}$, and $S(\vec{r}) = \sum_{\nu\in\mathcal{M}_f(\{1,\dots,p\})} a_\nu \cdot \vec{r}^{\nu}$ with non-negative coefficients a_μ, such that S is convergent on an interval $I = [-c,c]^p$, with $c > 0$.*

Then there exists $0 < b < c$, such that the function $g : \vec{r} \in]-b,b[^p \mapsto S(\vec{r}) \in \mathbb{R}$ is partially derivable in each of the r_i variables, and moreover:

$$\frac{\partial g}{\partial r_i}(\vec{r}) = \sum_{\nu\in\mathcal{M}_f(\{1,\dots,p\})|i\in\nu} a_\nu \cdot \vec{r}^{\nu-[i]} \cdot \nu(i).$$

Proof. We take $b = \frac{c}{2}$, and set $J =]-b,b[$. To simplify the notations, we suppose here that $i = 1$, but the proof is the same in other cases. We want to show that for any fixed $\vec{u} \in J^{p-1}$, the function $h_{\vec{u}} : r \mapsto g(r,\vec{u})$ is derivable on J. Let us fix $\vec{u} \in]-b,b[^{p-1}$. We are going to use Lemma 11.4.4 on $h_{\vec{u}}$. We see that $h_{\vec{u}}(r) = \sum_{n\in\mathbb{N}} h_n(r)$, where

$$h_n(r) = \Big(\sum_{\nu\in\mathcal{M}_f(2,\dots,p)} a_{\nu+[1^n]} \cdot \vec{u}^{\nu} \Big) \cdot r^n,$$

where $[1^n]$ is the multiset consisting of n occurrences of 1. We see that for every $n \in \mathbb{N}$, the function h_n is derivable on J, and:

$$h_n'(r) = \Big(\sum_{\nu\in\mathcal{M}_f(2,\dots,p)} a_{\nu+[1^n]} \cdot \vec{u}^{\nu} \Big) \cdot n \cdot r^{n-1}.$$

We see now that the series $\sum_{n\in\mathbb{N}} h'_n$ is uniformly convergent on J: for every $r \in J$, it holds that:

$$\begin{aligned}|h'_n(r)| &\leq (\sum_{\nu\in\mathcal{M}_f(2,\ldots,p)} a_{\nu+[1^n]}\cdot|\vec{u}|^\nu)\cdot n\cdot|r|^{n-1}\\ &= (\sum_{\nu\in\mathcal{M}_f(2,\ldots,p)} a_{\nu+[1^n]}\cdot|\vec{u}|^\nu\cdot c^n)\cdot n\cdot\frac{1}{c}\cdot\left(\frac{|r|}{c}\right)^{n-1}\\ &= (\sum_{\eta\in\mathcal{M}_f(1,\ldots,p)|\eta(1)=n} a_\eta\cdot|(c,\vec{u})|^\eta)\cdot n\cdot\frac{1}{c}\cdot\frac{|r|^{n-1}}{c^{n-1}}\end{aligned}$$

Since the series $S(\vec{r}) = \sum a_\mu\cdot\vec{r}^\mu$ is convergent on I, and $(c,|\vec{u}|)\in I$, it holds that there exists $M\geq 0$, with $\sum_{\nu\in\mathcal{M}_f(1,\ldots,p)|\nu(1)=n} a_\nu\cdot|(c,\vec{u})|^\nu \leq M$. As a consequence, and since moreover for each $r\in J$, it holds that $|r|\leq b$, we can now write:

$$\forall r\in J,\quad |h'_n(r)|\leq M\cdot\frac{n}{c}\cdot\left(\frac{|r|}{c}\right)^{n-1}\leq M\cdot\frac{n}{c}\cdot\left(\frac{b}{c}\right)^{n-1} \tag{11.18}$$

Since $b<c$, we know that the quantity in the right part of (11.18) defines a convergent series. As a consequence, the series $\sum_{n\in\mathbb{N}} h'_n$ is uniformly convergent on J, which means that we are able to apply Lemma 11.4.4: we see that $\frac{\partial g}{\partial r_1}$ exists on J^n, and moreover:

$$\begin{aligned}\frac{\partial g}{\partial r_1}(\vec{r}) &= \sum_{n\in\mathbb{N}} h'_n(r,(r_2,\ldots,r_p))\\ &= \sum_{n\in\mathbb{N}}(\sum_{\nu\in\mathcal{M}_f(\{2,\ldots,p\})} a_{\nu+[1^n]}(r_2,\ldots,r_n)^\nu)\cdot n\cdot r^{n-1}\\ &= \sum_{\nu\in\mathcal{M}_f(\{1,\ldots,p\})|1\in\nu} a_\nu\cdot\vec{r}^{\nu-[1]}\cdot\nu(1)\quad\text{and the result folds.}\end{aligned}$$

□

We iterate now Lemma 11.4.5 in order to look at higher-order partial derivatives.

Lemma 11.4.6 *Let $S(\vec{r}) = \sum_{\nu\in\mathcal{M}_f(\{1,\ldots,p\})} a_\nu\cdot(\vec{r})^\nu$ with $a_\nu\geq 0$. We suppose S convergent on $I =]-b,b[^p$, with $b>0$. Then for every multiset $\nu\in\mathcal{M}_f(\{,1\ldots,p\})$, there exists $0<b\leq a$, such that, when we define $g:\vec{r}\in[-b,b]^p\mapsto S(\vec{r})$, the partial higher-order derivative $\frac{\partial g^{card(\nu)}}{\partial {r_1}^{\nu(1)}\ldots\partial {r_p}^{\nu(p)}}$ exists, and moreover:*

$$\frac{\partial g^{card(\nu)}}{\partial {r_1}^{\nu(1)}\ldots\partial {r_p}^{\nu(p)}}(\vec{r}) = \sum_{\eta\in\mathcal{M}_f(\{1,\ldots,p\})} a_{\nu+\eta}\cdot\vec{r}^\eta\cdot\prod_{1\leq i\leq p}\frac{(\eta+\nu)(i)!}{\eta(i)!}.$$

Proof. The proof is by induction on $\mathrm{card}(\nu)$, and uses Lemma 11.4.5. It is clear that the result holds for $\nu=\emptyset$. Now, we suppose that it holds for every ν of cardinality N. Let κ be a multiset of cardinality $N+1$, and we take ν, and i such that $\kappa=\nu+[i]$. By the induction hypothesis, there exists $c>0$, such that, when we define $g:\vec{r}\in[-c,c]^p\mapsto S(\vec{r})$, the partial derivative $\frac{\partial g^{\mathrm{card}(\nu)}}{\partial {r_1}^{\nu(1)}\ldots\partial {r_p}^{\nu(p)}}$ exists, and is equal to:

$$T(\vec{r}) = \sum_{\eta\in\mathcal{M}_f(\{1,\ldots,p\})} a_{\eta+\nu}\cdot\vec{r}^\eta\cdot\prod_{1\leq j\leq p}\frac{(\eta+\nu)(j)!}{\eta(j)!}.$$

We see we can apply Lemma 11.4.5 with T as multiset series, and $I = [-c, c]^p$. It means that there exist $0 < d < c$, such that $\frac{\partial T}{\partial r_i}$ exists, and

$$\begin{aligned}\frac{\partial T}{\partial r_i}(\vec{r}) &= \sum_{\eta \in \mathcal{M}_f(\{1,\ldots,p\}) | i \in \eta} a_{\eta+\nu} \cdot \vec{r}^{\eta-[i]} \cdot \eta(i) \cdot \prod_{1 \leq j \leq p} \frac{(\eta+\nu)(j)!}{\eta(j)!} \\ &= \sum_{\iota \in \mathcal{M}_f(\{1,\ldots,p\})} a_{\iota+\kappa} \cdot \vec{r}^{\iota} \cdot (\iota + [i])(i) \cdot \prod_{1 \leq j \leq p} \frac{(\iota+\kappa)(j)!}{(\iota+[i])(j)!} \\ &= \sum_{\iota \in \mathcal{M}_f(\{1,\ldots,p\})} a_{\iota+\kappa} \cdot \vec{r}^{\iota} \cdot \prod_{1 \leq j \leq p} \frac{(\iota+\kappa)(j)!}{\iota(j)!}.\end{aligned}$$

□

We end the proof of Lemma 11.4.3 by using Lemma 11.4.6 for each $\vec{u} \in \mathbb{R}^n$ on the multiset series given by

$$S_{\vec{u}}(\vec{r}) = \sum_{\nu \in \mathcal{M}_f(\{1,\ldots,p\})} (\Xi^{\mathcal{X},\mathcal{Y}} \gamma(\vec{u}))_{\nu,b} \cdot \vec{r}^{\nu}.$$

We see that it is indeed absolutely convergent on $I =]-\frac{1}{p}, \frac{1}{p}[^p$, using the fact that for $\vec{r} \in [0, \frac{1}{p}]^p$, $S_{\vec{u}}(\vec{r}) = \psi^{\mu,b}(\vec{r}, \vec{u})$ for $\vec{r} \in \mathbb{R}^p$. □

We can now end the proof of Lemma 11.4.2. Indeed, since the class of real-valued measurable functions is closed by addition, multiplication by a scalar and pointwise limit, and that $\psi^{\mu,b} : K \times \mathbb{R}^p \to \mathbb{R}_+$ is measurable, it holds that the partial derivatives (when they exist) are measurable too. Indeed, observe that $\frac{\partial \psi^{\mu,b}}{\partial r_1}((r_1, \vec{r}), \vec{u}) = \lim_{n \to \infty} f_n((r_1, \vec{r}), \vec{u})$, with

$$f_n((r_1, \vec{r}), \vec{u}) = n \cdot f((r_1 + \frac{1}{n}, \vec{r}), \vec{u}) - f((r_1, \vec{r}), \vec{u}).$$

As a consequence, applying Lemma 11.4.3 with $\nu = \mu$ leads us to:

$$\vec{u} \in \mathbb{R}^n \mapsto \frac{\partial (\psi^{\mu,b})^{\mathrm{card}(\mu)}}{\partial {r_1}^{\mu(1)} \ldots \partial {r_p}^{\mu(p)}}(\vec{0}, \vec{u}) \text{ is measurable.}$$

Therefore (again by Lemma 11.4.3), $(\vec{u} \in \mathbb{R}^n \mapsto \Xi^{\mathcal{X},\mathcal{Y}}(\gamma(\vec{u}))_{\mu,b} \cdot \prod_{1 \leq i \leq p} \mu(i)!$ is measurable), and the result folds. □

Theorem 11.4.7 *$\mathcal{F}^m$ is a cartesian closed full and faithful functor.*

11.5 Conclusion

Our full embedding of $\mathbf{PCoh}_!$ into **Cstab** implies that every stable function f from $\mathrm{P}(\mathcal{X})$ to $\mathrm{P}(\mathcal{Y})$ can be characterized by an element $\Xi^{\mathcal{X},\mathcal{Y}}(f) \in \mathbb{R}^{\mathcal{M}_f(|\mathcal{X}|) \times |\mathcal{Y}|}$, that has to be seen as a power series. It gives us a *concrete representation* of stable functions on discrete cones, similar to the notion of trace introduced by Girard in [53] for stable functions on quantitative domains. There are well-known real analysis results on power series, as for instance the *uniqueness theorem*—any power series which is null on an open subset has all its coefficients equal to 0—on which is based the proof of full abstraction for $\mathsf{PCF}_{\oplus}$ in $\mathbf{PCoh}_!$ [46]. While we have not been able to extend such a concrete representation to cones which are not *directed-complete*, as for instance the cone $\mathrm{Meas}(\mathbb{R}) \Rightarrow_m \mathrm{Meas}(\mathbb{R})$, our result could hopefully be a first step in this direction. This kind of characterization could lead to a way towards a full abstraction result for the continuous language $\mathsf{PCF}_{\mathrm{sample}}$ in $\mathbf{Cstab}_{\mathbf{m}}$, and more generally gives us new tools to reason about continuous probabilistic programs.

Chapter 12

Perspectives

12.1 Abramsky's applicative bisimulation for an higher-order language with continuous probabilities.

The present thesis aimed to contribute to a better understanding of coinductive techniques for studying context equivalence and distance for higher-order languages with discrete probabilities. In this setting, we now have quite an extensive picture of the situation: we can cite for instance the seminal work of Dal Lago, Sangiorgi and Alberti [32] on state-based and distribution-based generalization of Abramsky's applicative bisimulation for call-by-name $\Lambda_{\oplus}$, the study presented in the present thesis for call-by-value $\Lambda_{\oplus}$, the development by Vignudelli and Sangiorgi [107] of an environmental bisimulation for a probabilistic λ-calculus with imperative features as local states... By contrast, operational techniques based on coinduction have not yet been extended to higher-order languages equipped with primitives handling *continuous* distributions. Such a step further would ask to replace the quantitative transition systems on which are based the discrete probabilistic generalization of Abramsky's applicative bisimulation with quantitative transition systems that also embed concepts of *measure theory*. This necessity already appears when defining the operational semantics of the language: for instance when Ehrhard Pagani and Tasson [45] define the operational semantics for their continuous language $\mathsf{PCF}_{\text{sample}}$, they need to express it as a Labeled Markov Process—thus generalizing to continuous probabilities the way they defined the operational semantics of the discrete probabilistic language $\mathsf{PCF}_{\oplus}$ using a Labeled Markov Chain. Coinductively defined notions of both equivalences and distances have been studied in depth for Labelled Markov Processes [39, 41, 48] —and in some cases, reasoning on these continuous transition systems has proved easier than reasoning on their discrete counterpart, as we have highlighted in Chapter 5 when looking at the testing characterization of equivalence on Labeled Markov Processes developed in [118].

12.2 State-based Applicative Bisimulation Distances for Probabilistic Higher-Order Languages.

As we presented in details in Chapter 4, Dal Lago, Sangiorgi and Alberti introduced [32] two generalizations of Abramsky's applicative bisimulation to a discrete probabilistic setting: the *state-based* and *distribution-based* applicative bisimulation. It may be noted that while the underlying transition system of the state-based applicative bisimulation has a more complex transition structure than the one obtained in the distribution-based

setting—in the sense that the transition are truly probabilistic—the states space of the former is by contrast simpler as the one of the latter, since it consists of all programs of the language instead of all distributions over all programs of the language. In Chapter 8, we looked at quantitative generalizations of *distribution-based* applicative bisimulation, for both a probabilistic affine λ-calculus and a probabilistic λ-calculus with copying capabilities, and in both cases we showed that the resulting distances on programs are fully-abstract with respect to context distance. In a joint work [29] with Ugo Dal Lago—that is not presented in the present thesis—we looked at a quantitative generalization of *state-based* applicative bisimulation, for our *affine* probabilistic λ-calculus, and we showed that we obtain this way a sound but not fully abstract notion of distance. To do so, we used the existence of a built-in notion of coinductive distance on the states of a Labelled Markov Chain [40, 42], and we applied it to the particular Labelled Markov Chain built to model interactively the operational semantics of our language. The next step would be to generalize the notion of state-based applicative bisimulation distance to the case of a probabilistic language with *copying abilities*. The definition of such a state-based applicative bisimulation distance for a λ-calculus with *copying abilities* would have two challenges to face. The first one is similar to the one we encountered when defining distribution-based context distance: it consists in the fact that we cannot keep the Labelled Markov Chain used to define state-based *equivalence* on $\Lambda_{\oplus}$, because we will then obtain an *unsound distance*. Similarly to what we did for the distribution-based distance, the solution may lay in embedding the ability to copy into the states space of the transition system. The second complication is more specific to the state-based setting. The soundness proof for the state-based distance in the affine case were greatly facilitated by the fact that the Labelled Markov Chain we used for the affine λ-calculus was *finitely branching*—i.e. there are a *finite* number of states that can be reached with non-zero probability when we fix the state of departure and the transition label, hence allowing us to use the results from [42]. However, it is not be the case anymore as soon as we consider a Labelled Markov Chain of a language at least as expressive as $\Lambda_{\oplus}$, because it becomes possible to write programs that can output an infinite numbers of possible values.

12.3 The Full Abstraction Problem for $\mathbf{PCF}_{\mathbf{sample}}$ in the category of measurable cones and measurable stable functions.

There is no known fully-abstract denotational model yet for an higher-order language with continuous probabilities. When introducing the model of measurable complete cones and measurable stable functions for $\mathsf{PCF}_{\text{sample}}$ [45], Ehrhard, Pagani and Tasson designed it as an extension of the denotational model based on probabilistic coherence spaces for the discrete probabilistic language $\mathsf{PCF}_{\oplus}$, already known to be fully abstract [46]. In Chapter 11, we formalized this connection categorically by defining a full and faithful functor from $\mathbf{PCoh}_!$—the co-Kleisli category of the LL model based on probabilistic coherence spaces—and $\mathbf{Cstab}_{\mathbf{m}}$—the category of complete measurable cones and stable measurable functions. The fullness of this functor means that when we consider two *discrete* cones, the stable measurable functions between them are *exactly* the power series with non-negative coefficients that are morphisms in $\mathbf{PCoh}_!$. This connection could hopefully be a first step towards generalizing the full abstraction proof for $\mathsf{PCF}_{\oplus}$ in $\mathbf{PCoh}_!$ into a full-abstraction proof for $\mathsf{PCF}_{\text{sample}}$ in $\mathbf{Cstab}_{\mathbf{m}}$. The ability of seeing measurable stable function as a generalization of power series is especially encouraging, because the full-abstraction proof for

$\mathsf{PCF}_\oplus$ uses crucially the power-series structure of program interpretation. More precisely, if f is a power series is the denotation of some program M, it is possible to build for any index a a sequence of arguments to pass to M that allow us to recover the coefficient f_a of the power series. From there, the aim would be to study stable functions with the aim of generalizing the reasoning done on power series to the case where the cones we consider are not discrete anymore.

12.4 Is $\mathbf{Cstab_m}$ the co-Kleisli category of a LL model ?

As highlighted in Chapter 9, the denotational model of $\mathsf{PCF}_\oplus$ lives actually in the co-Kleisli category of the **PCoh** model of Linear Logic(LL). By contrast, the category of measurable cones and measurable stable functions is a cartesian closed category, which is not yet known as being the co-Kleisli category of a model of LL. The connection between $\mathbf{PCoh}_!$ and $\mathbf{Cstab_m}$ that we highlighted in Chapter 11 lead to the question of whether we can draw a similar connection between **PCoh**, and a LL model that would generate $\mathbf{Cstab_m}$ as its co-Kleisli category. Such a construction would allow for instance to interpret a linear λ-calculus with primitive for handling continuous probabilities—e.g. a continuous variant of $\Lambda^!_\oplus$. We can also recall that Ehrhard and Tasson [116] gave a fully-abstract denotational semantics in **PCoh** to a discrete probabilistic extension of Levy's Call-by-Push-Value—a λ-calculus with a polarized type system, that allows for the call-by-name and call-by-value evaluation paradigm to coexist. Their construction of program interpretation use the structure of **PCoh** as a model of Linear Logic. Similarly, having a model of LL that would generate $\mathbf{Cstab_m}$ as its co-Kleisli category could potentially be used to give a denotational semantics to a call-by-push-value with *continuous* probabilities.

12.5 A Context Distance adapted to Polytime Adversaries.

In Chapter 6 we show that for an expressive enough calculus where moreover all programs stop, the context distance always trivializes, i.e. the distance between two programs is either 0 or 1. Termination of programs is a strong requirement, but nonetheless one which is often enforced on concrete programming situation. As a consequence, we can ask ourselves whether our notion of context distance is really adapted to the aim of talking about programs that are close without being equivalent, and in particular whether take *all* contexts into account is really significant. In some setting, as for instance computational cryptography, we do not want to consider all contexts, but only those that are run in polynomial time on some *security parameter*—e.g. the size of the encryption key. Formally, it would mean to have a more restricted language for contexts as the one for programs, possibly using a type system guaranteeing polytime execution in the spirit of Bounded Linear Logic [56]. Another option would be to explore *tamed* versions of the context distances, where context have a *price to pay* each time they use their input: concretely every time they call their input, they enter in an infinite loop with some fixed probability ε. These tamed distances are for instance considered for $\mathsf{PCF}_\oplus$ in a recent unpublished work by Ehrhard [44]. The problem would then become how to adapt the coinductive operational techniques for context distances that we explored in Chapter 8 to these new way of comparing quantitatively programs.

Bibliography

[1] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53(2-3):225–241, 1987.

[2] S. Abramsky. The Lazy λ-Calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.

[3] S. Abramsky. Computational interpretations of linear logic. *Theor. Comput. Sci.*, 111(1&2):3–57, 1993.

[4] S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105(2):159–267, 1993.

[5] R. Alur, T. A. Henzinger, and E. D. Sontag, editors. *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, volume 1066 of *Lecture Notes in Computer Science*. Springer, 1996.

[6] A. Azevedo de Amorim, M. Gaboardi, J. Hsu, S.-y. Katsumata, and I. Cherigui. A semantic account of metric preservation. In *ACM SIGPLAN Notices*, volume 52, pages 545–556. ACM, 2017.

[7] H. Barendregt. Lambda calculi with types, handbook of logic in computer science vol. ii, 1992.

[8] H. P. Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[9] H. P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

[10] T. Barker. A monad for randomized algorithms. *Electronic Notes in Theoretical Computer Science*, 325:47–62, 2016.

[11] M. Barr. **-Autonomous categories*, volume 752. Springer, 2006.

[12] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *International Workshop on Computer Science Logic*, pages 121–135. Springer, 1994.

[13] S. Bernstein. Constructive proof of weierstrass approximation theorem, comm. *Kharkov Math. Soc*, 13(1-2), 1912.

[14] G. Bierman, A. Pitts, and C. Russo. Operational properties of lily, a polymorphic linear lambda calculus with recursion. *Electronic Notes in Theoretical Computer Science*, 2001.

[15] G. M. Bierman. What is a categorical model of intuitionistic linear logic? In *International Conference on Typed Lambda Calculi and Applications*, pages 78–93. Springer, 1995.

[16] J. Borgström, U. Dal Lago, A. D. Gordon, and M. Szymczak. A lambda-calculus foundation for universal probabilistic programming. In *ACM SIGPLAN Notices*, volume 51, pages 33–46. ACM, 2016.

[17] M. Brandt and F. Henglein. Coinductive axiomatization of recursive type equality and subtyping. In P. de Groote and J. Roger Hindley, editors, *Typed Lambda Calculi and Applications*, volume 1210 of *Lecture Notes in Computer Science*, pages 63–81. Springer Berlin Heidelberg, 1997.

[18] T. Braüner. A simple adequate categorical model for pcf. In *International Conference on Typed Lambda Calculi and Applications*, pages 82–98. Springer, 1997.

[19] F. Breuvart, U. D. Lago, and A. Herrou. On higher-order probabilistic subrecursion. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, pages 370–386, 2017.

[20] F. Breuvart and M. Pagani. Modelling coeffects in the relational semantics of linear logic. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 41. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[21] A. Carraro, T. Ehrhard, and A. Salibra. Exponentials with infinite multiplicities. In *International Workshop on Computer Science Logic*, pages 170–184. Springer, 2010.

[22] S. Castellan, P. Clairambault, H. Paquet, and G. Winskel. The concurrent game semantics of probabilistic pcf. In *Logic in Computer Science (LICS), 2018 33rd Annual ACM/IEEE Symposium on, ACM/IEEE*, 2018.

[23] R. Crubillé. Probabilistic stable functions on discrete cones are power series. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 275–284, 2018.

[24] R. Crubillé and U. Dal Lago. On probabilistic applicative bisimulation and call-by-value λ-calculi. In *Proc. of ESOP*, pages 209–228, 2014.

[25] R. Crubillé and U. Dal Lago. Metric reasoning about λ-terms: The general case (long version). Available at `http://eternal.cs.unibo.it/mrltgc.pdf`, 2016.

[26] R. Crubillé and U. Dal Lago. Metric reasoning about \lambda -terms: The general case. In *European Symposium on Programming*, pages 341–367. Springer, 2017.

[27] R. Crubillé, U. Dal Lago, D. Sangiorgi, and V. Vignudelli. On applicative similarity, sequentiality, and full abstraction. In *Proc. of Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, pages 65–82, 2015.

[28] R. Crubillé, T. Ehrhard, M. Pagani, and C. Tasson. The free exponential modality of probabilistic coherence spaces. In *International Conference on Foundations of Software Science and Computation Structures*, pages 20–35. Springer, 2017.

[29] R. Crubillé and U. D. Lago. Metric reasoning about λ-terms: The affine case. In *Proc. of LICS*, pages 633–644, 2015.

[30] R. Culpepper and A. Cobb. Contextual equivalence for probabilistic programs with continuous random variables and scoring. In *European Symposium on Programming*, pages 368–392. Springer, 2017.

[31] U. Dal Lago and S. Martini. The weak lambda calculus as a reasonable machine. *Theoretical Computer Science*, 398(1-3):32–50, 2008.

[32] U. Dal Lago, D. Sangiorgi, and M. Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *Proc. of POPL*, pages 297–308, 2014.

[33] U. Dal Lago and M. Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.*, 46(3):413–450, 2012.

[34] V. Danos and T. Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Inf. Comput.*, 209(6):966–991, 2011.

[35] V. Danos and R. S. Harmer. Probabilistic game semantics. *ACM Transactions on Computational Logic (TOCL)*, 3(3):359–382, 2002.

[36] K. De Leeuw, E. F. Moore, C. E. Shannon, and N. Shapiro. Computability by probabilistic machines. *Automata studies*, 34:183–198, 1956.

[37] V. De Paiva. Categorical semantics of linear logic for all. In *Advances in Natural Deduction*, pages 181–192. Springer, 2014.

[38] Y. Deng, Y. Feng, and U. Dal Lago. On coinduction and quantum lambda calculi. In *International Conference on Concurrency Theory, CONCUR 2015*, 2015.

[39] J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled markov processes. *Information and Computation*, 179(2):163–193, 2002.

[40] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labeled markov systems. In *Proc. of CONCUR*, 1999.

[41] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating labelled markov processes. *Information and Computation*, 184(1):160–200, 2003.

[42] J. Desharnais, R. Jagadeesan, V. Gupta, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proc. of LICS*, pages 413–422, 2002.

[43] C. Dwork. Differential privacy. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, pages 1–12, 2006.

[44] T. Ehrhard. Differentials and distances in probabilistic coherence spaces. Available at `https://www.irif.fr/~ehrhard/pub/proba_diff.pdf`, 2019.

[45] T. Ehrhard, M. Pagani, and C. Tasson. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. *PACMPL*, 2(POPL):59:1–59:28, 2018.

[46] T. Ehrhard, C. Tasson, and M. Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *Proc. of POPL*, pages 309–320, 2014.

[47] M. Felleisen and M. Flatt. Programming languages and lambda calculi. *Unpublished lecture notes. http://www. ccs. neu. edu/home/matthias/3810-w02/readings. html*, 2003:1538, 1989.

[48] N. Ferns, P. Panangaden, and D. Precup. Metrics for markov decision processes with infinite state spaces. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 201–208. AUAI Press, 2005.

[49] D. Gebler and S. Tini. Fixed-point characterization of compositionality properties of probabilistic processes combinators. In *EXPRESS-SOS*, pages 63–78, 2014.

[50] A. Girard and G. J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.

[51] J. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[52] J. Girard and Y. Lafont. Linear logic and lazy computation. In *TAPSOFT'87: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Pisa, Italy, March 23-27, 1987, Volume 2: Advanced Seminar on Foundations of Innovative Software Development II and Colloquium on Functional and Logic Programming and Specifications (CFLP)*, pages 52–66, 1987.

[53] J.-Y. Girard. The system f of variable types, fifteen years later. *Theoretical computer science*, 45:159–192, 1986.

[54] J.-Y. Girard. Linear logic: its syntax and semantics. *London Mathematical Society Lecture Note Series*, pages 1–42, 1995.

[55] J.-Y. Girard. Between logic and quantic: a tract. *Linear logic in computer science*, 316:346–381, 2004.

[56] 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.

[57] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[58] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. In *UAI 2008*, pages 220–229, 2008.

[59] A. D. Gordon. Bisimilarity as a theory of functional programming. *Theoretical Computer Science*, 228(1-2):5–47, 1999.

[60] J. Goubault-Larrecq. Full abstraction for non-deterministic and probabilistic extensions of pcf i: The angelic cases. *Journal of Logical and Algebraic Methods in Programming*, 84(1):155–184, 2015.

[61] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM (JACM)*, 32(1):137–161, 1985.

[62] M. Hennesy and R. Milner. On observing non-determinism and concurrency lecture notes in computer science, 1980.

[63] C. Heunen, O. Kammar, S. Staton, and H. Yang. A convenient category for higher-order probability theory. In *Logic in Computer Science (LICS), 2017 32nd Annual ACM/IEEE Symposium on*, pages 1–12. IEEE, 2017.

[64] D. J. Howe. Equality in lazy computation systems. In *Logic in Computer Science, 1989. LICS'89, Proceedings., Fourth Annual Symposium on*, pages 198–203. IEEE, 1989.

[65] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996.

[66] J. M. E. Hyland and C.-H. Ong. On full abstraction for pcf: I, ii, and iii. *Information and computation*, 163(2):285–408, 2000.

[67] C. Jones. *Probabilistic non-determinism*. PhD thesis, University of Edinburgh, UK, 1990.

[68] C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *Prof. of LICS*, pages 186–195, 1989.

[69] V. Koutavas, P. B. Levy, and E. Sumii. From applicative to environmental bisimulation. *Electr. Notes Theor. Comput. Sci.*, 276:215–235, 2011.

[70] D. Kozen. Semantics of probabilistic programs. In *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pages 101–114. IEEE, 1979.

[71] S. G. Krantz and H. R. Parks. *A primer of real analytic functions*. Springer Science & Business Media, 2002.

[72] Y. Lafont. The linear abstract machine. *Theor. Comput. Sci.*, 59:157–180, 1988.

[73] Y. Lafont. *Logiques, catégories et machines*. PhD thesis, PhD thesis, Université Paris 7, 1988.

[74] U. D. Lago, D. Sangiorgi, and M. Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *Proc. of POPL*, pages 297–308, 2014.

[75] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

[76] S. B. Lassen. Relational reasoning about contexts. In *Higher Order Operational Techniques in Semantics, Publications of the Newton Institute*, pages 91–135. Cambridge University Press, 1998.

[77] S. B. Lassen. *Relational reasoning about functions and nondeterminism*. Citeseer, 1998.

[78] T. Leventis. Probabilistic böhm trees and probabilistic separation. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 649–658, 2018.

[79] S. Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.

[80] B. McMillan. Absolutely monotone functions. *Annals of Mathematics*, 60:467–501, 1954.

[81] P.-A. Mellies. Categorical semantics of linear logic. *Panoramas et syntheses*, 27:15–215, 2009.

[82] P.-A. Melliès, N. Tabareau, and C. Tasson. An explicit formula for the free exponential modality of linear logic. In *Automata, Languages and Programming*, volume 5556 of *LNCS*, pages 247–260. Springer, 2009.

[83] R. Milner. Communication and concurrency. international series in computer science, 1989.

[84] M. Mislove. Discrete random variables over domains, revisited. In *Concurrency, Security, and Puzzles*, pages 185–202. Springer, 2017.

[85] J. C. Mitchell, C. John, and K. Apt. *Concepts in programming languages*. Cambridge University Press, 2003.

[86] E. Moggi. Computational lambda-calculus and monads. In *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, 1989.

[87] J. H. Morris Jr. *Lambda-calculus models of programming languages.* PhD thesis, Massachusetts Institute of Technology, 1969.

[88] C.-H. L. Ong. Concurrent lambda calculus and a general precongruence theorem for applicative bisimulations. *Draft, Cambridge Un*, 1992.

[89] L. Ong and M. Vákár. S-finite kernels and game semantics for probabilistic programming. POPL.

[90] H. Paquet and G. Winskel. Continuous probability distributions in concurrent games.

[91] S. Park, F. Pfenning, and S. Thrun. A probabilistic language based on sampling functions. *ACM Trans. Program. Lang. Syst.*, 31(1), 2008.

[92] A. M. Pitts. Operationally-based theories of program equivalence. In *Semantics and Logics of Computation*, pages 241–298. Cambridge University Press, 1997.

[93] G. D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.

[94] G. D. Plotkin. Lcf considered as a programming language. *Theoretical computer science*, 5(3):223–255, 1977.

[95] G. D. Plotkin and K. Keimel. Mixed powerdomains for probability and nondeterminism. *Logical Methods in Computer Science*, 13, 2017.

[96] M. O. Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

[97] J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. *ACM Sigplan Notices*, 45(9):157–168, 2010.

[98] M. Rennela. Convexity and order in probabilistic call-by-name fpc. *arXiv preprint arXiv:1607.04332*, 2016.

[99] E. Ritter and A. M. Pitts. A fully abstract translation between a λ-calculus with reference types and standard ml. In *International Conference on Typed Lambda Calculi and Applications*, pages 397–413. Springer, 1995.

[100] H. L. Royden and P. Fitzpatrick. *Real analysis*, volume 32. Macmillan New York, 1988.

[101] N. Saheb-Djahromi. *Probabilistic LCF*. Springer Berlin Heidelberg, 1978.

[102] N. Saheb-Djahromi. Probabilistic LCF. In *Proc. of MFCS*, pages 442–451, 1978.

[103] D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8:447–479, 1998.

[104] D. Sangiorgi. Beyond bisimulation: The "up-to" techniques. In *International Symposium on Formal Methods for Components and Objects*, pages 161–171. Springer, 2005.

[105] D. Sangiorgi. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(4):15, 2009.

[106] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In *Logic in Computer Science, 2007. LICS 2007. 22nd Annual IEEE Symposium on*, pages 293–302. IEEE, 2007.

[107] D. Sangiorgi and V. Vignudelli. Environmental bisimulations for probabilistic higher-order languages. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 595–607, 2016.

[108] E. Schechter. *Handbook of Analysis and its Foundations*. Academic Press, San Diego, CA, USA, 1996.

[109] D. Scott. *Outline of a mathematical theory of computation*. Oxford University Computing Laboratory, Programming Research Group, 1970.

[110] D. S. Scott. A type-theoretical alternative to iswim, cuch, owhy. *Theoretical Computer Science*, 121(1-2):411–440, 1993.

[111] D. S. Scott and C. Strachey. *Toward a mathematical semantics for computer languages*, volume 1. Oxford University Computing Laboratory, Programming Research Group, 1971.

[112] R. A. Seely. *Linear logic,*-autonomous categories and cofree coalgebras*. Ste. Anne de Bellevue, Quebec: CEGEP John Abbott College, 1987.

[113] B. Simon. *Convexity: An analytic viewpoint*, volume 187. Cambridge University Press, 2011.

[114] A. K. Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In *Proc. of RTA*, pages 219–234, 2005.

[115] S. Staton, F. Wood, H. Yang, C. Heunen, and O. Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10. IEEE, 2016.

[116] C. Tasson and T. Ehrhard. Probabilistic call by push value. *Logical Methods in Computer Science*, 15, 2019.

[117] D. Tolpin, J.-W. van de Meent, and F. Wood. Probabilistic programming in anglican. In A. Bifet, M. May, B. Zadrozny, R. Gavalda, D. Pedreschi, F. Bonchi, J. Cardoso, and M. Spiliopoulou, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 308–311, Cham, 2015. Springer International Publishing.

[118] F. van Breugel, M. W. Mislove, J. Ouaknine, and J. Worrell. Domain theory, testing and simulation for labelled markov processes. *Theor. Comput. Sci.*, 333(1-2):171–197, 2005.

[119] F. van Breugel and J. Worrell. A behavioural pseudometric for probabilistic transition systems. *Theor. Comput. Sci.*, 331(1):115–142, 2005.

[120] J. van de Wiele. Manuscript. 1987.

[121] L. Vepstas. The minkowski question mark, psl (2, z) and the modular group. 2004.

[122] P. Wadler. A syntax for linear logic. In *Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA, April 7-10, 1993, Proceedings*, pages 513–529, 1993.

[123] M. Wand, R. Culpepper, T. Giannakopoulos, and A. Cobb. Contextual equivalence for a probabilistic language with continuous random variables and recursion. *PACMPL*, 2(ICFP):87:1–87:30, 2018.

[124] D. V. Widder. *Laplace Transform (PMS-6)*. Princeton University Press, Princeton, NJ, USA, 2015.