Structural Foundations for Probabilistic Programming Languages

Dario Maximilian Stein

Balliol College

University of Oxford

A thesis submitted for the degree of
DOCTOR OF PHILOSOPHY, COMPUTER SCIENCE
October 2021

The success of the probability theory in mathematics and theoretical physics is due not so much to its measure theoretic foundation but because it exploits and enhances symmetries of the structures it applies to.

— MIKHAIL GROMOV, Bernoulli Lecture - What is Probability? (EPFL)

Abstract

Probability theory and statistics are fundamental disciplines in a data-driven world. Synthetic probability theory is a general, axiomatic formalism to describe their underlying structures and methods in a elegant and high-level way, abstracting away from the mathematical foundations such as measure theory. We showcase the value of synthetic probabilistic reasoning to computer science by presenting a novel, synthetic analysis of three situations: the *Beta-Bernoulli process*, *exact conditioning on continuous random variables* and an investigation into the relationship of *random higher-order functions and fresh name generation*.

The synthetic approach to probability has merit not only by matching statistical practice and achieving greater conceptual clarity, but it also lets us transfer and generalize probabilistic ideas and language to other domains: A wide range of computational phenomena admit a probabilistic analysis, such as *nondeterminism*, *generativity* (e.g. name generation), *unification*, *exchangeable random primitives* and *local state*. This perspective is particularly useful if we wish to compare or combine those effects with actual probability: We develop a purely stochastic model of name generation (gensym) which is not only a tight semantic fit, but also uncovers striking parallels to the theory of random functions.

Probabilistic programming is an emergent flavor of Bayesian machine learning where complex statistical models are expressed through code. We argue that probabilistic programming and synthetic probability theory are two sides of the same coin. Not only do they share a similar goal, namely to act as an accessible, high-level interface to statistics, but synthetic probability theories are also the natural semantic domains for probabilistic programs. Conversely, every probabilistic language *defines* its own internal theory of probability. This is a probabilistic version of the Curry-Howard correspondence: Statistical models *are* programs. This opens up the analysis and optimization of statistical inference procedures to tools from programming language theory.

Much as category theory has served as a unifying model for logic and programming, we argue that probabilistic programming languages arise precisely as the internal languages of categorical probability theories. Our methods and examples draw from diverse areas of computer science and mathematics such as rewriting theory, logical relations, linear and universal algebra, categorical logic, measure theory and descriptive set theory.

Acknowledgements

I was lucky to have Sam Staton as a supervisor and to learn from his vast knowledge and inspiring style of research. I am grateful for the many stimulating discussions and everyday interactions with my colleagues at the Computer Science department, among them Carmen Constantin, Swaraj Dash, Mathieu Huot, Younesse Kaddar, Ohad Kammar, Carol Mak, Cristina Matache, Sean Moss, Hugo Paquet, Philip Saville, Jesse Sigal, Sam Speight, Ned Summers, Dominik Wagner and Fabian Zaiser. I too wish to thank Luke Ong and Alex Simpson for examining this thesis.

Thanks to Tobias Fritz, Tomáš Gonda and Paolo Perrone for introducing me to Markov categories and great collaboration. My work has benefited and drawn inspiration from discussions with many further people, some of whom are Zeinab Galal, Alexander Kechris, Paul Levy, Alex Lew and Prakash Panangaden.

Much of the material of this thesis is based on work that was conducted collaboratively (as fully outlined in Section 1.2). I'd like to thank my previously unmentioned coauthors Nathanael Ackerman, Nicholas Gauguin Houghton-Larsen, Cameron Freer, Daniel Roy, Marcin Sabok, Michael Wolman and Hongseok Yang, with whom it has been a pleasure to work. I also enjoyed the opportunity to present my work at various talks and venues. My DPhil has been funded by the Engineering and Physical Science Research Council (EPSRC) and the Oxford-DeepMind Graduate Scholarship.

I wish to thank Balliol College and the extraordinary graduate community I found at Holywell Manor for the amazing time I had there. Deepest affection goes to my family, my parents Christian and Marlies, and all my friends in Oxford, back home and across the world: you are my dearest treasures and I wouldn't be who I am without you.

This thesis was typeset using LaTeX. The typesetting was greatly aided by the applications QUIVER for commutative diagrams, and TikZiT for string diagrams.

Contents

List of Symbols

We list the following nonstandard symbols and notations which are used in the thesis. When notation is overloaded, we make sure that the meanings are either distinct (so that no confusion is possible) or related in an evocative way (like in the various uses for ν or $[x]$).

Categorial notation	
$\mathbb{C}, \mathbb{D}$	categories ($\mathbb{C}$ means complex numbers in Section 4.4)
$\mathbb{C}(C, D)$	homset
$\mathbb{V} \to \mathbb{C}$	Freyd category
$[\mathbb{C}, \mathbb{D}]$	functor category
$\otimes$	monoidal product, product distribution; product-σ-algebra
$\alpha, \rho, \mathsf{swap}$	coherence isomorphisms in symmetric monoidal categories
$f : C \to D$	morphism
$f : C \rightsquigarrow D$	kernel, Kleisli map, morphism in Leak (Section 30.5) or Cond (Section 19.2)
Monads and Programming Languages	
η	unit of a monad
δ	Dirac distribution; unit of a probability monad
st	strength of a monad
$[x]$	unit of a monad, return x
$[\phi]$	Iverson bracket (1 if ϕ else 0)
join	join of a monad
$\mathbb{C}_T$	Kleisli category of monad T
f^+	Kleisli extension of f
$\mathsf{let}\, x \leftarrow u$	monadic sequencing (do-notation)
$\mathsf{let}\, x = u$	call-by-value sequencing
$u[t/x]$	substitution of x by t
$u[t!x]$	substitution of the unique free occurrence of x by t
ground	non-function types like booleans, reals, names
first-order	non-nested function types $\tau_1 \to \ldots \to \tau_n$ with τ_i ground
higher-order	function types (first-order types are the simplest higher-order types!)
Specific categories	
Set	sets and functions
Fin	finite sets and functions
Inj	finite sets and injections
Meas	measurable spaces and measurable maps
Sbs	standard Borel spaces and measurable maps
SMeas	standardly generated measurable spaces and measurable maps
Qbs	quasi-Borel spaces and morphisms
SQbs	standardly generated quasi-Borel spaces and morphisms
Top	topological spaces and continuous maps
Pol	Polish spaces and continuous maps
CH	compact Hausdorff spaces and continuous maps
MIU	commutative C*-algebras and multiplicative involution-preserving unital maps
PU	commutative C*-algebras and positive unital maps

FinStoch	finite sets and stochastic matrices
Stoch	measurable spaces and probability kernels
BorelStoch	standard Borel spaces and probability kernels
SfKer	measurable spaces and s-finite kernels
Aff	affine spaces $\mathbb{R}^n, n \in \mathbb{N}$ and affine-linear maps
Gauss	affine spaces $\mathbb{R}^n, n \in \mathbb{N}$ and affine-linear maps with Gaussian noise
Nom	nominal sets and equivariant functions
Atoms	nominal sets which are finite coproducts of representables
FinFam	finite family construction (free finite coproduct completion)

Specific notation

$\mathbb{A}, \mathbb{A}^{\#n}$	nominal set of atoms/n distinct atoms
$A * B$	separated product of nominal sets; tuple type
$\mathcal{B}$	Borel σ-algebra on a topological space
$\mathbf{C}_{\text{det}}$	deterministic subcategory
f_X	marginal of X
$f\|_X$	conditional on X
$\langle f, g \rangle$	tupling of morphisms: $(f \otimes g) \circ \text{copy}$
$f =_\mu g$	almost-sure equality with respect to μ
$f \restriction s$	restriction of function or relation f to domain s
$R : s_1 \leftrightharpoons s_2$	partial bijection ("span") between finite sets
dom, cod	domain and codomain of relations
col	column space of a matrix
ϵ	Frobenius multiplication, exact conditioning
ν_{ij}	fresh urn creation, Beta sample in Chapter III
ν	random variable allocation, standard Gaussian sample in Chapter IV
ν	atomless probability measure, fresh name generation in Chapter V
$\mathbb{N}$	natural numbers $\mathbb{N} = \{0, 1, \ldots\}$
$\mathcal{N}(\mu, \sigma^2)$	normal distribution with mean μ and variance σ^2
$\mathcal{N}(\vec{\mu}, \Sigma)$	multivariate normal distribution with mean $\vec{\mu}$ and covariance matrix Σ
$\mathcal{N}(\Sigma)$	multivariate normal distribution with mean zero and covariance matrix Σ
$+$	coproduct, disjoint union of sets; addition
$x +_p y, x +_{i:j} y$	probabilistic choice, abstract convex combination
$x \,?_p\, y$	probabilistic choice by drawing from urn p
$x \vee y$	nondeterministic choice, semilattice join
$\oplus$	disjoint union of sets of names or partial equivalence relations (Chapter V)
$\int f(x)\mu(\mathrm{d}x)$	Lebesgue integral, Kleisli composition of probability monads (Section 5)
$f_*\mu$	pushforward distribution, functorial action of probability monads
Σ_X	induced or equipped σ-algebra on X
M_X	induced or equipped quasi-Borel structure on X
$\mathcal{C}$	set of continuous functions; countable-cocountable σ-algebra
I	monoidal unit; abstract unit interval type in Chapter III
$\ll$	absolute continuity
supp	support in nominal sets; support of distributions
$\mathcal{G}$	Giry monad

$\mathcal{R}$	Radon monad
P	probability monad
$\Pr(A)$	the probability of event A
$\mathcal{P}$	powerset
$p(y\|x)$	alternative notation for stochastic matrices; synthetic notation for $p : X \to Y$
Y^X	exponential object
$2^{\mathbb{R}}$	the set of Borel subsets of $\mathbb{R}$ (this is an exponential object in Qbs)
$\Rightarrow$	logical implication, exponential object, natural transformation
$X \sim \nu$	statistical notation for random variables and distributions
$(\mathrel{=:=}), (:=)$	exact conditioning in Chapter IV
$\mathbb{S}^1, \mathbb{T}$	unit circle (1-dimensional torus)
ω	least infinite ordinal; operation symbols in universal algebra
Ω	(weak) subobject classifier

1 Introduction

> *At a purely formal level, one could call probability theory the study of measure spaces with total measure one, but that would be like calling number theory the study of strings of digits which terminate.*
>
> — TERENCE TAO, Topics in random matrix theory

What is probability really about? The central claim of this thesis is that this question can be approached using ideas from programming language theory:

"**Probabilistic Programming Languages** are the internal languages of **Categorical Probability Theories**."

We begin by giving a nontechnical overview of the background of this claim:

Probabilistic programming is an emergent programming paradigm which can express *statistical models through code*. This is enabled by the addition of two capabilities to the language

(i) *sampling* draws random values from a target distribution

(ii) *observing* ties the random draws to empirical likelihood, letting the system *learn from data*

While any programming language with access to a pseudorandom number generator implements the *sampling* part, *observing* is more far-reaching in nature: It seeks to alter the random choices made by the program to match the occurring observations, enabling the program to learn from data. This has the semblance of running a simulation backwards, inverting it and discovering which earlier choices led to later outcomes. In statistical terminology, the forward (sampling) part of the program defines a *generative model*, which is then *conditioned* on the observations. In terms of Bayes' law

$$\text{posterior} \propto \text{prior} \cdot \text{likelihood}$$

the forward part defines a prior and the observations a likelihood. The outcome of running the program is then a solution to the *inference problem* it poses, for example by sampling (approximately) from the posterior. Therefore, probabilistic programming is sometimes seen as a flavor of machine learning called *Bayesian machine learning*.

The following example (adapted from [Goodman et al., 2016, Chapter 3]) in the probabilistic programming language WebPPL defines a small medical decision model using two possible diseases (A and B), which each may lead to developing a symptom. We model the presence of each disease using biased coin flips (`flip`). The line `condition(symptom == true)` conditions only on those cases where a symptom has been developed. The following program thus describes the inference problem: Given that I have a symptom, what is the probability I have disease A?

```
Infer({method: 'enumerate'},
  function () {
    // two diseases and their baseline probabilities
    var diseaseA = flip(0.01)
    var diseaseB = flip(0.2)

    // probabilities of developing a symptom
    var symptom = (diseaseA && flip(0.8)) || (diseaseB && flip(0.5))

    condition(symptom == true)
    return diseaseA
});
```

The call `Infer({method: 'enumerate'}, ...)` invokes a general-purpose inference algorithm to compute the desired posterior described by the probabilistic program – in this case, exhaustive enumeration of all possibilities. For more complex statistical models, exhaustive search would soon be intractable and more sophisticated algorithms such as particle filters and Monte Carlo simulation can be used. To be precise, every probabilistic programming system comprises two aspects:

(i) a declarative language for describing inference problems

(ii) a set of general-purpose inference algorithms for solving these problems

The power of the declarative approach is apparent: Code is an excellent way to develop and communicate complex models. It is highly modular, compositional and we have a host of software engineering best practices available to help organize it. In using an expressive general-purpose programming language, we can leverage pre-existing libraries or invoke complicated simulations that would be impossible to write down in a statistical model by hand. Probabilistic programming enables everyone with basic scientific coding skill to write flexible statistical models. Users can experiment with their models and even try to run inference on them. That being said, generating high-performance inference code is still far from automatic. There is No Free Lunch in machine learning, and no set of general-purpose inference algorithms will address every situation optimally. However, probabilistic programming still helps decouple the domain expert writing the model from a statistician dedicated to running efficient inference.

This explains the crucial role that **programming language theory** holds to probabilistic programming: One can often replace a probabilistic program by an equivalent one where inference runs more efficiently. For example, we can predict the unknown bias `p` of a coin which came up heads once using *rejection sampling*

```
Infer({method: 'rejection'},
  function () {
    var p = uniform(0, 1)
```

```
    var x = flip(p)
    condition(x == true) // Reject current execution trace if false
    return p
  })
```

The same program is more efficiently implemented using *importance sampling*, where `score(p)` multiplies the likelihood `p` to the current execution trace

```
Infer({method: 'importance'},
  function () {
    var p = uniform(0, 1)
    score(p) // always accept, but weight the current execution trace
    return p
  })
```

Verifying such program transformations is subtle. Studying the correctness of inference algorithms necessarily combines techniques from programming language theory and statistics.

Denotational semantics is an important tool to study meaning-preserving transformations of programs. We associate to every program a mathematical object, its *denotation*. As probabilistic programs concern random values, their denotations will use the mathematical language of probability theory. For example, if `normal`$(\mu,\ \sigma^2)$ samples from a normal (Gaussian) distribution with mean μ and variance σ^2, then the program `normal(0, 1) + normal(0,1)` differs from `normal(0, 2)` in that it samples twice instead of once. Yet we consider those programs equivalent, because mathematically, the variance of independent random variables is additive and so both programs denote the same probability distribution over the real numbers. Note that to give a proper mathematical account of such programs, we're required to use a mathematical axiomatization of probability such as Kolmogorov's measure-theoretic axioms.

The idea of denotational semantics is to understand programs in terms of probability theory. We are interested to go the other way, and propose to understand probability theory though code:

Synthetic Probability Theory: We contend that statistical modelling assumptions correspond to properties of programs, and theorems of probability theory to properties of the language. For example, in measure theory, random variables X, Y are *independent* if for all measurable sets A, B we have

$$\Pr(X \in A \wedge Y \in B) = \Pr(X \in A) \cdot \Pr(Y \in B).$$

Every programming language comes with a natural syntactic notion of independence: The outputs of a program `joint` returning pairs `(x,y)` are deemed independent if they can be obtained by subprograms `px`, `py` as

$$\texttt{joint} \approx \texttt{(px,py)} \tag{Ind}$$

This is the case if and only if `px` and `py` can be obtained as the marginals

$$\texttt{px} \approx (\texttt{var (x,y) = joint; x}) \qquad \texttt{py} \approx (\texttt{var (x,y) = joint; y})$$

These equations are examples of *synthetic definitions*: We have captured a concept from probability purely in terms of a dataflow property of a program. Such a definition makes sense wherever our language makes sense, even without reference to measure theory. When interpreted in the denotational semantics, this definition recovers the measure-theoretic meaning of independence.

Some dataflow properties are always expected of random sampling [Staton, 2017]: Firstly, the order of sampling does not matter, that is independent lines of code can be reordered as

$$\begin{pmatrix}\texttt{var x = u}\\ \texttt{var y = v}\end{pmatrix} \approx \begin{pmatrix}\texttt{var y = v}\\ \texttt{var x = u}\end{pmatrix} \tag{Comm}$$

whenever `x` does not occur freely in `v` and `y` not in `u`. Secondly, unused samples can be discarded, i.e.

$$(\texttt{var x = u; v}) \approx \texttt{v} \tag{Disc}$$

whenever `x` is not free in `v`. These properties called *commutativity* and *discardability* are expected to hold in every *purely probabilistic language* which has no effects other than random sampling. Note that commutativity corresponds to *Fubini's theorem* in measure theory (see (21)). Probabilistic programs with observations are still commutative but **`observe`** statements are no longer discardable.

Commutativity and discardability are generally *invalid* in stateful programs. For example, the following lines using JavaScript's post-increment operator `i++`,

```
var x = i++
var y = i
```

cannot be interchanged. Many languages provide a pseudorandom number generator (PRNG) to implement sampling. Such generators are stateful, because they maintain and update a random seed, but their interface *does* verify the commutativity and discardability equations up to observable statistics. For this, it is crucial that the random seed remains abstract and cannot be inspected by the client program.

In Chapter III, we will meet Pólya's urn, which is a stateful urn whose observable behavior nonetheless satisfies (Comm) and (Disc) because of a statistical property called *exchangeability*. Again, it is crucial that the contents of the urn cannot be inspected, which we enforce via abstract types. Name generation (Chapter V) is another example of a process with a commonly stateful implementation which we can interpret using pure probability.

Because of central importance of commutativity and discardability, we will take these as the minimal requirement for any system which we can analyze in probabilistic terms:

Soft Definition 1.0 A *synthetic model of probability* is any structure where we can compose computations in sequence, in parallel, as well as copy and discard data. Composition must be commutative and discardable.

This definition is purely concerned with the abstract structure of probabilistic computation. For that reason, we will use the terms *structural* and *synthetic probability theory* interchangeably.

In Chapter II, we will meet precise instances of this soft definition, principally Markov categories and commutative monads. A large number of probabilistic and statistical concepts can be defined inside this structural language alone, with no reference to measure theory at all. Among those are determinism, (conditional) independence, support, almost-sure equality, conditional probability, sufficiency of statistics, zero-one laws and De Finetti's theorem. Such a formulation offers a variety of advantages:

(i) it is high-level and intuitive

(ii) it is agnostic to mathematical foundations

(iii) it reduces the redundant difficulties of expressing ourselves in both code *and* statistics

(iv) it lends itself to an analysis with tools of programming language theory, and plays well with program transformation techniques discussed before

(v) it opens probabilistic reasoning up to generalizations

The point about foundations needs elaboration: So far we've only mentioned a single mathematical formalism of probability – measure theory. In fact, there are plenty of different *models of probability theory*, with measure theory being one among those models. Synthetic probability theory is the right language to compare the properties of these models. A programming language then has an *implementation* in measure theory, much like an ordinary language compiles to assembly. Synthetic reasoning abstracts away from this implementation dependence, leading to greater conceptual clarity.

Our soft definition is general enough that it applies to a range of phenomena beyond probability and statistics in a narrow sense. Examples of these are computational effects like nondeterminism, logic programming, name generation and various forms of generativity, but also exotic notions such as negative probability. In this thesis, we consider all these phenomena as generalized probability theories, and analyze them in the unifying framework of synthetic probability theory.

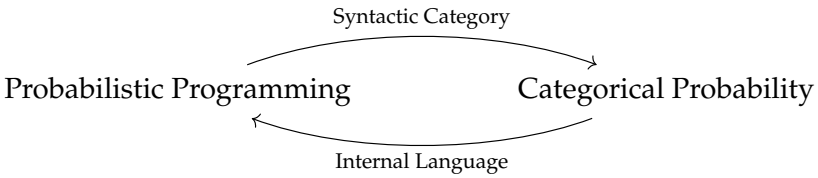
Category theory is the language in which the soft definition is most naturally formalized, to the extent that we will often use *synthetic probability* and *categorical probability* interchangeably. The tight relation between programming languages, type theories and categories is well known in theoretical computer science. Adapting John Bell's aphorism,

> "a category may be thought of as a type theory shorn of its syntax." [Bell, 2012]

we contend that a categorical model of probability theory is a probabilistic programming language shorn of its syntax. In the style of the Curry-Howard correspondence for λ-calculus, we posit the relation

$$\frac{\lambda\text{-calculus}}{\text{cartesian closed categories}} = \frac{\textbf{probabilistic programming languages}}{\textbf{categorical probability theories}}$$

Every model of probability has an *internal language*, which is a probabilistic programming language, and every probabilistic programming language in turn defines its own model of probability by its *syntactic category*:



1.1 Contributions

The objective of this thesis is to demonstrate the value of exploring categorical models and synthetic reasoning for probabilistic programming, substantiating the main claim from the introduction. Our aims are the following

(i) To argue for synthetic probability theory as a foundation of probabilistic programming (Section 7), as well as a shared language for a wider scope of probability-like phenomena like nondeterminism, unification, name generation or generativity (Section 5).

(ii) To further the understanding of synthetic probability theory itself, by proving novel theorems and characterizations (Section 9)

(iii) To present techniques for the construction of new categorical models, such as second-order algebra (Section 3.7), the Cond construction (Section 19.2) and the Leak construction (Section 30.5)

While Fritz [2020] argues for the general merits of synthetic reasoning, each of the main chapters III-V of this thesis will center around one particular categorical model, which each exemplifies a *characteristic strength* of such reasoning:

Simplicity of models (Chapter III) Simple things should be simple. Yet, formally, defining even the simplest continuous probability distributions like Beta or Gaussian requires heavy machinery like Borel measurability and integration. Synthetic probability allows us to construct minimalistic categories which capture such distributions purely combinatorially. We give one such axiomatization for the relationship between the Beta and Bernoulli distributions, and formulate their conjugate-prior relationship completely algebraically.

Well-behavedness of properties (Chapter IV) Specialized synthetic models tend to be more well-behaved than general-purpose ones. Notions like *support* or *conditioning* are ill-behaved in general measure-theoretic probability or require arbitrary choices. In restricted settings like finite or Gaussian probability, statisticians work with particular constructions

that get rid of these choices. By considering these construction in the appropriate categorical models, we see that they follow elegant universal properties. This tackles the problem of "well-behaved disintegrations" posed by Shan and Ramsey [2017]. We use this as a starting point for developing a theory of Bayesian inference purely synthetically, addressing phenomena like Borel's paradox in a type-theoretic way. This allows us to define a specialized probabilistic programming language for Gaussian processes with a powerful notion of conditioning. This is an example of how denotational semantics can actually guide language design. This work can be seen as the semantic side of the symbolic disintegration techniques of Narayanan and Shan [2019]; Shan and Ramsey [2017].

Generality of constructions (Chapter V) Synthetic models may model constructions which lie outside the scope of conventional, measure-theoretic probability! *Higher-order functions* are commonplace in programming, yet there is no satisfactory account of random functions in measure theory – the category of measurable spaces is not cartesian closed. One can remedy this by passing to the category of *quasi-Borel spaces*, which conservatively extends standard Borel probability with function spaces.

This thesis contains one of the first detailed investigations into random higher-order functions. An important result is that a random function $\mathbb{R} \to 2$ like

$$\mathsf{let}\, a \;=\; \mathsf{normal}() \,\mathsf{in}\, \lambda x.(x == a)$$

is indistinguishable from the function $\lambda y.\mathsf{false}$, even though the second function is always constant and the first one never is. For a more involved example, the random transposition function $\mathbb{R} \to \mathbb{R}$

$$\mathsf{let}\, a \;=\; \mathsf{normal}() \,\mathsf{in}\, \mathsf{let}\, b \;=\; \mathsf{normal}() \,\mathsf{in}\, \lambda x.\mathsf{if}\; x = a \;\mathsf{then}\; b \;\mathsf{else}\; \mathsf{if}\; x = b \;\mathsf{then}\; a \;\mathsf{else}\; x$$

can be proved semantically equal to the identity function.

This behavior is reminiscent of identities for name generation: The values of a,b are hidden or private inside the body of the functions. It turns out that the probability theory on function spaces intrinsically features such subtle effects as privacy and information hiding. The language of categorical probability theory offers the right tool to make this connection precise. We develop a probabilistic model of name generation and show that two name generating functions behave the same if and only if the corresponding random functions do.

We hope that different audiences may take away different things from this thesis

To a computer scientist there is a powerful interplay between languages and their semantic models. The construction of models furthers the understanding of the language and lets one justify meaning-preserving transformations such as compiler optimizations. On the other hand, understanding a model may inspire the addition of new features to the language like the exact conditioning operator in Chapter IV. Synthetic probability is a new perspective on statistics with tools and ideas from programming language theory. Likewise, it offers new perspectives on a variety of generalized probabilistic effects.

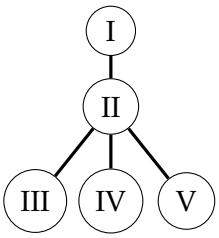
To a mathematician the interest lies in the models themselves: We are working with non-trivial structures from set theory (Section 25), linear algebra (Section 21), functional analysis (Section 4.4) and descriptive set theory (Section 28). The categorical formalism gives an alternative, high-level picture of working with these structures. String diagrams and the various internal languages from Section 3.1 serve as *elementary calculi* for reasoning about these structures in an intuitive way, without loss of precision.

To a statistician probabilistic programming may be a powerful tool for the creation of modular large-scale models. Our work also lets us see familiar structures (statistical models, Bayesian networks) in another light, such as string diagrams and probabilistic programs. It also lets us transfer statistical thinking from probability to other computational phenomena, such as generating a fresh name or a logic variable. This is particularly interesting when unifying these effects, for example choosing fresh names for clusters in a Dirichlet process (Section 30.1).

1.2 Technical Summary

We now give an outline of the structure of this thesis and its relation with previously published work of the author. The thesis builds on material the author developed in four articles with various coauthors. It presents this work in its most developed form and within the unifying context of synthetic probability theory. We generally won't cite these articles explicitly, but will make clear their relation to the different parts of the thesis.

Chapters I–II establish the common theoretical background for the main chapters III–V, which each focus on a particular model and application of synthetic probability theory. The latter are largely independent and can be read in any order, though they exhibit various interconnections:



We now briefly summarize the contents of each chapter and refer to the beginning of each chapter for a less technical introduction.

Chapter I: Background As this thesis makes connections between various areas of maths and computer science, we begin by compiling an overview over the assumed background. We will review probabilistic programming systems (Section 2), general semantical and categorical notions (Section 3) and classical models of probability theory (Section 4). We introduce the various internal languages we will employ (Moggi's monadic metalanguage, computational λ-calculus, fine-grained call-by-value and string diagrams), establishing shared notation for the later chapters. Chapter I is purely expository and contains no novel results. It may be skipped by the knowledgeable reader or consulted later.

Chapter II: Categorical Probability Theory We introduce the two main flavors of categorical probability theory that we will be using throughout the thesis:

(i) generalized probability monads due to Kock [2011]

(ii) CD and Markov categories due to Cho and Jacobs [2019]; Fritz [2020]

We define the internal language of CD categories in Section 7, which is a ground computational λ-calculus with an appealing equational theory, and argue that this is the prototypical probabilistic programming language. We then revisit the relevant notions from synthetic probability, such as determinism, independence, almost sure equality and conditionals (Section 8). We expand on these definitions and prove novel characterizations, such as the characterization of representable supports in Section 8.3 and the implications between the dataflow axioms of Section 9. These results are based on an upcoming article with Tobias Fritz, Tomáš Gonda, Nicholas Gauguin Houghton-Larsen and Paolo Perrone.

Chapter III: The Beta-Bernoulli Process and Algebraic Effects We give an algebraic presentation of the Beta-Bernoulli process and prove various normalization theorems about the theory. This allows us to make a formal connection between a stateful implementation (Polya's urn) and the original stateless process, in the spirit of De Finetti's theorem. We obtain a purely combinatorial model of synthetic probability which encodes the Beta and Bernoulli distributions. This chapter is based on joint work with Sam Staton, Hongseok Yang, Nathanael Ackerman, Cameron Freer and Daniel Roy [Staton et al., 2018].

Chapter IV: Compositional Semantics for Conditioning We introduce a probabilistic programming language with Gaussian random variables and a powerful exact conditioning construct $(=:=)$. We analyze the equational properties of this operator and generalize them by developing a general theory of exact conditioning in Markov categories $\mathbb{C}$. The challenge is now to internalize conditioning, which is a meta-operation on $\mathbb{C}$, to a morphism in a larger category, thereby making conditioning compositional. We achieve this by introducing the concept of *conditioning channels*, which form a category $\mathsf{Cond}(\mathbb{C})$ with good formal properties, in Section 19. We then return to the Gaussian language to give an algebraic presentation of contextual equivalence similar to Beta-Bernoulli. This chapter is based on joint work with Sam Staton [Stein and Staton, 2021]. An implementation of the Gaussian language is available under [Stein, 2021].

Chapter V: Name Generation and Probability on Function Spaces We view the ν-calculus [Pitts and Stark, 1993], a λ-calculus for fresh name generation, as a minimalistic probabilistic language which only knows perfect correlation and independence, and revisit nominal sets, a traditional model of name generation, from this probabilistic viewpoint (Section 25). We recognize that the phenomena of privacy and information hiding, which are crucial to name generation, violate the dataflow axioms of Section 9, making name generation the canonical example of a *non-positive* probabilistic effect (Section 26).

We then give a novel semantic model of the ν-calculus by interpreting fresh name generation as random sampling, unifying the two effects (Section 26.3). This requires us to consider cartesian-closed models of probability. We review quasi-Borel spaces [Heunen et al., 2017],

and prove our main theorem, that is probabilistic semantics is fully abstract up to first-order function types (Section 29). This requires an in-depth analysis of quasi-Borel function spaces, and tools ranging from logical relations over normal forms to descriptive set theory. Because of the full abstraction result, the exotic dataflow properties of name generation carry over to quasi-Borel spaces, with profound impact on the existence of conditional probabilities. Our result generalizes; ideas and intuitions of name generation automatically apply in every higher-order probabilistic programming language. This chapter is based on joint work with Marcin Sabok, Sam Staton and Michael Wolman [Sabok et al., 2021].

Chapter I
Background

This chapter contains a brief exposition of background material for the thesis. We begin with a brief historical overview over probabilistic programming languages (Section 2) and establish examples and terminology around them. The remaining sections are more technical collections of theorems and notations. In Section 3, we review the semantics of programming languages, in particular denotational semantics. We recall the relevant flavors of internal languages (monadic metalanguage, computational λ-calculus and fine-grained call-by-value) and their categorical models, as well as the framework of algebraic effects. In Section 4.2, we review the traditional foundations of probability theory through measure theory. We emphasize the importance of probability monads, foreshadowing their generalization in Section 5.

2 Overview of Probabilistic Programming

One of the pioneers of the semantic study of probabilistic programs was Kozen [1981]. The subject has received renewed interest as a principled and explainable form of ´Bayesian' machine learning. For an introduction to the subject, we refer to [van de Meent et al., 2018; Rainforth, 2017]. For a fuller history, see [Panangaden, 2016]. An interactive textbook with motivations from neuroscience and cognition is available under [Goodman et al., 2016; Goodman and Stuhlmüller, 2014]. We'll now attempt a rough taxonomy of common probabilistic languages, which is by no means exhaustive:

Every probabilistic programming system comprises two parts: A declarative language to specify a statistical inference problem, and a set of inference algorithms to 'run' the program, thereby (approximately) solving the inference problem. General classes inference algorithms are

Monte-Carlo simulation including rejection sampling and importance sampling. More sophisticated methods are Markov chain Monte-Carlo methods (MCMC) such as Metropolis-Hastings or Hamiltonian Monte Carlo (HMC).

Optimization techniques such as variational inference parameterize the posterior as a program whose parameters are then optimized to fit the true posterior

Exact inference In restricted situations, the inference problem can be solved exactly by symbolic inference techniques or even exhaustive enumeration of all possibilities.

In this thesis, we will mainly focus on the declarative language rather than the inference algorithm (though Chapters III and IV contain symbolic inference techniques). Yet, the choice of an inference algorithm dictates a crucial tradeoff in the language design: Highly specialized inference algorithms require restricted languages, while more general-purpose, Turing-complete languages are often left with generic inference procedures.

Classical examples of restricted modelling languages are WINBUGS [Lunn et al., 2000] and JAGS [Plummer, 2003] based on Gibbs sampling, or the more modern STAN [Carpenter et al., 2017] which features Hamiltonian Monte Carlo simulation. The requirement of gradient information for HMC poses an interesting challenge to language design, combining probabilistic and differentiable programming. Another specialized language is BLOG [Russell and Milch, 2006] for modelling unknown classes of objects.

On the side of general-purpose languages, we mention WEBPPL [Goodman and Stuhlmüller, 2014] based on purely-functional JavaScript, ANGLICAN [Tolpin et al., 2016] and its precursor CHURCH [Goodman et al., 2008] based on Clojure/Scheme and MONAD-BAYES [Ścibior et al., 2017], which is a Haskell library. A recent trend is *programmable inference* which extends a general-purpose probabilistic language with fine-grained facilities to customize the inference process, such as PYRO [Bingham et al., 2018] (Python) and GEN [Cusumano-Towner et al., 2019] (Julia).

Another difference between probabilistic languages is how the dependence on data is expressed. We extensively discuss the distinction between *soft conditioning* (scoring) and *exact conditioning* in Chapter IV. We notice that HAKARU [Narayanan et al., 2016] and BIRCH [Murray and Schön, 2018] employ symbolic inference techniques, while INFER.NET [Minka et al., 2018] allows exact conditioning together with approximate inference.

For the purposes of semantics, our reference general-purpose probabilistic language with scoring is taken from [Staton, 2017]. Its types and terms are given by

$$\begin{aligned} A ::= \mathbb{R} \mid \mathsf{P}(A) \mid 1 \mid A \times A \mid \sum_{i \in I} A_i \qquad t ::= {} & x \mid [i,t] \mid \mathsf{case}\, t\, \mathsf{of} \{[i,x] \Rightarrow t\}_{i \in I} \\ & \mid () \mid (t,t) \mid \pi_i(t) \\ & \mid \mathsf{return}(t) \mid \mathsf{let}\, x = t\, \mathsf{in}\, t \\ & \mid f(t) \mid \mathsf{sample}(t) \mid \mathsf{score}(t) \end{aligned}$$

with constructs for sampling and scoring. There are two judgements $\Gamma \vdash_d t : A$ for deterministic and $\Gamma \vdash_p t : A$ for probabilistic terms, following the fine-grained call-by-value paradigm (Section 3.4). The language has fully-definable denotational semantics in s-finite kernels (Section 4.2) and can in fact be seen as the internal language of that category (Section 3.4). We can also extend the language with higher-order functions. Giving denotational semantics to random higher-order functions is subtle, but the category of quasi-Borel spaces (Section 27) is a natural semantic domain for this language [Staton et al., 2016; Heunen et al., 2017].

3 Semantics of Programming Languages

In this section, we will review general techniques in the semantics of programming languages. We will particularly focus on categorical denotational semantics and its interplay with various internal languages (Section 3.1), string diagrams (Section 3.5) and discuss the role of algebraic theories for the presentation of programming languages (Section 3.6).

The meaning of programs is traditionally analyzed from the following viewpoints [Moggi, 1991]

Operational semantics describes how *programs* (closed terms) reduce to *values*. This approach most closely mirrors the execution of a program on a machine, though effects like nondeterminism or random choice may occur.

Denotational semantics associates to every term t a mathematical object $[\![t]\!]$, its *denotation*. Denotation is typically an invariant of reduction, though this requires care when the reduction relation is nondeterministic.

Axiomatic semantics specifies a set of axioms and deduction rules for establishing properties of programs. For example, *program equations* may axiomatize when two terms are considered equal and may be interchanged.

Operational semantics induces a notion of *contextual equivalence* on open programs: Two terms are deemed contextually equivalent if they can be interchanged in every closed context. Contextual equivalence is often difficult to work with directly, because of the quantification over all contexts. Denotational semantics or derivable program equations define a stronger notion of equivalence, which may be easier to work with: If two terms can be shown to be denotationally or derivably equal, they must be contextually equivalent. The converse is known as *full abstraction*: Every two terms which are contextually equivalent can be proven denotationally or derivably equal. This means that the chosen mathematical model or axiomatic system has precisely captured the dynamics of the language.

We define a simple operational semantics for Gaussian probability in Section 17.3 and make use of the operational semantics of name generation in Chapter V. We will not require operational considerations of general-purpose probabilistic languages but point to [Borgström et al., 2016; Staton et al., 2016] as a reference.

Some languages have a single, *intended* denotational model. For others, one specifies a class of possible models. This blends axiomatic and denotational semantics, because two terms are derivably equal if and only if they have the same denotation in *all* possible models.

3.1 Categorical Semantics and Internal Languages

An elegant method of constructing denotational semantics is by interpreting the language as the *internal language* of a category $\mathbb{C}$ with structure. Types A of the language are interpreted as objects $[\![A]\!]$ of $\mathbb{C}$, and terms-in-context $\Gamma \vdash t : A$ are interpreted as a morphism $[\![t]\!] : [\![\Gamma]\!] \to [\![A]\!]$ in $\mathbb{C}$, in a way that composition of morphisms corresponds to substitution in the language of some sort. Constructs of the language then correspond to structure on the category, and reduction rules and program equations translate to properties which the category satisfies. Famously, the simply-typed λ-calculus can be interpreted in cartesian-closed categories this way.

Given a category $\mathbb{C}$, we can form a language where all objects and morphisms are available in the language as types and terms. We can then reason *in the language* instead of in the mathematical structure $\mathbb{C}$. In this way, programming languages naturally give rise to elementary[1] calculi or *metalanguages* for mathematical structures. The advantages are manifold:

[1] as in "of elements", i.e. the sense of Goldblatt, not Sherlock Holmes

(i) the internal language hides nonessential bookkeeping like coherence isomorphisms from view

(ii) we can harness elementary intuitions

(iii) we make mathematical structure amenable to programming language techniques

Objects in a category do not *have elements* the way a set does. Yet internal languages let us pretend they do in a meaningful way: For example, the elementary phrasing of the associativity axiom

$$a : X, b : X, c : X \vdash m(m(a,b),c) = m(a,m(b,c)) \tag{1}$$

is equivalent in the internal language of a cartesian category (a category with finite products) to the statement that the following diagram commutes

$$\begin{array}{ccccc} X \times (X \times X) & \xrightarrow{\mathsf{id}_X \times m} & X \times X & \xrightarrow{m} & X \\ {\scriptstyle \langle\langle \pi_1, \pi_1\pi_2\rangle, \pi_2\pi_2\rangle}\downarrow & & & & \| \\ (X \times X) \times X & \xrightarrow[m \times \mathsf{id}_X]{} & X \times X & \xrightarrow[m]{} & X \end{array}$$

Variables aren't taken to actually represent elements, but act as convenient placeholders for the dataflow described by the program. In our analysis of probabilistic programming, the same fate will befall the notion of *sample* in statistics. Samples are not necessarily actually drawn, instead they are convenient indicators for the dataflow obtained by composing probabilistic computation. In the diagrammatic picture of Markov categories (Section 6), samples are merely wires.

We'll now introduce several important styles of internal language that will be used throughout the thesis.

3.2 Monadic Metalanguage

Moggi's monadic metalanguage [Moggi, 1991] is the internal language of a category together with a *strong monad* T. It pioneered the familiar way Haskell treats effectful computation, by wrapping it in a monad. We'll thus say effects here are *explicit*, tracked by the type system.

The types and terms of the metalanguage are

$$\begin{aligned} A &::= \tau \mid T(A) \mid 1 \mid A \times A \\ t &::= x \mid () \mid (t,t) \mid \pi_i(t) \mid f(t) \mid [t] \mid \mathsf{let}\, x \leftarrow t\, \mathsf{in}\, t \quad i \in \{1,2\} \end{aligned}$$

where τ stands for base types and f stands for function symbols of given arity $A \to B$. The language extends the usual constructions of a cartesian category (tuples, projections) with the return statement $[t]$ and monadic sequencing $\mathsf{let}\, x \leftarrow e\, \mathsf{in}\, t$. The typing rules read

$$\frac{}{\Gamma \vdash x : A}\ (x : A \in \Gamma) \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash [t] : TA} \qquad \frac{\Gamma \vdash e : TA \quad \Gamma, x : A \vdash t : TB}{\Gamma \vdash \mathsf{let}\, x \leftarrow e\, \mathsf{in}\, t : TB}$$

and the following equations are valid

$$\mathsf{let}\, x \leftarrow t \,\mathsf{in}\, [x] = t \tag{M1}$$

$$\mathsf{let}\, x \leftarrow [u] \,\mathsf{in}\, t = t[u/x] \tag{M2}$$

$$\mathsf{let}\, x_2 \leftarrow (\mathsf{let}\, x_1 \leftarrow e_1 \,\mathsf{in}\, e_2) \,\mathsf{in}\, t = \mathsf{let}\, x_1 \leftarrow e_1 \,\mathsf{in}\, \mathsf{let}\, x_2 \leftarrow e_2 \,\mathsf{in}\, t \tag{M3}$$

where in (M3) we assume $x_1 \notin \mathsf{fv}(t), x_2 \notin \mathsf{fv}(e_1)$.

Recall that a monad is an endofunctor $T : \mathbb{C} \to \mathbb{C}$ equipped with natural transformations[2] $\eta : 1 \Rightarrow T$ and join : $TT \Rightarrow T$ satisfying conditions [MacLane, 1971]. For a morphism $f : X \to TY$, we denote by $f^+ = \mathrm{join} \circ Tf : TX \to TY$ its *Kleisli extension* (or *bind* in functional programming terminology).

Now, assume that $\mathbb{C}$ has products and recall that a *strength* for T is a natural transformation with components

$$\mathsf{st}_{X,Y} : X \times TY \to T(X \times Y)$$

again satisfying conditions [Kock, 1972]. A *strong monad* is a monad equipped with such a strength. Fixing an interpretation of base types $[\![X]\!]$ as objects of $\mathbb{C}$, we let $[\![TA]\!] = T([\![A]\!])$ and define for contexts $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ that

$$[\![\Gamma]\!] = [\![A_1]\!] \times \ldots \times [\![A_n]\!].$$

For any term $\Gamma \vdash t : A$ in the metalanguage, an interpretation $[\![t]\!] : [\![\Gamma]\!] \to [\![A]\!]$ is defined using the cartesian structure of $\mathbb{C}$ and the following monadic operations

$$[\![[t]]\!] : [\![\Gamma]\!] \xrightarrow{[\![t]\!]} [\![A]\!] \xrightarrow{\eta_{[\![A]\!]}} [\![TA]\!]$$

$$[\![\mathsf{let}\, x \leftarrow e \,\mathsf{in}\, t]\!] : [\![\Gamma]\!] \xrightarrow{\langle \mathsf{id}_{[\![\Gamma]\!]}, [\![e]\!] \rangle} [\![\Gamma]\!] \times T[\![A]\!] \xrightarrow{\mathsf{st}_{[\![\Gamma]\!],[\![A]\!]}} T([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{T[\![t]\!]} TT[\![B]\!] \xrightarrow{\mathrm{join}_{[\![B]\!]}} [\![TB]\!]$$

The equations (M1)-(M3) morally correspond to the monad axioms of unitality and associativity.

When used as the internal language of a category with a strong monad $(\mathbb{C}, T)$, we add basic types for all objects of $\mathbb{C}$ and function symbols for all morphisms in $\mathbb{C}$. If $\mathbb{C}$ is cartesian closed, we employ the usual λ-syntax

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash f\, t : B}$$

for function types $A \to B$ which are interpreted as the function spaces of $\mathbb{C}$

$$[\![A \to B]\!] = [\![A]\!] \Rightarrow [\![B]\!]$$

Lastly, we note that one can extend let syntax to work with algebras of a monad. If the interpretation of a type B has a dedicated T-algebra structure $([\![B]\!], \beta)$, we allow the judgement

$$\frac{\Gamma \vdash e : TA \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \mathsf{let}\, x \leftarrow e \,\mathsf{in}\, t : B}$$

[2] we write join because μ already heavily overloaded in this thesis.

whose interpretation is

$$[\![\Gamma]\!] \xrightarrow{\langle \mathsf{id}_{[\![\Gamma]\!]}, [\![e]\!]\rangle} [\![\Gamma]\!] \times T[\![A]\!] \xrightarrow{\mathsf{st}_{[\![\Gamma]\!],[\![A]\!]}} T([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{T[\![t]\!]} T[\![B]\!] \xrightarrow{\beta} [\![B]\!] \tag{2}$$

This subsumes the previous monadic sequencing using the canonical T-algebra structure $(T[\![A]\!], \mathsf{join}_{[\![A]\!]})$ on $[\![TA]\!]$. Laws (M2)-(M3) remain valid as per the algebra laws.

3.3 Computational λ-calculus

The monadic metalanguage is a *language of values* where effectful sequencing is explicitly invoked through monadic let, and effects are tracked in the type system by the monad T. We contrast this with the computational λ-calculus (λ_c-calculus) of Moggi [1989], where computational effects are implicit in the type system and may happen at any point. This is reminiscent of the ML family of functional programming languages. Care is needed to reflect the call-by-value strategy of the language in its theory, by distinguishing computations from values and fixing an evaluation order. The types and terms are

$$\begin{aligned} A &::= \tau \mid 1 \mid A \times A \\ t &::= x \mid () \mid (t,t) \mid \pi_i(t) \mid f(t) \mid \mathsf{let}\ x\ =\ t\ \mathsf{in}\ t \end{aligned}$$

where the let-binding now has the type

$$\frac{\Gamma \vdash e : A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \mathsf{let}\ x\ =\ e\ \mathsf{in}\ t : B}$$

This is a language of computations. Familiar equations hold, for example

$$\begin{gathered} (\mathsf{let}\ x\ =\ e\ \mathsf{in}\ x) = e \\ (\mathsf{let}\ x\ =\ x\ \mathsf{in}\ e) = e \\ (\mathsf{let}\ x_2\ =\ (\mathsf{let}\ x_1\ =\ e_1\ \mathsf{in}\ e_2)\ \mathsf{in}\ e) = (\mathsf{let}\ x_1\ =\ e_1\ \mathsf{in}\ \mathsf{let}\ x_2\ =\ e_2\ \mathsf{in}\ e) \end{gathered}$$

where $x_1 \notin \mathsf{fv}(e), x_2 \notin \mathsf{fv}(e_1)$. On the other hand, unlike in the monadic metalanguage, λ_c-programs are written in direct style that allows arbitrary nesting of effectful computations. This makes the equational theory more complex and we need to add explicit sequencing information such as

$$(s,t) = (\mathsf{let}\ x\ =\ s\ \mathsf{in}\ \mathsf{let}\ y\ =\ t\ \mathsf{in}\ (x,y))$$

as well as defining a special value predicate V to capture substitution as in

$$(\mathsf{let}\ x\ =\ V\ \mathsf{in}\ u) = u[V/x]$$

For more detail, see Moggi [1989] and our simpler presentation of the ground *commutative* fragment of λ_C in Section 7.

A convenient approach to studying the computational λ-calculus is translating it into the monadic metalanguage. Following Moggi [1991], we recursively define for every $\Gamma \vdash t : A$ a

translation $\Gamma^\circ : t^\circ : T(A^\circ)$ by

$$\begin{aligned} x^\circ &= [x] & ()^\circ &= [()] \\ (s,t)^\circ &= \mathsf{let}\, x \leftarrow s^\circ \,\mathsf{in}\, \mathsf{let}\, y \leftarrow t^\circ \,\mathsf{in}\, [(s,t)] & (\pi_i(t))^\circ &= \mathsf{let}\, x \leftarrow t^\circ \,\mathsf{in}\, [\pi_i(x)] \\ (f(t))^\circ &= \mathsf{let}\, x \leftarrow t^\circ \,\mathsf{in}\, f(x) & (\mathsf{let}\, x = e \,\mathsf{in}\, t)^\circ &= \mathsf{let}\, x \leftarrow e^\circ \,\mathsf{in}\, t^\circ \end{aligned}$$

We notice how that translation makes all sequencing information explicit. Care is needed when adding λ-abstraction to the language. It is important that effectful call-by-value function spaces are translated as

$$(A \to B)^\circ = A^\circ \to T(B^\circ).$$

One consequence is that while currying is valid in the monadic metalanguage, i.e.

$$(A \to B \to C) \cong A \times B \to C$$

the same does *not* hold in the computational λ-calculus. First-order (i.e. non-nested) function types thus already encode considerable amounts of complexity.

3.4 Fine-grained call-by-value

A different analysis of the computational λ-calculus has been presented in [Levy et al., 2003]. It is particularly appealing for the semantics of ground programs because it avoids all mention of a monad. Being a dedicated call-by-value analysis, it formulates two separate judgements $\vdash_v$ for values and $\vdash_c$ for computations. We extend the λ_c-calculus by a term return t and have judgements

$$\frac{}{\Gamma \vdash_v x : A}\ (x : A \in \Gamma) \qquad \frac{\Gamma \vdash_v t : A}{\Gamma \vdash_c \mathsf{return}\, t : A} \qquad \frac{\Gamma \vdash_c e : A \quad \Gamma, x : A \vdash_c t : B}{\Gamma \vdash_c \mathsf{let}\, x = e \,\mathsf{in}\, t : B}$$

That is, return t promotes a value term t to a computation, and let sequences computations together. Fine-grained call-by-value is the internal language of *Freyd categories*, which comprise a category $\mathbb{V}$ of values, a category $\mathbb{C}$ of computations and an identity-on-objects functor $J : \mathbb{V} \to \mathbb{C}$ promoting value morphisms to computations. The computation category $\mathbb{C}$ comes equipped with the following structure:

Recall that a *symmetric premonoidal category* [Power and Robinson, 1997] is a category $\mathbb{C}$ equipped with the following structure

- an object I called *unit*
- a tensor construction on objects $(A, B) \mapsto A \otimes B$
- for every object A, an endofunctor $A \rtimes (-)$ sending B to $A \otimes B$
- for every object A, an endofunctor $(-) \ltimes A$ sending B to $B \otimes A$
- isomorphisms

$$\begin{aligned} \rho_A &: A \otimes I \cong A \\ \alpha_{A,B,C} &: (A \otimes B) \otimes C \cong A \otimes (B \otimes C) \\ \mathsf{swap}_{A,B} &: A \otimes B \cong B \otimes A \end{aligned}$$

satisfying coherence conditions. In such a category, we can compose morphisms $f : A \to B$ and $f' : A' \to B'$ in parallel as

$$(B \rtimes f') \circ (f \ltimes A') : A \otimes A' \to B \otimes B' \tag{3}$$

but a fixed order of sequencing needs to be chosen. This way, premonoidal categories formalize the evaluation order necessary for noncommutative effects. In particular (3) needs not be equal to the composite $(f \ltimes B') \circ (A \rtimes f')$. A morphism $f : A \to B$ is called *central* if for every other $f' : A' \to B'$, the order of parallel composition is unambiguous, i.e.

$$(B \rtimes f') \circ (f \ltimes A') = (f \ltimes B') \circ (A \rtimes f')$$
$$(f' \ltimes B) \circ (A' \rtimes f) = (B' \rtimes f) \circ (f' \ltimes A)$$

If every morphism is central, then $\mathbb{C}$ becomes a *symmetric monoidal category* (Section 3.5) and we write $f \otimes g$ for the unambiguous parallel composite of morphisms. The construction $\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ is then a bifunctor, that is it satisfies the so-called *interchange law*

$$(g \otimes g') \circ (f \otimes f') = (g \circ f) \otimes (g' \circ f') \tag{4}$$

Because synthetic probability is only concerned with commutative effects, we won't be needing premonoidal categories much except to show that they are monoidal. We have decided to recall the full definition because of various connections to the programming language literature:

Definition 3.1 (Levy et al. [2003, 4.1]) A *Freyd category* consists of a cartesian category $\mathbb{V}$, a symmetric premonoidal category $\mathbb{C}$ and an identity-on-objects functor $J : \mathbb{V} \to \mathbb{C}$ which preserves strict symmetric premonoidal structure and whose image is central.

Freyd categories subsume the earlier monadic models by the following construction:

Proposition 3.2 (Power and Robinson [1997, 3.5]) *If $\mathbb{V}$ is cartesian and T a strong monad on $\mathbb{V}$, then the Kleisli category $\mathbb{V}_T$ is symmetric premonoidal and the functor $J : \mathbb{V} \to \mathbb{V}_T$ is a Freyd category.*

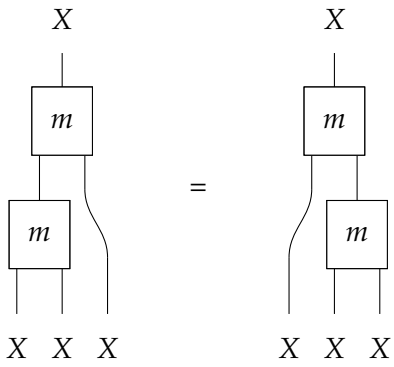
Not every Freyd arises as a Kleisli category. A relevant example for probabilistic computation is the Freyd category $\mathsf{Meas} \to \mathsf{SfKer}$ of measurable maps and s-finite kernels, as defined in Definition 4.8. This construction is not known to arise from a monad on Meas. The internal language of this Freyd category is a fragment of the reference probabilistic language from Section 2, and recovers the semantics given in [Staton, 2017].

We say that a Freyd category is *commutative* if every morphism in $\mathbb{C}$ is central. A Kleisli category $\mathbb{V}_T$ is commutative as a Freyd category if and only if the monad T is commutative in the sense of Definition 3.3. Commutative Freyd categories are very close to one of the central notions of this thesis, CD categories. We'll discuss their precise differences in Section 6.3. It should not come as a surprise that the internal language of CD categories (Section 7) resembles fine-grained call-by-value. Due to commutativity, its theory can be simplified and presented very concisely.

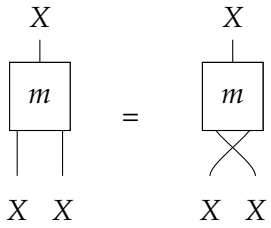
3.5 Graphical Language: String Diagrams

Let $(\mathbb{C}, I, \otimes)$ be a symmetric monoidal category [MacLane, 1971]. The tensor differs from a product in that information cannot be copied or discarded at will. We call $\mathbb{C}$ *cartesian* if $\otimes$ is a categorical product, and *semicartesian* if I is a terminal object.

The interchange law (4) in monoidal categories makes them amenable to a different type of internal language: The graphical calculus of string diagrams. Here, morphisms are drawn as boxes in the plane, and wires are labeled with objects. In our convention, 'time flows upwards'. For example, the associativity equation (1) for a morphism $m : X \otimes X \to X$ uses its variables linearly and can be interpreted in any monoidal category as the string diagram

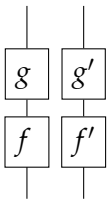


Commutativity requires us to use the swap isomorphism, which is depicted as crossing wires.



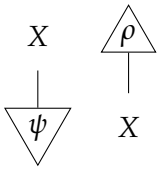
The appeal of string diagrams is that all coherence isomorphisms are hidden in the geometry of the plane: One can show that a transformation of string diagrams is derivable from the axioms of symmetric monoidal categories if and only if it can be obtained from certain geometric movements in the plane [Selinger, 2011]. For further work on coherence of monoidal categories and the rigorous manipulation of string diagrams, see [Joyal and Street, 1991, 1993].

For example, the *interchange law* (4) corresponds to the un-ambiguity of reading the string diagram



In contrast, premonoidal categories are usually not rendered graphically. If one does, extra 'control flow wires' must be attached to keep track of the order of effects [Jeffrey, 1998;

Møgelberg and Staton, 2014]. In string diagrams, all coherence isomorphisms are completely absent from the graphical presentation, and wires typed I need not be drawn. This leads to *states* $\psi : I \to X$ and *effects* $\rho : X \to I$ being drawn as follows



(5)

There seems to be something inherently two-dimensional about monoidal categories which makes them cumbersome to use when compressed in one-dimensional programming notation (though attempts exist [Staton et al., 2017a]). In the presence of additional structure for copying and discarding, programming syntax becomes useful again. We exploit this when we define the internal language of CD categories (Section 7).

3.6 Monads and Algebraic Effects

The importance of monads to effectful programming has been understood since Moggi. An important class of monads arises from algebraic theories, which makes universal algebra an important tool for presenting effects in programming languages. This paradigm has been termed *algebraic effects*, going back to Plotkin and Power [2003]. It allows us to give a focused analysis of the phenomena of interest, while making it routine to combine these features with other constructs to obtain a full programming language (e.g. [Ahman and Staton, 2013; Johann et al., 2010; Kammar and Plotkin, 2012; Pretnar, 2010]). Algebraic effects are fundamental to the development of languages such as EFF [Bauer and Pretnar, 2015] and KOKA [Leijen, 2014].

There is a well-known equivalence between finitary monads on Set, algebraic theories and Lawvere theories [Lawvere, 1963]. We will now recall parts of this equivalence, as algebraic theories are a recurring tool in this thesis. First-order algebraic theories will be generalized to the second-order algebraic theories in Section 3.7.

An algebraic theory can be presented using operations and equations. For example, the theory of (real) vector spaces can be presented using a binary operation $(+)$, a constant 0 and unary operations $r \cdot (-)$ for every $r \in \mathbb{R}$. A well-known system of axioms is

$$\begin{aligned} x + (y + z) &= (x + y) + z & a \cdot (b \cdot x) &= (a \cdot b) \cdot x \\ x + y &= y + x & 1 \cdot x &= x \\ x + 0 &= x & a \cdot (x + y) &= a \cdot x + a \cdot y \\ 0 \cdot x &= 0 & (a + b) \cdot x &= a \cdot x + b \cdot x \end{aligned}$$

Such presentations are not unique, that is, different systems of axioms can be chosen for the same theory. Furthermore, not even the operation signature is unique. We could have included, say, a unary operation $(-)$, which is expressible in our presentation as $-x = (-1) \cdot x$. Note that we could *not* have dispensed with the constant 0, as doing so would

make the empty set a valid vector space. To overcome the non-uniqueness of presentations, we can seek to give an *unbiased presentation* of a theory which doesn't prefer any particular set of operation symbols. We do this by considering all possible *derived operations* at once, that is all terms $x_1, \ldots, x_n \vdash t$ of the theory, modulo equations. In the theory of vector spaces, to give a term-modulo-equations over $x_1, \ldots, x_n$ is to give a formal linear combination

$$a_1 x_1 + \ldots + a_n x_n \text{ with } a_i \in \mathbb{R} \tag{6}$$

Once we have identified the derived operations, we need to specify how these combine under substitution. This substitution structure can be organized in several equivalent ways, notably abstract clones, Lawvere theories and monads.

Monads For a set X, we define

$$T(X) = \{\text{terms } x_1, \ldots, x_n \vdash t \text{ modulo equations} \mid x_1, \ldots, x_n \in X\}$$

The construction T has the structure of a monad $\mathsf{Set} \to \mathsf{Set}$ whose unit considers an element $x \in X$ as a one-variable term, and Kleisli composition is substitution. Monads of this form are strong and finitary (see (16)), and every such monad arises from an algebraic theory.

Lawvere theories We define a category $\mathbb{L}$ whose objects are natural numbers $n \in \mathbb{N}$ and whose morphisms are tuples of terms

$$\mathbb{L}(m, n) = \{(t_1, \ldots, t_n) \mid x_1, \ldots, x_m \vdash t_i \text{ modulo equations }\}$$

The identity is given by variables $(x_1, \ldots, x_n)$ and composition of morphisms is substitution.

There is well-known equivalence between the following structures (e.g. [Hyland and Power, 2007])

(i) Finitary monads and their algebras

(ii) Lawvere theories and their models (product-preserving functors into Set)

(iii) Unbiased algebraic theories and their models

The simplest consistent algebraic theory has no operation symbols at all; this is sometimes called the *theory of equality*. Its associated monad is the identity monad $T(X) \cong X$; its associated Lawvere theory is $\mathsf{Fin}^{\mathsf{op}}$, the dual of finite sets and functions.

Generic effects Given a monad T, we recover the operations of the theory as certain well-behaved transformations [Plotkin and Power, 2003]: An *n-ary algebraic operation* for T is a set-indexed family of functions $\omega_X : (T(X))^n \to T(X)$ satisfying for all $f : X \to T(Y)$ and $t_1, \ldots, t_n \in T(X)$ that

$$f^+ \omega_X(t_1, \ldots, t_n) = \omega_Y(f^+ t_1, \ldots, f^+ t_n)$$

where $f^+ : T(X) \to T(Y)$ is the Kleisli extension of f. The algebraic operations are in one-to-one correspondence with the elements $\xi \in T(\{1, \ldots, n\})$, called *generic effects*, where we set $\xi = \omega_n([1], \ldots, [n])$ and conversely reconstruct

$$\omega_X(t_1, \ldots, t_n) = (\lambda n.t_n)^+ \xi$$

The correspondence of algebraic operations and generic effects is crucial when we axiomatize programming languages using algebra, e.g. Sections 11.2 and 21.2.

3.6.1 Example: Monads of Linear Combinations

Important running examples in this thesis are derived from the monad of linear combinations, which corresponds to the unbiased presentation of vector spaces in (6): For a set X, a formal linear combination over X can be identified with a coefficient function $p : X \to \mathbb{R}$ which has *finite support*, that is $p(x) = 0$ for all but finitely many x. The monad defined by

$$F(X) = \{p : X \to \mathbb{R} \text{ finitely supported } \}$$

is thus called the linear combination monad or free vector space monad. Its unit is the "Dirac" linear combination

$$\eta_X(x) = \delta_x \text{ where } \delta_x(y) = [x = y]$$

and given $p \in F(X)$ and $f : X \to F(Y)$, their Kleisli extension is

$$f^+ p(y) = \sum_{x \in X} p(x) \cdot f(x)(y)$$

The Lawvere theory $\mathbb{L}_{\mathsf{Vec}}$ of vector spaces admits a similarly nice description. To give a morphism $m \to n$ is to give a linear function $\mathbb{R}^n \to \mathbb{R}^m$ (note the contravariance), and substitution corresponds to reverse function composition. Equivalently, morphisms $m \to n$ are matrices $\mathbb{R}^{m \times n}$ and composition is reverse matrix multiplication.

We can use the previous paragraph to give concise unbiased presentations of important algebraic theories in the form of submonads of F.

Commutative monoids A *formal sum* is a formal linear combination with nonnegative integer coefficients. Other words for this are *finite multiset*, or *bag*. The bag monad is thus defined as

$$B(X) = \{p : X \to \mathbb{N} \text{ finitely supported } \} \tag{7}$$

This monad gives an unbiased presentation of the theory of commutative monoids. The corresponding Lawvere theory is given by matrices over the natural numbers (under reverse composition)

$$\mathbb{L}_{\mathsf{CMon}}(m, n) = \mathbb{N}^{m \times n}$$

Affine spaces An affine combination is a linear combination of total weight 1. The affine combination monad is thus defined as

$$D^{\pm}(X) = \{p : X \to \mathbb{R} \text{ finitely supported } \mid \sum_{x \in X} p(x) = 1\} \tag{8}$$

An affine space is precisely an algebra for this monad, and homomorphisms of affine spaces are called affine or affine-linear functions. Note that because $D^{\pm}(\emptyset) = \emptyset$, the theory of affine spaces has no constant symbol. So unlike vector spaces, the empty set *is* a valid affine space. In the Lawvere theory, $\mathbb{L}_{\mathsf{Aff}}(m, n)$ consists of $(m \times n)$-matrices whose columns sum to 1.

Convex sets A convex combination is a nonnegative linear combination of total weight 1. The convex combination monad is thus defined as

$$D(X) = \{p : X \to [0,1] \text{ finitely supported } \mid \sum_{x \in X} p(x) = 1\} \tag{9}$$

In the corresponding Lawvere theory, we have

$$\mathbb{L}_{\mathsf{Cvx}}(m, n) = \{A \in \mathbb{R}^{m \times n} \text{ column-stochastic}\}.$$

An (abstract) *convex set* is a set where convex combinations may be taken. The theory can be presented using a family of operations $(+_p)$ for $p \in [0,1]$ corresponding to the binary convex combination

$$x +_p y = (1-p)x + py \tag{10}$$

subject to the following axioms due to Stone [1949]

$$x +_0 y = x \qquad x +_p x = x$$
$$x +_{(1-p)} y = y +_p x \qquad x +_p (y +_q z) = (x +_r y) +_{pq} y \text{ when } p(1-q) = (1-pq)r$$

Every convex subset A of a real vector space obtains the structure of an abstract convex set by (10) and Stone has characterized the abstract convex sets that arise that way. The free convex set $D(n)$ on n generators is isomorphic to the $(n-1)$-dimensional simplex.

We will give a probabilistic interpretation of the convex combination monad in Definition 4.1. Under the probabilistic reading, the generic effect of the convex combination $(+_p)$ is the coin flip $\mathsf{flip}_p \in D(2)$ with bias p. The affine combination monad will have an interpretation as a generalized probability theory, potentially allowing negative probabilities, in Section 5.4.

3.6.2 Commutative Monads

If our monad of interest comes from an algebraic theory, then the dataflow properties of the monad reflect properties of the theory. Recall the following definitions:

Definition 3.3 A strong monad T is called *commutative* if the following diagram commutes

$$\begin{array}{ccccc} T(X) \times T(Y) & \xrightarrow{\mathsf{st}_{TX,Y}} & T(TX \times Y) & \xrightarrow{T(\mathsf{st}'_{X,Y})} & T^2(X \times Y) \\ \downarrow{\scriptstyle \mathsf{st}'_{X,TY}} & & & & \downarrow{\scriptstyle \mathrm{join}_{X \times Y}} \\ T(X \times TY) & \xrightarrow[T(\mathsf{st}_{X,Y})]{} & T^2(X \times Y) & \xrightarrow[\mathrm{join}_{X \times Y}]{} & T(X \times Y) \end{array} \tag{11}$$

where $\mathsf{st}'_{X,Y}$ is a *costrength* defined by

$$TX \times Y \xrightarrow{\mathsf{swap}_{TX,Y}} Y \times TX \xrightarrow{\mathsf{st}_{Y,X}} T(Y \times X) \xrightarrow{T(\mathsf{swap}_{Y,X})} T(X \times Y)$$

Under the monadic metalanguage, this condition is equivalent to commutativity in the programming language sense. That is, the monad T is commutative if and only if

$$(\mathsf{let}\, x \leftarrow u \,\mathsf{in}\, \mathsf{let}\, y \leftarrow v \,\mathsf{in}\, t) = (\mathsf{let}\, y \leftarrow v \,\mathsf{in}\, \mathsf{let}\, x \leftarrow u \,\mathsf{in}\, t) \tag{12}$$

holds whenever $y \notin \mathsf{fv}(u)$ and $x \notin \mathsf{fv}(v)$.

We say an algebraic theory is commutative if its associated monad is. We can spell this condition out concretely in terms of a presentation: Two operation symbols ω,ϕ of arities m, n are said to *commute* if the following equation is derivable

$$\omega(\phi(x_{11}, \ldots, x_{1n}), \ldots, \phi(x_{m1}, \ldots, x_{mn})) = \phi(\omega(x_{11}, \ldots, x_{m1}), \ldots, \omega(x_{1n}, \ldots, x_{mn})) \tag{13}$$

An algebraic theory is commutative if all operation symbols commute.

The free vector space monad is commutative; the composite morphism

$$\otimes : F(X) \times F(Y) \to F(X \times Y)$$

in (11) corresponds to taking the pointwise product

$$(p \otimes q)(x, y) = p(x) \cdot q(y) \tag{14}$$

The convex combination monad D is immediately seen to be commutative as a submonad of F. One can also verify this using Stone's presentation: Instantiating (13) with the operations $(+_p)$ and $(+_q)$, the following equation is indeed derivable from the axioms:

$$(x +_p y) +_q (z +_p w) = (x +_q z) +_p (y +_q w)$$

3.6.3 Affine Monads

Definition 3.4 A monad T on a category $\mathbb{C}$ with terminal object 1 is called *affine* if $T(1) \cong 1$.

This means that computations whose result is unused can be discarded. T is affine if and only if the equation

$$(\mathsf{let}\, x \leftarrow u \,\mathsf{in}\, t) = t \tag{15}$$

holds whenever x is not free in t. Given the presentation of an algebraic theory, the associated monad is affine if and only if all operations ω are *idempotent*, i.e.

$$\omega(x, \ldots, x) = x$$

is derivable (constant symbols are never idempotent). The theories of affine spaces and convex sets are indeed affine, but the theory of vector spaces and abelian groups are not. For example $x +_p x = x$ holds for all x, but $x + x = x$ is not derivable. For a treatment of commutative and idempotent theories from the perspective of universal algebra, see [Linton, 1966; Romanowska and Smith, 2002].

3.6.4 Finitary Monads

Starting from an algebraic theory, every term will only make use of finitely many variables. This condition makes the associated monad T *finitary*, meaning it preserves filtered colimits[3]. In particular, the set $T(X)$ can be written as a coend

$$T(X) \cong \int^{n \in \mathsf{Fin}} T(n) \times X^n \tag{16}$$

Conversely, every finitary monad on Set comes from an algebraic theory. Note that such monads usually do not preserve other colimits such as coproducts. For example, the presence of a nontrivial binary operation ω introduces cross-terms like $\omega(x, y)$ in $T(\{x\} + \{y\})$ which do not lie in $T(\{x\})$ or $T(\{y\})$ respectively. From the programmer's perspective, this phenomenon corresponds to a branching of control flow such as for probabilistic choice $x +_p y$. The only cocontinuous monad on Set is the writer monad (Section 5.2), however will find interesting examples over other categories in Section 21.2 and Section 25.2.

3.7 Second-order Algebraic Theories

Second-order algebraic theories are an equational fragment of second-order logic, which extends the usual algebraic theories to encompass variable-binding operations such as λ-abstraction or logical quantifiers. Such theories will serve as crucial tools in this thesis for describing generative effects in programming languages such as urn creation (Section 11.1), random sampling (Section 21.2) or fresh name generation (Section 25.2).

Second-order theories were originally introduced in [Fiore and Hur, 2010] and [Fiore and Mahmoud, 2010]. Here, we are going to use a particular flavor called *parameterized algebraic theories* due to Staton [2013c].

We shall begin with an extended example along the lines of [Staton, 2013a]: In predicate logic, we distinguish two types of expressions, namely terms and formulas.

(i) terms are built from variables x, y, z ranging over the domain of discourse, and operations such as $x + y$.

(ii) formulas are built from variables φ, χ standing for predicates, and operations such as $\varphi \vee \chi$. Predicates may themselves take parameters, such as $\psi[x, x + y]$, and some operations *bind* variables, like the quantifier in $\exists x.\psi[x, x + y]$. Note that $\varphi[]$ is the formula built from predicate variable φ taking zero parameters.

We can also discuss two types of equations: Those between terms, like $x + y = y + x$, are traditional first-order equations. Those between formulas like

$$\varphi[] \vee (\chi[] \vee \varphi[]) = \varphi[] \vee \chi[] \qquad \text{or} \qquad (\exists x.\exists y.\psi[x, y]) = (\exists y.\exists x.\psi[x, y])$$

are second-order equations, which express that certain formulas (with free predicates) are equivalent. We can reason algebraically using such equations, both by specializing terms for

[3] technically, the condition which generalizes is the preservation of *sifted* colimits. This makes no difference for monads on Set

variables, and formulas for predicates (explained below).

We now generalize the predicate logic example using the following generic terminology:

(i) variables x, y, z will be called *parameters*, and expressions constructed from them are *terms* $x + y$

(ii) variables φ, χ will be called *metavariables*. Each metavariable has an arity $\varphi : n$. We'll use *computation* as a generic name for second-order expressions built up from metavariables and parameters.

We fix a first-order algebraic theory $\mathbb{S}$ which serves as the *theory of parameters*, and provides us with the construction and equality judgement $x_1, \ldots, x_n \vdash s = t$ for *terms*.

A *signature* $\mathbb{T}$ *parameterized by* $\mathbb{S}$ is given by a set of function symbols $F, G, \ldots$ with an arity written $F : (p \mid n_1, \ldots, n_k)$ where p is a natural number and $(n_1, \ldots, n_k)$ a list of natural numbers. We read this arity as follows: F takes p terms and k computations, where the i-th computation itself expects n_i parameters. If $n_i > 0$, the operation F can be thought of as binding these parameters.

In the predicate logic example, we have the following signature: Disjunction is $\vee : (0 \mid 0, 0)$ because it takes two closed formulas and no variables. The existential quantifier has signature $\exists : (0 \mid 1)$ because it takes one formula with one free parameter. We could also introduce a ternary operation $(- = -) \wedge (-) : (2 \mid 0)$ which compares two parameters for equality and combines the result with some other formula.

Computations are constructed in a two-zone context $\Gamma \mid \Delta$ where $\Gamma = (x_1, \ldots, x_n)$ is a context of parameters and Δ is a context of metavariables. The rules for forming computations are plugging terms into metavariables

$$\frac{\Gamma \vdash t_1 \quad \cdots \quad \Gamma \vdash t_{n_i}}{\Gamma \mid \varphi_1 : n_1 \ldots \varphi_k : n_k \vdash \varphi_i[t_1, \ldots, t_{n_i}]} \tag{MVar}$$

and by applying operations of arity $F : (p \mid n_1, \ldots, n_k)$ as

$$\frac{\Gamma \vdash t_1 \quad \ldots \quad \Gamma \vdash t_p \quad \Gamma, a_{11}, \ldots, a_{1n_1} \mid \Delta \vdash u_1 \quad \ldots \quad \Gamma, a_{k1}, \ldots, a_{1n_k} \mid \Delta \vdash u_k}{\Gamma \mid \Delta \vdash F(t_1, \ldots, t_p, \vec{a}_1.u_1, \ldots, \vec{a}_k.u_k)} \tag{F-App}$$

Weakening and exchange rules for contexts are derivable and the names of the parameters bound in the (F-App)-rule are considered up to α-equivalence.

An example computation over the signature of predicate logic is $x, y \mid \varphi : 2 \vdash \varphi[y, x]$. Applying the operation $\exists$ gives $x \mid \varphi : 2 \vdash \exists(y.\varphi[y, x])$ which is more idiomatically rendered as $\exists y.\varphi[y, x]$.

In second-order algebra, we can define two notions of substitution, terms-for-parameters and computations-for-metavariables. The latter substitution is capture-avoiding.

Term-for-variable Given a computation u and a term t, we form $u[t/a]$ by replacing every occurrence of the free parameter a by t. For example $(\exists y.\varphi[x, x+y])[0/x] = \exists y.\varphi[0, 0+y]$. The following typing rule is derivable

$$\frac{\Gamma, a \mid \Delta \vdash u \quad \Gamma \vdash t}{\Gamma \mid \Delta \vdash u[t/a]}$$

Computation-for-metavariable Given a computation u and another computation w, we form $u[\vec{a}.w/\varphi]$ by replacing every occurrence of the metavariable $\varphi[a_1, \ldots, a_n]$ (of arity n) with w; that is, the extra free parameters $a_1, \ldots, a_n$ of w get bound to the respective arguments of φ. Instructive examples:

$$(\exists x.\varphi[x, x+y])[a\, b.\psi[a-b]/\varphi] = \exists x.\psi[x-(x+y)]$$
$$(\exists x.\psi[x])[y.\chi[x-y]/\psi] = \exists y.\chi[x-y]$$

The derivable typing rule for this kind of substitution reads

$$\frac{\Gamma \mid \Delta, \varphi : n \vdash u \qquad \Gamma, a_1, \ldots, a_n \mid \Delta \vdash w}{\Gamma \mid \Delta \vdash u[\vec{a}.w/\varphi]}$$

Equational logic An equational logic $\equiv$ is formed by closing a set of axioms under all substitution instances, weakening, reflexivity, symmetry, transitivity and the congruence rules

$$\frac{\Gamma \vdash t_1 = t_1' \quad \cdots \quad \Gamma \vdash t_{n_i} = t_{n_i}'}{\Gamma \mid \varphi_1 : n_1 \ldots \varphi_k : n_k \vdash \varphi_i[t_1, \ldots, t_{n_i}] = \varphi_i[t_1', \ldots, t_{n_i}']}$$

$$\frac{\Gamma \vdash t_1 = t_1' \quad \ldots \quad \Gamma \vdash t_p = t_p' \qquad \Gamma, \vec{a}_1 \mid \Delta \vdash u_1 \equiv u_1' \quad \ldots \quad \Gamma, \vec{a}_k \mid \Delta \vdash u_k \equiv u_k'}{\Gamma \mid \Delta \vdash F(t_1, \ldots, t_p, \vec{a}_1.u_1, \ldots \vec{a}_k.u_k) \equiv F(t_1', \ldots, t_p', \vec{a}_1.u_1', \ldots \vec{a}_k.u_k')}$$

Note that these rules make use of the equality relation $\Gamma \vdash t_1 = t_1'$ of S to let us simplify terms of parameters. One subtlety is that while weakening is valid for equations, strengthening is not; therefore we must take care to always annotate our equations with contexts: For example, the following equation is valid for predicate logic

$$x \mid \varphi : 0 \vdash \exists y.\varphi[] \equiv \varphi[] \tag{17}$$

Even though the equation does not use the parameters x, we cannot remove x from context: The following equation is not valid

$$- \mid \varphi : 0 \vdash \exists y.\varphi[] \equiv \varphi[] \tag{18}$$

Intuitively, the second equation is invalid if the domain of discourse is empty (see Example 3.5).

Semantics The interpretation of second-order algebra is straightforward in cartesian closed categories $\mathbb{C}$. We choose an object C of computation outcomes and an object of parameters P (which is a model of $\mathbb{S}$), and interpretations

$$[\![F]\!] : P^{\ell} \times C^{P^{n_1}} \times \ldots \times C^{P^{n_k}} \to C$$

for every operation symbol $F : (\ell \mid n_1, \ldots, n_k)$. We then recursively extend the interpretation to all computations, making use of λ-abstraction for parameter binding.

For example, the standard boolean semantics of predicate logic in $\mathbb{C} = \mathsf{Set}$ takes $P = X$ as the domain of discourse and $C = 2$ as the object of outcomes. The semantics associates to every formula $x_1, \ldots, x_\ell \mid \varphi_1 : n_1, \ldots, \varphi_k : n_k \vdash u$ the obvious second-order predicate

$$[\![u]\!] : X^p \times 2^{X^{n_1}} \times \ldots \times 2^{X^{n_k}} \to 2$$

Here, the interpretation of the operation symbol $\exists$ is the function $[\![\exists]\!] : 2^X \to 2$ with $[\![\exists]\!](p) = \mathsf{true}$ iff there is some $x \in X$ with $p(x) = \mathsf{true}$.

Example 3.5 Both sides of (17) denote the same function

$$p : X \times 2 \to 2, (x, b) \mapsto b$$

However the left hand side of (18) denotes the function

$$[\![\exists y.\varphi[]]\!] : 2 \to 2, b \mapsto \mathsf{false}$$

when $X = \emptyset$. This demonstrates why the context for equations may not be strengthened.

It can be shown that $\equiv$ is a sound and complete deduction system with respect to semantics in cartesian closed categories [Staton, 2013b]. Full cartesian closure is not necessary to give a sound model of second-order algebra, and we will see simpler types of models in Section 12 and Section 21.2. Term models do however naturally live in presheaf categories, which are cartesian closed. Parameterized theories can be understood from the following viewpoints

(i) a presentation of finitary monads on presheaf categories [Staton, 2013a, Corollary 1]

(ii) a presentation of an enriched Lawvere theory or Freyd category [Staton, 2014]

We won't make the enriched category theory precise here but it should not come as a surprise that parameterized theories are useful in the algebraic presentation of programming languages (Section 3.3) and Markov categories (Section 7). We briefly sketch how to obtain the monad: Let $\mathbb{S}$ denote the Lawvere theory for the theory of parameters, and let $\widehat{\mathbb{S}} = [\mathbb{S}^{\mathsf{op}}, \mathsf{Set}]$ denote the category of presheaves on $\mathbb{S}$. If $X \in \widehat{\mathbb{S}}$ is such a presheaf, the set $X(\Gamma)$ can be understood as the elements of X in the context of variables Γ, and the functorial action is variable substitution. The representable presheaf $\mathbb{S}(-, n)$ corresponds to a metavariable of arity n, for

$$\mathbb{S}(\Gamma, n) \cong \{(t_1, \ldots, t_n) : \Gamma \vdash t_i\}$$

We define the strong monad $T : \widehat{\mathbb{S}} \to \widehat{\mathbb{S}}$ on finite sums of representables as terms-modulo-equations

$$T(\mathbb{S}(-, n_1) + \ldots + \mathbb{S}(-, n_k))(\Gamma) \stackrel{\text{def}}{=} \{\Gamma \mid \varphi_1 : n_1, \ldots, \varphi_k : n_k \vdash u\} / \equiv$$

and extend this to arbitrary presheaves by preservation of sifted colimits as in (16). The unit of the monad is given by the variable rule (MVar) and Kleisli extension is metasubstitution.

4 Traditional Models of Probability

Here, we review the traditional foundations of probability theory. They are all instances of Kolmogorov's measure-theoretic axioms, but we emphasize that there are many different levels of expressivity ranging from finite probability to continuous probability to fully general probability measures. Typically, statistical notions center around the concepts of probability spaces and random variables. We will instead shift towards the more dynamic view of stochastic maps (probability kernels), which are closer to the needs of program semantics. An important tool will be the study of probability monads.

Random variables and probability spaces A *probability space* is a space Ω together with a probability distribution P on it. A random variable X taking values in a space V is a function $X : \Omega \to V$ (satisfying conditions like measurability). The random variable X induces a probability distribution $P_X \stackrel{\text{def}}{=} X_* P$ on V by pushforward, called *the law of* X. Two variables $X, Y : \Omega \to X$ are called *equal in distributions*, written $X \stackrel{\text{d}}{=} Y$ if they have the same law. The elements $\omega \in \Omega$ are thought of as seeds or sources of randomness, and the values $X(\omega), Y(\omega)$ are realizations or *samples* of the random variables under the seed ω. When we write $\Pr(X = Y)$ we actually evaluate the probability of the event $\{\omega : X(\omega) = Y(\omega)\})$ under the distribution P. *Statistical model notation* is a concise way to introduce random variables and their joint distributions, for example

$$\begin{aligned} X &\sim \mathsf{Bernoulli}(1/2) \\ Y &\sim \mathsf{Bernoulli}(1/2) \\ Z &= X \oplus Y \end{aligned}$$

postulates the existence of variables $X, Y, Z : \Omega \to \mathbb{R}$ where X, Y are independent fair coin flips, and Z is their exclusive-or. The role of the probability space Ω is somewhat mysterious. One treats it as an abstract entity and works with the specified joint laws only. However there are subtleties which are often glossed over, for example when allocating new random variables, one must be able to consistently enlarge Ω to supply the extra randomness [Tao, 2010].

We will now switch from the random variable picture to a view of statistics that emphasizes stochastic computation and maintains sources of randomness explicitly via monads. Nonetheless, the idea of random variables and their laws will be fundamental in the development of quasi-Borel spaces (Section 27). We will for now give a high-level introduction of what we expect from a probability monad, and provide concrete definitions in Definitions 4.1 and 4.9.

Probability monads Given a space X, the probability monad $P(X)$ is the space of all probability distributions on X. The unit of the monad assigns to an element $x \in X$ the *Dirac distribution* δ_x which returns x deterministically. A Kleisli map $f : X \to P(Y)$ (sometimes written $X \rightsquigarrow Y$) is called a *stochastic map* or *probability kernel*, because it returns for every input $x \in X$ a distribution over possible outputs in Y. Kleisli composition corresponds to

running stochastic computations in sequence, aggregating their randomness. We note that *samples* are not emphasized in this picture, other than serving as names (types) for the outputs of distributions.

Monads naturally appear in the structure of effectful computation. Even in a programming language that doesn't explicitly track effects in the type system, the thunk type `() -> a` has the structure of a monad. This is used to great effect by the use of samplers in WEBPPL [Goodman et al., 2016]. The fine control over effects makes it particularly convenient to express *hierarchical models* [Goodman et al., 2016, Chapter 12] in probabilistic programming. Distributions like the Beta or Dirichlet distribution are very naturally expressed as *distributions over distributions*. We will study those distributions in detail in Chapter III.

Important structure of probabilistic computation can be expressed using only the monad.

Probability is covariant Given a map $f : X \to Y$ and a distribution $\pi \in P(X)$, we obtain a pushforward $f_*\pi = P(f)(\pi)$ from the functorial action of the probability monad. In particular the identities $\mathsf{id}_*\pi = \pi$ and $f_*(g_*\pi) = (f \circ g)_*\pi$ hold.

Stochastic computation is commutative and discardable The probability monad P is strong, i.e. we have a map

$$\mathsf{st}_{X,Y} : X \times PY \to P(X \times Y)$$

The two ways from Definition 3.3 of using the strength to run stochastic computations in parallel agree, meaning that P is commutative. The resulting map

$$\otimes : P(X) \times P(Y) \to P(X \times Y)$$

takes two distributions and constructs their product distribution. The commutativity claim is nontrivial; for measurable spaces, we will see that it corresponds to Fubini's theorem (21). Furthermore stochastic computation is discardable, which corresponds to the fact that P is affine: The only probability distribution on the one-point space $1 = \{\star\}$ is $\delta_\star$.

Stochastic computation admits copying and discarding Let $\Delta : X \to X \times X$ denote the diagonal map. Composing with δ results in a stochastic map $X \to P(X \times X)$ which copies its input. Pushing a distribution forward along Δ results in perfectly correlated samples. Similarly, the unique map $X \to 1$ results in a stochastic map $X \to P(1) \cong 1$ which discards the input X.

The properties above are instances (and in fact motivation) of our Soft Definition 1.0 of probabilistic system; they will the basis of the generalizations in Chapter II.

We will now introduce the most important flavors of probability through their respective probability monads.

4.1 Finite Probability

A *finite distribution* or *probability mass function* on a set X is a finitely supported function $\pi : X \to [0,1]$ which satisfies

$$\sum_{x \in X} \pi(x) = 1$$

Definition 4.1 The *finite distribution monad* D assigns to a set X the set of probability mass functions on X. The unit of the monad assigns to an element x the *Dirac distribution* δ_x with mass function $\delta_x(y) = [x = y]$.[4]

We recognize D as the convex combination monad of (9). We will not repeat the definitions of the monad structure from Section 3.6.1, but instead offer a different view on these operations in terms of convex combinations. Notice that probability mass functions are closed under taking convex combinations. It will be useful to denote the Dirac distribution δ_x as $[x]$, following the 'return' of the monadic metalanguage. Every element $\pi \in DX$ can then be written as a finite convex combination of Dirac distributions[5]

$$\pi = \sum_{i=1}^{n} p_i[x_i] \text{ with } p_1 + \cdots + p_n = 1$$

The join of the monad acts in the following way on formal convex combinations,

$$\sum_{i \in I} p_i \left[\sum_{j \in J_i} q_{ij}[x_{ij}] \right] \mapsto \sum_{i \in I} \sum_{j \in J_i} p_i q_{ij}[x_{ij}].$$

and pushforward by $f : X \to Y$ takes the form

$$f_* \left(\sum_{i \in I} p_i[x_i] \right) = \sum_{i \in I} p_i[f(x_i)]$$

If X, Y are finite, a probability kernel $p : X \to D(Y)$ can be described as a *column stochastic matrix* $P \in \mathbb{R}^{Y \times X}$ where $P_{yx} = p(x)(y)$. Kleisli composition then coincides with matrix multiplication. Probability kernels are sometimes written in the evocative "conditional probability notation" $p(y|x) \stackrel{\text{def}}{=} p(x)(y)$ where the vertical bar has no other formal meaning than to separate the outputs from the inputs. In this notation, kernel composition takes the form of the *Kolmogorov-Chapman equation*

$$(qp)(z|x) = \sum_y q(z|y)p(y|x) \tag{19}$$

The probability monad D has a strength given by

$$\left(x, \sum_{i \in I} p_i[y_i] \right) \mapsto \sum_{i \in I} p_i[(x, y_i)]$$

[4] Recall Iverson bracket notation, i.e. $[\phi] = 1$ if ϕ is true, and $[\phi] = 0$ otherwise

[5] in a non-unique way

As seen in (14), the monad D is commutative: In terms of convex combinations, this amounts to checking the equation

$$\sum_{i\in I}\sum_{j\in J} p_i q_j[(x_i, y_j)] = \sum_{j\in J}\sum_{i\in I} q_j p_i[(x_i, y_j)]$$

that is, the order of summation can be interchanged.

4.2 Measure-Theoretic Probability

In order to formalize infinite sequences of random variables or distributions with uncountable support like Gaussians, one traditionally employs measure theory. We refer to [Kallenberg, 1997] for an introduction.

A *σ-algebra* on a set X is a collection of subsets of X which contains $\emptyset$ and is closed under complementation and countable union. A *measurable space* is a set X together with a σ-algebra Σ_X on X, sometimes written (X, Σ_X). We call a set $U \subseteq X$ *measurable* if $U \in \Sigma_X$. A function $f : X \to Y$ between measurable spaces is called *measurable* if for all measurable subsets $A \subseteq Y$, the preimage $f^{-1}(A)$ is measurable in X. The measurable spaces and measurable functions form a category Meas.

Given σ-algebras Σ_X, Σ_Y, the *product σ-algebra* is the σ-algebra generated by the rectangles $A \times B$ with $A \in \Sigma_X, B \in \Sigma_Y$. In Meas, $(X \times Y, \Sigma_X \otimes \Sigma_Y)$ is a categorical product. The category Meas has all limits and colimits. A measurable space X is *discrete* if every subset is measurable, i.e. $\Sigma_X = \mathcal{P}(X)$.

Every topology induces a σ-algebra: If X is a topological space, the *Borel σ-algebra* $\mathcal{B}(X)$ is generated by the open sets of X. We will consider topological spaces equipped with their Borel σ-algebra by default. This makes all continuous functions measurable, giving a faithful functor $\mathsf{Top} \hookrightarrow \mathsf{Meas}$. We will mainly consider the particularly nice class of Polish spaces.

Definition 4.2 A *Polish space* is a topological space homeomorphic to a complete metric space with a countable dense subset.

Examples include $\mathbb{R}^n$, the open interval $(0, 1)$, Cantor space 2^ω and countable discrete topological spaces.

Definition 4.3 A *standard Borel space* is a measurable space isomorphic to $(X, \mathcal{B}(X))$ for some Polish space X. We will sometimes refer to measurable functions $X \to Y$ between standard Borel spaces as *Borel*, as well as their measurable subsets as Borel sets.

Standard Borel spaces form a well-behaved subcategory $\mathsf{Sbs} \subseteq \mathsf{Meas}$ which contains most relevant examples like $\mathbb{R}^n$ and is closed under important constructions like countable products and coproducts. Yet, these spaces admit a strong characterization

Theorem 4.4 (e.g. [Kechris, 1987, Ch. 15]) *The following are equivalent for a measurable space X*

(i) *X is a standard Borel space*

(ii) *X is either empty or a retract of $\mathbb{R}$ in Meas*

(iii) *X is measurably isomorphic to $\mathbb{R}$, or countable and discrete*

Furthermore, if X is a standard Borel space and $A \subseteq X$ a Borel subset, then A is itself a standard Borel space when equipped with the subspace-σ-algebra $\Sigma_A = \{A \cap X \mid X \in \Sigma_X\}$.

Definition 4.5 Let X be a measurable space. A *measure* on X is a function $\mu : \Sigma_X \to [0, \infty]$ such that

$$\mu(\emptyset) = 0, \qquad \mu\left(\sum_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mu(A_i) \tag{20}$$

where $\sum_i A_i$ denotes disjoint union.

A measure is called *finite* if $\mu(X) < \infty$, *σ-finite* if X is a countable union of sets of finite measure, *s-finite* if it is a countable sum of finite measures and a *probability measure* if $\mu(X) = 1$. If μ is a measure on X and $f : X \to Y$ is measurable, the *pushforward measure* $f_*\mu$ is defined by $f_*\mu(A) = \mu(f^{-1}(A))$. For $x \in X$, the *Dirac measure* δ_x is the probability measure on X defined by $\delta_x(A) = [x \in A]$. The Borel-Lebesgue measure is the unique measure on $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ assigning every interval its length, that is $\ell([a, b]) = b - a$ for all $a \leq b$.

Note that measures on a space X are closed under pointwise nonnegative linear combinations, while probability measures are closed under convex combinations only.

Integration: For a measurable function $f : X \to [0, \infty)$ and a measure μ, its integral is written

$$\int_X f(x)\mu(\mathrm{d}x) \in [0, \infty]$$

and we may omit the subscript X for convenience. The integral extends to measurable functions $f : X \to \mathbb{R}$ that are integrable, i.e. satisfy $\int |f(x)|\mu(\mathrm{d}x) < \infty$. For a measurable set A, we define

$$\int_A f(x)\mu(\mathrm{d}x) \stackrel{\text{def}}{=} \int_X f(x)[x \in A]\mu(\mathrm{d}x)$$

It holds for every measurable function f that

$$\int f(x)\delta_{x_0}(\mathrm{d}x) = f(x_0)$$

Definition 4.6 If μ, ν are two measures on X, we say μ is *absolutely continuous* with respect to ν, written $\mu \ll \nu$, if for all measurable sets A, $\nu(A) = 0$ implies $\mu(A) = 0$.

Theorem 4.7 (Radon-Nikodým) *Is μ, ν are σ-finite measures on X, then $\mu \ll \nu$ if and only if there is a measurable function $f : X \to [0, \infty)$ such that*

$$\mu(A) = \int_A f(x)\nu(\mathrm{d}x)$$

The function f is called a density function *for μ.*

Density functions are important practical tools for defining measures. For example, the standard normal distribution on $\mathbb{R}$ is defined as having the density function φ with respect to the Lebesgue measure, where

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

If μ, ν are probability measures on X, Y respectively, their *product measure* is the unique probability measure $\mu \otimes \nu$ on $X \times Y$ which satisfies

$$(\mu \otimes \nu)(A \times B) = \mu(A) \cdot \nu(B) \quad \forall A \in \Sigma_X, B \in \Sigma_Y$$

Fubini's theorem says that for all integrable functions $f : X \times Y \to \mathbb{R}$, the partial integration functions

$$\int_X f(x, -)\mu(\mathrm{d}x), \int_Y f(-, y)\nu(\mathrm{d}y)$$

are themselves integrable and the order of integration can be interchanged

$$\int_Y \int_X f(x,y)\mu(\mathrm{d}x)\nu(\mathrm{d}y) = \int_{X\times Y} f(x,y)(\mu \otimes \nu)(\mathrm{d}(x,y)) = \int_X \int_Y f(x,y)\nu(\mathrm{d}y)\mu(\mathrm{d}x) \quad (21)$$

The construction of product distributions and Fubini's theorem extend to s-finite measures. Fubini's theorem need not hold for more general measures without further assumptions.

Definition 4.8 A kernel $X \rightsquigarrow Y$ between measurable spaces is a function $f : X \times \Sigma_Y \to [0, \infty]$ such that $f(-, A)$ is measurable for all $A \in \Sigma_Y$ and $f(x, -)$ is a measure for all $x \in X$.

Kernels $f : X \rightsquigarrow Y$ and $g : Y \rightsquigarrow Z$ compose as

$$(g \bullet f)(x, A) = \int_Y g(y, A) f(x, \mathrm{d}y) \quad (22)$$

and any measurable map $f : X \to Y$ induces a Dirac kernel $\delta f : X \rightsquigarrow Y$ via

$$\delta f(x, A) = [f(x) \in A]$$

This defines a category Ker of kernels, whose identities $X \rightsquigarrow X$ are given by $\delta \mathsf{id}_X$.

A *probability kernel* or *Markov kernel* is a kernel such that $f(x, -)$ is a probability measure for all x. A kernel is *finite* if $f(x, X) < C$ for some constant $C < \infty$ not depending on x, and *s-finite* if it is a countable sum of finite kernels. Both Markov kernels and s-finite kernels form subcategories of Ker, called Stoch and SfKer respectively.

Definition 4.9 (Giry monad) There is a monad $\mathcal{G} : \mathsf{Meas} \to \mathsf{Meas}$ due to Giry that assigns to X the space of probability measures $\mathcal{G}X$, endowed with the least σ-algebra making all evaluations

$$\mathrm{ev}_A : \mathcal{G}X \to [0, 1], \mu \mapsto \mu(A)$$

measurable for $A \in \Sigma_X$. The unit of this monad takes the Dirac measure $x \mapsto \delta_x$. Kleisli composition takes the average measure via integration, that is for $f : X \to \mathcal{G}Y$ and $\mu \in \mathcal{G}X$, we have

$$f^\dagger(\mu)(A) = \int_X f(x)(A)\mu(\mathrm{d}x) \quad (23)$$

For $h : X \to Y$, the functorial action $\mathcal{G}(h)(\mu)$ is precisely the pushforward measure $h_*\mu$.

Like the distribution monad, the Giry monad is strong, affine and commutative, where commutativity follows from Fubini's theorem (21). If X is standard Borel, so is $\mathcal{G}X$, giving a restricted monad $\mathcal{G}^{\mathsf{sbs}} : \mathsf{Sbs} \to \mathsf{Sbs}$. A Kleisli arrow $X \to \mathcal{G}Y$ is the same as a Markov kernel $X \rightsquigarrow Y$, and Kleisli composition (23) agrees with kernel composition (22). s-finite kernels are not known to arise as the Kleisli category of any monad.

Example 4.10 For $p \in [0,1]$, let $\mathsf{flip}(p) \in \mathcal{G}(2)$ denote the coin flip with bias p. Every probability measure on the booleans is of this form, making

$$\mathsf{flip} : [0,1] \to \mathcal{G}(2) \tag{24}$$

an isomorphism in Meas. In fact, if we give $[0,1]$ the structure of a $\mathcal{G}$-algebra by the expectation (barycenter) map $\epsilon : \mathcal{G}([0,1]) \to [0,1]$

$$\epsilon(\mu) = \int x\mu(\mathrm{d}x)$$

then (24) becomes an isomorphism of $\mathcal{G}$-algebras.

4.3 Higher-order Probability

By *higher-order probability*, we understand the study of probability on function spaces Y^X, which are the spaces of all measurable functions $X \to Y$. An important special case is the study of *random Borel subsets* of the reals, which are probability measures on the space $2^{\mathbb{R}}$.

Providing a foundation for random functions is difficult in measure-theoretic probability, because the category of measurable spaces is not cartesian closed: For example, it is unclear which σ-algebra to put on the space $2^{\mathbb{R}}$ of Borel sets. A result of Aumann shows that no answer can be satisfactory

Theorem 4.11 (Aumann [1961]) *There exists no σ-algebra $\Sigma_{2^{\mathbb{R}}}$ on the space $2^{\mathbb{R}}$ of Borel sets such that the evaluation map $(\ni) : 2^{\mathbb{R}} \times \mathbb{R} \to 2$ is measurable.*

Higher-order functions are a common and useful feature of programming languages. This makes developing a theory of random functions an important problem in the semantics of probabilistic languages. *Quasi-Borel spaces* [Heunen et al., 2017] are an example of a category which conservatively extends standard Borel spaces while being cartesian closed. The space $2^{\mathbb{R}}$ of Borel sets is a genuine function space in quasi-Borel spaces, and the evaluation map $2^{\mathbb{R}} \times \mathbb{R} \to 2$ is a morphism. The theory of quasi-Borel spaces also induces a canonical σ-algebra on $2^{\mathbb{R}}$, which consists of what is known in descriptive set theory as *Borel-on-Borel* families (Section 28). We will extensively review quasi-Borel spaces in Chapter V and analyze their theory of random functions in detail.

Other recent models combining probability and higher-order functions are probabilistic coherence spaces [Ehrhard et al., 2014], stable cones model [Ehrhard et al., 2018], a function analytic model [Dahlqvist and Kozen, 2020], game semantics [Paquet and Winskel, 2018], geometry of interaction [Dal Lago and Hoshino, 2019], boolean-valued sets [Bacci et al., 2018] and a boolean topos model [Simpson, 2017]. For an introduction to random Borel sets see [Tsirelson, 2012].

4.4 Continuous Kernels, Duality, GNS Construction

If the measurable space in question arises from a topology, we can impose further continuity restrictions on kernels.

Definition 4.12 (e.g. [Doberkat, 2004]) If X is a Polish space, we topologize the space $\mathcal{G}X$ with the least topology making all integration maps

$$\mu \mapsto \int_X f(x)\mu(\mathrm{d}x)$$

continuous for $f : X \to \mathbb{R}$ continuous and bounded. With this topology $\mathcal{G}X$ is again Polish, and the Giry monad restricts to a monad $\mathcal{G}^{\mathrm{pol}} : \mathsf{Pol} \to \mathsf{Pol}$. Its Borel σ-algebra coincides with Definition 4.9, making the following diagram of faithful functors commute

$$\begin{array}{ccccc} \mathsf{Pol} & \hookrightarrow & \mathsf{Sbs} & \hookrightarrow & \mathsf{Meas} \\ \downarrow{\scriptstyle \mathcal{G}^{\mathrm{pol}}} & & \downarrow{\scriptstyle \mathcal{G}^{\mathrm{sbs}}} & & \downarrow{\scriptstyle \mathcal{G}} \\ \mathsf{Pol} & \hookrightarrow & \mathsf{Sbs} & \hookrightarrow & \mathsf{Meas} \end{array}$$

We will henceforth write $\mathcal{G}$ indiscriminately for all Giry constructions.

Note that a Kleisli map $X \to \mathcal{G}X$ in Pol is required to be continuous. The corresponding notion of convergence is *weak convergence* of measures (e.g [Kallenberg, 1997, Chapter 3]). Other notions of continuous kernel are known for metric spaces, see [Breugel, 2005; Fritz and Perrone, 2017].

A remarkable aspect of continuity is that we can set up measure theory as dual to functional analysis, giving an entirely different perspective on why measures arise. This view also naturally generalizes to the noncommutative world of quantum computation.

If X is a compact Hausdorff space, we call a measure μ on X *Radon* if it is finite and the relations hold for all Borel sets A.

$$\mu(A) = \inf\{\mu(U) : A \subseteq U, U \text{ open}\}$$
$$\mu(A) = \sup\{\mu(K) : K \subseteq A, K \text{ compact}\}$$

Let $\mathcal{C}(X, \mathbb{R})$ denote the Banach space of continuous functions $X \to \mathbb{R}$ with the supremum norm. Then every Radon measure μ induces a linear functional $\phi : \mathcal{C}(X, \mathbb{R}) \to \mathbb{R}$ by integration

$$\phi(f) = \int f(x)\mu(\mathrm{d}x). \tag{25}$$

This functional is *positive* in the sense that if $f(x) \geq 0$ for all x, then $\phi(f) \geq 0$. Positive functionals are automatically continuous; indeed one can easily see that ϕ is bounded with norm $\mu(X)$. Importantly, a converse holds:

Theorem 4.13 (Riesz-Markov-Kakutani) *Let X be a compact Hausdorff and $\phi : \mathcal{C}(X, \mathbb{R}) \to \mathbb{R}$ a positive linear functional. Then there is a unique Radon measure μ on X such that ϕ is of the form (25).*

Radon *probability* measures correspond precisely to those positive linear functionals that are *unital*, meaning $\phi(x \mapsto 1) = 1$. We can give this representation theorem a categorical form, following [Furber and Jacobs, 2015]: A commutative C*-algebra is a (complex, unital) commutative Banach algebra E with an involution $(-)^* : E \to E$ satisfying properties. An

element $x \in E$ is called *positive* if $x = y^* \cdot y$ for some $y \in E$. A linear function $\phi : E \to F$ between commutative C*-algebras is called MIU if it preserves multiplication, involution and the unit. We write MIU for the category of commutative C*-algebras and MIU maps.

A classical example of a commutative C*-algebra is the space of complex-valued continuous functions $\mathcal{C}(X, \mathbb{C})$ on a compact Hausdorff space X; here multiplication is pointwise and the involution is complex conjugation. These are in fact the only examples – every commutative C*-algebras is isomorphic to $\mathcal{C}(X, \mathbb{C})$ for a unique compact Hausdorff space X. The space X can be taken to be the set $X = \mathsf{MIU}(C, \mathbb{C})$ endowed with a suitable topology. In fact, the notion of continuous map is also recovered by MIU homomorphisms:

Theorem 4.14 (Gelfand duality) *We have an equivalence of categories*

$$\mathsf{CH} \underset{\mathsf{MIU}(-,\mathbb{C})}{\overset{\mathcal{C}(-,\mathbb{C})}{\rightleftarrows}} \mathsf{MIU}^{\mathsf{op}}$$

where CH *is the category of compact Hausdorff spaces.*

Probabilistic computation arises by relaxing the preservation of products between C*-algebras. A linear function ϕ is called positive unital (PU) if it preserves positive elements and the unit. Any MIU map is also PU. Denote by PU the category of commutative C*-algebras and positive unital maps, and define the *Radon monad* $\mathcal{R} : \mathsf{CH} \to \mathsf{CH}$ as

$$\mathcal{R}(X) = \{\mu \text{ Radon probability measure on } X\} \tag{26}$$

topologised as in Definition 4.12 (if X is furthermore Polish, every probability measure is Radon, and $\mathcal{R}X$ and $\mathcal{G}X$ coincide).

Theorem 4.15 (Furber-Jacobs) *The Radon monad induces an equivalence of categories*

$$\mathsf{CH}_{\mathcal{R}} \underset{\mathsf{PU}(-,\mathbb{C})}{\overset{\mathcal{C}_{\mathcal{R}}}{\rightleftarrows}} \mathsf{PU}^{\mathsf{op}}$$

where the functor $\mathcal{C}_{\mathcal{R}}$ *sends the object* $X \in \mathsf{CH}$ *to* $\mathcal{C}(X, \mathbb{C})$ *and the Kleisli morphism* $\kappa : X \to \mathcal{R}Y$ *to the integration map*

$$\mathcal{C}(Y, \mathbb{C}) \mapsto \mathcal{C}(X, \mathbb{C}), f \mapsto \lambda x. \int f(y)\kappa(x)(\mathrm{d}y)$$

In particular, to give a positive unital map $\mathcal{C}(X, \mathbb{C}) \to \mathbb{C}$ is to give a Radon probability measure on X, generalizing Theorem 4.13.

Example 4.16 For a finite sets X, Y, to give a probability kernel $X \to DY$ is the same as to give a PU map $\mathbb{C}^Y \to \mathbb{C}^X$. The kernel is described by a column stochastic matrix in $\mathbb{R}^{Y \times X}$ while the PU map is a row stochastic real-valued matrix in $\mathbb{C}^{X \times Y}$. The duality here is matrix transposition. The direction $\mathbb{R}^X \to \mathbb{R}^Y$ is sometimes called the *distribution transformer* picture of probabilistic computation, while $\mathbb{C}^Y \to \mathbb{C}^X$ is the *expectation transformer* picture.

The functional-analytic picture is a useful source of intuition in Section 6.2.3. We will make frequent use of the equivalence Theorem 4.15 throughout Section 12.

As an aside, the setup using C*-algebras allows for a natural generalization from probabilistic to quantum computation: Here we drop the commutativity assumption on the C*-algebras. Let $M_n := \mathsf{End}((\mathbb{C}^2)^{\otimes n}) \cong \mathbb{C}^{2^n \times 2^n}$ denote the noncommutative C*-algebra of $2^n \times 2^n$ matrices under matrix multiplication, then a *quantum channel* from m to n qubits is given by a *completely positive unital map* $M_n \to M_m$. The dual (covariant) notion is that of a *completely positive trace-preserving map* $M_m \to M_n$. See [Staton, 2015] for an introduction from a programming perspective. Because information cannot be copied in quantum computation, we need to weaken our axioms of synthetic probability, which be briefly discuss in Section 6.2.4. We remark that the 'noncommutative' nature of quantum computation is unrelated to the notation of 'commutativity' for monads: The theory of quantum computation *is* commutative in the sense of monads.

Chapter II
Categorical Probability Theory

Arguably, the category language, some call it "abstract", reflects mental undercurrents that surface as our "intuitive reasoning"; a comprehensive mathematical description of this "reasoning", will be, probably, even farther removed from the "real world" than categories and functors.

MIKHAIL GROMOV, In a Search for a Structure, Part 1: On Entropy

Categorical probability theory is the central theme of this thesis. It seeks to abstract the underlying structure of the various models of probability we have encountered, and opens them up to generalization. A selection of recent work that uses synthetic reasoning is [Fritz and Perrone, 2017; Fritz and Rischel, 2020; Fritz et al., 2021; Ścibior et al., 2017; St Clere Smithe, 2020; Jacobs, 2020].

Our first two sections are expository: We will begin by recalling two particular formulations of categorical probability. In Section 5, we review commutative monads due to [Kock, 2011] as generalized monads of measures. We will give probabilistic interpretations of the monads in Section 3.6.1 and introduce further running examples that are of interest in computer science, like nondeterminism (Section 5.5), unification (Section 5.6) and name generation (Section 5.7).

In Section 6, we review a more general formalism of categorical probability, namely CD & Markov categories due to [Cho and Jacobs, 2019; Fritz, 2020], give further examples and discuss their relationship to Freyd categories. As they are monoidal categories with extra structure, we can make convenient use of string diagrams (Section 3.5).

In Section 7, we introduce the CD-calculus, which is the internal language of CD categories. This is a ground computational λ-calculus with a simple theory. The language is novel despite similarities with fine-grained call-by-value. We emphasize the transparent translation between string diagrams and the CD-calculus, giving both formalisms the same expressive power. We argue that this makes it the prototypical backbone of a ground probabilistic programming language, and will later base our Gaussian language (Section 17.2) on it.

In Section 8, we summarize the relevant technical concepts from synthetic probability theory, such as independence (Section 8.1), almost-sure equality (Section 8.2) and conditionals (Section 8.4). Our characterization of supports in Section 8.3 is novel.

In Section 9, we discuss a family of synthetic axioms called *positivity*, *strong affineness* and *causality* due to [Fritz, 2020; Jacobs, 2016; Cho and Jacobs, 2019] under the umbrella of 'dataflow axioms'. Those axioms are interesting in that they can distinguish traditional probability from more exotic theories such as negative probability, and impact the existence

of conditional distributions. We give an equivalent characterization of positivity (Proposition 9.3), prove that strong affineness coincides with it under common circumstances (Proposition 9.7) and show that causality implies positivity (Proposition 9.12). We develop a reading of positivity that concerns information hiding, which will be crucial in recognizing name generation as a prime example of a nonpositive probability theory in Section 26. Section 9 is based on joint work in an upcoming article with Tobias Fritz, Tomáš Gonda, Nicholas Gauguin Houghton-Larsen and Paolo Perrone.

5 Generalized Probability Monads

The oldest line of generalized probability theory goes back to [Kock, 2011] who postulates that any commutative monad may serve as a generalized monad of measures. A related notion is that of a 'measure category' from [Ścibior et al., 2017, 4.3]. We summarize key points of this development.

Definition 5.1 (Kock [2011]) Let $\mathbb{C}$ be a category with finite products. We consider every commutative monad (Definition 3.3) $M : \mathbb{C} \to \mathbb{C}$ a *generalized measure monad*. We consider every commutative and affine monad $T : \mathbb{C} \to \mathbb{C}$ a *generalized probability monad* (Definition 3.4).

This definition alone allows for a surprising amount of structure. The first step is to leverage probabilistic intuitions by introducing synthetic measure-theoretic notation

Notation 5.2 The unit of the monad may be written δ, as it is the synthetic version of the Dirac distribution. If $f : X \to Y$ and $\mu : A \to MX$, we write the functorial action $M(f)\mu : A \to MY$ as a pushforward $f_*\mu$. Lastly, we can use a variant of the monadic metalanguage where the let-binding $\mathsf{let}\, x \leftarrow \mu\, \mathsf{in}\, f$ is written as a formal integral

$$\int f(x)\mu(\mathrm{d}x) \quad \text{or} \quad \int \mu(\mathrm{d}x) f(x)$$

This notation works for both monadic computations and M-algebras like in (2). In the latter case, the integral can also be written as an expectation

$$\mathbb{E}_{X\sim\mu}[f(X)] \overset{\text{def}}{=} \int f(x)\mu(\mathrm{d}x)$$

The monad or algebra laws respectively read as the familiar equations

$$\int f(x)\delta_{x_0}(\mathrm{d}x) = f(x_0)$$

$$\int \delta_x \mu(\mathrm{d}x) = \mu$$

$$\int g(y) \left[\int f(x)\mu(\mathrm{d}x)\right](\mathrm{d}y) = \int\int g(y)f(x)(\mathrm{d}y)\mu(\mathrm{d}x)$$

In this guise, commutativity becomes precisely Fubini's theorem – the order of integration does not matter for generalized integrals.

$$\int f(x,y)\mu(\mathrm{d}x)\nu(\mathrm{d}y) = \int f(x,y)\nu(\mathrm{d}y)\mu(\mathrm{d}x)$$

Affineness corresponds to *normalization*, if α does not depend on x then

$$\int \alpha \mu(\mathrm{d}x) = \alpha.$$

Example 5.3 Kernel composition (22) corresponds to an averaging of measures, which can be concisely written using generalized integrals. If $f : X \to \mathcal{G}Y$ and $g : Y \to \mathcal{G}Z$, their composite is

$$(g \bullet f)(x) = \int g(y) f(x)(\mathrm{d}y) = \mathbb{E}_{Y \sim f(x)}[g(Y)]$$

where we make use of the $\mathcal{G}$-algebra structure $(\mathcal{G}Z, \mathrm{join})$ on $\mathcal{G}Z$.

Many ideas arise from this setup alone. For example, the *object of scalars* $R \stackrel{\mathrm{def}}{=} M(1)$ generalizes the role the real numbers play in ordinary measure theory [Kock, 2011, Section 14,15]. It can be shown to that R carries a commutative monoid structure corresponding to multiplication of reals. If the underlying category is cartesian closed, one can map the monad M to the continuation monad using a monad morphism

$$t_X : MX \to ((X \Rightarrow R) \Rightarrow R)$$

via the integration pairing

$$t_X(\mu) = \lambda f : (X \Rightarrow R) \,.\, \int f(x) \mu(\mathrm{d}x)$$

This is a synthetic analogue of Schwartz distributions and the double-dualization techniques in Section 4.4.

We now give examples of important commutative monads and their probabilistic reading:

5.1 Traditional Probability Monads

The probability monads of Section 4 were, of course, the original inspiration for monadic models. We recall the categories Set, Sbs, Meas and Pol all equipped with their respective affine and commutative probability monads. Generalized integral and expectation here are the usual measure-theoretic integral.

Of these, only Set is cartesian closed. It was a challenge to find a model of both continuous probability and higher-order functions (Section 4.3). One solution is the category of quasi-Borel spaces, which we will treat in detail in Chapter V. The probability monad P on quasi-Borel spaces is commutative and affine.

We will now introduce running examples of other generalized measure and probability monads, meaning the computational effects they describe can be interpreted using the language of synthetic probability:

5.2 Writer

A simple commutative monad is given by the writer monad

$$WX = R \times X$$

where R has the structure of a commutative monoid. This monad can be thought of as a generalized measure monad which supports only scoring

$$\text{score} : R \to W1, r \mapsto (r, ())$$

but no probabilistic choice (this is reflected in the fact that W is cocontinuous, see Section 3.6.4). The writer monad is affine only if $R \cong 1$.

5.3 Multisets (Bags)

A multiset or *bag* is like a set where elements are allowed to appear multiple times, or with multiplicity. The functor $B : \text{Set} \to \text{Set}$ defined by

$$B(X) = \{ \text{ bags on } X \}$$

has the structure of a commutative monad given by singletons and unions of bags. In fact, B is isomorphic to the monad of commutative monoids considered in (7).

The bag monad has been used to model probabilistic databases and stochastic processes [Dash and Staton, 2021; Jacobs and Staton, 2020]. An element of the composite $p \in D(B(X))$ can be seen as a *point process*, and DB can be given the structure of a (noncommutative!) monad on its own right to compose such processes [Dash and Staton, 2020; Jacobs, 2021a]. One can in certain circumstances represent a bag as an $\mathbb{N}$-valued measure, which further underlines the probabilistic reading of this monad and motivates generalizations to measurable probability (e.g. [Dash and Staton, 2020, 6.1]).

5.4 Negative Probabilities

One can formally compute with negative numbers in place of probabilities, as long as these numbers sum to 1. Such negative probabilities have occasionally appeared as useful tools in quantum physics [Feynman, 1987] or queueing theory [Tijms, 2007]. De Finetti's thereom (which is of interest to us in Section 10.3) is known to admit a finitary version featuring negative probabilities [Janson et al., 2016], though we won't use this further.

As an example, consider the hypothetical joint distribution over coins X, Y given by

$$\Pr(X = 0, Y = 0) = -\frac{1}{2} \qquad \Pr(X = 0, Y = 1) = \frac{1}{2}$$
$$\Pr(X = 1, Y = 0) = \frac{1}{2} \qquad \Pr(X = 1, Y = 1) = \frac{1}{2}$$

Negative probabilities produce curious phenomena through "destructive interference": Both variables X, Y almost surely take the value 1, as

$$\Pr(X = 0) = -\frac{1}{2} + \frac{1}{2} = 0 \qquad \Pr(X = 1) = \frac{1}{2} + \frac{1}{2} = 1$$

and similarly $\Pr(Y = 1) = 1$. Yet, X and Y are *perfectly anticorrelated*, as

$$\Pr(X = Y) = \Pr(X = 0, Y = 0) + \Pr(X = 1, Y = 1) = -\frac{1}{2} + \frac{1}{2} = 0$$

This challenges classical notions of determinism and independence. Yet negative probabilities form a valid model of categorical probability in the following sense: A signed distribution over the set X corresponds precisely to a formal *affine* combination (8), and the associated monad monad $D^{\pm}$ is a affine and commutative: Our coin example arises as the formal affine combination

$$\mu = -\frac{1}{2}[(0,0)] + \frac{1}{2}[(0,1)] + \frac{1}{2}[(1,0)] + \frac{1}{2}[(1,0)] \in D^{\pm}(2 \times 2)$$

Negative probability will serve as an important counterexample to the 'dataflow axioms' of Section 9, which aim to axiomatically describe familiar properties of ordinary probability. We will see in Section 26 that name generation and quasi-Borel space probability share formal similarities with $D^{\pm}$ despite all probabilities being nonnegative.

5.5 Nondeterminism

The classic primitive for nondeterministic programming is John McCarthy's `amb` operator [Abelson et al., 1996, 4.3.3], which tries out all of its arguments and backtracks once it triggers `fail()`. Backtracking plus unification (Section 5.6) are the basis for logic programming.

On Set, we model nondeterministic choice by relations in place of functions. The powerset functor $\mathcal{P} : \mathsf{Set} \to \mathsf{Set}$ with monad structure

$$\eta_X(x) = \{x\}, \qquad \mathrm{join}_X(\mathcal{A}) = \bigcup \mathcal{A}$$

is commutative and provides the nondeterministic primitives $\mathsf{fail} : 1 \to \mathcal{P}(1)$ and $\mathsf{amb} : 1 \to \mathcal{P}(2)$ by

$$\mathsf{fail} = \emptyset, \qquad \mathsf{amb} = \{\mathsf{true}, \mathsf{false}\}$$

Note that the generalized integral is precisely existential quantification, e.g. if $\mu \in \mathcal{P}X$ and $f : X \to \mathcal{P}Y$ then

$$y \in \int \mu(\mathrm{d}x) f(x) \Leftrightarrow \exists x \in \mu.\, y \in f(x).$$

The powerset monad is not affine because $\mathcal{P}1 = \{\emptyset, 1\}$, that is powerset nondeterminism allows a computation to return multiple or zero results (`fail`). We can remedy this by taking the nonempty powerset monad $\mathcal{P}^+ : \mathsf{Set} \to \mathsf{Set}$ which forces *one* or more results.

Both monads admit finitary variants $\mathcal{P}_\mathsf{f}$, $\mathcal{P}^+_\mathsf{f}$ of finite and finite nonempty subsets respectively. Those monads correspond to the algebraic theories of $\vee$-semilattices and $\vee$-semilattices with a bottom element. Mathematically, nondeterminism differs from probability in that mere possibility is recorded; for example, nondeterministic choice $(\vee)$ is associative while probabilistic choice $(+)$ is not. We elaborate on the relationship between the two theories in Section 13.3.

Aside on supports: It is always possible to *collapse* probability to possibility. The following construction is folklore, but we include it for lack of reference

Proposition 5.4 (Possibilistic collapse) *Every semilattice can be considered a convex set where* $x +_p y = x \vee y$ *for all* $0 < p < 1$. *The inclusion functor* $\mathsf{SL} \to \mathsf{Cvx}$ *has a left adjoint* $(-)^{\mathsf{sl}} : \mathsf{Cvx} \to \mathsf{SL}$ *which identifies all "intermediate mixtures", e.g.* $a \sim b$ *if there are* c, d *and* $0 < p, q < 1$ *such that* $a = c +_p d, b = c +_q d$.

For example $[0,1]^{\mathsf{sl}}$ has three elements, "surely 0", "surely 1" and "mixture" $(0 \vee 1)$. As a left adjoint, the collapse must send free convex sets to free semilattices, and the unit of the adjunction

$$D(X) \to \mathcal{P}_{\mathsf{f}}^{+}(X), \quad p \mapsto \{x \in X \mid p(x) > 0\}$$

can be interpreted as taking the support of a distribution (we return to supports in Section 8.3).

We briefly note that a topological variant of nondeterminism has been described as Markov categories in [Fritz and Rischel, 2020]. It serves as a counterexample in Section 9.2, and can be related to the support of probability measures as another way of collapsing probability to nondeterminism [Fritz et al., 2019].

5.6 Logic Programming and Unification

A particular form of nondeterminism forms the basis of logic programming languages like PROLOG: We can present logic variables and unification in the framework of synthetic probability, following the analysis of Staton [2013a]. The second-order algebraic theory (see Section 3.7) in that paper serves as an inspiration for our treatment of conditional probability in Section 21.2; in Section 20.2 we draw further analogies between improper priors and Prolog. Outside of Chapter IV, the allocation of fresh logic variables is another example of generativity just like name generation (Section 5.7) or urn creation (Section 10.3).

Unification is a fundamental operation between symbolic expressions. For example, the two types

$$\alpha \to (\beta \to \alpha) \quad \text{and} \quad (\mathsf{int} \to \mathsf{int}) \to \gamma$$

unify by the assignment of type variables

$$\alpha = \mathsf{int} \to \mathsf{int} \quad \gamma = \beta \to (\mathsf{int} \to \mathsf{int}).$$

For a more involved example, we consider the following Prolog implementation of proof search for the implicational fragment of natural deduction.

```
% Does Xs contain X?
mem([X|_],  X).
mem([_|Xs], X) :- mem(Xs,X).

% Can we derive Ps |- P?
ded(Ps, P)      :- mem(Ps, P).                    % axiom rule
```

```
ded(Ps, P -> Q) :- ded([P|Ps], Q).                % (->) introduction
ded(Ps, P)      :- ded(Ps, Q -> P), ded(Ps, Q).   % (->) elimination

% ?- ded([a], (a -> b) -> b).
% true.
```

In order to express the Prolog semantics in a functional language with effects, we make use of an abstract type `term` of *logic variables* (*terms* in Prolog parlance) admitting a constructor `impl : term -> term -> term` forming the implication terms $(a \to b)$. We need to explicitly invoke unification as an effect `(=:=) : term -> term -> unit`. Then the Prolog program can be translated as follows

```
val mem : term list -> term -> unit
let rec mem ps q =
   match ps with
   | []        -> fail()
   | (p :: ps) -> if amb [true;false] then p =:= q else mem ps q

val ded : term list -> term -> unit
let rec ded ps p =
  match amb [1;2;3] with
  | 1 -> mem ps p
  | 2 -> let q = free() in let r = free() in
         p =:= impl(q,r); ded (q :: ps) r
  | 3 -> let q = free() in
         ded ps (impl(q,p)); ded ps q
```

In addition to explicit unification, we make use of nondeterminism (`amb`, `fail`) to model Prolog's disjunctive choice between different rules. The effect `free : unit -> term` allocates a new logic variable. The addition of a unification effect to a functional language leads to the hybrid paradigm of *functional-logic programming* [Antoy and Hanus, 2010] as exemplified by the languages *Curry* [Hanus et al., 2000] and *Mercury* [Somogyi et al., 1996].

Semantically, the new language constructs are modeled by a strong monad T on a category of presheaves (Section 3.7), an object L of terms and effects

$$\exists : 1 \to T(L) \qquad (=:=) : L \times L \to T(1)$$

Staton proved that the monad T is commutative[6]. This not only lets us think of generativity as a form of randomness, we will later even explore fresh logic variables ($\exists$) as a synthetic version of improper priors (Section 20.2).

[6] T cannot be affine as we have nontrivial effects like $(=:=) : L \times L \to T(1)$

5.7 Name Generation

Name generation is one of the core examples of this thesis, which we set out to analyze from a probabilistic viewpoint. A name is an abstract entity which has no properties other than its identity, that is whether or not it is equal to other names. We can generate *fresh names*, that is create a new name which is distinct from all other names. Examples of names are *GUIDs*, *database IDs* or *URLs*, but also *variable names* in metaprogramming (`gensym`), *locations* for memory allocation (**`new`**) and *cluster names* in statistics.

This makes name generation *the* simplest instance of *generativity*, which is an abundant phenomenon in computer science; other examples in this thesis are urn creation, latent random variables, unification variables and memory cells. A classical semantical model is the 'name-generation monad' T on nominal sets[7], which we discuss in detail in Section 25.2.

Much of Chapter V is dedicated to investigating the relationship between name generation and probability. For now, let us state that name generation is a commutative and discardable probabilistic effect, as reflected in Definition 24.7. Name generation combines features of probability and nondeterminism. For example, the equation

$$(\mathsf{let}\, x \;=\; \mathsf{fresh}()\, \mathsf{in}\, \mathsf{let}\, y \;=\; \mathsf{fresh}()\, \mathsf{in}\, (x = y)) \;=\; \mathsf{false} \tag{27}$$

shows that fresh names will be distinct, while nondeterministically chosen names can coincide. As a rule of thumb, a statement about name generation will be true if it holds for some and equivalently for all suitably fresh names. This 'some/any' principle is important [Pitts, 2013b, 3.9] and reminiscent of zero-one laws in probability.

The freshness equation (27) is reminiscent of probabilistic choice from an atomless probability distribution. For example, if X, Y are sampled independently from a Gaussian distribution, then $\Pr(X = Y) = 0$. Indeed (27) holds in Meas when we take $\mathsf{fresh} : 1 \to \mathcal{G}(\mathbb{R})$ to be an atomless probability distribution. Finding probabilistic models of name generation is achieved in Chapter V. The interest then lies in studying the interaction of name generation with higher-order function (Section 4.3). For example, the following name-generating functions are identified

$$(\mathsf{let}\, x \;=\; \mathsf{fresh}()\, \mathsf{in}\, \lambda y.(x = y)) \;=\; \lambda y.\mathsf{false} \tag{28}$$

because the name x remains private in the body of the closure. We show that the same equation holds when interpreted as random higher-order functions (Section 28). This showcases curious abstract properties of names name generation and higher-order probability shared with negative probability (Section 26).

6 CD- and Markov Categories

6.1 Definition

Monadic models of probability correspond to probabilistic languages with explicit effects, thunking and higher-order functions if the underlying category is cartesian closed. A simpler formalism for ground probabilistic programming formalizes categories of stochastic

[7] for the knowledgeable reader, T is the free restriction set monad, see [Pitts, 2013a, Chapter 9.5]

maps directly, in a way reminiscent of Freyd categories. The three probabilistic features copying, discarding and commutativity are modeled transparently by a symmetric monoidal category with extra structure. We recall two kinds of structures

CD categories due to [Cho and Jacobs, 2019] which model unnormalised probabilistic computation and

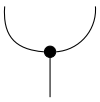
Markov categories due to [Fritz, 2020] which model normalized probabilistic computation

Both theories are phrased in symmetric monoidal categories, and will make use of the graphical language of string diagrams (Section 3.5). We will also define a programmatic internal language which is reminiscent of fine-grained call-by-value (Section 7). Markov categories have been used to formalize various theorems of probability and statistics, such as sufficient statistics (Fisher-Neyman, Basu, Bahadur) [Fritz, 2020], stochastic dominance (Blackwell-Sherman-Stein) [Fritz et al., 2020] and zero-one laws [Fritz and Rischel, 2020]. We'll study them as a foundation for programming language semantics (Section 7), a theory of conditioning (Chapter IV) and for their dataflow properties (Section 9).

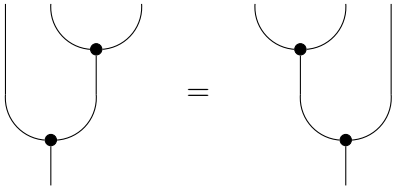
Definition 6.1 (CD category) A *copy-delete category* (CD category) is a symmetric monoidal category $(\mathbb{C}, \otimes, I)$ where every object X is equipped with the structure of a commutative comonoid

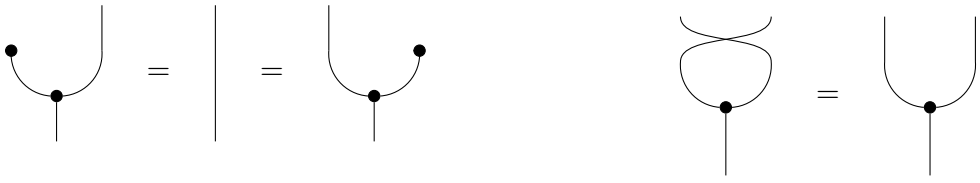
$$\mathrm{copy}_X : X \to X \otimes X \quad \mathrm{del}_X : X \to I$$

graphically depicted as

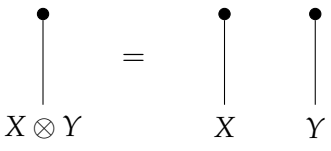
$\mathrm{copy}_X =$  $\mathrm{del}_X =$ 

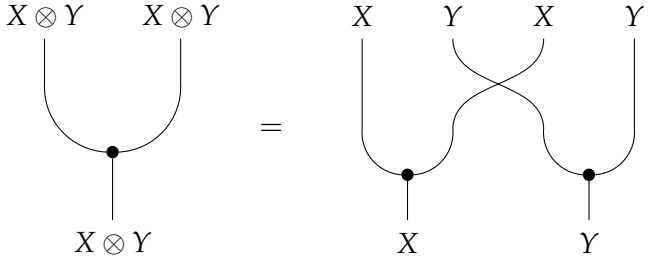
satisfying the axioms



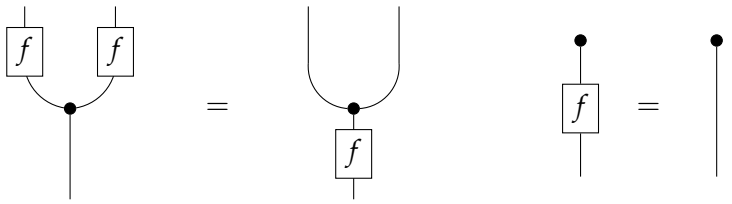


We require that the comonoid structure be compatible with the monoidal structure as follows





It is important that copy and del are *not* assumed to be natural; explicitly the equations



(29)

need *not* hold in general. We'll give special names to situations where they do hold.

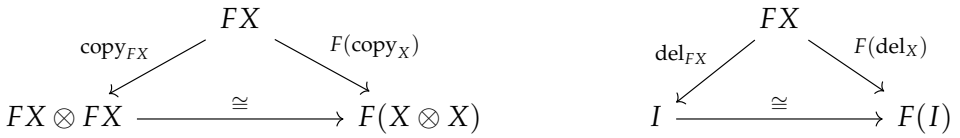
Definition 6.2 (Copyable and discardable morphism) A morphism $f : X \to Y$ is called *copyable* if

$$\mathrm{copy}_Y \circ f = (f \otimes f) \circ \mathrm{copy}_X.$$

A morphism $f : X \to Y$ is called *discardable*[8] if

$$\mathrm{del}_Y \circ f = \mathrm{del}_X.$$

The appropriate notion of functor between CD categories $F : \mathbb{C} \to \mathbb{D}$ is a strong symmetric monoidal functor which preserves the comonoid structure in the sense that



The correct notion of natural transformation between these functors is slightly more subtle and will be discussed in Remark 6.11.

Definition 6.3 (Markov category) A Markov category is a CD category $\mathbb{C}$ in which the following equivalent properties hold

(i) $\mathbb{C}$ is semicartesian, i.e. I is terminal

(ii) every morphism is discardable

(iii) del is natural

Notation: The presence of explicit copying and discarding maps lets us appropriate product-like syntax for CD categories: If $f : A \to X$, $g : A \to Y$, we write a tupling

$$\langle f, g \rangle \stackrel{\mathrm{def}}{=} (f \otimes g) \circ \mathrm{copy}_A$$

and define projection maps

$$\pi_X : X \otimes Y \to X \quad \pi_Y : X \otimes Y \to Y$$

[8] this is sometimes called *causal* in physics, but that word will have a different meaning in this thesis

via discarding. Recall the terminology of states and effects (5) in symmetric monoidal categories: In Markov categories, all effects $X \to I$ are trivial, while they will be relevant in CD categories.

In Markov categories, we'll additionally make use of the following probabilistic terminology: If $f : A \to X \otimes Y$, we define its *marginal* (on X) $f_X : A \to X$ as $\pi_X f$. Of course, we generally have $f \neq \langle f_X, f_Y \rangle$ unless $\mathbb{C}$ is cartesian. We call states $\mu : I \to X$ *distributions*, which lets us think of general morphisms $f : A \to X$ as *parametrized distributions*.

The comonoid structure allows us to copy and discard variables freely; this lets us interpret CD categories in a nonlinear programming language. We formally introduce the internal language in Section 7. We argue this is the prototypical probabilistic programming language. This gives us a total of 3 useful calculi for synthetic probability – ordinary algebraic notation, string diagrams and probabilistic programs.

We begin by establishing a selection of key examples for Markov categories.

6.2 Examples of Markov Categories

6.2.1 Kleisli Categories

CD and Markov categories subsume the generalized probability monads from before by considering their Kleisli categories. This is the same construction as in Proposition 3.2 or [Fritz, 2020, 3.1], but we'll sketch out the constructions explicitly from a programming perspective.

Proposition 6.4 *Let T be a commutative monad on a category $\mathbb{C}$ with products. Then the Kleisli category $\mathbb{C}_T$ becomes a CD category by inheriting the comonoid structure from $(\mathbb{C}, \times, 1)$ as*

$$\mathrm{copy}_X(x) = [(x, x)] \quad \mathrm{del}_X(x) = [()]$$

If T is furthermore affine, $\mathbb{C}_T$ is a Markov category.

PROOF It is known that the Kleisli category is symmetric monoidal if T is commutative. On objects, tensor is cartesian product

$$X \otimes Y \overset{\text{def}}{=} X \times Y$$

while the tensor of Kleisli maps $f : X \to TY$, $g : Z \to TW$ is

$$(f \otimes g)(x, z) \overset{\text{def}}{=} \mathsf{let}\, y \leftarrow f(x)\, \mathsf{in}\, \mathsf{let}\, w \leftarrow g(z)\, \mathsf{in}\, [(z, w)]$$

Commutativity lets us verify the interchange law. The comonoid laws are evident, and discardability follows from the affineness of T (15). ∎

6.2.2 Traditional Models of Probability

All classical models of probability from Section 4 can be presented as Markov categories. Fritz [2020] has established names for some of them

(i) Stoch is the Kleisli category of the Giry monad on Meas

(ii) BorelStoch is the Kleisli category of the Giry monad on Sbs

We could make similar names like for the Kleisli category of the probability monads on Pol, but will often leave this implicit when the obvious probability monad used.

The category FinStoch consists of finite sets and stochastic matrices between them. Note that a stochastic matrix is the same as a Kleisli arrow $X \to D(Y)$ for the distribution monad, but FinStoch is not formally a Kleisli category because $D(Y)$ is not finite. Similarly FinSetMulti (an example of nondeterminism) consists of finite sets and Kleisli arrows $X \to \mathcal{P}^+(Y)$ for the nonempty powerset monad.

The category SfKer of s-finite kernels is a CD category which is not known to be of Kleisli form. All measure-theoretic examples in this thesis are in fact CD subcategories of SfKer. An interesting avenue of research is to identify well-behaved subcategories which may admit a compact description without any measure theory, and enjoy stronger formal properties: Crucial examples are the Beta-Bernoulli process (Chapter III) and Gaussian probability (Definition 18.2). The category of quasi-Borel spaces Section 27 on the other hand conservatively extends standard Borel probability with a function space construction.

6.2.3 Categories of Comonoids

A conceptually interesting class of Markov categories is obtained as follows: Let $\mathbb{D}$ be a symmetric monoidal category and $\mathbb{C} = \mathsf{CComon}(\mathbb{D})$ be the category of commutative comonoids in $\mathbb{D}$, i.e. triples (X, μ_X, e_X) with $\mu : X \to X \otimes X$ and $e_X : X \to I$ satisfying the usual equations. Morphism in $\mathbb{C}$ are all *counit-preserving maps* $f : X \to Y$, i.e. $e_X = e_Y f$. Then $\mathbb{C}$ is symmetric monoidal and each object (X, μ_X, e_X) is equipped with a comonoid structure, namely its very own (μ_X, e_X). The monoidal structure is affine because morphisms are assumed counit-preserving. This makes $\mathbb{C}$ into a Markov category [Fritz, 2020, Section 9].

Dually, if $\mathbb{C}$ is a category of commutative *monoids* and unital maps, its opposite category $\mathbb{C}^{\mathrm{op}}$ will be a Markov category. This fits well with the functional-analytic approach from Section 4.4. C*-algebras are monoids of sorts (equipped with a multiplication) and the opposite categories $\mathsf{MIU}^{\mathrm{op}}$ and $\mathsf{PU}^{\mathrm{op}}$ are Markov categories of (continuous) deterministic and probabilistic computation respectively. Recall that deterministic computation is characterized by preserving the multiplication; this is precisely the (dual) definition of *determinism* which we will introduce Section 6.3.

6.2.4 Aside on Semicartesian Theories

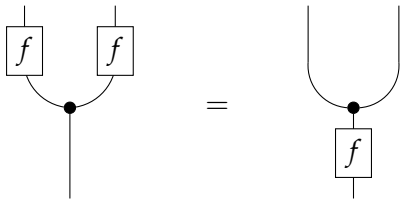
As a small digression, we mention that there exist variations of synthetic probability theory which are even more general than Markov categories. For example, for the purposes of quantum computation, information cannot be copied but it can be discarded in a unique way. A *theory* in the sense of Houghton-Larsen [2021] is a semicartesian monoidal category, without any supply of comonoids. An crucial example is the category QIT (for quantum information theory) of finite-dimensional C*-algebras and completely positive trace-preserving maps. It turns out that several concepts in Markov categories have equivalent formulations which generalize to a purely semicartesian setting. In particular, the concept of *dilation* has similarities to our developments of Leak and Cond in Section 19 and promises to be an interesting avenue of investigation. We refer to [Staton, 2015] for a presentation of quantum computation

using a linear variant of second-order algebra and to [Huot and Staton, 2018; Heunen and Kaarsgaard, 2021] for further categorical background. For another way to combine Markov categories and quantum computation see [Parzygnat, 2020].

6.3 Determinism

The following definition is so crucial to the theory of CD and Markov categories that we will discuss it before defining the internal language. Let $\mathbb{C}$ be a CD category:

Definition 6.5 (Fritz [2020, 10.1]) We call a morphism $f : X \to Y$ *deterministic* if it is copyable and discardable (Definition 6.2), that is the equations (29) hold. If $\mathbb{C}$ is a Markov category, discardability is automatic, so determinism is the equation



A morphism $f : X \to Y$ is deterministic if and only if it is a comonoid homomorphism $(X, \mathrm{copy}_X, \mathrm{del}_X) \to (Y, \mathrm{copy}_Y, \mathrm{del}_Y)$. We recall that respecting comonoid structure is precisely the dual of the condition witnessed in the C*-algebra formalism (Section 4.4): A PU map $f : \mathcal{C}(Y) \to \mathcal{C}(X)$ represents a stochastic map, and that map is deterministic iff it respects the multiplication (monoid structure).

In programming terms, deterministic expressions behave like values: Running an effectful expression twice is the same as running it once and reusing the result; not running the expression (discarding it) has no effect.

It is easy to show that deterministic morphisms are closed under composition and that all structure maps of $\mathbb{C}$ are deterministic. We write $\mathbb{C}_{\mathrm{det}}$ for the wide subcategory of $\mathbb{C}$ consisting only of deterministic morphisms. It is easy to see that a Markov category is cartesian if and only if every morphism is deterministic. It immediately follows that $\mathbb{C}_{\mathrm{det}}$ is itself a cartesian subcategory.

Synthetic probability theory can thus be seen as studying the ability of computation to produce nontrivial correlations: If T is a computational monad that preserves products, i.e. $T(X \times Y) \cong T(X) \times T(Y)$, then its Kleisli category is cartesian, hence making it uninteresting from a *probabilistic viewpoint*, because every morphism is deterministic. We study further this difference between determinism and purity (effect-freeness) in Kleisli categories.

Determinism versus purity: From the programming perspective, deterministic computations behave like values. It is important to note that there are other notions of valuehood which may not agree with determinism. In a Kleisli category, we call a morphism $f : X \to TY$ *pure* if it factors through the unit, i.e. $f = \eta_Y \circ f_0$ for $f_0 : X \to Y$. Every pure

morphism is deterministic, because by a simple calculation in the monadic metalanguage

$$\begin{aligned}\mathsf{let}\, y \leftarrow f(x)\, \mathsf{in}\, [(y,y)] &= \mathsf{let}\, y \leftarrow [f_0(x)]\, \mathsf{in}\, [(y,y)] \\ &= [(f_0(x), f_0(x)] \\ &= \mathsf{let}\, y_1 \leftarrow [f_0(x)]\, \mathsf{in}\, \mathsf{let}\, y_2 \leftarrow [f_0(x)]\, \mathsf{in}\, [(y_1,y_2)] \\ &= \mathsf{let}\, y_1 \leftarrow f(x)\, \mathsf{in}\, \mathsf{let}\, y_2 \leftarrow f(x)\, \mathsf{in}\, [(y_1,y_2)]\end{aligned}$$

The converse to this need not hold, as the following example in measurable spaces shows:

Proposition 6.6 (Fritz [2020, 10.4]) *In* Stoch, *a probability measure* $\mu \in \mathcal{G}(X)$ *(seen as a morphism* $1 \to X$*) is deterministic iff it is zero-one valued, i.e.* $\mu(A) \in \{0,1\}$ *for all* $A \in \Sigma_X$.

On a standard Borel space, every $\{0,1\}$-valued measure must be a Dirac measure. This need not be the case for more exotic σ-algebras – a canonical example is given as follows

Example 6.7 Let X be an uncountable set and Σ_X consist of those subsets of X that are countable or co-countable. Then Σ_X is a σ-algebra and we have a probability measure $\nu : \Sigma_X \to [0,1]$ given by

$$\nu(A) = [A \text{ is cocountable}].$$

Because $\nu(A) \in \{0,1\}$, this measure is deterministic as state $1 \to X$ in Stoch by Proposition 6.6, but it is not pure.

The measure ν has aspects reminiscent of fresh name generation, so we'll revisit it in Section 26.3. However X is not a suitable domain for name generation, because the equality test $(=) : X \times X \to 2$ is not measurable: if it was, we would obtain an easy contradiction, because by determinism of ν we compute in the monadic metalanguage

$$(\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, y \leftarrow \nu\, \mathsf{in}\, [x = y]) = (\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [x = x]) = [\mathrm{true}]$$

while on the other hand $\nu(\{x\}) = 0$ implies that

$$(\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, y \leftarrow \nu\, \mathsf{in}\, [x = y]) = (\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [\mathrm{false}]) = [\mathrm{false}]$$

The non-pathological case in which determinism coincides with purity has been called *representability* [Fritz et al., 2020]. The condition that the only deterministic distributions are Dirac is easily translated into a pullback condition:

Definition 6.8 (Fritz et al. [2020, 3.4]) A commutative and affine monad T satisfies the *representability condition* if the following square is a pullback for every X.

$$\begin{array}{ccc} X & \xrightarrow{\delta_X} & TX \\ {\scriptstyle \langle \delta_X, \delta_X \rangle}\downarrow & \lrcorner & \downarrow{\scriptstyle T(\langle \mathsf{id}_X, \mathsf{id}_X \rangle)} \\ TX \times TX & \xrightarrow[\otimes]{} & T(X \times X) \end{array} \tag{30}$$

Proposition 6.9 (Fritz et al. [2020, 3.4]) *Let T satisfy the representability condition (30), then a Kleisli morphism $f : X \to TY$ is deterministic if and only if it is pure.*

Proposition 6.10 *The following monads satisfy the representability condition*

(i) the distribution monad D

(ii) the nonempty finite powerset monad $\mathcal{P}_{\mathsf{f}}^{+}$

(iii) the Giry monad on standard Borel spaces

The Giry monad on Meas *does not satisfy the representability condition.*

PROOF The first two results follow from Fritz et al. [2020, 3.6], applied to the 'entire semirings' $[0, \infty)$ and $\{0, 1\}$. The result for the Giry monad on Sbs is [Fritz, 2020, 10.5]. The Giry monad on Meas fails the representability condition because of Example 6.7. ∎

CD versus Freyd categories: We finish with a remark on the relation between CD categories and Freyd catgories (Section 3.4). Following [Fritz, 2020, 10.19], we see that for every CD category $\mathbb{C}$, the inclusion $\mathbb{C}_{\text{det}} \to \mathbb{C}$ is a commutative Freyd category. Conversely, every object in a commutative Freyd category $J : \mathbb{V} \to \mathbb{C}$ has a canonical comonoid structure induced by the cartesian structure on $\mathbb{V}$. These constructions are in general *not* inverses of each other, as seen when $\mathbb{V} \to \mathbb{V}_T$ is the Freyd category for a monad violating the representability condition. In that case, $(\mathbb{V}_T)_{\text{det}}$ is strictly larger than $\mathbb{V}$.

To summarize, a Freyd category $\mathbb{V} \to \mathbb{C}$ comes equipped with a notion of effect-free morphism, while in a CD category, determinism is a derived notion. We consider the differences between these approaches minor. Freyd categories might be conceptually cleaner, but the definition of a CD category is simpler because there is only one sort of morphism. This makes the internal language of CD categories (Section 7) more naturally expressible in direct style than fine-grained call-by-value. Furthermore, because determinism (unlike valueness) is determined equationally, one can in fact prove nontrival theorems to the effect that certain morphisms are deterministic, e.g. in the formulation of categorical zero-one laws [Fritz and Rischel, 2020].

Remark 6.11 The notion of determinism is important to define equivalence of CD categories: We must ask that the components of a natural transformations between CD functors are deterministic [Fritz, 2020, 10.14]. This is because isomorphisms in a CD categories need not be deterministic (Example 9.8). This is however regarded a pathological situation which will not occur in common circumstances (Proposition 9.9).

7 Internal Language of CD Categories

In this section, we introduce the *CD calculus*, which is the internal language of CD and Markov categories. We'll argue that this is the prototypical backbone of a ground probabilistic programming language. This connects the developments on Markov categories, primarily found in mathematics literature, directly to computer science. We obtain three formalisms for synthetic probability which are freely convertible into each other, having the same expressive power but different strengths and weaknesses:

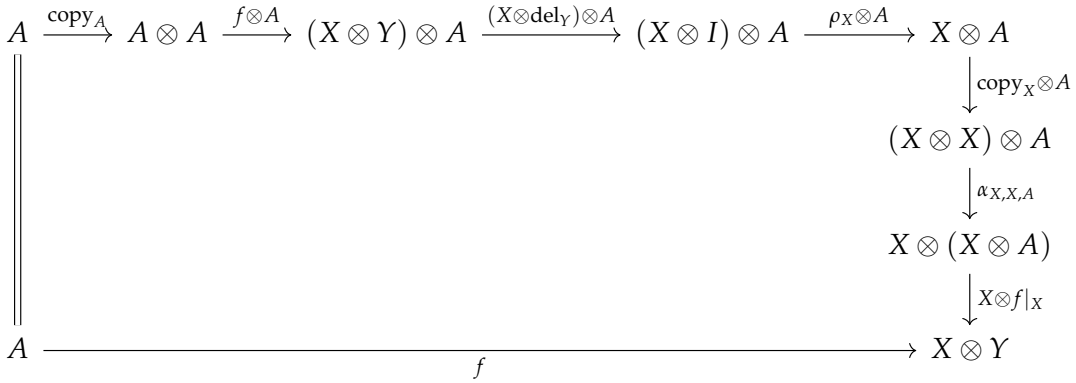
(i) plain categorical notation

(ii) string diagrams

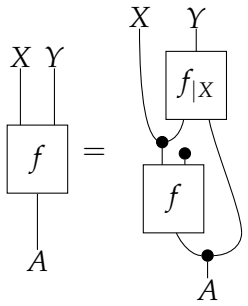
(iii) CD calculus

Categorical notation is the most explicit but also most verbose of the formalisms, explicitly featuring all coherence, copy and delete operations. String diagrams hide the coherence and allow intuitive global geometric reasoning, while still being explicit about copying and deleting information. In a programming language, even copying and deleting is implicit in the nonlinear use of variables in context. Nonetheless, explicit control over variable scope can be achieved through nested lets.

As an example, we compare the definition of parameterized conditionals (Definition 8.13) as a commutative diagram



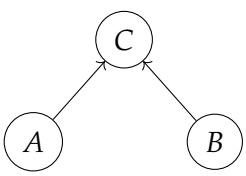
string diagram



and program equation

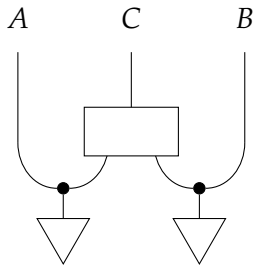
$$a : A \vdash \mathsf{let}\,(x,y) \;=\; f(a)\,\mathsf{in}\,(x, f|_X(x,a)) \equiv f(a) : X * Y$$

There are several further formalisms which translate into the above, like statistical notation (Section 4) and directed graphical models. For example, the independence structure[9] conveyed by the graphical model



[9] as will be clarified in Section 8.1

translates into the following string diagram



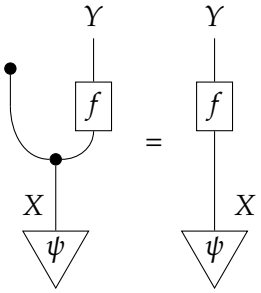
We refer to [Fong, 2012] for more information. Not every string diagram comes from a directed model. An important obstruction is that all variables are public, for example in the model

$$\begin{aligned} X &\sim \psi \\ Y &\sim f(X) \end{aligned}$$

it is impossible to consider the variable Y in isolation of X. In the other hand, nested lets

$$\mathsf{let}\, y \;=\; (\mathsf{let}\, x \;=\; \psi\, \mathsf{in}\, f(x))\, \mathsf{in}\, \cdots$$

or string diagrams



give us precise control over the scope of random variables. This is important in the analysis of subtle phenomena like nonpositivity and leaking (Sections 9 and 30.5).

Lastly, we notice that in [Fritz, 2020, 2.8], the author formally extends the kernel notation $p(y|x)$ from (19) to arbitrary morphisms in Markov categories. A morphism $f : A \otimes B \to X \otimes Y$ would be written $f(x, y|a, b)$ where the vertical bar serves no further formal function than separating inputs and outputs. Composition is denoted using the synthetic analogue of the Kolmogorov-Chapman equation

$$(gf)(z|x) = \sum_y g(z|y) f(y|x) \tag{31}$$

One might even omit the summation sign following Einstein summation convention. This notation has not been formalized and we won't use it in what follows. However, we point out some interesting features: Variables may be used nonlinearly in the *inputs* of morphisms; for example $p(z|x, x)$ translates to $p \circ \mathrm{copy}_X$. However, variables must be used linearly in the outputs, for example the expression $q(y, y, |x)$ is meaningless. We'll briefly revisit this in Section 20.2 in relation to conditioning and unification in Prolog.

Technically an internal language for CD categories is the ground fragment of the Moggi's λ_C-calculus (Section 3.3) extended with a commutativity equation (Figure 1). It can equivalently be described by the ground fragment of fine-grained call-by-value plus commutativity, which is unsurprising given the relationship between CD categories and Freyd categories. We choose to give an independent presentation of the language for the following reasons

(i) the CD calculus has a single judgement unlike the separate computation and value judgement of fine-grained call by value. Accordingly, unlike Freyd categories, a CD category does not come equipped with a notion of value subcategory, but instead, value-like terms are *recovered* as deterministic ones. The relation between determinism and effect-freeness has been addressed in Section 6.3

(ii) fine-grained call-by-value makes sequencing information completely explicit in an almost CPS-like manner. In a commutative calculus, this information is redundant, making direct style more a natural fit. In fact, picking an evaluation order introduces arbitrary choices.

(iii) unlike premonoidal categories, commutativity unlocks string diagrams, allowing a definition of our semantics in string-diagrammatic terms

(iv) the restriction to the ground fragment of λ_C plus commutativity allows us to subsume much of the theory of the λ_C calculus under two general axiom schemes for substitution. The theory of the CD calculus is thus particularly simple and interesting in its own right.

Definition A CD signature $\mathfrak{S} = (\tau, \omega)$ consists of sets τ of base types and function symbols ω. A *type* is recursively defined by closing the base types under tuple formation

$$A ::= \tau \mid \mathsf{unit} \mid A * A$$

Each function symbol $f \in \omega$ is equipped with a unary *arity* of types, written $f : A \to B$. The terms of the CD-calculus are given by

$$t ::= x \mid () \mid (t, t) \mid \pi_i\, t \mid f\, t \mid \mathsf{let}\, x \;=\; t\, \mathsf{in}\, t \qquad (i = 1, 2)$$

subject to the typing rules $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ given in Figure 1.

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A} \qquad \frac{}{\Gamma \vdash () : \mathsf{unit}}$$

$$\frac{\Gamma \vdash s : A \quad \Gamma \vdash t : B}{\Gamma \vdash (s, t) : A * B} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash f\, t : B}\ (f : A \to B)$$

$$\frac{\Gamma \vdash t : A_1 * A_2}{\Gamma \vdash \pi_i\, t : A_i} \qquad \frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash e : A}{\Gamma \vdash \mathsf{let}\, x \;=\; e\, \mathsf{in}\, t : B}$$

Figure 1: Typing rules for the CD calculus

Syntactic sugar We employ standard syntactic sugar, for example sequencing

$$s; t \stackrel{\text{def}}{=} \mathsf{let}\, x \;=\; s \,\mathsf{in}\, t \qquad (x \notin \mathsf{fv}(t))$$

We also define a pattern-matching let as syntactic sugar

$$(\mathsf{let}\,(x, y) \;=\; s \,\mathsf{in}\, t) \stackrel{\text{def}}{=} (\mathsf{let}\, p \;=\; s \,\mathsf{in}\, \mathsf{let}\, x \;=\; \pi_1\, p \,\mathsf{in}\, \mathsf{let}\, y \;=\; \pi_2\, p \,\mathsf{in}\, t)$$

from which we can provably recover the projection constructs once we discussed the theory Section 7.1.

$$(\pi_1\, s) = (\mathsf{let}\,(x, y) \;=\; s \,\mathsf{in}\, x)$$
$$(\pi_2\, s) = (\mathsf{let}\,(x, y) \;=\; s \,\mathsf{in}\, y)$$

We prefer the projections over pattern-matching when presenting the equational theory, because this means one less binding construct.

7.1 Equational Theory

In call-by-value languages, the substitution

$$(\mathsf{let}\, x \;=\; e \,\mathsf{in}\, u) \equiv u[e/x]$$

is generally only admissible if e is a *value*. In the CD calculus, another powerful substitution scheme is valid: We can replace $(\mathsf{let}\, x \;=\; e \,\mathsf{in}\, u) \equiv u[e/x]$ whenever u uses x *linearly*, i.e. exactly once, *even if* e is an effectful computation. Whenever we say "use" or "occurrence", we mean free use and occurrence. Substitution is always capture-avoiding.

The linear substitution scheme becomes invalid in the presence of non-commutative effects like printing, say $e \stackrel{\text{def}}{=} \mathrm{print}(1)$ and $u \stackrel{\text{def}}{=} (\mathsf{let}\, y \;=\; \mathrm{print}(2) \,\mathsf{in}\, x)$ where

$$\mathsf{let}\, x \;=\; \mathrm{print}(1) \,\mathsf{in}\, \mathsf{let}\, y \;=\; \mathrm{print}(2) \,\mathsf{in}\, x \not\equiv \mathsf{let}\, y \;=\; \mathrm{print}(2) \,\mathsf{in}\, \mathrm{print}(1)$$

Similarly the scheme is invalidated in the presence of higher-order constructs like λ-abstraction or thunks, e.g $e \stackrel{\text{def}}{=} \mathrm{rnd}()$, $u \stackrel{\text{def}}{=} \lambda y.x$ as

$$\mathsf{let}\, x \;=\; \mathrm{rnd}() \,\mathsf{in}\, \lambda y.x \not\equiv \lambda y.\mathrm{rnd}()$$

Using the linear- and value substitution schemes, the theory of the CD calculus can be presented concisely as in Figure 2. Note that we omit the context of equations when unambiguous and identify bound variables up to α-equivalence. The theory is given

Proposition 7.1 *All axioms of the ground λ_c-calculus and commutativity are derivable.*

$$(\mathsf{let}\, x_2 \;=\; (\mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, e_2) \,\mathsf{in}\, e) \equiv (\mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, \mathsf{let}\, x_2 \;=\; e_2 \,\mathsf{in}\, e) \quad x_1 \notin \mathsf{fv}(e) \qquad \text{(assoc)}$$
$$(\mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, \mathsf{let}\, x_2 \;=\; e_2 \,\mathsf{in}\, e) \equiv (\mathsf{let}\, x_2 \;=\; e_2 \,\mathsf{in}\, \mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, e) \quad x_1 \notin \mathsf{fv}(e_2), x_2 \notin \mathsf{fv}(e_1) \qquad \text{(comm)}$$
$$(\mathsf{let}\, x \;=\; e \,\mathsf{in}\, x) \equiv e \qquad \text{(id)}$$
$$(\mathsf{let}\, x_1 \;=\; x_2 \,\mathsf{in}\, e) \equiv e[x_2/x_1] \qquad \text{(let.}\beta\text{)}$$
$$f\, e \equiv (\mathsf{let}\, x \;=\; e \,\mathsf{in}\, f\, x) \qquad \text{(let.f)}$$
$$(s, t) \equiv (\mathsf{let}\, x \;=\; s \,\mathsf{in}\, \mathsf{let}\, y \;=\; t \,\mathsf{in}\, (x, y)) \qquad \text{(let.}*\text{)}$$

$$\equiv \text{ is reflexive, symmetric and transitive} \tag{equiv}$$

$$\frac{e_1 \equiv e_1' \quad e_2 \equiv e_2'}{(\mathsf{let}\, x \;=\; e_1 \,\mathsf{in}\, e_2) \equiv (\mathsf{let}\, x \;=\; e_1' \,\mathsf{in}\, e_2')} \tag{let.ξ}$$

A *value* is a term of the form

$$V ::= x \mid () \mid (V, V) \mid \pi_i V \mid \mathsf{let}\, x \;=\; V \,\mathsf{in}\, V$$

The axioms of the CD calculus are

$$(\mathsf{let}\, x \;=\; e \,\mathsf{in}\, t) \equiv t[e!x] \tag{let.lin}$$
$$(\mathsf{let}\, x \;=\; V \,\mathsf{in}\, t) \equiv t[V/x] \tag{let.val}$$
$$\pi_i\,(x_1, x_2) \equiv x_i \tag{$*.\beta$}$$
$$(\pi_1\, x, \pi_2\, x) \equiv x \tag{$*.\eta$}$$
$$x \equiv () \tag{unit.η}$$

where we write $t[x!e]$ for substituting a unique free occurrence of x. For the internal language of Markov categories, extend (let.lin) to all substitutions targeting *at most one* free occurrence of x.

Figure 2: Axioms of the CD-calculus

Note that by commutativity (comm)*, the order of evaluation in* (let.$*$) *does not matter.*

PROOF We will invoke (let.ξ) implicitly throughout. (let.β) follows immediately from (let.val) because x_2 is a value. (id), (let.f), (let.$*$) follow by applying (let.lin) one or two times.

For (comm), we notice that because $x_2 \notin \mathsf{fv}(e_1)$, the expression $\mathsf{let}\, x_1 \;=\; e_2 \,\mathsf{in}\, \mathsf{let}\, x_2 \;=\; x_2 \,\mathsf{in}\, e$ has a unique free occurrence of x_2, hence by linear substitution

$$\begin{aligned}
&\mathsf{let}\, x_2 \;=\; e_2 \,\mathsf{in}\, \mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, e \\
\overset{(\mathrm{let}.\beta)}{\equiv}\ &\mathsf{let}\, x_2 \;=\; e_2 \,\mathsf{in}\, \mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, \mathsf{let}\, x_2 \;=\; x_2 \,\mathsf{in}\, e \\
\overset{(\mathrm{let.lin})}{\equiv}\ &\mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, \mathsf{let}\, x_2 \;=\; e_2 \,\mathsf{in}\, e
\end{aligned}$$

For (assoc), if $x_1 \notin \mathsf{fv}(e)$ then $\mathsf{let}\, x_2 \;=\; (\mathsf{let}\, x_1 \;=\; x_1 \,\mathsf{in}\, e_2) \,\mathsf{in}\, e$ has a unique free occurrence of x_1, hence

$$\begin{aligned}
&\mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, \mathsf{let}\, x_2 \;=\; e_2 \,\mathsf{in}\, e \\
\overset{(\mathrm{let}.\beta)}{\equiv}\ &\mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, \mathsf{let}\, x_2 \;=\; (\mathsf{let}\, x_1 \;=\; x_1 \,\mathsf{in}\, e_2) \,\mathsf{in}\, e \\
\overset{(\mathrm{let.lin})}{\equiv}\ &\mathsf{let}\, x_2 \;=\; (\mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, e_2) \,\mathsf{in}\, e
\end{aligned}$$

■

Linear substitution lets us control general nonlinear substitution in the following way: If t has n free occurrences of the variable x, let $\hat{t}$ denote the term t with those occurrences replaced with distinct fresh variables $x_1, \ldots, x_n$ (their order does not matter). By repeated application of (let.val), we have

$$t \equiv \mathsf{let}\, x_1 \;=\; x \,\mathsf{in}\, \cdots \mathsf{let}\, x_n \;=\; x \,\mathsf{in}\, \hat{t} \tag{32}$$

We can now substitute some or all occurrences of x using (let.lin) as follows

$$t[e/x] \equiv \hat{t}[e!x_1] \cdots [e!x_n] \equiv \mathsf{let}\, x_1 \;=\; e\, \mathsf{in} \cdots \mathsf{let}\, x_n \;=\; e\, \mathsf{in}\, \hat{t} \tag{33}$$

This means we can reduce questions about substitution to the copying behavior of the term e. We adapt the definitions from [Führmann, 2002; Kammar and Plotkin, 2012].

Definition 7.2 A term e is called *copyable* if

$$(\mathsf{let}\, x \;=\; e\, \mathsf{in}\, (x, x)) \equiv (e, e) \tag{34}$$

is derivable. A term e is called *discardable* if

$$(\mathsf{let}\, x \;=\; e\, \mathsf{in}\, ()) \equiv () \tag{35}$$

is derivable. We call e *deterministic* if it is both copyable and discardable.

Proposition 7.3 *The substitution equation*

$$(\mathsf{let}\, x \;=\; e\, \mathsf{in}\, t) \equiv t[e/x]$$

is derivable in any of the following circumstances:

(i) t *uses* x *exactly once*

(ii) t *uses* x *at least once, and* e *is copyable*

(iii) t *uses* x *at most once, and* e *is discardable*

(iv) e *is deterministic*

PROOF The first case is (let.lin) and the last case follows from the combination of the previous ones.

Copyable We begin with the special case that t has precisely two occurrences of x. Then

$$\begin{aligned}
& t[e/x] \\
\overset{(33)}{\equiv}\ & \mathsf{let}\, x_1 \;=\; e\, \mathsf{in}\, \mathsf{let}\, x_2 \;=\; e\, \mathsf{in}\, \hat{t} \\
\overset{(\mathrm{let.val}),(*.\beta)}{\equiv}\ & \mathsf{let}\, p \;=\; (e, e)\, \mathsf{in}\, \mathsf{let}\, x_1 \;=\; \pi_1\, p\, \mathsf{in}\, \mathsf{let}\, x_2 \;=\; \pi_2 p\, \mathsf{in}\, \hat{t} \\
\overset{(34)}{\equiv}\ & \mathsf{let}\, p \;=\; (\mathsf{let}\, x \;=\; e\, \mathsf{in}\, (x, x))\, \mathsf{in}\, \mathsf{let}\, x_1 \;=\; \pi_1\, p\, \mathsf{in}\, \mathsf{let}\, x_2 \;=\; \pi_2 p\, \mathsf{in}\, \hat{t} \\
\overset{(\mathrm{assoc})}{\equiv}\ & \mathsf{let}\, x \;=\; e\, \mathsf{in}\, \mathsf{let}\, p \;=\; (x, x)\, \mathsf{in}\, \mathsf{let}\, x_1 \;=\; \pi_1\, p\, \mathsf{in}\, \mathsf{let}\, x_2 \;=\; \pi_2 p\, \mathsf{in}\, \hat{t} \\
\overset{(\mathrm{let.val}),(*.\beta)}{\equiv}\ & \mathsf{let}\, x \;=\; e\, \mathsf{in}\, \mathsf{let}\, x_1 \;=\; x\, \mathsf{in}\, \mathsf{let}\, x_2 \;=\; x\, \mathsf{in}\, \hat{t} \\
\overset{(33)}{\equiv}\ & \mathsf{let}\, x \;=\; e\, \mathsf{in}\, t
\end{aligned}$$

Repeating this process, any chain of repeated let bindings of a copyable term e

$$\mathsf{let}\, x_1 \;=\; e\, \mathsf{in} \cdots \mathsf{let}\, x_n \;=\; e\, \mathsf{in}\, \ldots$$

can be replaced by

$$\mathsf{let}\, x \;=\; e\, \mathsf{in}\, \mathsf{let}\, x_1 \;=\; x\, \mathsf{in} \cdots \mathsf{let}\, x_n \;=\; x\, \mathsf{in}\, \ldots$$

Discardable Let t have no free occurrence of x, and e be discardable. Then

$$\begin{aligned}
& \mathsf{let}\, x \;=\; e \,\mathsf{in}\, t \\
\overset{\text{(let.val)}}{\equiv}\; & \mathsf{let}\, x \;=\; e \,\mathsf{in}\, \mathsf{let}\, y \;=\; () \,\mathsf{in}\, t \\
\overset{\text{(assoc)}}{\equiv}\; & \mathsf{let}\, y \;=\; (\mathsf{let}\, x \;=\; e \,\mathsf{in}\, ()) \,\mathsf{in}\, t \\
\overset{(35)}{\equiv}\; & \mathsf{let}\, y \;=\; () \,\mathsf{in}\, t \\
\overset{\text{(let.val)}}{\equiv}\; & t
\end{aligned}$$

■

7.2 Semantics

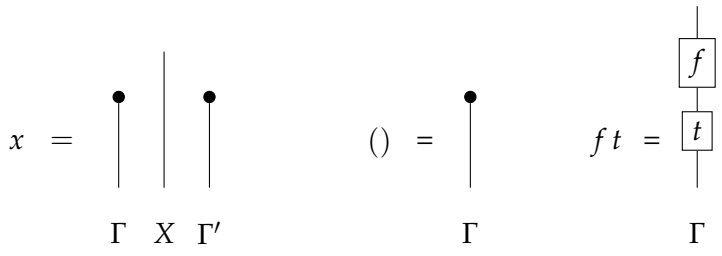
Every CD category models the CD calculus in a way that validates the axioms of the theory. Formally, a *model* of signature (τ, ω) is a CD category $\mathbb{C}$ together with an assignment of objects $[\![A]\!] \in \mathbb{C}$ for each basic type and morphisms $[\![f]\!] : [\![A]\!] \to [\![B]\!]$ for each function symbol $f : A \to B$. Here we extend $[\![-]\!]$ to arbitrary types and contexts by

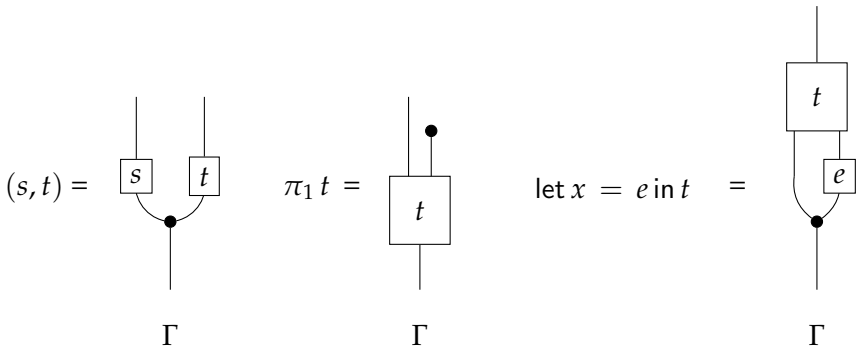
$$[\![\mathsf{unit}]\!] = I \quad [\![A_1 * A_2]\!] = [\![A_1]\!] \otimes [\![A_2]\!] \quad [\![A_1, \ldots, A_n]\!] = [\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!]$$

For any model, the interpretation of a term $\Gamma \vdash t : A$ is defined recursively as

- $[\![x]\!]$ is the discarding map $[\![\Gamma, A, \Gamma']\!] \cong [\![\Gamma]\!] \otimes [\![A]\!] \otimes [\![\Gamma']\!] \to I \otimes [\![A]\!] \otimes I \cong [\![A]\!]$
- $[\![()]\!]$ is the discarding map $\mathrm{del}_{[\![\Gamma]\!]} : [\![\Gamma]\!] \to I$
- $[\![(s,t)]\!]$ is the map $[\![\Gamma]\!] \xrightarrow{\mathrm{copy}_{[\![\Gamma]\!]}} [\![\Gamma]\!] \otimes [\![\Gamma]\!] \xrightarrow{[\![s]\!] \otimes [\![t]\!]} [\![A]\!] \otimes [\![B]\!] = [\![A * B]\!]$
- $[\![\pi_i t]\!]$ is marginalization $[\![\Gamma]\!] \xrightarrow{[\![t]\!]} [\![A_1]\!] \otimes [\![A_2]\!] \to [\![A_i]\!]$
- $[\![f\, t]\!]$ is the composite $[\![\Gamma]\!] \xrightarrow{[\![t]\!]} [\![A]\!] \xrightarrow{[\![f]\!]} [\![B]\!]$
- $[\![\mathsf{let}\, x \;=\; e \,\mathsf{in}\, t]\!]$ is given by $[\![\Gamma]\!] \xrightarrow{\mathrm{copy}_{[\![\Gamma]\!]}} [\![\Gamma]\!] \otimes [\![\Gamma]\!] \xrightarrow{\mathrm{id}_{[\![\Gamma]\!]} \otimes [\![e]\!]} [\![\Gamma]\!] \otimes [\![A]\!] \xrightarrow{[\![t]\!]} [\![B]\!]$

Semantics can be seen as a procedure for translating every term into a string diagram as follows, where we omit brackets for readability.





Proposition 7.4 (Structural rules) *Weakening and exchange are implemented through discarding of variables, and swap isomorphisms respectively.*

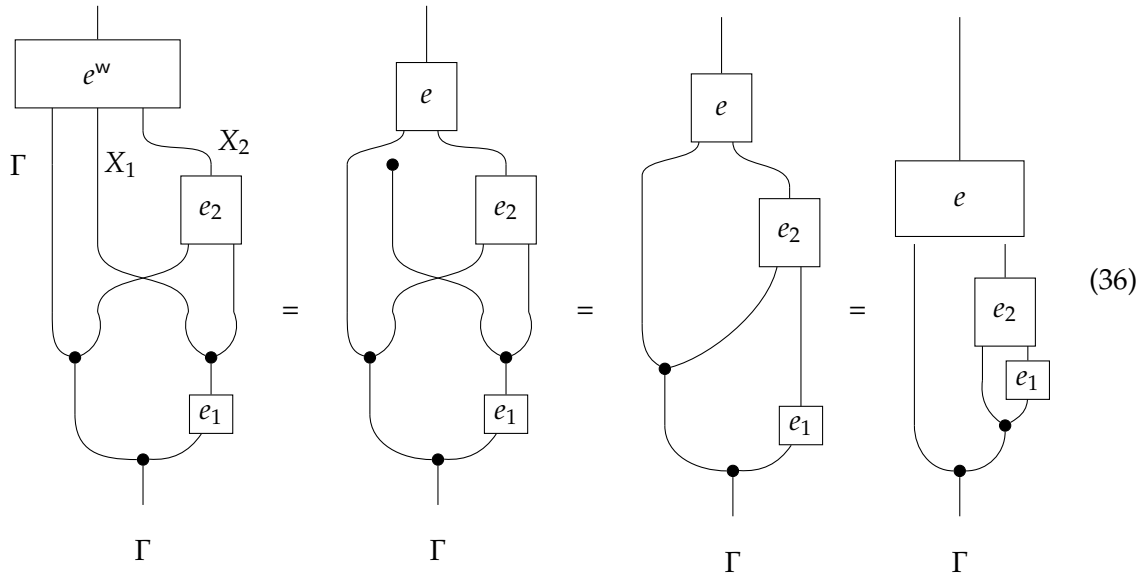
PROOF Straightforward induction using the comonoid axioms. ∎

Proposition 7.5 (Soundness) *Every CD model validates the axioms of the CD calculus. That is if* $\Gamma \vdash e_1 \equiv e_2 : A$ *then* $[\![e_1]\!] = [\![e_2]\!] : [\![\Gamma]\!] \to [\![A]\!]$.

PROOF The proofs are straightforward if tedious string diagram manipulations. We showcase the validation of one interesting equation, (assoc), here and move the remaining derivations to the appendix (Section 31). Let $\Gamma \vdash e_1 : X_1$, $\Gamma, x_1 : X_1 \vdash e_2 : X_2$ and $\Gamma, x_2 : X_2 \vdash e : Y$. Then

$$(\mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, \mathsf{let}\, x_2 \;=\; e_2 \,\mathsf{in}\, e^{\mathsf{w}}) \equiv (\mathsf{let}\, x_2 \;=\; (\mathsf{let}\, x_1 \;=\; e_1 \,\mathsf{in}\, e_2)\,\mathsf{in}\, e)$$

translates to the following equation in string diagrams



(36)

Note that we formally write e^{w} to emphasize the weakening of e; its denotation discards the unused X_1-wire as per Proposition 7.4. ∎

7.3 Syntactic Category

Every string diagram can be turned into a program, and the theory of $\equiv$ proves all ways of reading a diagram equivalent. Fix a signature $\mathfrak{S}$.

Definition 7.6 The *syntactic category* Syn consists of the following data

(i) objects are types A

(ii) morphisms are equivalence classes of terms $x : A \vdash t : B$ modulo $\equiv$

(iii) identities are variables $x : A \vdash x : A$

(iv) composition is let binding; if $x : A \vdash s : B$ and $x : B \vdash t : C$, their composite is

$$x : A \vdash \mathsf{let}\, x \;=\; e \,\mathsf{in}\, t$$

Composition is well-defined because of (let.ξ), and the category axioms follow from (let.val), (let.lin), (assoc). Note that formally, the free variable of morphisms is the fixed choice x, but we'll allow slight abuse of notion by allowing α-renaming.

Proposition 7.7 Syn *can be given the structure of a CD category (as below).*

PROOF Tensor on objects is defined as $A \otimes B = A * B$ with unit type unit. The tensor on morphisms of $x_1 : A_1 \vdash s_1 : B_1$, $x_2 : A_2 \vdash s_1 : B_2$ is

$$x : A_1 * A_2 \vdash \mathsf{let}\, x_1 \;=\; \pi_1\, x \,\mathsf{in}\, \mathsf{let}\, x_2 \;=\; \pi_2\, x \,\mathsf{in}\, (s_1, s_2)$$

This defines a symmetric monoidal structure with coherence terms

$$\begin{aligned}
\alpha_{A,B,C} &= x : (A * B) * C \vdash (\pi_1(\pi_1(x)), (\pi_2(\pi_1(x), \pi_2(x)))) : A * (B * C) \\
\alpha^{-1}_{A,B,C} &= x : A * (B * C) \vdash ((\pi_1(x), \pi_1(\pi_2(x)), \pi_2(x))) : (A * B) * C \\
\rho_A &= x : A * \mathsf{unit} \vdash \pi_1(x) : A \\
\rho^{-1}_A &= x : A \vdash (x, ()) : A * \mathsf{unit} \\
\mathsf{swap}_{A,B} &= x : A * B \vdash (\pi_2 x, \pi_1 x) : A * B
\end{aligned}$$

CD structure is given by the terms

$$\begin{aligned}
\mathrm{copy}_A &= x : A \vdash (x, x) : A * A \\
\mathrm{del}_A &= x : A \vdash () : \mathsf{unit}
\end{aligned}$$

The verification of the CD category axioms is tedious but standard. Note that we can build on existing work because our axioms prove all equations of the ground fragment of λ_c (Proposition 7.1). ∎

We expect that the syntactic category is an initial model over a given signature and the definition of the semantics $[\![-]\!]$ is forced by preserving CD structure, but we won't formalize this here.

8 Concepts of Synthetic Probability

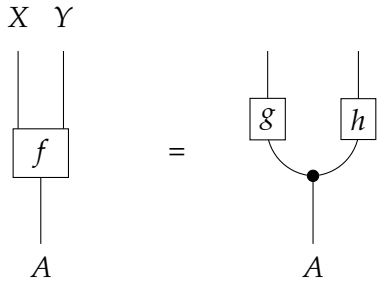
8.1 Independence

Stochastic independence is expressed in Markov categories by composing computation in parallel (independently) using the tensor product. The presence of parameters leads to the notion of *conditional independence*, of which we distinguish two flavors: Independence given inputs and given outputs. An early categorical formulation of these concepts can be found in [Coecke and Spekkens, 2012].

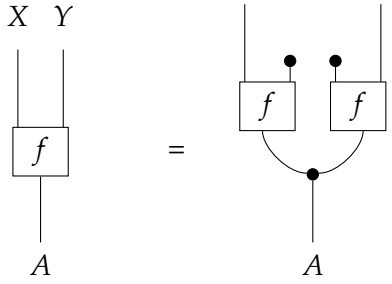
Independence given Inputs

Proposition 8.1 (Fritz [2020, 12.11]) *In a Markov category, the following are equivalent for* $f : A \to X \otimes Y$

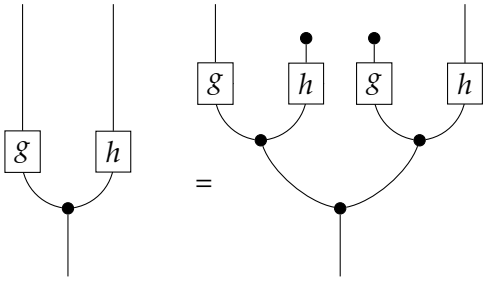
(i) there exist morphisms $g : A \to X$ *and* $h : A \to Y$ *such that* $f = \langle g, h \rangle$*, i.e.*



(ii) f *is the product of its marginals* $f = \langle f_X, f_Y \rangle$*, i.e.*



PROOF (ii) of course implies (i). For the converse, we observe the following equality



■

Definition 8.2 (Fritz [2020, 12.12]) A morphism $f : A \to X \otimes Y$ *displays the conditional independence* $X \perp Y \,||\, A$ if the equivalent conditions of Proposition 8.1 hold.

If f is understood, we will speak of X, Y as conditionally independent. In programming syntax, conditional independence takes the form of independent lines of code, i.e. we can obtain the variables x, y from expressions g, h such that $x \notin \mathsf{fv}(h), y \notin \mathsf{fv}(g)$ as

$$f = (\mathsf{let}\, x \;=\; g\, \mathsf{in}\, \mathsf{let}\, y \;=\; h\, \mathsf{in}\, (x, y))$$

In the distribution case $A = I$, conditional independence reduces to plain *independence*: $\psi : I \to X \otimes Y$ displays the independence $X \perp Y$ if μ is a product distribution, i.e.

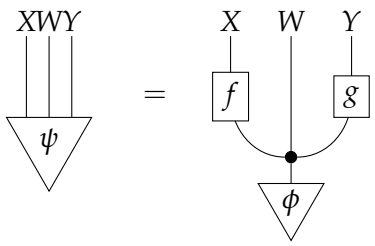
$$\psi = \psi_X \otimes \psi_Y.$$

Remark 8.3 In cartesian Markov categories, all morphisms $f : A \to X \times Y$ display the conditional independence $X \perp Y \,||\, A$. Note that a morphism $f : A \to X$ is deterministic if and only if its output is "independent of itself" in the sense that $\mathrm{copy}_X \circ f$ displays the conditional independence $X \perp X \,||\, A$.

Note that the double bar " $||$ " indicates conditional independence from the parameter or *input* A. This is conceptually different from conditional independence of an *output*, which is written with a single bar ' $|$ '.

Independence given Outputs

Definition 8.4 (Fritz [2020, 12.1], Cho and Jacobs [2019, 6.9(2)]) A distribution $\psi : I \to X \otimes W \otimes Y$ displays the conditional independence $X \perp Y \,|\, W$ of X, Y given the *output* W if there exist $\phi : I \to W$, $f : W \to X$, $g : W \to Y$ such that



where the triple-copy stands for any repetition of copying operations, which are all equal by associativity. The distribution ϕ must equal the marginal ψ_W. Note that f, g act as conditional distributions given W as in Section 8.4. The combinatorial relationship of this notion of conditional independence to graphical models in terms of "semigraphoids" has been discussed in [Fritz, 2020, Section 12] and [Coecke and Spekkens, 2012, Section 5].

8.2 Almost-sure Equality, Absolute Continuity and Supports

It is perhaps surprising that the notions of equality almost everywhere and absolute continuity (Definition 4.6) have elegant abstract characterizations in Markov categories. We can also use these to develop a synthetic notion of the support of a distribution.

Definition 8.5 ([Cho and Jacobs, 2019, 5.1], Fritz [2020, 13.1]) Let $\mu : I \to X$ be a distribution. Parallel morphisms $f, g : X \to Y$ are called *μ-almost surely equal* (written $f =_\mu g$) if

$$\langle \mathsf{id}_X, f \rangle \mu = \langle \mathsf{id}_X, g \rangle \mu$$

This is stronger than demanding f, g have the same pushforward $f\mu = g\mu$, because a copy of X is leaked (cf. Section 30.5) alongside. The expression $\langle \mathsf{id}_X, f \rangle \mu$ is reminiscent of the graph of a deterministic function f.

Example 8.6 The synthetic definition recovers the intended meaning in familiar cases

(i) In $\mathsf{FinStoch}$, $f, g : X \to D(Y)$ are μ-almost surely equal if the distributions $f(x) = g(x)$ agree for all x with $\mu(x) > 0$

(ii) In $\mathsf{BorelStoch}$, $f, g : X \to \mathcal{G}(Y)$ are μ-almost surely equal if $f(x) = g(x)$ as measures for μ-almost all x.

PROOF In [Fritz, 2020, 13.2] and [Fritz et al., 2020, 3.19]. ∎

Absolute continuity can be naturally phrased in terms of almost-sure equality:

Definition 8.7 (Fritz et al. [2020, 2.8]) Given two distributions $\mu, \nu : I \to X$, we say that μ is *absolutely continuous* with respect to ν, written $\mu \ll \nu$, if for all $f, g : X \to Y$ we have

$$f =_\nu g \text{ implies } f =_\mu g$$

Example 8.8 (Fritz et al. [2020, 2.9]) In $\mathsf{BorelStoch}$, this recovers the standard notion of absolute continuity (Definition 4.6), i.e. $\mu \ll \nu$ if and only if for all measurable sets A, $\nu(A) = 0$ implies $\mu(A) = 0$.

On the one hand, almost-sure equality and absolute continuity are simple categorical properties which naturally appear in many manipulations: Absolute continuity lets us strengthen statements about almost-sure equality to actual equality. For example if $f =_\mu g$ and $x \ll \mu$ then $fx = gx$. This will be a crucial ingredient in our theory of conditioning in (Section 18).

On the other hand, $\ll$ will often have a useful geometric interpretation, which motivates us to adopt the following terminology:

Convention 8.9 If $x : I \to X$ is a *deterministic state* and $x \ll \mu$, we will informally say that x *lies in the support of* μ.

Examples will be discussed in detail in Proposition 18.6, but for example in $\mathsf{BorelStoch}$, we have $x \ll \mu$ if and only if $\mu(\{x\}) > 0$. The notion of support depends on a the surrounding category. Restricting to a smaller Markov category like in Proposition 18.5 may give rise to more instances of $\ll$. Convention 8.9 will be our main formal meaning when speaking of 'supports' in this thesis, such as in Definition 19.4.

We will now outline a stronger notion of 'support' due to Fritz [2020], which may exist in a Markov category. To set this apart from Convention 8.9, we will sometimes refer to the stronger notion as 'representable supports'. The existence of these supports is often a prohibitively strong assumption, while the relation $\ll$ is applicable to any Markov category. We will therefore phrase our theory of conditioning (Chapter IV) in terms of $\ll$ only.

8.3 Representable Supports

As explained above, supports are way of reducing questions about almost-sure equality to actual equality. This leads to the following categorical definition of support:

Definition 8.10 (Fritz [2020, 13.20]) Let $\mathbb{C}$ be a Markov category and $\mu : I \to X$ be a distribution. There is a functor $H_\mu : \mathbb{C} \to \mathrm{Set}$ given by equivalence classes of morphisms modulo μ-almost sure equality

$$H_\mu(Y) \stackrel{\text{def}}{=} \mathbb{C}(X, Y) / =_\mu$$

with functorial action given by postcomposition

$$H_\mu(f)([g]) = f \cdot [g] \stackrel{\text{def}}{=} [fg]$$

A *(representable) support* for μ is a representation of the functor H_μ, that is a pair (S, α) of an object S and a natural isomorphism

$$\alpha : \mathbb{C}(S, -) \cong H_\mu$$

As usual for representing objects, the pair (S, α) is unique up to unique isomorphism. We elaborate on the definition by giving an explicit characterization of supports using a Yoneda-style argument.

Proposition 8.11 *To give a support for $\mu : I \to X$ is to give morphisms*

$$i : S \to X \qquad\qquad p : X \to S \tag{37}$$

such that

(i) for all $f, g : X \to Y$ we have $f =_\mu g \Leftrightarrow fi = gi$

(ii) $pi = \mathsf{id}_S$

(iii) $ip =_\mu \mathsf{id}_X$

PROOF Given $\alpha : \mathbb{C}(S, -) \cong H_\mu$, consider $\alpha_S(\mathsf{id}_S) \in H_\mu(S)$ and pick any representative $p : X \to S$ for it. Let $i \stackrel{\text{def}}{=} \alpha_X^{-1}([\mathsf{id}_X])$. The naturality equations for α, α^{-1} state that for every $g : X \to Y$ and $h : S \to Y$ we have

$$\begin{aligned} \alpha_Y(h) &= \alpha_Y(h \circ \mathsf{id}_S) = h \cdot [p] = [hp] \\ \alpha_Y^{-1}([g]) &= \alpha^{-1}(g \cdot [\mathsf{id}_X]) = gi \end{aligned}$$

From the latter equation, we can conclude that if $[f] = [g]$ then $fi = gi$. From the bijectivity of the transformations

$$\begin{aligned} \mathsf{id}_X &= \alpha_X^{-1}(\alpha_X(\mathsf{id}_X)) = \mathsf{id}_X pi = pi \\ [id_X] &= \alpha_X(\alpha_X^{-1}([\mathsf{id}_X])) = [\mathsf{id}_X ip] = [ip] \end{aligned}$$

hence $ip =_\mu \mathsf{id}_X$. Therefore if $fi = gi$ then $fip = gip$ hence $f =_\mu g$. ∎

Example 8.12 (Fritz [2020, 13.23]) In FinStoch, the support of $\mu : I \to X$ is

$$S = \{x \in X : \mu(x) > 0\}$$

More precisely, i is the deterministic inclusion $i : S \hookrightarrow X$ and p any retraction of it.

The support of multivariate Gaussian distributions $\mathcal{N}(\vec{\mu}, \Sigma)$ on $\mathbb{R}^n$, as formalized in the Markov category Gauss (Definition 18.2), will indeed turn out to be the affine subspace $\{\vec{\mu} + \Sigma\vec{x} \mid \vec{x} \in \mathbb{R}^n\}$ as expected (Proposition 18.5). On the other hand, supports do not exist in BorelStoch (Proposition 18.6). This is consistent with our slogan from the introduction that smaller Markov categories admit stronger universal properties.

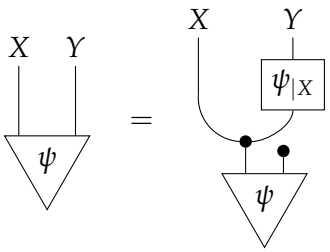
Remark: Following up on Convention 8.9, the existence of representable supports is generally too strong an assumption to require. It is often sufficient to work with the absolute continuity relation $x \ll \mu$ for $x : I \to X$ deterministic. If a support $i : S \to X$ of μ exists, this condition simplifies to checking that x factors through S, which can be seen as follows

- x factors through i if and only if $x = ipx$.
- Assume $x = ipx$ and consider morphisms with $f =_\mu g$, then by (i) $fi = gi$ hence $fx = fipx = gipx = gx$. Because f, g were arbitrary, we conclude $x \ll \mu$.
- Conversely, assume $x \ll \mu$, then by (iii) we have $ip =_\mu \mathrm{id}_X$ hence by absolute continuity $ip =_x \mathrm{id}_X$, so $ipx = x$, so x factors through i.

8.4 Conditionals

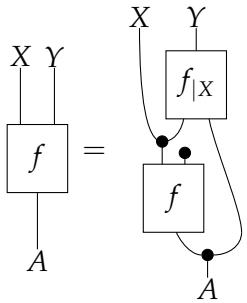
Conditioning means recovering a joint distribution only given access to part of its information. Its categorical formulations trace back to Golubtsov and Cho-Jacobs.

Definition 8.13 (Fritz [2020, 11.1]) A *conditional distribution* for $\psi : I \to X \otimes Y$ (given X) is a morphism $\psi|_X : X \to Y$ such that



(38)

A *(parameterized) conditional* for $f : A \to X \otimes Y$ is a morphism $f|_X : X \otimes A \to Y$ such that

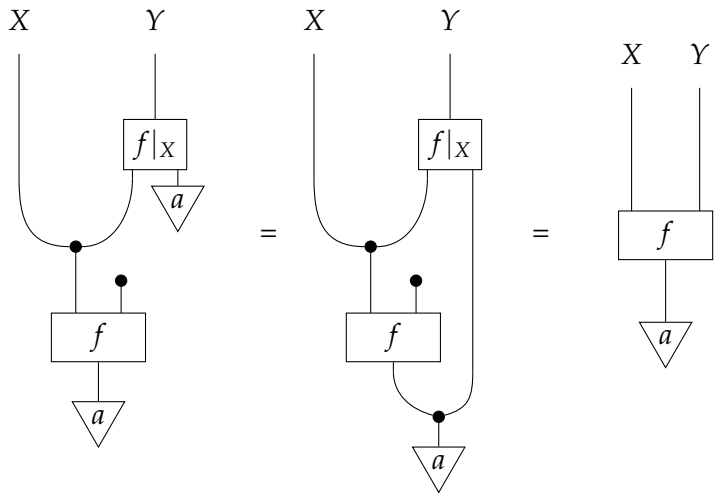


(39)

Parameterized conditionals can again be specialized to conditional distributions by fixing a parameter

Proposition 8.14 *If $f : A \to X \otimes Y$ has conditional $f|_X : X \otimes A \to Y$ and $a : I \to A$ is a deterministic state, then $f|_X(\mathsf{id}_X \otimes a)$ is a conditional distribution for fa.*

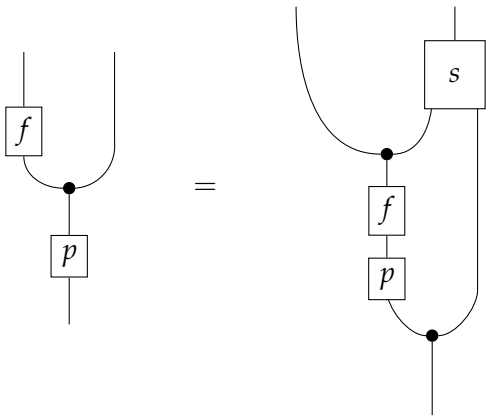
PROOF Using determinism of a, we check that



■

Conditionals allow to recover the joint distribution from only one marginal. The existence of conditionals is a very strong property of a Markov category: Given a joint distribution over X, Y, they allow us to form a generative story whereby X is sampled first, and Y is then sampled dependent on X. If the conditional on Y exists, we can similarly imagine Y sampled first. In programming terms, conditionals are a powerful way of restructuring dataflow to our liking.

Disintegration Fritz [2020, 11.17] gives an equivalent formulation of conditionals using a different shape of diagram. A *disintegration* of a morphism $p : A \to X$ with respect to $f : X \to Y$ is a morphism $s : A \otimes Y \to X$ such that



This corresponds to the usual definition of disintegration or regular conditional probability in Stoch [Fritz, 2020, 11.18]. Because conditionals and disintegrations are interdefinable, we

will make no further use of that definition and sometimes speak of disintegration as a synonym to conditioning.

Conditionals are not unique. However we note that if $\psi|_X, \psi|'_X$ are two morphisms satisfying (38) then we obtain immediately from the definitions that

$$\psi|_X =_{\psi_X} \psi|'_X \tag{40}$$

That is, conditional distributions are unique almost surely with respect to the marginal ψ_X of the variable we conditioned on. We will give a detailed account of how to use the categorical machinery to build a programming language with conditioning in Section 18.

Proposition 8.15 FinStoch, BorelStoch *and nondeterminism have conditionals.*

PROOF In FinStoch, conditionals are given by the traditional conditional distribution [Fritz, 2020, 11.2]

$$\psi|_X(y|x) = \frac{\psi(x,y)}{\psi(x)}, \pi(x) > 0$$

Note that this is defined on the support of ψ_X only (and can be extended arbitrarily outside of it). In Stoch, the definition of conditionals recovers *regular conditional distributions* which are known to exist for standard Borel spaces. Given a relation $\Psi \subseteq X \times Y$, a conditional $f : X \to \mathcal{P}(Y)$ is given by the slicing

$$f(x) = \{y \in Y : (x,y) \in \Psi\}$$

■

9 Dataflow Axioms

A wide range of formal structures fit the axioms of Markov categories. Further properties can then be imposed to recover various familiar aspects of traditional probability. The properties we summarize under "dataflow axioms" in this section are information-theoretic in nature: They capture that information flow behaves classically in certain ways. These properties do for example distinguish classical probability from negative probabilities, but cannot distinguish between probability and nondeterminism.

We recall the notions of *positivity* and *causality*. We then give novel characterizations of each and establish that causality implies positivity. We will later see that failure of positivity is linked to information leaking, which makes it a crucial tool in analyzing name generation (Section 26). This section is based on an upcoming article with Tobias Fritz, Tomáš Gonda, Nicholas Gauguin Houghton-Larsen and Paolo Perrone.

9.1 Positivity

Positivity arose as a candidate axiom to rule out the "destructive interference" seen in negative probability Fritz [2020, 11.22]. We will give novel characterizations which suggest the axiom is rather about the absence of *information hiding* than negativity, and discover non-examples involving strictly nonnegative probability. Yet, we'll stick with the name *positivity* for historic reasons.

Definition 9.1 (Fritz [2020, 11.22]) A Markov category $\mathbb{C}$ is *positive* if whenever $f : X \to Y$ and $g : Y \to Z$ are such that $g \circ f$ is deterministic, then

$$(g \otimes \mathrm{id}_Y) \circ \mathrm{copy}_Y \circ f = ((g \circ f) \otimes f) \circ \mathrm{copy}_X \tag{41}$$

This formulation of the axiom is somewhat mysterious, but it suggests that irrelevant intermediate results cannot introduce correlations. If gf is deterministic, the intermediate output of f may be resampled rather than copied. We'll discuss examples once we've arrived at our equivalent characterization:

In statistics, constants are independent of all other variables. For example, given a joint distribution (X, Y), if $X \overset{\mathrm{d}}{=} x_0$ then $Y \perp X$. We can adapt this into the following definition for Markov categories.

Definition 9.2 (Deterministic marginal property) A Markov category $\mathbb{C}$ has the *deterministic marginal property* if for every $f : A \to X \otimes Y$, whenever one marginal f_X is deterministic then f is the product of its marginals

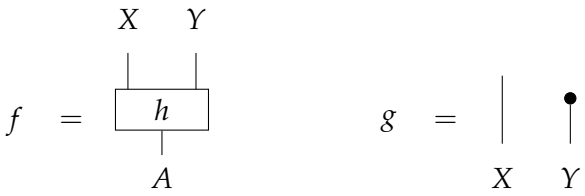
$$f = \langle f_X, f_Y \rangle \tag{42}$$

Proposition 9.3 *The following are equivalent for a Markov category* $\mathbb{C}$*.*

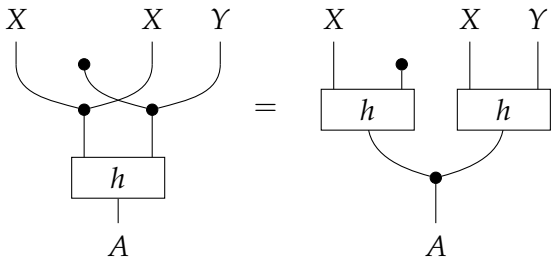
(i) $\mathbb{C}$ *is positive*

(ii) $\mathbb{C}$ *has the deterministic marginal property*

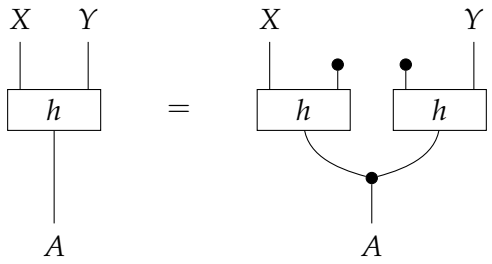
PROOF (i)⇒(ii): Let $h : A \to X \otimes Y$ be given with deterministic marginal h_X; instantiate the positivity axiom with



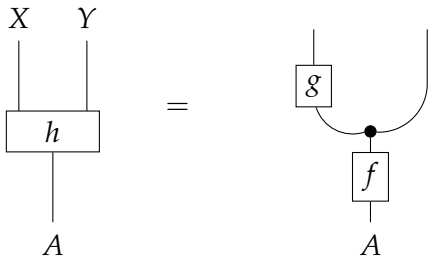
Then $gf = h_X$ is deterministic by assumption, so (41) reads



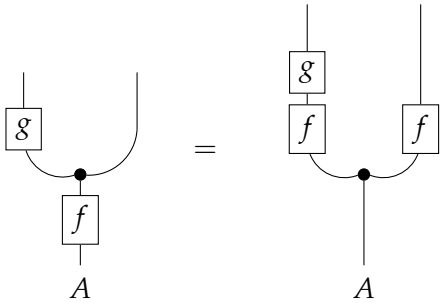
From this, we obtain by further marginalizing the middle wire



(ii)⇒(i): Let $f : A \to X$, $g : X \to Y$ be such that gf is deterministic. Define



By assumption, the marginal h_X is deterministic, so the deterministic marginal property allows us to conclude that



which is precisely the positivity equation (41). ∎

We can now intuitively see why negative probabilities break positivity. Consider again the joint distribution $\mu : 1 \to 2 \times 2$ in the Kleisli category of $D^{\pm}$ (Section 5.4) given by

$$\mu = -\frac{1}{2}[(0,0)] + \frac{1}{2}[(0,1)] + \frac{1}{2}[(1,0)] + \frac{1}{2}[(1,1)]$$

Its marginals are both deterministic, $\mu_X = \mu_Y = [1]$ but μ is *not* a product distribution, for X and Y are perfectly anti-correlated, that is $\Pr(X = Y) = 0$.

Measure theory on the other hand *is* positive [Fritz, 2020, 11.25]. We can easily rederive this using the deterministic marginal property: Let μ be a probability measure on $X \times Y$, we can compare it with the product of its marginals in measurable rectangles $A \times B$ by

$$(\mu_X \otimes \mu_Y)(A \times B) = \mu_X(A) \cdot \mu_Y(B) = \mu(A \times Y) \cdot \mu(X \times B)$$

If μ_X is deterministic, i.e. $\mu(A \times Y) \in \{0, 1\}$, we obtain

$$\mu(A \times Y) \cdot \mu(X \times B) = \mu(A \times B)$$

because $\mu(U) \cdot \mu(V) = \mu(U \cap V)$ holds whenever $\mu(U) \in \{0, 1\}$. This proof certainly uses the fact that $\mu(U)$ is nonnegative; for example $\{X = 1\}$ and $\{Y = 1\}$ are non-independent probability 1-events in the negative probability example. However, it also relies heavily on the fact that joint distributions are defined on product σ-algebras, which allows us to reduce their behavior to measurable rectangles. It is this aspect that breaks in quasi-Borel spaces (see Section 27.2), allowing non-positivity while maintaining nonnegative probabilities. A prototypical example of non-positivity is name generation, which will be discussed extensively in Section 26.

Proposition 9.4 *Every Markov category with conditionals is positive.*

PROOF This is [Fritz, 2020, 11.24]. Note that this is consistent with Proposition 9.12 and the fact that every Markov category with conditionals is causal. ∎

Proposition 9.5 *The following Markov categories are positive:* FinStoch, BorelStoch, Stoch, FinSetMulti

PROOF We have shown that Stoch has the deterministic marginal property, and positivity restricts to its subcategories. For the case of FinSetMulti, it is easy to see that if a relation $R \subseteq X \times Y$ satisfies

$$\{x : (x, y) \in R\} = \{x_0\}$$

then R is the product of its marginals $R = \{x_0\} \times \{y : (x, y) \in R\}$. Alternatively, FinSetMulti is known to have conditionals (Proposition 8.15). ∎

We remark that a notion of Jacobs [2016] can now be related to positivity.

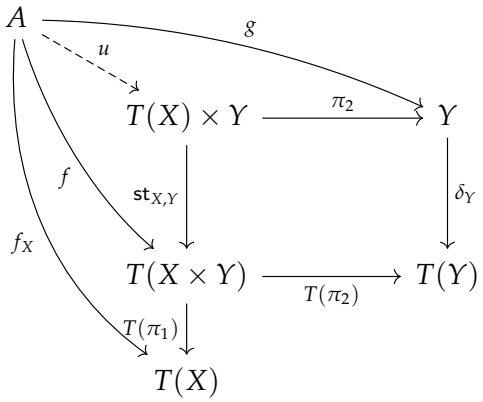
Definition 9.6 (Jacobs [2016, Def. 1]) A commutative monad on a category $\mathbb{C}$ with binary products is called *strongly affine* if the following square is a pullback for all X, Y

$$\begin{array}{ccc} T(X) \times Y & \xrightarrow{\pi_2} & Y \\ {\scriptstyle \mathsf{st}_{X,Y}}\downarrow & \lrcorner & \downarrow{\scriptstyle \delta_Y} \\ T(X \times Y) & \xrightarrow[T(\pi_2)]{} & T(Y) \end{array} \tag{43}$$

Jacobs shows that all strongly affine monads are in fact affine, justifying the terminology. The pullback can be read as a direct categorification of the deterministic marginal property: The only distributions in $T(X \times Y)$ that have a deterministic Y-marginal are those that come from constants Y. It is not surprising to obtain the following characterization of positive Kleisli categories.

Proposition 9.7 *Let T satisfy the representability condition (Definition 6.8). Then the Kleisli category $\mathbb{C}_T$ is positive iff T is strongly affine.*

PROOF Let T be strongly affine and $f : A \to T(X \times Y)$ have a deterministic marginal f_Y. By representability, $f_Y = \delta_Y g$ for some $g : A \to Y$. So the outer square commutes and we obtain a factorization u



which is necessarily of the form $u = \langle f_X, g \rangle$ as seen by composing with the appropriate projections. It follows that f is the product of its marginals.

Conversely let a commutative outer square be given, then $f_Y = \delta_Y g$ is deterministic and by the deterministic marginal property, we obtain that $u = \langle f_X, g \rangle$ is the (necessarily unique) factorization, making the square (43) a pullback. ∎

Combining Proposition 6.10 and Proposition 9.5, we see that the monads D, $\mathcal{P}_{\mathsf{f}}^{+}$ and $\mathcal{G}^{\mathsf{Sbs}}$ are strongly affine. The Giry monad on Meas is strongly affine despite violating representability, and the affine combination monad $D^{\pm}$ is not *not* strongly affine [Jacobs, 2016, Ex. 1,2].

We conclude with a somewhat subtle pathology of certain Markov categories, and show that this situation can *not* arise if the Markov categories are positive.

Example 9.8 (Fritz [2020, 10.10]) Isomorphisms in Markov categories are not necessarily deterministic.

PROOF Let $\mathbb{C}$ be the category of commutative monoids in Set. Take for example $X = \{0, 1, 2\}$, then we can find two distinct commutative monoid multiplications $m_1, m_2 : X \times X \to X$ which both have 0 as the neutral element. The opposite category $\mathbb{C}^{\mathrm{op}}$ is a Markov category (Section 6.2.3), and

$$\mathsf{id}_X : (X, m_1, 0) \to (X, m_2, 0)$$

is a counit-preserving isomorphism. It is however not deterministic because it is not comultiplication preserving ($m_1 \neq m_2$). ∎

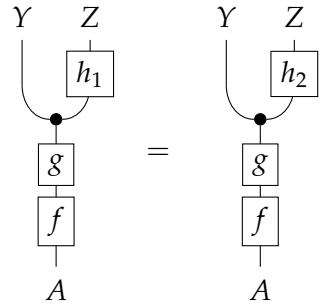
We regard this as highly pathological, because it means Markov structures aren't necessarily stable under monoidal equivalence (cf. Remark 6.11). This phenomenon is uncommon though, and it won't occur in nice cases:

Proposition 9.9 (Fritz [2020, 11.29]) *In a positive Markov category, all isomorphisms are deterministic.*

9.2 Causality

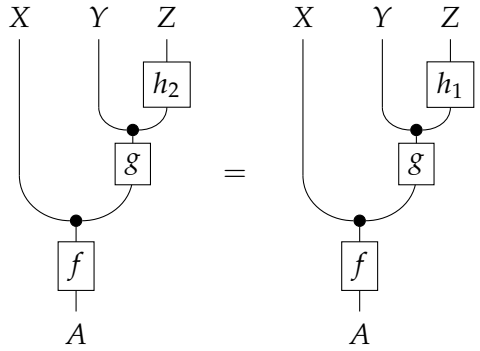
Causality is another dataflow axiom identified in [Fritz, 2020]. As remarked by Fritz, this axiom is equivalent to the notion of *equality strengthening* discussed by Cho and Jacobs [2019, Def. 5.7], but we won't need that formulation. We briefly remark that causality is useful for reasoning about supports, and prove that it is a stronger property than positivity (Proposition 9.12).

Definition 9.10 (Fritz [2020, 11.31]) A Markov category $\mathbb{C}$ is called *causal*[10] if every equation



(44)

implies the stronger equation



(45)

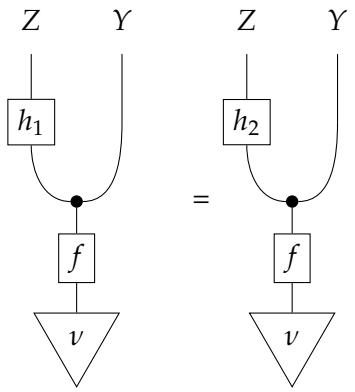
Causality implies that supports behave sensibly under pushforward. This will be useful for reasoning about the satisfiability of conditions in Theorem 19.6.

Proposition 9.11 *Let $\mathbb{C}$ be causal, then the absolute continuity relation $\ll$ (Definition 8.7) respects composition in the sense that for all $f : X \to Y$ and $\mu, \nu : I \to X$*

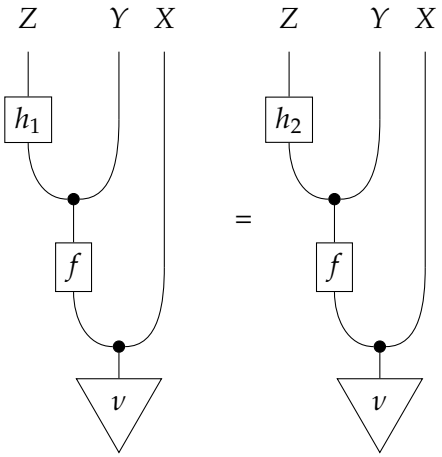
$$\mu \ll \nu \Rightarrow f\mu \ll f\nu$$

[10] This definition is unrelated to the use of the word *causal* in categorical quantum foundations, which means terminality of the unit.

PROOF Let $\mu \ll \nu$ and consider arbitrary $h_1, h_2 : Y \to Z$ such that $h_1 =_{f\nu} h_2$, i.e.



Causality implies the stronger equation

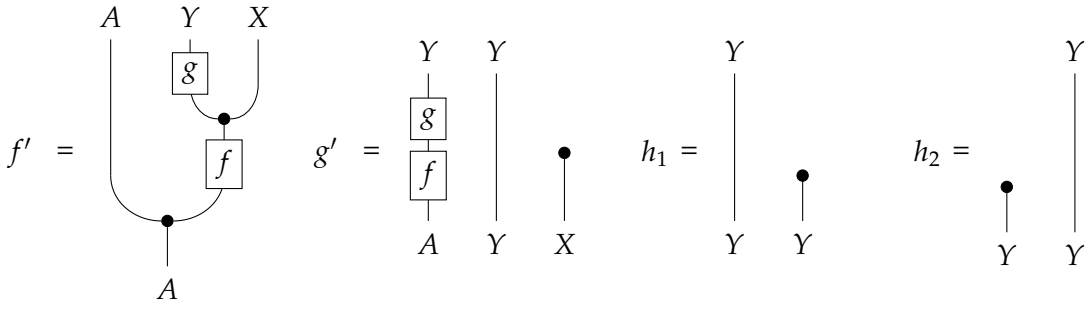


By the absolute continuity assumption, we may replace ν by μ in this equation; marginalizing the X-wire shows $h_1 =_{f\mu} h_2$. ∎

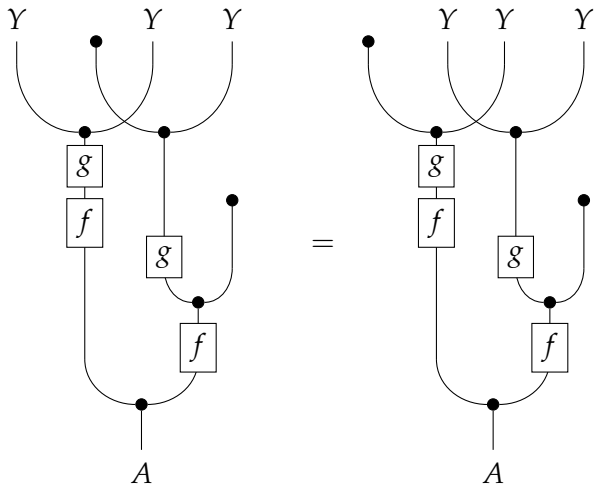
Despite having rather different shapes, we show that the causality axiom can be used to derive positivity.

Proposition 9.12 *Every causal Markov category is positive.*

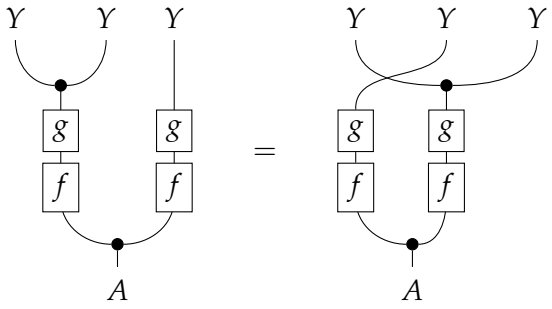
PROOF Let f, g be such that gf is deterministic. We make the following choices of morphisms to which to apply the causality axiom (writing f',g' in place of f, g).



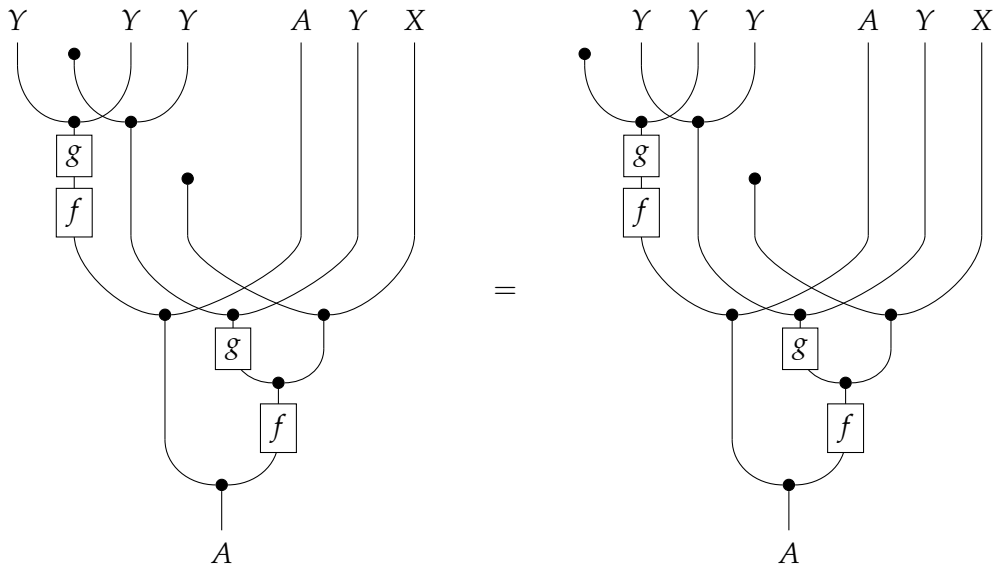
We check that the assumption (44) holds, that is



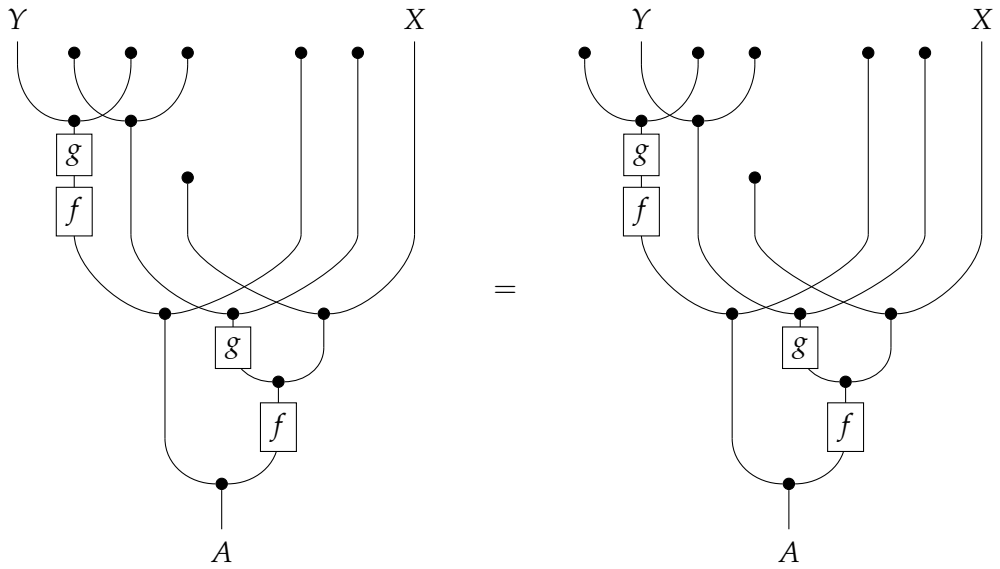
which simplifies to



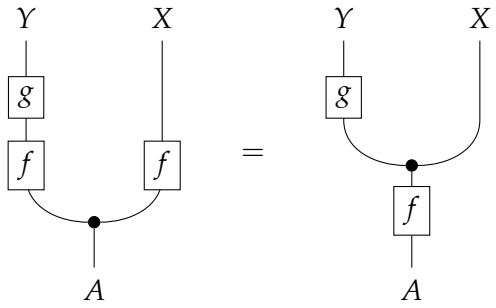
which holds by the fact that gf is deterministic. The causality axiom now implies the stronger equation (45), which reads



We simplify by marginalizing over all occurrences of Y except the first one, obtaining



which can be simplified to yield (41)



■

The converse to Proposition 9.12 is shown to be false in [Fritz and Rischel, 2020].

Proposition 9.13 (Fritz [2020, 11.34]) *Every Markov category with conditionals is causal.*

Again, the converse is false: It is known that Stoch is causal [Fritz, 2020, 11.35] but does not have all conditionals.

Chapter III

The Beta-Bernoulli Process and Algebraic Effects

SUMMARY: This chapter is about a synthetic presentation of the interaction of the Beta- and Bernoulli distributions. These distributions are important primitives in Bayesian statistics, because they model biased choices as well as beliefs over the biases. Our presentation is purely algebraic and combinatorial, and avoids all mention of measure theory. We will however prove the theory complete with respect to measure theory (Theorem 12.10) as well as syntactically complete (Theorem 13.5). Our theory is an instance of a general recipe for presenting interesting Markov categories (Section 14).

From the programming side, our analysis can be conducted in terms of a minimalistic probabilistic programming language and the framework of algebraic effects (Section 3.6). In addition to exchangeability and discardability, it reveals the importance of abstract datatypes and a program equation called conjugacy. Monadic programming naturally lends itself to expressing hierarchical models (beliefs over beliefs).

Our theory allows us to make formal connections to a different, stateful implementation of the Beta-Bernoulli process through *Pólya's urn*: Our syntactic completeness result implies that both implementations must satisfy precisely the same program equations. The link between sampling and generativity (here: creating a new urn) is a persistent theme throughout the thesis (e.g. random variables in Chapter IV, names in Chapter V).

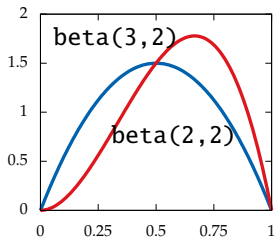
This chapter is based on joint work with Sam Staton, Hongseok Yang, Nathanael Ackerman, Cameron Freer and Daniel Roy [Staton et al., 2018]. It formalizes and proves a conjecture from [Staton et al., 2017b].

We begin with a leisurely introduction to the topics of interest:

10 Introduction

10.1 Beta-Bernoulli

Alice and Bob watch a football player score a penalty. Alice has seen the player score on 80 out of 100 kicks, while Bob – newer to football – has only seen them succeed 8 out of 10 times. Both observers estimate that the player will on average score 80% of kicks. However, Alice's assessment of this probability is *more certain*, as she has access to more prior information. On the other hand, Bob *learns more* from the latest observation, revising his opinion more than Alice.



Bob's *prior belief* of the player's skill (before the latest kick) can be modeled by the distribution $\mathsf{Beta}(7,2)$ where the parameters record the previous observations of 7 successes versus 2 failures. The distribution $\mathsf{Beta}(\alpha,\beta)$ has the following density over the interval

$[0,1]$

$$\mathsf{pdf}_{\alpha,\beta}(p) = \frac{1}{B(\alpha,\beta)} p^{\alpha-1}(1-p)^{\beta-1} \tag{46}$$

where the normalization constant $\mathrm{B}(\alpha,\beta)$ is given in terms of the Gamma function as

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

In the remainder of this chapter, we will focus on integer parameters with $\alpha+\beta > 1$. In this case, the normalization constant can be written more simply as

$$B(\alpha,\beta) = \frac{(\alpha-1)!(\beta-1)!}{(\alpha+\beta-1)!}$$

To predict the outcome X of the latest kick, Bob first samples the success probability p for the player according to his belief, and then models the kick succeed with probability p

$$\begin{aligned} p &\sim \mathsf{Beta}(7,2) \\ X &\sim \mathsf{Bernoulli}(p) \end{aligned}$$

Conditioning this model on the outcome $X = 1$, the variable $p|(X=1)$ has density

$$\begin{aligned} f(p) &= \left(\int_0^1 p \cdot \mathsf{pdf}_{7,2}(p)\mathrm{d}p\right)^{-1} \cdot p \cdot \mathsf{pdf}_{7,2}(p) \\ &= \left(\int_0^1 p^7(1-p)^1\mathrm{d}p\right)^{-1} p^7(1-p)^1 \\ &= \frac{1}{B(8,2)} p^7(1-p)^1 \end{aligned}$$

which we recognize as the density of $\mathsf{Beta}(8,2)$. This is consistent with the interpretation that Bob has now observed 8 successes and 2 failures. Conditioned on a negative outcome $X = 0$, Bob would instead have the posterior belief $\mathsf{Beta}(7,3)$. Alice's *posterior belief* is modeled by $\mathsf{Beta}(80,20)$. Both $\mathsf{Beta}(8,2)$ and $\mathsf{Beta}(80,20)$ have mean 0.8, but the latter has a much sharper peak. This situation that a Beta prior conditioned on a Bernoulli outcome is again Beta, is referred to as a **conjugate prior relationship**.

We work towards giving a synthetic presentation of the Beta-Bernoulli relationship. Consider a probabilistic language with a type `bool` for booleans and an abstract type `I` for the unit interval $[0,1]$. Then Bob's belief update can be expressed as the program

```
let p : I = beta(7,2)
let x : bool = flip(p)
condition(x == true)
return p
```

The conjugate prior relationship implies that this program can be simplified to

```
let p = beta(8,2) // conjugate prior update
let x = true // we conditioned on x being true
return p
```

The condition has been removed, to the effect of revising a prior in the past. Let's ask Bob to predict the outcome of the *next* penalty, that is the query

```
let p = beta(7,2)
let x = flip(p)
condition(x == true)
return flip(p) // predict another kick
```

We notice that this is a *hierarchical model*: The true bias p is latent, and Bob makes two coin flips with that unknown bias. Because of that, the coin flips are not independent, but only conditionally independent given p. By the previous consideration, this model simplifies as

```
let p = beta(8,2)
let x = true
return flip(p)
```

The value of `x` is irrelevant and can be discarded by (Disc). If we sample $p \sim \mathsf{Beta}(\alpha, \beta)$ and immediately perform a coin flip $X \sim \mathsf{Bernoulli}(p)$, then that coin comes up heads with probability $\alpha/(\alpha+\beta)$. In code

$$\texttt{let p = beta}(\alpha,\beta)\ \texttt{in flip(p)} \approx \texttt{flip}(\alpha/\alpha{+}\beta)$$

The whole program thus reduced to the answer `flip(8/10)`. This technique of evaluating a probabilistic program is called **symbolic inference**.

10.2 Towards an Algebraic Theory

We use these ideas to give a model of Beta-Bernoulli that is purely based on algebra and combinatorics; no measure theory or analysis is required. To simplify reasoning with program equations, we write the probabilistic language in continuation passing style and introduce the following shorthand.

Shorthand	Long expression
$\nu_{\alpha,\beta} p.u$	`let p = beta(`α`,`β`) in u`
$u \,?_p\, v$	`if flip(p) then u else v`
$u +_{m:n} v$	`if flip(m/(m+n)) then u else v`

We can axiomatize these operations equationally using an algebraic theory. This theory needs to be second-order because $\nu_{\alpha,\beta}$ is a binder (Section 3.7). For example, following (Disc),

there is an axiom stating that unused samples are discardable, i.e. for $p \notin \mathrm{fv}(u)$ we have

$$\nu_{\alpha,\beta} p.u \equiv u \tag{Disc}$$

A special role among the axioms is played by the *conjugacy axiom*, which expresses the symbolic update rule for Beta distributions:

$$\nu_{\alpha,\beta} p.(u(p) \mathbin{?}_p v(p)) \equiv (\nu_{\alpha+1,\beta} p.u(p)) +_{\alpha:\beta} (\nu_{\alpha,\beta+1} p.v(p)) \tag{Conj}$$

Making a coin flip following a Beta draw means randomly choosing an outcome, but inside each branch, the parameters of the Beta distribution have been updated to reflect the new information. Using the shorthand, we may express Bob's prediction of the next two kicks as the term

$$\nu_{7,2} p.\ (x \mathbin{?}_p y) \mathbin{?}_p z$$

which models one kick, and then another one ($x \mathbin{?}_p y$) if the first one succeeds. Using the two axioms above, we can simplify this term to

$$\begin{aligned} \nu_{7,2} p.\ (x \mathbin{?}_p y) \mathbin{?}_p z &\equiv (\nu_{8,2} p.x \mathbin{?}_p y) +_{7:2} (\nu_{7,3} p.z) \\ &\equiv ((\nu_{9,2} p.x) +_{8:2} (\nu_{8,3} p.y)) +_{7:2} z \\ &\equiv (x +_{8:2} y) +_{7:2} z \end{aligned}$$

showing both the posterior 8:2 in case the first kick succeeds, and the model evidence 7:2.

We give the full theory in Section 11.3. Its term model defines a synthetic model of probability theory. This model is purely combinatorial and yet contains interesting probability distributions like the Beta family which ordinarily require density functions and measure theory (Section 14).

We can now at least informally state our two main theorems (with references to the formal statement): Firstly, our theory is complete with respect to measure-theoretic semantics:

Informal Theorem (Completeness – Theorem 12.10) *An equation $u \equiv v$ is derivable from the axioms if and only if it holds in measure theory.*

In fact, we can show a stronger completeness result stating that the axioms are maximally consistent in a certain way, meaning that every nondegenerate model is complete:

Informal Theorem (Syntactic completeness – Theorem 13.5) *For every equation $u \equiv v$,*

(i) either the equation $u \equiv v$ is derivable from the axioms of the theory

(ii) if not, then assuming *$u \equiv v$ and its substitution instances, we can logically derive an inconsistency of finite probabilities. That is we can derive*

$$x +_{i:j} y \equiv x +_{k:l} y \text{ for all } i, j, k, l \geq 1$$

The syntactic completeness result is interesting because it allows us to make a precise formal connection between the Beta-Bernoulli process and a different stochastic process (Pólya's urn). Those processes are related by way of De Finetti's theorem, as we explain now.

10.3 Pólya's Urn, Exchangeability and Abstraction

Pólya's urn is a statistical urn which implements *drawing with duplication*: We select a ball from the urn, record its color and return it to the urn *along with a copy* of the same color. Let $U_0 = (R_0, B_0)$ record the number of red and black balls the urn contains at time 0, then we let

$$X_n \sim \mathsf{Bernoulli}(R_n/(R_n + B_n))$$
$$U_{n+1} = \begin{cases} (R_0 + 1, B_n) & \text{if } X_n = 1 \\ (R_0, B_n + 1) & \text{if } X_n = 0 \end{cases}$$

The draws (X_i) are *not* independent. If we start with one red and one black ball, observing three red draws in a row will make a future red draw much more likely. It does however not matter in which order the outcomes were observed: A sequence $X_0, X_1, \ldots$ of random variables is called *exchangeable* if for any permutation σ on $\{0, \ldots, n\}$, the vectors $(X_0, \ldots, X_n)$ and $(X_{\sigma(0)}, \ldots, X_{\sigma(n)})$ have the same distribution. One can show that the draws (X_n) of Pólya's urn indeed form an exchangeable sequence. The connection to Beta-Bernoulli is made through *De Finetti's theorem* (e.g. [Schervish, 1995]):

Theorem 10.1 (De Finetti) *Let $X_0, \ldots$ be an infinite exchangeable sequence of $\{0,1\}$-valued random variables. Then we can model the X_i as conditionally independent coin flips. That is, there is a unique distribution π on $[0,1]$ such that the sequence (X_i) has the same distribution as*

$$p \sim \pi$$
$$X_i \sim \mathsf{Bernoulli}(p) \text{ } iid.$$

That is, we choose the bias p of a coin at random, and the X_i are drawn independently given *that p.*

Applying this result to Pólya's urn precisely recovers the Beta-Bernoulli process:

Proposition 10.2 *Let $U_0 = (R_0, B_0)$, then the sequence (X_i) arises as*

$$p \sim \mathsf{Beta}(R_0, B_0)$$
$$X_i \sim \mathsf{Bernoulli}(p) \text{ } iid.$$

Pólya's urn has an obvious implementation in an impure probabilistic language with coin flips and state.

```
module Polya = struct
  type process = (int * int) ref // a mutable urn (number of red, black balls)
  let new (i,j) = ref (i,j) // urn creation is memory allocation
  let get p =  // drawing from the urn is stateful
    let (i,j) = !p in
    if flip(i/(i+j))
      then p := (i+1,j); true
      else p := (i,j+1); false
end
```

Recall that the interface to a pseudorandom number generator looks commutative and discardable despite its use of state as long as the seed is kept abstract. The same properties hold for Pólya's urn as long as the contents of an urn cannot be inspected. We can enforce this sort of opacity by hiding the urns in an abstract type `process` in the following signature

```
module AbstractProcess = sig
  type process // abstract type of urns
  val new : int * int -> process // initialize process
  val get : process -> bool // draw from the process
end
```

The `AbstractProcess` module has another implementation which is *stateless* but uses continuous probability: Beta-Bernoulli

```
module BetaBern = struct
  type process = I // unit interval
  let new (i,j) = beta(i,j)
  let get p = flip(p)
end
```

In which way are the two implementations related? This is akin to asking for a finitary, programming-language theoretic instance of De Finetti's theorem. The module `Polya` has a straightforward operational semantics (which we won't formalize here). By contrast, `BetaBern` has a straightforward denotational semantics using measure theory (Section 4.2).

A connection between the two semantics can be made by showing that our algebraic theory is sound for both accounts. By the syntactic completeness theorem, both models are then also complete for the theory, and must thus satisfy the exact same equations:

Informal Theorem (Section 13.4) `Polya` *and* `BetaBern` *have the same equational theory.*

It is however not straightforward to verify that `Polya` validates our axioms! We elaborate a strategy for doing this in Section 13.4, which relies on the methods of Section 13.

Exchangeable random processes are an important primitive in Bayesian nonparametric statistics (see [Ackerman et al., 2016a; Staton et al., 2017b] and the references therein). We contend that the commutativity and discardability equations are very close to this notion of exchangeability. A client program for the `BetaBern` module is clearly commutative by Fubini's theorem. For the `Polya` module, an elementary calculation is needed: it is not trivial because memory is involved.

Our findings points to an interesting phenomenon which will occur repeatedly throughout this thesis. Pólya's urn satisfies (Comm), (Disc), that is its theory looks like that of a pure probabilistic language. We can therefore already speak of `new` and `get` as *synthetic probabilistic operations*. From this perspective, it is maybe not surprising to find that a synthetic probabilistic effect (urn creation) can be implemented in terms of an actual probabilistic effect (Beta sampling).

10.4 Algebraic Effects, Monads and Models of Synthetic Probability

Algebraic effects (Section 3.6) provide a concise way to axiomatize the specific features of an effect while putting aside the general properties of programming languages, such as β/η laws. A full programming language will feature other constructs, but it is routine to combine these with an algebraic theory of effects (e.g. [Ahman and Staton, 2013; Johann et al., 2010; Kammar and Plotkin, 2012; Pretnar, 2010]).

In Section 14, we spell out how our algebraic theory is a convenient tool to present categorical models of probability. This means we have a variety of internal languages at our disposal (Sections 3.1 and 7) and can analyze the theory using the concepts of synthetic probability theory.

It is straightforward to turn the terms of the algebraic theory into morphisms of a Markov category. Using general methods (Section 3.7), the theory also induces a generalized probability monad P on the presheaf category $[\mathsf{Fin}, \mathsf{Set}]$. That category is also cartesian closed and serves as a natural domain for semantics of a fully-fledged programming language.

Double distributions: Monadic programming offers a particularly natural way of talking about Beta-Bernoulli. Classically, the Beta distribution is defined to take values $p \in [0,1]$ in the unit interval, but recall from Example 4.10 that every such value can be understood as a distribution flip_p on the booleans. The Beta distribution becomes a *distribution over distributions*, that is a morphism

$$\mathsf{beta}_{i,j} : 1 \to P(P(2)) \tag{47}$$

This is consistent with the Bayesian intuition for Beta as belief over beliefs. For example, $\mathsf{beta}_{1,1}$ generates a coin p of uniformly random bias. Such a coin may itself be tossed repeatedly using monadic sequencing such as

$$\begin{aligned}
&\mathsf{let}\, p : P(2) \leftarrow \mathsf{beta}_{1,1}\, \mathsf{in} \\
&\mathsf{let}\, x_1 : 2 \leftarrow p\, \mathsf{in} \\
&\mathsf{let}\, x_2 : 2 \leftarrow p\, \mathsf{in} \\
&[(x_1, x_2)]
\end{aligned}$$

Note that the *average coin* produced by $\mathsf{beta}_{1,1}$ is a fair one; in terms of the monadic join

$$\mathsf{join}(\mathsf{beta}_{1,1}) = (\mathsf{let}\, p \leftarrow \mathsf{beta}_{1,1}\, \mathsf{in}\, p) = \mathsf{flip}_{0.5}$$

This layering of effects is typical for hierarchical models [Goodman et al., 2016, Chapter 12]. Note that the abstract type `I` and the Bernoulli distribution have been subsumed completely in the structure of monadic computation. The monadic language module can naturally express the multi-outcome generalizations of Beta-Bernoulli consisting of 'categorical' distributions $\pi \in P(n)$ and Dirichlet distributions $\mathsf{dir}_{i_1,\dots,i_k} \in P(P(n))$.

10.5 Outline

In summary, our main contribution is that the axioms — commutativity, discardability, conjugacy, and finite probability — entirely determine the equational theory of the Beta-Bernoulli process, in the following sense:

- *Model completeness:* Every equation that holds in the measure theoretic interpretation is derivable from our axioms (Theorem 12.10);
- *Syntactical completeness:* Every equation that is not derivable from our axioms is inconsistent with finite discrete probability (Theorem 13.5).

In Section 11, we introduce the language and algebraic theory of Beta-Bernoulli. In Section 12, we give the intended semantics of the theory in measure theory and show that this interpretation is complete. The proof relies on a normalization procedure and the linear independence of Bernstein basis polynomials. In Section 13, we use the model completeness result to derive syntactical completeness. Here we make use of techniques from mathematical analysis to prove the existence of definable distinguishing contexts.

In Section 14, we showcase the general procedure by which algebraic theories present models of synthetic probability. Our language can be seen as a combinatorial characterization of a Markov subcategory of BorelStoch built from the Beta and Bernoulli kernels.

11 An Algebraic Presentation of the Beta-Bernoulli Process

In this section, we present syntactic rules for well-formed client programs for an `AbstractProcess`, and axioms for deriving equations on those programs. We will refer to elements of the abstract type `process`, which may represent urns or Beta samples depending on the implementation, as *parameters*. This is consistent with the terminology of parameterized algebraic theories (Section 3.7).

11.1 An Algebraic Presentation of Finite Probability

We begin by presenting the Bernoulli distribution, that is biased coin flips. Because of the discrete nature of urns, it suffices to treat rational probabilities. We will also write *odds* $(i : j)$ for integers $i + j > 0$ instead of probabilities, as this makes the conjugacy axiom slightly more convenient. The odds $(i : j)$ always represent the ratio $i/(i + j)$.

The coin flip $\texttt{flip}(i : j)$ with odds $(i : j)$ defines a binary operation $+_{i:j}$ on programs by

$$u +_{i:j} v \stackrel{\text{def}}{=} \textbf{if } \texttt{flip}(i : j) \textbf{ then } u \textbf{ else } v$$

Conversely, given the operations $+_{i:j}$, we can recover $\texttt{flip}(i : j) = \textbf{true} +_{i:j} \textbf{false}$. This is the usual correspondence of algebraic operations $(+_{i:j})$ and generic effects $(\texttt{flip}(i : j))$. The algebraic laws for $+_{i:j}$ due to Stone [1949] are analogous to the usual laws for convex sets (see Section 3.6), restricted to natural odds

Definition 11.1 The *theory of rational convexity* is the first-order algebraic theory with binary operations $(+_{i:j})$ for all $i, j \in \mathbb{N}$ such that $i + j > 0$, subject to the axiom schemes

$$(w +_{i:j} x) +_{i+j:k+l} (y +_{k:l} z) = (w +_{i:k} y) +_{i+k:j+l} (x +_{j:l} z)$$
$$x +_{i:j} y = y +_{j:i} x \qquad x +_{i:0} y = x \qquad x +_{i:j} x = x$$

Commutativity of all pairs of operations $(+_{k:l})$ and $(+_{i:j})$ is a derivable equation,

$$(w +_{i:j} x) +_{k:l} (y +_{i:j} z) = (w +_{k:l} y) +_{i:j} (x +_{k:l} z)$$

and so is scaling

$$x +_{ki:kj} y = x +_{i:j} y, \qquad \forall k > 0$$

As usual, commutativity and discardability ($x +_{i:j} x = x$) in this algebraic sense precisely correspond to the program equations (Comm), (Disc). As an aside, the monad $D^{\mathbb{Q}}$ of the theory is given by rational convex combinations, but we won't need this fact in what follows (see also Section 12.3)

$$D^{\mathbb{Q}}(X) = \{p : X \to [0,1] \cap \mathbb{Q} \text{ finitely supported} \mid \sum_{x \in X} p(x) = 1\} \tag{48}$$

11.2 A Parameterized Signature for Beta-Bernoulli

In the signature `AbstractProcess` (Section 10.3), the arguments to `get` are first-class values (parameters) of the abstract type `process`. This type is instantiated to urns in case of `Polya`, and to real probabilities in `BetaBern`. On the other hand, we treat the integer arguments to `new` and `flip` as hyperparameters, which are not first-class. In a more complex hierarchical system with hyperpriors, we might treat them as first-class in turn.

For a full theory of Beta-Bernoulli, we define the algebraic operations

$$\nu_{i,j} p.t \stackrel{\text{def}}{=} \texttt{let } p\texttt{=new}(i,j) \texttt{ in } t \qquad u \,?_p\, v \stackrel{\text{def}}{=} \texttt{if get}(p) \texttt{ then } u \texttt{ else } v$$

There is nothing lost because we can recover the generic effects from their operations as

$$\texttt{new}(i,j) = \nu_{i,j} p.\, p \qquad \texttt{get}(p) = \texttt{true} \,?_p\, \texttt{false}$$

Note that $(?_p)$ is a parameterized binary operation and $\nu_{i,j}$ is a binder. Formally, our syntax requires two kinds of variables: variables x, y ranging over programs (continuations), and now also p, q ranging over parameters. Continuation variables x may themselves expect parameters, and we write their arity as $x : n$. Using the formalism of second-order algebraic theories (Section 3.7), the signature of Beta-Bernoulli consists of the following families of operations

$$(+_{k:l}) : (0 \mid 0,0) \quad (?) : (1 \mid 0,0) \quad \nu_{i,j} : (0 \mid 1) \qquad k + l \geq 1, i, j \geq 1$$

Note that there are no operations combining parameters to new parameters; the type `process` is purely abstract. Formally, this means our theory is parameterized over the theory of equality (Section 3.6). We spell out the term formation in detail

Definition 11.2 The term formation rules for the theory of Beta-Bernoulli are:

$$\frac{-}{\Gamma \mid \Delta, x : m, \Delta' \vdash x(p_1, \cdots, p_m)}\ (p_1 \ldots p_m \in \Gamma) \qquad \frac{\Gamma, p \mid \Delta \vdash t}{\Gamma \mid \Delta \vdash \nu_{i,j} p.t}\ (i,j > 0)$$

$$\frac{\Gamma \mid \Delta \vdash t \quad \Gamma \mid \Delta \vdash u}{\Gamma \mid \Delta \vdash t \,?_p\, u}\ (p \in \Gamma) \qquad \frac{\Gamma \mid \Delta \vdash t \quad \Gamma \mid \Delta \vdash u}{\Gamma \mid \Delta \vdash t +_{i:j} u}\ (i + j > 0)$$

Contexts have two zones; Γ is a parameter context of the form $\Gamma = (p_1, \ldots, p_\ell)$ and Δ is a context of continuation variables the form $\Delta = (x_1 \colon m_1, \ldots, x_k \colon m_k)$. For ease of notation, we often abbreviate $x()$ by x.

Recall the two types of substitution in second-order algebra: terms for parameters, and programs for continuation variables (Section 3.7). Substitution of computations is capture-avoiding, and we work up to α-conversion of bound parameters. For example, substituting $x \,?_p\, y$ for w in $\nu_{1,1}p.w$ yields $\nu_{1,1}q.(x \,?_p\, y)$, while substituting $x \,?_p\, y$ for $z(p)$ in $\nu_{1,1}p.z(p)$ yields $\nu_{1,1}p.(x \,?_p\, y)$. For the sake of a well-defined notion of dimension in 12.6, we disallow the formation of $\nu_{i,0}$ and $\nu_{0,i}$ (see Appendix, Section 32.2).

11.3 Axioms for Beta-Bernoulli

The axioms for the Beta-Bernoulli theory comprise the axioms for rational convexity (Definition 11.1) together with the following axiom schemes.

Commutativity. All the operations commute with each other:

$$p, q \mid w, x, y, z : 0 \vdash (w \,?_q\, x) \,?_p (y \,?_q\, z) = (w \,?_p\, y) \,?_q (x \,?_p\, z) \tag{C1}$$

$$- \mid x \colon 2 \vdash \nu_{i,j}p.\nu_{k,l}q.x(p,q) = \nu_{k,l}q.\nu_{i,j}p.x(p,q) \tag{C2}$$

$$q \mid x, y \colon 1 \vdash \nu_{i,j}p.(x(p) \,?_q\, y(p)) = (\nu_{i,j}p.x(p)) \,?_q (\nu_{i,j}p.y(p)) \tag{C3}$$

$$- \mid x, y : 1 \vdash \nu_{i,j}p.(x(p) +_{k:l} y(p)) = (\nu_{i,j}p.x(p)) +_{k:l} (\nu_{i,j}p.y(p)) \tag{C4}$$

$$p \mid w, x, y, z : 0 \vdash (w +_{i:j} x) \,?_p (y +_{i:j} z) = (w \,?_p\, y) +_{i:j} (x \,?_p\, z) \tag{C5}$$

Discardability. All operations are idempotent:

$$- \mid x \colon 0 \vdash \nu_{i,j}p.x = x \tag{D1}$$

$$p \mid x \colon 0 \vdash x \,?_p\, x = x \tag{D2}$$

Conjugacy.

$$- \mid x, y : 1 \vdash \nu_{i,j}p.(x(p) \,?_p\, y(p)) = (\nu_{i+1,j}p.x(p)) +_{i:j} (\nu_{i,j+1}p.y(p)) \tag{Conj}$$

A theory of equality for terms in context is built, as usual, by closing the axioms under substitution instances, weakening, congruence, reflexivity, symmetry and transitivity. It immediately follows from conjugacy and discardability that $x +_{i:j} y$ is definable for $i, j > 0$, as

$$\nu_{i,j}p.(x \,?_p\, y) = (\nu_{i+1,j}p.x) +_{i:j} (\nu_{i,j+1}p.y) = x +_{i:j} y$$

For a more involved example, consider the operation

$$t(r) = (r \,?_p\, x) \,?_p (y \,?_p\, r)$$

that represents tossing a coin with bias p twice under the Beta-Bernoulli interpretation, continuing with x or y if the results are different, or with r otherwise. In the appendix (Section 32.1), we include a proof that $x +_{1:1} y$ is the unique fixed point of t, i.e. $x +_{1:1} y = t(x +_{1:1} y)$. This is exactly von Neumann's trick [von Neumann, 1951] to simulate a fair coin toss from a biased one.

12 A Complete Interpretation in Measure Theory

In this section we define the intended semantics of terms using measure theory (Section 4.2) and we show that this interpretation is complete (Theorem 12.10). We will make use of the completeness to establish purely syntactical results about the theory in Section 13.

By the Riesz–Markov–Kakutani representation theorem (Section 4.4), there are two equivalent ways to view probabilistic programs: as probability kernels and as linear functionals. We will frequently switch between both views;

Programs as probability kernels: Let I denote the standard Borel space $[0,1]$ and let $\beta_{i,j} \in \mathcal{G}(I)$ denote the Beta distribution with parameters i, j, as given by the density function $\mathsf{pdf}_{i,j}$ as in (46). For contexts of the form $\Gamma = (\mathfrak{p}_1, \dots, \mathfrak{p}_\ell)$ and $\Delta = (x_1 : m_1, \dots, x_k : m_k)$, we let $[\![\Delta]\!] \stackrel{\text{def}}{=} \sum_{i=1}^{k} I^{m_i}$ consist of a copy of I^{m_i} for every variable $x_i : m_i$. We'll for the moment emphasize parameters as fraktur $\mathfrak{p}$ to distinguish them from their interpretations as probabilities $p \in [0,1]$ in the semantics.

Definition 12.1 We interpret terms $\Gamma \mid \Delta \vdash t$ as probability kernels $[\![t]\!] : I^\ell \rightsquigarrow [\![\Delta]\!]$ inductively, for $\vec{p} \in I^\ell$ and $U \subseteq [\![\Delta]\!]$ measurable, as follows[11]

$$\begin{aligned}
[\![x_i(\mathfrak{p}_{j_1}, \dots, \mathfrak{p}_{j_m})]\!](\vec{p}, U) &= [(i, p_{j_1} \dots p_{j_m}) \in U] \\
[\![u +_{i:j} v]\!](\vec{p}, U) &= \tfrac{1}{i+j}\Big(i([\![u]\!](\vec{p}, U)) + j([\![v]\!](\vec{p}, U))\Big) \\
[\![u \,?_{p_j}\, v]\!](\vec{p}, U) &= p_j([\![u]\!](\vec{p}, U)) + (1 - p_j)([\![v]\!](\vec{p}, U)) \\
[\![\nu_{i,j}\mathfrak{q}.t]\!](\vec{p}, U) &= \int_0^1 [\![t]\!]((\vec{p}, q), U)\, \beta_{i,j}(\mathrm{d}q)
\end{aligned}$$

Proposition 12.2 *The interpretation is sound: if $\Gamma \mid \Delta \vdash t = u$ is derivable then $[\![t]\!] = [\![u]\!]$ as probability kernels $[\![\Gamma]\!] \rightsquigarrow [\![\Delta]\!]$.*

PROOF One must check that the axioms are sound under the interpretation. Each of the axioms are elementary facts about probability. For instance, commutativity (C2) amounts to Fubini's theorem, and the conjugacy axiom (Conj) is the well-known conjugate-prior relationship of Beta- and Bernoulli distributions. ∎

We notice that all kernels $[\![t]\!]$ are continuous when seen as maps between Polish spaces $[\![\Gamma]\!] \to \mathcal{G}([\![\Delta]\!])$. We can therefore give the dual analysis in terms of functional analysis (see Section 4.4). To keep things simple, we will use real Banach algebras $\mathcal{C}(X, \mathbb{R})$ instead of complex C*-algebras $\mathcal{C}(X, \mathbb{C})$. We'll make natural use of both measure-theoretic and functional-analytic semantics in our proofs.

Interpretation as functionals: Write $\mathbb{R}^X$ for the commutative Banach algebra of continuous functions $X \to \mathbb{R}$, endowed with the supremum norm. Given a continuous kernel $\kappa : [\![\Gamma]\!] \to [\![\Delta]\!]$, we define linear map $\phi : \mathbb{R}^{[\![\Delta]\!]} \to \mathbb{R}^{[\![\Gamma]\!]}$ by considering κ as an integration operator

$$\phi(f)(\vec{p}) = \int f(r)\kappa(\vec{p}, \mathrm{d}r)$$

[11] recall Iverson bracket notation, i.e. $[\phi] = 1$ if ϕ is true, and $[\phi] = 0$ otherwise

The resulting map ϕ is a positive unital linear map. Recall that a map ϕ is positive if $f \geq 0$ implies $\phi(f) \geq 0$, and unital if $\phi(1) = 1$.

It is informative to spell out the interpretation of terms $\mathfrak{p}_1, \ldots, \mathfrak{p}_\ell \mid x_1 : m_1, \ldots, x_k : m_k \vdash t$ as maps $[\![t]\!] : \mathbb{R}^{I^{m_1}} \times \ldots \times \mathbb{R}^{I^{m_k}} \to \mathbb{R}^{I^\ell}$ directly. This semantics fits the continuation passing style of the term language: we may think of computation variables $x : m$ as ranging over functions in $\mathbb{R}^{I^m}$.

Definition 12.3 The functional interpretation is inductively given by

$$\begin{aligned}
[\![x_i(\mathfrak{p}_{j_1}, \ldots, \mathfrak{p}_{j_m})]\!](\vec{f})(\vec{p}) &= f_i(p_{j_1}, \ldots, p_{j_m}) \\
[\![u +_{i:j} v]\!](\vec{f})(\vec{p}) &= \tfrac{1}{i+j}\Big(i([\![u]\!](\vec{f})(\vec{p})) + j([\![v]\!](\vec{f})(\vec{p}))\Big) \\
[\![u \,?_{p_j}\, v]\!](\vec{f})(\vec{p}) &= p_j([\![u]\!](\vec{f})(\vec{p})) + (1 - p_j)([\![v]\!](\vec{f})(\vec{p})) \\
[\![\nu_{i,j}\mathfrak{q}.t]\!](\vec{f})(\vec{p}) &= \int_0^1 [\![t]\!](\vec{f})(\vec{p}, q)\, \beta_{i,j}(\mathrm{d}q)
\end{aligned}$$

For example, $[\![- \mid x, y : 0 \vdash x +_{1:1} y]\!] : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the function $(x, y) \mapsto \frac{1}{2}(x + y)$, and $[\![- \mid x : 1 \vdash \nu_{1,1}\mathfrak{p}.x(p)]\!] : \mathbb{R}^I \to \mathbb{R}$ is the integration functional, $f \mapsto \int_0^1 f(p)\mathrm{d}p$. The analogous soundness property to Proposition 12.2 holds for the functional interpretation. We use the same brackets $[\![-]\!]$ for both interpretations, as the intended semantics will be clear from context.

12.1 Background on Bernstein polynomials

We review well-known facts about Bernstein polynomials, which enjoy various applications from probability theory to computer graphics [Farouki, 2012]. In our work, they feature in two related semantical roles, as decision trees (Section 12.4) and the density functions of the Beta distribution.

Definition 12.4 (Bernstein polynomials) For $i = 0, \ldots, k$, we define the i-th *Bernstein basis polynomial* $b_{i,k}$ *of degree* k as[12]

$$b_{i,k}(p) \stackrel{\text{def}}{=} \binom{k}{i} p^{k-i}(1 - p)^i$$

For a multi-index $I = (i_1, \ldots, i_\ell)$ with $0 \leq i_j \leq k$, we let $b_{I,k}(\vec{p}) = b_{i_1,k}(p_1) \cdots b_{i_\ell,k}(p_\ell)$. A *Bernstein polynomial* refers to any polynomial expanded in the Bernstein basis.

The family $\{b_{i,k} : i = 0, \ldots, k\}$ forms a basis of the polynomials of maximum degree k and also a partition of unity, i.e. $\sum_{i=0}^k b_{i,k} = 1$. Every Bernstein basis polynomial of degree k can be expressed as a nonnegative rational linear combination of degree $k + 1$ basis polynomials.

The density function of the distribution $\beta_{i,j}$ on $[0, 1]$ for $i, j > 0$ is proportional to a Bernstein basis polynomial of degree $i + j - 2$. We can conclude that the family of distributions $\{\beta_{i,j} : i, j > 0, i + j = n\}$ is linearly independent for every n.

For multi-indices I, the polynomials $\{b_{I,k}\}$ are linearly independent for fixed k. Equivalently, products of beta distributions β_{i_r,j_r} are linearly independent as long as $i_r + j_r = k$

[12] we use here the mirrored version of the common definition, where i is replaced with $k - i$

holds for some common k. This will be the crucial ingredient for establishing the uniqueness of normal forms for Beta-Bernoulli terms.

Pedantic point: When we say that a family of distributions (μ_i) on a measurable space (X, Σ_X) is linearly independent, we must formally interpret this statement in a vector space, like the space of all functions $\Sigma_X \to \mathbb{R}$ (this is because measures don't form a vector space). The relevant point however is that coefficients of linear combinations are uniquely determined, that is if $\sum_i w_i \mu_i = \sum_i w'_i \mu_i$ then $w_i = w'_i$ for all i.

12.2 Normal Forms and Completeness

For the completeness proof of the measure-theoretic model, we proceed as follows: To decide $\Gamma \mid \Delta \vdash t = u$ for two terms t, u, we transform them into a common normal form whose interpretations can be given explicitly. We then use a series of linear independence results to show that if the interpretations agree, the normal forms must be syntactically equal. Normalization happens in three stages:

- If we think of a term as a syntax tree of binary choices and ν-binders, we use the conjugacy axiom to push all occurrences of ν towards the leaves of the tree.
- We use commutativity and discardability to stratify the use of free parameters $?_p$.
- The leaves of the tree will now consist of chains of ν-binders, variables and fixed choices $+_{i:j}$. Those can be collected into a canonical form.

We will describe these normalization stages in reverse order because of their increasing complexity.

12.3 Stone's Normal Form for Rational Convex Sets

Normal forms for the theory of rational convex sets have been described by Stone [1949]. We note that if $- \mid x_1 \dots x_k : 0 \vdash t$ is a term in the theory of rational convex sets (Def. 11.1) then its semantics (Definition 12.3) $[\![t]\!] : \mathbb{R}^k \to \mathbb{R}$ is a unital positive linear map that takes rationals to rationals. From the perspective of measures, this corresponds to a categorical distribution with k categories.

Proposition 12.5 (Stone) *The interpretation of Definition 12.3 exhibits a bijective correspondence between terms*

$$- \mid x_1 \dots x_k : 0 \vdash t$$

built from $+_{i:j}$, modulo equations, and unital positive linear maps $\mathbb{R}^k \to \mathbb{R}$ that take rationals to rationals.

Example 12.6 The map $\phi(x, y, z) = \frac{1}{10}(2x + 3y + 5z)$ is unital positive linear, and arises from the term $t \stackrel{\text{def}}{=} x +_{2:8} (y +_{3:5} z)$. This is the only term that gives rise to the ϕ, modulo equations.

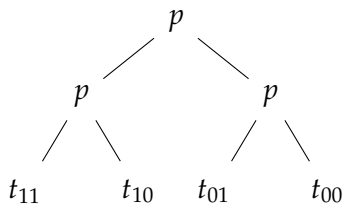
In brief, one can recover t from ϕ by looking at $\phi(1, 0, 0) = \frac{2}{10}$, then $\phi(0, 1, 0) = \frac{3}{10}$, then $\phi(0, 0, 1) = \frac{5}{10}$. We will write $\left(\bigoplus \begin{matrix} x_1 & \dots & x_k \\ w_1 & \dots & w_k \end{matrix}\right)$ for the term corresponding to the linear

map $(x_1 \dots x_k) \mapsto \frac{1}{\sum_{i=1}^k w_k}(w_1 x_1 + \cdots + w_k x_k)$. These are normal forms for the theory of rational convex sets. This normal form is dual to the rational probability monad $D^{\mathbb{Q}}$ from (48), and expressed in terms of odds w_i rather than probabilities.

12.4 Normalization of ν-free Terms

This section concerns the normalization of terms using free parameters but no ν. Consider a single parameter p. If we think of a term t as a syntactic tree, commutativity can be used to move all occurrences of $?_p$ to the root of the tree. Also by discardability, we can expand the tree to become a full binary tree of some depth k, which we call a *tree diagram*. Let us label the 2^k leaves with $t_{a_1 \cdots a_k}$, $a_i \in \{0, 1\}$ as follows

$$(t_{11} ?_p t_{10}) ?_p (t_{01} ?_p t_{00}) \qquad \approx$$



As a programming language expression, this corresponds making k successive choices

$$\texttt{let } a_1\texttt{=get(}p\texttt{) in ... let } a_k\texttt{=get(}p\texttt{) in match } (a_1,\dots,a_k) \Rightarrow t_{a_1 \cdots a_k}$$

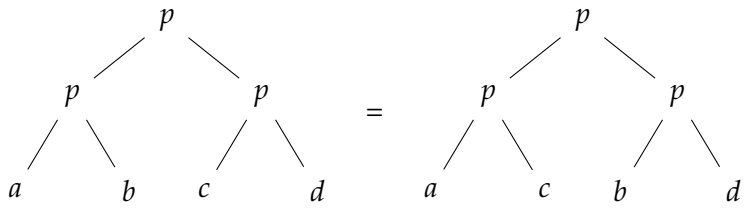
By commutativity (C1), we may reorder these choices. That is, the permutation group S_k acts on tree diagrams without changing their meaning by permuting the leaves according to the rule

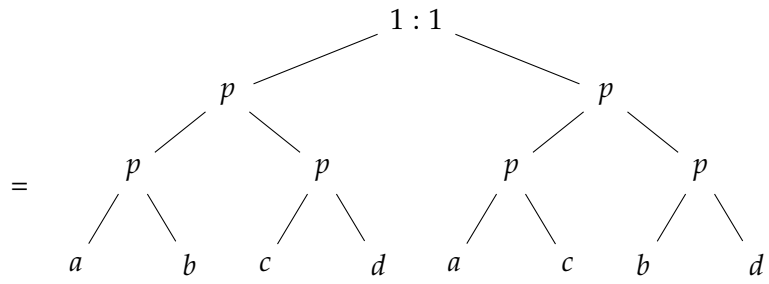
$$(\sigma \in S_k) : t_{a_1 \cdots a_k} \mapsto t_{a_{\sigma(1)} \cdots a_{\sigma(k)}}$$

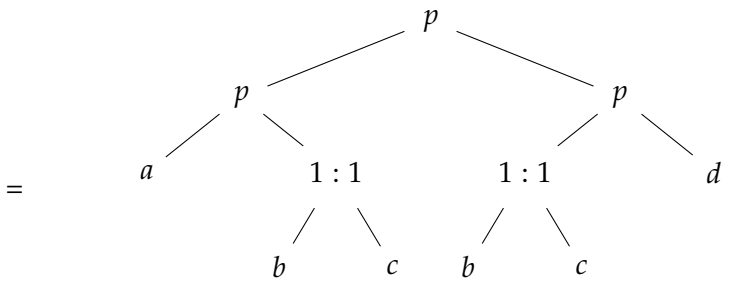
It is a standard trick in representation theory to obtain an invariant element by averaging over an orbit. Since rational choice is idempotent, we can indeed replace the tree diagram t by the average over all its permutations. The average commutes down to the leaves by (C5), so we obtain a tree diagram with leaves

$$m_{a_1 \cdots a_k} = \frac{1}{k!} \sum_{\sigma \in S_k} t_{a_{\sigma(1)} \cdots a_{\sigma(k)}}$$

where the average is to be read as a rational choice with all weights 1. This new tree diagram is derivably equal to t, and by construction invariant under permutation of levels in the tree, in particular $m_{a_1 \cdots a_k}$ only depends on the sum $a_1 + \cdots + a_k$. That is to say, the counts are a sufficient statistic. For example







We write $C_k^p(t_0, \ldots, t_k)$ for the unique permutation invariant tree diagram of k successive p-choices with leaves $t_{a_1 \cdots a_k} = t_{a_1 + \cdots + a_k}$, then the example shows

$$(a \,?_p\, b) \,?_p (c \,?_p\, d) = C_2^p(a, b +_{1:1} c, d)$$

We can give the following normalization procedure for terms $p_1 \ldots p_\ell \mid x_1 \ldots x_n : 0 \vdash t$: Bring t into the form $C_k^{p_1}(t_0, \ldots, t_k)$ where each t_i is p_1-free. Then recursively normalize each t_i in the same way, collecting the next parameter. By discardability, we can choose the height of all these tree diagrams to be a single maximum k, such that the resulting term is a nested structure of tree-diagrams $C_k^{p_j}$. We will use multi-indices $I = (i_1, \ldots, i_\ell)$ to write the whole stratified term as $C_k((t_I))$ where each leaf t_I *only contains rational choices* and no more use of the parameters p_i. The interpretation of such a term can now be written down explicitly in terms of Bernstein basis polynomials as

$$[\![C_k((t_I))]\!](\vec{x})(\vec{p}) = \sum_I b_{I,k}(\vec{p}) \cdot [\![t_I]\!](\vec{x}) \tag{49}$$

We can now give the following completeness result, stating that the stratified tree diagrams $C_k((t_I))$ are normal forms.

Proposition 12.7 *There is a bijective correspondence between equivalence classes of ν-free terms*

$$p_1 \dots p_\ell \mid x_1 \dots x_n : 0 \vdash t$$

and linear unital maps $\phi : \mathbb{R}^n \to \mathbb{R}^{I^\ell}$ such that for every standard basis vector e_j of $\mathbb{R}^n$, $\phi(e_j)$ is a multivariate Bernstein polynomial with nonnegative rational coefficients.

PROOF Clearly every $[\![C_k((t_I))]\!]$ is of the desired form. Given ϕ, we can assume that all basis polynomial have the same, large-enough degree k. If

$$\phi(e_j) = \sum_I w_{I,j} b_{I,k} \tag{50}$$

then the unitality condition $\phi(1, \dots, 1) = 1$ means $\sum_I \left(\sum_j w_{I,j}\right) b_{I,k} = 1$, and hence by the linear independence of Bernstein basis polynomials and partition of unity, we have $\sum_j w_{I,j} = 1$ for every I. Thus if by Proposition 12.5, we can take t_I to be the rational convex combination with interpretation

$$[\![t_I]\!](\vec{x}) = \sum_j w_{I,j} x_j$$

and recover $[\![C_k((t_I))]\!] = \phi$. Again by linear independence, the weights $w_{I,j}$ in (50) are uniquely defined by ϕ. In particular, two normal forms $C_k((t_I))$, $C_k((t'_I))$ are have the same interpretation if and only if all $t_I = t'_I$ are derivably equal. ∎

Geometric characterizations for the assumption of this theorem exist in [Powers and Reznick, 2000; Alves Diniz et al., 2016]. For example, a univariate polynomial is a Bernstein polynomial with nonnegative coefficients if and only if it is positive on $(0, 1)$. More care is required in the multivariate case.

12.5 Normalization of Full Terms

For arbitrary terms $p_1 \dots p_\ell \mid x_1 : m_1, \dots, x_s : m_s \vdash t$, we employ the following normalization procedure.

(i) Using conjugacy and the commutativity axioms (C2–C4), we can push all uses of ν towards the leaves of the syntax tree, until we end up with a tree of ratios and free parameter choices only.

(ii) Using the conjugacy and discardability, we can freely increase the sum $i + j$ of the indices in any binder $\nu_{i,j}$ appearing in t, as

$$\nu_{i,j} p.x(p) = \nu_{i,j} p.(x(p) \,?_p\, x(p)) = (\nu_{i+1,j} p.x(p)) +_{i:j} (\nu_{i,j+1} p.x(p))$$

We do this until every instance of $\nu_{i,j}$ in our term satisfies $i + j = k$ for some large enough k.

(iii) We now use the symmetrization procedure from Section 12.4 to stratify the use of free parameters. The resulting term can be written as $C_s((t_I))$ where the leaves t_I consist of only rational choices and ν's.

We call a term without any occurrence of $(+_{i:j})$ and $(?_p)$ a *chain*, because it must be of the form

$$\nu_{i_1,j_1} p_{\ell+1} \ldots \nu_{i_d,j_d} p_{\ell+d}.x_j(p_{\tau(1)}, \ldots, p_{\tau(m)}) \text{ for some } \tau : m_j \to \ell + d \tag{51}$$

By discardability, we can assume that the chains are minimal, i.e. every bound parameter $p_{\ell+i}$ for $i = 1, \ldots, d$ gets used. We consider two chains equal if they are α-convertible into each other, meaning τ differs by a permutation of $\{\ell+1, \ldots, \ell+d\}$. The geometric meaning of these chains will be discussed in Section 12.6.

For now, it sufficed to note that after our normalization procedure, each t_I is a convex combination of chains. Furthermore, by Proposition 12.5, if $c_1, \ldots, c_m$ is a list of the distinct chains that occur in any of the leaves, we can give the leaves t_I the uniform shape

$$t_I = \left(\bigoplus \begin{matrix} c_1 & \ldots & c_m \\ w_{I1} & \ldots & w_{Im} \end{matrix} \right) \text{ for appropriate weights } w_{Ij} \in \mathbb{N}$$

We will show that this representation is a unique normal form for $(?_p)$-free terms.

12.6 Proof of Completeness

Let $x : m$. We shall develop a geometric intuition for a chain

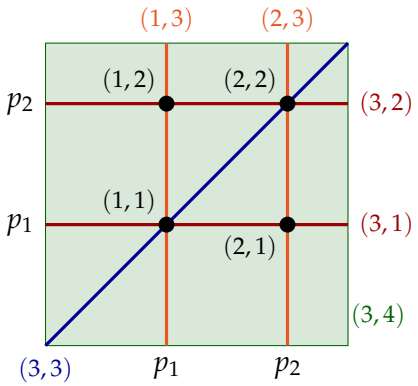
$$c = \nu_{i_1,j_1} p_{\ell+1} \ldots \nu_{i_d,j_d} p_{\ell+d}.\, x(p_{\tau(1)}, \ldots, p_{\tau(m)})$$

The measure-theoretic interpretation of c is a probability kernel $[\![c]\!] : I^\ell \to \mathcal{G}(I^m)$ where $I = [0,1]$. If we fix the free parameters $\vec{p} \in \mathbb{R}^\ell$, the measure $[\![c]\!](\vec{p})$ is a pushforward of d Beta-distributions to a hyperplane segment in I^m along the inclusion map

$$h_{\tau,\vec{p}} : I^d \to I^m, h_\tau(p_{\ell+1}, \ldots, p_{\ell+d}) = (p_{\tau(1)}, \ldots, p_{\tau(m)})$$

By the minimality assumption on the chain, $h_{\tau,\vec{p}}$ is injective and an isomorphism with its image hyperplane segment $H_{\tau,\vec{p}}$, which is exactly d-dimensional. Hence we call d the *dimension of the chain* and $H_{\tau,\vec{p}}$ the *support* of the distribution $[\![c]\!](\vec{p})$. The position of $H_{\tau,\vec{p}}$ gets translated by the coordinates of $\vec{p}$. Because all this geometric information is captured in the map τ, we call τ the *support type* of the chain.

For example, each chain with two free parameters p_1, p_2 and a variable $x : 2$ gives rise to a parameterized distribution on the unit square. On the right, we illustrate the ten possible supports that such distributions can have, as subspaces of the square. In the graphic we write (i, j) for $\nu p_3.\nu p_4.x(p_i, p_j)$, momentarily omitting the subscripts of ν because they do not affect the support. For instance, the upper horizontal line corresponds to $\nu p_3.x(p_3, p_2)$; the bottom-right dot corresponds to $x(p_2, p_1)$; the diagonal corresponds to $\nu p_3.x(p_3, p_3)$; and the entire square corresponds to $\nu p_3.\nu p_4.x(p_3, p_4)$. There are four subspaces of dimension $d = 0$, five with $d = 1$, and one with $d = 2$. Notice that all subspaces are all distinct as long as $p_1 \neq p_2$.



The core of our normalization argument is that chains are linearly independent, i.e. mixtures of chains can be uniquely decomposed into their parts. For chains on the same subspace, this restricts to the usual independence of products of Beta distributions, and chains on different supports are too different to interfere.

Proposition 12.8 *Let $c_1, \ldots, c_n$ be distinct chains*

$$p_1 \ldots p_\ell \mid x_1 : m_1, \ldots, x_s : m_s \vdash c_i$$

where all indices of ν_{i_r,j_r} add up to $i_r + j_r = k$. Then the family of measures

$$\{[\![c_i]\!](\vec{p}) \in \mathcal{G}(\mathbb{R}^{m_1} + \ldots + \mathbb{R}^{m_s})\}_{i=1,\ldots,n}$$

is linearly independent whenever all parameters p_i are distinct.

PROOF Fix $\vec{p} \in I^\ell$. Measures on different copies of $\mathcal{I}^{m_i}$ are clearly independent, so we can restrict ourselves to a single variable $x : m$.

Let the chain c_i have dimension d_i and support type τ_i. Its denotation is the measure $[\![c_i]\!](\vec{p}) = (h_i)_* \mu_i$ on I^m where μ_i is a product of d_i beta distributions, and $h_i : I^{d_i} \to I^m$ is the injective affine map $h_i(p_{\ell+1}, \ldots, p_{\ell+d}) = (p_{\tau_i(1)}, \ldots, p_{\tau_i(m)})$. Now consider a vanishing linear combination

$$\psi \stackrel{\text{def}}{=} \sum_i a_i [\![c_i]\!](\vec{p}) \stackrel{!}{=} 0 \tag{52}$$

We show by induction over a dimension d that all a_i must vanish: Our inductive hypothesis is that $a_i = 0$ whenever $d_i < d$. The base case $d = 0$ is empty, so now consider a chain c_j of dimension d. Let $A \subseteq I^d$ be an arbitrary Borel set, then the image $h_j(A) \subseteq I^m$ is Borel because h_j is a measurable injection (e.g. [Kechris, 1987, §15A]). We now analyze the summands of the following expression (corresponding to a restriction of ψ along h_j)

$$\psi(h_j(A)) = \sum_i a_i \mu_i(h_i^{-1}(h_j(A)))$$

- If c_i has dimension $d_i < d$, then $a_i = 0$ by the inductive hypothesis, so the summand vanishes
- If c_i has dimension $d_i > d$, the set $h_i^{-1}(h_j(A))$ has at most dimension d. It is therefore a nullset for μ_i, and the summand vanishes.
- If c_i has dimension d but a different type, and all $p_1, \ldots, p_\ell$ are assumed distinct, then the supports H_i and H_j are not identical. Therefore their intersection is at most $(d-1)$-dimensional and $h_i^{-1}(h_j(A))$ is a nullset for μ_i. The summand vanishes
- If c_i has support type τ_j then $h_i = h_j$, so $h_i^{-1}(h_j(A)) = A$.

By assumption (52), we thus obtain the equation

$$\sum_{c_i \text{ has support type } \tau_j} a_i \mu_i(A) = 0$$

Because all μ_i are multivariable Beta distributions with $i_r + j_r = k$ on I^d belonging to distinct chains, they are linearly independent (Section 12.1). Therefore $a_i = 0$ for all c_i with subspace type τ_j, in particular $a_j = 0$ as desired. ∎

Corollary 12.9 *Under the same assumptions as Proposition 12.8, the functionals*

$$\{[\![c_i]\!](-)(\vec{p}) : \mathbb{R}^{I^{m_1}} \times \ldots \times \mathbb{R}^{I^{m_s}} \to \mathbb{R}\}_{i=1,\ldots,n}$$

are linearly independent whenever all coordinates p_i are distinct.

PROOF From the linear isomorphism between measures and integration functionals (Section 4.4). ∎

Theorem 12.10 (Completeness) *If $\Gamma \mid \Delta \vdash t, t'$ and $[\![t]\!] = [\![t']\!]$, then $\Gamma \mid \Delta \vdash t = t'$.*

PROOF From the normalization procedure, we find numbers k, n, a list of distinct chains $c_1, \ldots, c_s$ with $i + j = n$ and weights $(w_{I,j}), (w'_{I,j})$ such that $\Gamma \mid \Delta \vdash t = C_k((t_I))$ and $\Gamma \mid \Delta \vdash t' = C_k((t'_I))$ where $t_I = \left(\oplus \begin{matrix} c_1 & \ldots & c_s \\ w_{I,1} & \ldots & w_{I,s} \end{matrix}\right)$ and $t'_I = \left(\oplus \begin{matrix} c_1 & \ldots & c_s \\ w'_{I,1} & \ldots & w'_{I,s} \end{matrix}\right)$. The interpretations of these normal forms are given explicitly by

$$[\![t]\!](\vec{f})(\vec{p}) = \sum_{I,j} \frac{w_{I,j}}{w_I} \cdot b_{I,k}(\vec{p}) \cdot [\![c_j]\!](\vec{f})(\vec{p}) \text{ where } w_I = \sum_j w_{I,j}$$

and analogously for t'. Then $[\![t]\!] = [\![t']\!]$ implies that for all $\vec{f}$

$$\sum_j \left(\sum_I \left(\frac{w_{I,j}}{w_I} - \frac{w'_{I,j}}{w'_I} \right) b_{I,k}(\vec{p}) \right) [\![c_j]\!](\vec{f})(\vec{p}) = 0.$$

By Corollary 12.9, this implies

$$\sum_I \left(\frac{w_{I,j}}{w_I} - \frac{w'_{I,j}}{w'_I} \right) b_{I,k}(\vec{p}) = 0 \tag{53}$$

for every j whenever the parameters p_i are distinct. However, since the left hand side of (53) is continuous and the condition of distinct p_i is dense, the equation must in fact hold for *all* $\vec{p} \in I^\ell$. By linear independence of the Bernstein basis polynomials, we obtain $w_{I,j}/w_I = w'_{I,j}/w'_I$ for all I, j. Thus, all weights agree up to rescaling and we can conclude $\Gamma \mid \Delta \vdash t = t'$. ∎

13 Extensionality and Syntactical Completeness

In this section we use the model completeness of the previous section to establish some syntactical results about the theory of Beta-Bernoulli. Although the model is helpful in informing the proofs, the statements of the results in this section are purely syntactical.

The ultimate result of this section is equational syntactical completeness (Theorem 13.5), which says that there can be no further equations in the theory without it becoming inconsistent with discrete probability. In other words, assuming that the axioms we have included are appropriate, they must be sufficient, regardless of any discussion about semantic models or intended meaning. This kind of result is sometimes called 'Post completeness' after Post proved a similar result for propositional logic.

The key steps towards this result are two extensionality results. These are related to the programming language idea of 'contextual equivalence' (Section 3) as follows: Recall that in a programming language, we often define a basic notion of equivalence from operational considerations on closed ground terms: these are programs with no free variables that return (say) booleans. We extend this to contextual equivalence on open terms by saying that $t \approx u$ if, for all closed ground contexts $\mathcal{C}$ we have $\mathcal{C}[t] = \mathcal{C}[u]$.

In our algebraic framework, our notion of equality too induces a basic notion of equivalence on closed ground terms; those are terms $- \mid x : 0, y : 0 \vdash u$ which correspond to closed programs of boolean type, for they have two possible continuations x and y corresponding to **`true`** and **`false`**. The extensionality results of this section say that, assuming one is content with this basic notion of equivalence, the equations that we axiomatize coincide with contextual equivalence.

13.1 Extensionality for Closed Terms

We first show that when considering equality of terms, we may eliminate free parameters p by closing them off using $\nu_{i,j} p.(-)$ for all possible values of i, j.

Proposition 13.1 (Extensionality for closed terms) *Suppose $\Gamma, q \mid \Delta \vdash t$ and $\Gamma, q \mid \Delta \vdash u$. If $\Gamma \mid \Delta \vdash \nu_{i,j} q.t = \nu_{i,j} q.u$ for all i, j, then also $\Gamma \mid \Delta \vdash t = u$.*

PROOF We show the contrapositive; by the model completeness theorem (Theorem 12.10), we can reason in the model rather than syntactically. The idea is that if two continuous functions have the same integral with respect to all Beta distributions, they must be equal.

So we consider t and u such that $[\![t]\!] \neq [\![u]\!]$ as functions $\mathbb{R}^{I^{m_1}} \times \mathbb{R}^{I^{m_k}} \to \mathbb{R}^{I^{l+1}}$, and show that there exist i, j such that $[\![\nu_{i,j} q.t]\!] \neq [\![\nu_{i,j} q.u]\!]$. By assumption there are $\vec{f}$ and $\vec{p}, q$ such that $[\![t]\!](\vec{f})(\vec{p}, q) \neq [\![u]\!](\vec{f})(\vec{p}, q)$ as real numbers.

We can find increasing sequences (i_n), (j_n) of natural numbers such that $\frac{i_n}{i_n + j_n} \to q$ as $n \to \infty$. The sequence of Beta distributions β_{i_n, j_n} satisfies the assumptions of Lemma 13.2 and therefore converges weakly to the Dirac distribution δ_q. This means for any continuous $h : I \to \mathbb{R}$, the integral $\int h(r) \beta_{i_n, j_n}(\mathrm{d}r)$ converges to $h(q)$ as $n \to \infty$. In particular, $\int \left([\![t]\!](\vec{f})(\vec{p}, r) - [\![u]\!](\vec{f})(\vec{p}, r) \right) \beta_{i_n, j_n}(\mathrm{d}r)$ must become non-zero for sufficiently large n. Therefore, there exists an n such that $\int [\![t]\!](\vec{f})(\vec{p}, r) \, \beta_{i_n, j_n}(\mathrm{d}r) \neq \int [\![u]\!](\vec{f})(\vec{p}, r) \, \beta_{i_n, j_n}(\mathrm{d}r)$. So $[\![\nu_{i_n, j_n} q.t]\!] \neq [\![\nu_{i_n, j_n} q.u]\!]$. ∎

In the proof, we make use of the following lemma about convergence to a Dirac distribution. We won't need convergence of random variables in the rest of our work, but refer to [Kallenberg, 1997, Chapter 3] for reference.

Lemma 13.2 *Let X_n be real-valued random variables with expectations μ_n such that $\mu_n \to c$ and $\mathsf{Var}(X_n) \to 0$ for $n \to \infty$. Then the X_n converge in distribution to c.*

PROOF We show the stronger statement that X_n converges to c in probability [Kallenberg, 1997, 3.7]: For all $\epsilon > 0$, Chebyshev's inequality implies that

$$\begin{aligned} \Pr(|X_n - c| > \epsilon) &\leq \Pr(|X_n - \mu_n| + |\mu_n - c| > \epsilon) \\ &= \Pr(|X_n - \mu_n| > \epsilon - |\mu_n - c|) \\ &\leq \mathsf{Var}(X_n) \cdot (\epsilon - |\mu_n - c|)^{-2} \end{aligned}$$

for all sufficiently large n such that $\epsilon - |\mu_n - c| > 0$. By the assumptions, the last bound vanishes for $n \to \infty$. ∎

13.2 Extensionality for Ground Terms

Next we show that we can eliminate continuation variables $x_1, \ldots, x_k$ by substituting them with ground (boolean) expressions $v_1, \ldots, v_k$: If $\vdash t[^{v_1 \ldots v_k}/_{x_1 \ldots x_k}] = u[^{v_1 \ldots v_k}/_{x_1 \ldots x_k}]$ for all suitable ground $v_1 \ldots v_k$, then $\vdash t = u$. From a programming perspective, this says that for closed t, u, if $\mathcal{C}[t] = \mathcal{C}[u]$ for all boolean contexts $\mathcal{C}$, then $t = u$.

Proposition 13.3 (Extensionality for ground terms) *Consider closed terms*

$$- \mid x_1 \colon m_1 \ldots x_k \colon m_k \vdash t, u$$

Suppose that whenever $v_1 \ldots v_k$ are terms with

$$(p_1 \ldots p_{m_1} \mid y, z : 0 \vdash v_1), \ldots, (p_1 \ldots p_{m_k} \mid y, z : 0 \vdash v_k)$$

we have $- \mid y, z : 0 \vdash t[^{v_1 \ldots v_k}/_{x_1 \ldots x_k}] = u[^{v_1 \ldots v_k}/_{x_1 \ldots x_k}]$. Then we also have $- \mid x_1 \colon m_1 \ldots x_k \colon m_k \vdash t = u$.

PROOF Again, we show the contrapositive. Let $\Delta = (x_1 \colon m_1 \ldots x_k \colon m_k)$. Suppose we have t and u such that $\neg(- \mid \Delta \vdash t = u)$. Then by the model completeness theorem (Theorem 12.10), we have $[\![t]\!] \neq [\![u]\!]$ as linear functions $\mathbb{R}^{I^{m_1}} \times \cdots \times \mathbb{R}^{I^{m_k}} \to \mathbb{R}$. Since the functions are linear, there is an index $i \leq k$ and a continuous function $f : I^{m_i} \to \mathbb{R}$ with $[\![t]\!](0 \ldots 0, f, 0 \ldots 0) \neq [\![u]\!](0 \ldots 0, f, 0 \ldots 0)$. By the Stone-Weierstraß theorem, every such f is a limit of polynomials, and so since $[\![t]\!]$ and $[\![u]\!]$ are continuous and linear, there has to be a Bernstein basis polynomial $b_{I,k} : \mathbb{R}^{m_i} \to \mathbb{R}$ that already distinguishes them. This function is definable, i.e. there is a a term $p_1, \ldots, p_{m_i} \mid y, z : 0 \vdash w$ with $[\![w]\!](1, 0) = b_{I,k}$. Define terms $v_j = w$ for $i = j$ and $v_j = z$ for $i \neq j$. Then

$$[\![t[^{v_1 \ldots v_k}/_{x_1 \ldots x_k}]]\!](1, 0) = [\![t]\!](0, \ldots, b_{I,k}, \ldots, 0) \neq [\![u]\!](0, \ldots, b_{I,k}, \ldots, 0) = [\![u[^{v_1 \ldots v_k}/_{x_1 \ldots x_k}]]\!](1, 0).$$

Then $\neg\big(- \mid y, z : 0 \vdash t[^{v_1 \ldots v_k}/_{x_1 \ldots x_k}] = u[^{v_1 \ldots v_k}/_{x_1 \ldots x_k}]\big)$ follows from the model soundness property (Proposition 12.2). ∎

Note that in programming terms, we have invoked the Stone-Weierstraß theorem to obtain a definable boolean context which distinguishes t and u.

13.3 Relative Syntactical Completeness

We can now prove our syntactical completeness result; that is, no equations can be added to our theory without creating a contradiction with rational probability. Rational probability admits a similar theorem, which we discuss first; we can then use our extensionality results to reduce Beta-Bernoulli to that case.

Proposition 13.4 (Neumann [1970]) *If t, u are terms in the theory of rational convexity (Definition 11.1), then either $t = u$ is derivable or it implies $x +_{i:j} y = x +_{i':j'} y$ for all nonzero i, i', j, j'.*

As an aside, we shall give the following interpretation of this result by borrowing some standard terminology from universal algebra: Fix an algebraic signature such as the signature $(+_{i:j})$ of rational convexity. A *theory* $\mathcal{T}$ is a set of formal equations $(t = u)$ between terms which is *deductively closed* under the rules of equational logic. The largest theory is the *inconsistent theory* $\mathcal{T}_{\mathsf{Max}}$ which equates all terms.

In Proposition 5.4, we have seen that we can collapse probability to possibility by replacing all operations $(+_p)$ for $0 < p < 1$ by a semilattice operation $(\vee)$. The same holds for rational convexity once we posit

$$x +_{i:j} y = x +_{i':j'} y \text{ for all } i, i', j, j' > 0 \tag{54}$$

Let $\mathcal{T}_{\mathsf{Cvx}}$ denote the theory of rational convexity, which is generated by the axioms of Definition 11.1, and let $\mathcal{T}_{\mathsf{SL}}$ denote the theory generated by convexity and (54). Then Proposition 13.4 states that the inclusion $\mathcal{T}_{\mathsf{Cvx}} \subset \mathcal{T}_{\mathsf{SL}}$ is an immediate succession in the poset of theories. There exists no theory strictly between the two; adding *any* equation to $\mathcal{T}_{\mathsf{Cvx}}$ results in (54). In fact, it is folklore that $\mathcal{T}_{\mathsf{Cvx}} \subset \mathcal{T}_{\mathsf{SL}} \subset \mathcal{T}_{\mathsf{Max}}$ is an immediate succession: The theory $\mathcal{T}_{\mathsf{SL}}$ is therefore called syntactically complete or maximally consistent.

A consequence of syntactic completeness is that every nontrivial model of the theory is complete. If a model validates an equation $(t = u)$ which is not derivable, then that model must be inconsistent. This result must be relativized for rational probability: Every model is either complete, or a semilattice or inconsistent.

We show that the theory of Beta-Bernoulli is syntactically complete in the following sense:

Theorem 13.5 *The theory of Beta-Bernoulli is syntactically complete relative to the theory of rational convexity: For all terms t and u, either the equation* $(t = u)$ *is derivable, or it implies* (54).

PROOF If $(t = u)$ is *not derivable*, we use Propositions 13.1 and 13.3 to find a closed ground substitution instances $- \mid x : 0, y : 0 \vdash \tilde{t}, \tilde{u}$ such that $(\tilde{t} = \tilde{u})$ is still not derivable. The normalization procedure reduces $\tilde{t}$ and $\tilde{u}$ to terms in the language of rational convexity only. We apply Proposition 13.4 to derive (54). ∎

This shows that every model of Beta-Bernoulli which does not collapse rational probability is automatically complete for the entire theory.

Example 13.6 Consider the hypothetical equation

$$\nu_{1,1} p.x(p,p) = \nu_{1,1} p.\nu_{1,1} q.x(p,q)$$

expressing the statement that $\nu_{1,1}$ is deterministic (Section 6.3). This is obviously not derivable, because it is incorrect measure-theoretically. Syntactically, this fact can be witnessed by the ground substitution $x(p,q) \stackrel{\text{def}}{=} (y \,?_q\, z) \,?_p\, z$. Normalizing the result yields the equation $y +_{1:2} z = y +_{1:3} z$, which collapses rational probability (the normalization procedure is showcased in Section 32.1). In programming syntax, the example equation would be written

```
(let p = new(1,1) in (p,p)) = (new(1,1), new(1,1))
```

and the distinguishing context is

```
C[−] = let (p,q)=(−) in if get(p) then get(q) else false
```

That is to say, the closed ground programs `C[LHS]` and `C[RHS]` necessarily have different observable statistics.

13.4 Verification of Pólya's urn

In the introduction we recalled the stateful implementation of the Beta-Bernoulli process using the module `Polya`. Our equational theory gives a recipe for relating that implementation to `BetaBernoulli`:

The first step is to prove our equational theory sound with respect to `Polya`. We give an operational semantics to `Polya` which introduces a notion of observational equivalence on closed ground terms. We extend this to a notion of contextual equivalence $\approx_{\mathsf{ctx}}$ between open terms [Bizjak and Birkedal, 2015, §6]. We must now show that all ground substitution instances of our axioms hold up to contextual equivalence*. Because contextual equivalence is a congruence, we obtain that $u = v$ implies $u \approx_{\mathsf{ctx}} v$ for all ground terms u, v. From the extensionality results (Propositions 13.1 and 13.3), we obtain that all derivable equations are in fact validated in `Polya`. Finally, because contextual equivalence doesn't collapse rational probabilities, the syntactical completeness result implies that $u = v$ is derivable iff $u \approx_{\mathsf{ctx}} v$.

$\star$: It remains prove all ground instances of the axioms hold up to contextual equivalence. A verification in this style has successfully been conducted by Marcin Szymczak (private communication with Sam Staton). This task is simplified further as it is not necessary to check that axioms (C1) and (D2) hold in contextual equivalence, because the axiomatized equality on closed ground terms is independent of these axioms. To see this, notice that our normalization procedure does not use (C1) or (D2) when the terms are closed and ground. The remaining axioms are fairly straightforward, e.g. (Conj) is the essence of the urn scheme and (D1) is garbage collection.

14 A Model of Synthetic Probability

By the general methods sketched in Section 3.7, we can think of our algebraic theory as a combinatorial presentation a synthetic model of probability with contains the Beta and Bernoulli distributions. Because this is an important technique, we will spell out some details. This also lets us use the internal languages Sections 3.1 and 7 as extended calculi for Beta-Bernoulli and lets us make precise connections to the concepts of synthetic probability.

As a monad: Our theory gives rise to a monad on the (covariant) presheaf category $[\mathsf{Fin}, \mathsf{Set}]$, where Fin is the category of finite sets and functions. Because the theory is commutative and discardable, the associated monad is a generalized probability monad P. Note that the domain of the presheaves encodes contexts of parameters and their substitutions. In our case, simple renamings encoded by Fin suffice. (Formally, $\mathsf{Fin}^{\mathsf{op}}$ is the Lawvere theory of the theory of equality).

Let $\mathcal{I}$ denote the representable presheaf $\mathcal{I}(\vec{p}) = \vec{p}$. Every other representable $\mathsf{Fin}(n, -)$ is isomorphic to $\mathcal{I}^n$. The terminal presheaf is $1 = \mathcal{I}^0$, and we encode the booleans as $2 = 1 + 1$. We interpret $\mathcal{I}$ to stand for an abstract unit interval object; syntactically, $\mathcal{I}^n$ stands for a computation variable of arity n. A coproduct of representables $\sum_{k=1}^{n} \mathcal{I}^{m_k}$ thus represents a context of computation variables $(x_1 : m_1, \ldots, x_k : m_k)$. Let $\mathcal{F} \subseteq [\mathsf{Fin}, \mathsf{Set}]$ denote the full subcategory of such presheaves. We can begin defining the monad P on $\mathcal{F}$ as terms modulo equations

$$P(\mathcal{I}^{m_1} + \ldots + \mathcal{I}^{m_k})(\vec{p}) \stackrel{\text{def}}{=} \{\vec{p} \mid x_1 : m_1, \ldots, x_k : m_k \vdash u\}/= \tag{55}$$

The unit is given by the variable terms, and monadic bind is substitution of computations. The structure thus defined is an enriched clone [Staton, 2013a]. Every presheaf is a sifted colimit of $\mathcal{F}$-objects, and we extend the definition (55) to a monad P on the full presheaf category by preservation of that colimit. Note that P does not preserve coproducts because there are nontrivial interactions between computations, like $x +_{i:j} y$.

Applying the Yoneda lemma to (55) shows that to give a natural transformation $\mathcal{I}^n \to P(\mathcal{I}^{m_1} + \ldots + \mathcal{I}^{m_k})$ is to give a term-modulo-equations $p_1, \ldots, p_n \mid x_1 : m_1, \ldots, x_k : m_k \vdash u$. The monad can be seen to support the following operations

$$\beta_{i,j} : 1 \to P(I) \quad \mathsf{get} : I \to P(2) \quad \mathsf{flip}_{m:n} : 1 \to P(2), \quad i, j > 0,\ m + n > 0$$

given syntactically by the terms

$$x : 1 \vdash v_{i,j} p.x(p) \quad p \mid x : 0, y : 0 \vdash x \,?_p\, y \quad x : 0, y : 0 \vdash x +_{m:n} y$$

Proposition 14.1 *To give a point $1 \to P(2)$ is to give a rational probability $0 \leq p \leq 1$.*

PROOF Terms modulo equations $- \mid x : 0, y : 0 \vdash u$ are precisely of the form $x +_{m:n} y$, by the normalization procedure and Proposition 12.5. ∎

This is in contrast with the local behavior of the presheaf $P(2)$. For example, the morphism $\mathsf{get} : I \to P(2)$ is not merely a rational coin flip. By Proposition 12.7, the set $P(2)(\vec{p})$ consists of tree diagrams, which can be described in terms of Bernstein polynomials.

It is interesting to consider the difference between the interval object $\mathcal{I}$ and $P(2)$ in the model. In measure-theoretic probability, those are isomorphic (Example 4.10), but not here: For example, $\mathcal{I}$ has no points while $P(2)$ contains the rational probabilities. In accordance with Section 10.4, we could eschew the abstract type $\mathcal{I}$ altogether and define a variant of the Beta distribution as a *distribution over distributions* $\mathsf{beta}_{i,j} : 1 \to P(P(2))$,

$$\mathsf{beta}_{i,j} \stackrel{\text{def}}{=} \mathsf{let}\ p \leftarrow \beta_{i,j}\ \mathsf{in}\ [\mathsf{get}(p)]$$

This removes the need for get in the language, as we sample a double distribution by mere monadic sequencing, e.g.

$$\mathsf{let\ coin} \leftarrow \mathsf{beta}_{1,1}\ \mathsf{in\ coin} = \mathsf{flip}_{1:1}$$

By turning the algebraic theory into a monad, we get semantics for a more fully-fledged programming language which supports standard features such as if-then-else, pairs, thunks

and higher-order functions. Such semantics is adequate, but not designed with full abstraction in mind (such questions will concern us in Chapter V). With Beta and Bernoulli in the language, more complicated distributions such as the categorical and Dirichlet distributions are definable. This makes Beta-Bernoulli an important building blocks for Bayesian models.

As a Markov category: If we are merely interested to model ground Beta-Bernoulli computation, we can turn our theory into a Markov category in a hands-on way: Recall the subcategory $\mathcal{F} \subseteq [\mathsf{Inj}, \mathsf{Set}]$ of sums of representables from before. We note that $\mathcal{F}$ is closed under products of presheaves, because $\mathcal{I}^m \times \mathcal{I}^n \cong \mathcal{I}^{m+n}$ and this extends to sums of representables by distributivity[13]. We can thus form a Markov category BetaBern on the objects of $\mathcal{F}$ whose morphisms are the Kleisli maps $X \to PY$. As remarked by Staton in [Fritz, 2020, 6.2], we can use (55) to cast this definition in a purely combinatorial way: Objects are lists of natural numbers $(m_1, \dots, m_k)$ and morphisms $(n_1, \dots, n_r) \to (m_1, \dots, m_k)$ are collections of terms-modulo-equations

$$(p_1, \dots, p_{n_i} \mid x_1 : m_1, \dots, x_k : m_k \vdash u_i)_{i=1,\dots,r}$$

The product operation defines a tensor on objects,

$$(m_1, \dots, m_k) \otimes (n_1, \dots, n_r) \stackrel{\text{def}}{=} (m_1 + n_1, \dots, m_1 + n_r, \dots m_i + n_j, \dots, m_k + m_r)$$

and the copy-delete structure is given by nonlinear use of variables, e.g. $\mathrm{copy}_{(1)} : (1) \to (2)$ is given by copying of parameters $p \mid x : 2 \vdash x(p, p)$.

Proposition 14.2 *The measure-theoretic semantics (Definition 12.1)*

$$[\![-]\!] : \mathsf{BetaBern} \to \mathsf{CH}_{\mathcal{R}} \quad [\![(m_1, \dots, m_k)]\!] = [0,1]^{m_1} + \dots + [0,1]^{m_k} \tag{56}$$

induces a faithful functor to the Kleisli category of the Radon monad (26), which preserves Markov structure. Similarly, the (complexified[14]) functional-analytic semantics (Definition 12.3) induces a faithful Markov functor

$$[\![-]\!] : \mathsf{BetaBern} \to \mathsf{PU}^{\mathsf{op}} \quad [\![(m_1, \dots, m_k)]\!] = \mathbb{C}^{[0,1]^{m_1}} \times \dots \times \mathbb{C}^{[0,1]^{m_k}} \tag{57}$$

By construction of our semantics, these functors are identified under the equivalence of Theorem 4.15.

From this perspective, BetaBern can be understood as a combinatorial characterization of those probability kernels which are definable using only Beta, Bernoulli and if-then-else.

Conjugate priors Having turned our theory into a categorical model of probability, we can apply the concepts of Chapter II to it. For example, the (Conj) axiom expresses the conjugate-prior relationship between Beta- and Bernoulli in the categorical sense of Jacobs [2020].

[13] $\mathcal{F}$ is equivalent to $\mathsf{FinFam}(\mathsf{Fin}^{\mathsf{op}})$, the finite coproduct completion of the Lawvere theory $\mathsf{Fin}^{\mathsf{op}}$

[14] using C*-algebras $\mathbb{C}^X$ rather than $\mathbb{R}^X$ to fit the setup of Section 4.4

Proposition 14.3 *The distribution $\psi : 1 \to P(\mathcal{I} \times 2)$ given by a Beta sample and successive coin flip*

$$\psi = \mathsf{let}\, p \leftarrow \beta_{i,j}\, \mathsf{in}\, \mathsf{let}\, x \leftarrow \mathsf{get}(p)\, \mathsf{in}\, [(p, x)]$$

has a conditional distribution $\psi|_X : 2 \to P(\mathcal{I})$ given by

$$\psi|_X(x) = \mathsf{if}\ x\ \mathsf{then}\ \beta_{i+1,j}\ \mathsf{else}\ \beta_{i,j+1}$$

PROOF We use distributivity to identify $\mathcal{I} \times 2$ with $\mathcal{I} + \mathcal{I}$. The distribution $\psi : 1 \to P(\mathcal{I} + \mathcal{I})$ is then represented by the term

$$x : 1, y : 1 \vdash \nu_{i,j} p.x(p)\, ?_p\, y(p) \tag{58}$$

and its marginal $\psi|_X : 1 \to P(1 + 1)$ can be seen to equal $\mathsf{flip}_{i:j}$, for

$$\nu_{i,j} p.x\, ?_p\, y = x +_{i:j} y$$

The defining equation (38) of conditionals

$$\mathsf{let}\, x \leftarrow \mathsf{flip}_{i:j}\, \mathsf{in}\, \mathsf{let}\, p \leftarrow \psi|_X(x)\, \mathsf{in}\, [(p, x)]$$

indeed recovers (58) by an application of (Conj), as

$$(\nu_{i+1,j} p.x(p)) +_{i:j} (\nu_{i,j+1} p.y(p)) = \nu_{i,j} p.x(p)\, ?_p\, y(p)$$

■

We conjecture that in BetaBern, any distribution $\psi : 1 \to X \times n$ has a conditional $\psi|_n : n \to X$, where $n = (0, \ldots, 0)$ is a discrete sample space. The existence of these categorical conditionals serves as a starting point for the inclusion of conditioning as a first-class construct (see Chapter IV).

15 Conclusion and Related Work

We have demonstrated that the central notion of *exchangeable random processes* in Bayesian inference admits an analysis in terms of basic concepts from programming language theory: abstract types, commutativity and discardability. Those two axioms naturally places them within the domain of synthetic probability theory.

We have illustrated this approach by showing that adding the conjugacy law to these ingredients leads to a complete equational theory for the Beta-Bernoulli process. We proved that this theory has a canonical syntactic and axiomatic status, regardless of the measure-theoretic foundation. We have used the syntactic completeness result to formally relate the Beta-Bernoulli process to a stateful implementation using Pólya's urn. Other examples of effects with stateful implementations that have the interface of pure probability are pseudorandom number generators or name generation (`gensym`). These can be seen as a variant of De Finetti's theorem for program modules. Other recent treatments of this theorem approach it from the perspective of coalgebra [Jacobs and Staton, 2020] and Markov categories [Fritz et al., 2021].

We have made use of parameterized algebraic theories as a way of presenting models of synthetic probability in a concise way. We use similar techniques to present Gaussian

probability with conditioning (Chapter IV) and name generation (Proposition 25.19). Unlike in [Staton, 2013b], our urns cannot be compared for equality, and adding such an operation $?_{p=q} : (2 \mid [0,0])$ would in fact break the conjugacy axiom in its current form. Semantically, the equality test $(=) : \mathcal{I} \times \mathcal{I} \to 2$ is not a natural transformation in $[\mathsf{Fin}, \mathsf{Set}]$. In nominal sets, the category Inj of finite sets and injections replaces Fin to allow for equality testing (Proposition 25.19).

Normal forms play a crucial role in all chapters of this thesis. They can variously be seen as compiler optimizations, garbage collection schemes or symbolic inference techniques. Our results furthermore open up the following avenues of research:

Firstly, our methods may be generalized to more complex hierarchical models. For example, the Chinese Restaurant Franchise [Teh et al., 2006] can be implemented as a module with three abstract types, `f` (franchise), `r` (restaurant), `t` (table), and functions `newFranchise:()->f`, `newRestaurant:f->r`, `getTable:r->t`, `sameDish:t*t->bool`. Its various exchangeability properties correspond to commutativity/discardability in the presence of type abstraction. For further examples, see [Staton et al., 2017b].

A more practical motivation for our work is to inform the design of module systems for probabilistic programming languages. For example, ANGLICAN, CHURCH, HANSEI and VENTURE already support nonparametric primitives [Kiselyov and Shan, 2010; Wu, 2013; Mansinghka et al., 2014], although CHURCH and ANGLICAN do not have type systems. We contend that abstract types are a crucial concept from the perspective of exchangeability.

Chapter IV
Compositional Semantics for Conditioning

SUMMARY: We develop a synthetic treatment of Bayesian inference which works in any Markov category $\mathbb{C}$ with conditionals and a mild regularity assumption on the absolute continuity relation (Definition 19.4); this can be seen as a semantic validation of the symbolic disintegration techniques of Shan and Ramsey [2017].

We introduce *conditioning channels* $X \rightsquigarrow Y$ which represent open programs with conditioning effects up to observational equivalence. Our main result is that the category of such conditioning channels forms a CD category $\mathsf{Cond}(\mathbb{C})$ (Theorem 19.6) which encapsulates convenient properties of conditioning, such as commutativity, substitutivity and an initialization principle (Section 19.3).

As an application, we apply this construction to give compositional semantics to a probabilistic programming language with Gaussian random variables and a first-class exact conditioning operator (Section 21.1). We give an algebraic presentation of that language (Section 21) and draw analogies between exact conditioning and unification in logic programming (Section 20).

This chapter is based on joint work with Sam Staton [Stein and Staton, 2021]. An implementation of the Gaussian language is available under [Stein, 2021].

16 Introduction

We identify two ways of expressing dependence between model and observed data in probabilistic programming: scoring and exact conditioning.

Under *scoring*, we understand a construct like **score** or **observe** which records a likelihood to re-weight the current execution trace of the probabilistic program. Such constructs are wide-spread and available in popular languages such as STAN [Carpenter et al., 2017] or WEBPPL [Goodman and Stuhlmüller, 2014]. Scoring with likelihoods from $\{0, 1\}$ is sometimes called a *hard constraint*, as opposed to more general *soft constraints*. The prototypical way of performing inference on scoring programs is likelihood-weighted importance sampling. Hard constraints turn this into mere rejection sampling, because likelihood-zero traces are discarded entirely. Replacing hard constraints by equivalent soft ones can thus be beneficial for inference efficiency.

Under *exact conditioning*, we understand a construct here denoted $E_1 =:= E_2$ which expresses that expressions E_1, E_2 shall be conditioned to be exactly equal. In finite probability, this can be implemented in terms of a hard constraint by writing

```
score(if E1 == E2 then 1 else 0)
```

In a continuous setting, exact conditions are more powerful than that and cannot generally be expressed as scoring any more. For example, the program

```
let x = normal(0,1) in x =:= 42; x
```

should return 42 deterministically. The scoring program

```
let x = normal(0,1) in score(if x == 42 then 1 else 0); x
```

will however reject almost every execution trace, because the probability that $x = 42$ is zero. A major goal of this chapter is to rigorously define and study the properties of exact conditioning. Aspects of this construct are present in other frameworks; exact conditioning queries can be addressed using symbolic disintegration in HAKARU [Shan and Ramsey, 2017] but (=:=) is no first-order construct in Hakaru. INFER.NET [Minka et al., 2018] does allow exact conditioning on data in the following style[15] and employs an approximate inference algorithm, which we believe gives exact answers for problems involving Gaussians:

```
Variable<double> x = Variable.GaussianFromMeanAndPrecision(0, 1);
x.ObservedValue = 42;
```

We provide two semantic analyses of exact conditioning in a simple Gaussian language: a denotational semantics, and an equational axiomatic semantics, which we prove coincide for closed programs. Our denotational semantics is based on the notion of *conditioning channels*, which are a new and general construction on Markov categories.

Case study: A Gaussian Programming Language with Exact Conditions Exact conditioning has the advantage of decoupling the generative model cleanly from the data observations phase. Consider the following example of a Gaussian process regression (a.k.a. "kriging"):

A Gaussian process is a type of distribution over functions. We'll evaluate that function at a fixed list xs of inputs of interest, say `ys = gp_sample(xs)`. The distribution of `ys` is now an N-dimensional multivariate Gaussian vector where $N = |\texttt{xs}|$; different samples of `ys` are shown as the graphs on the left of Figure 4.

Say we know the values $\hat{y}_1, \ldots, \hat{y}_k$ of the random function at fixed inputs $x_1, \ldots, x_k$, and wish to predict how that affects the distribution of the unknown values. This is completely straightforward to express using exact conditioning (posterior in Figure 4):

```
ys = gp_sample(xs)
for i = 1 to k:
  ys[x_i] =:= ŷ_i
```

Figure 3: Exact inference for a Gaussian process

[15] see https://dotnet.github.io/infer/userguide/Learning%20a%20Gaussian%20tutorial.html

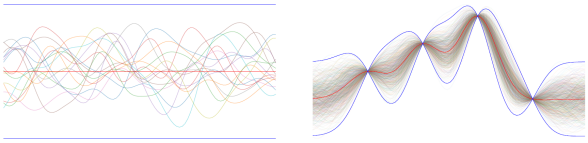


Figure 4: GP prior and posterior with $k = 4$ exact observations

The same program is difficult to express compositionally without exact conditioning: Using soft conditions, the observations $(x_i, \hat{y}_i)$ would need to be passed into **gp_sample** and **observe** commands need to be issued in-place, breaking the separation of model and inference.

No style of probabilistic modelling is immune to fallacies and paradoxes. Exact conditioning is indeed sensitive in this regard in general (Section 22.1), and so it is important to show that where it is used, it is consistent in a compositional way. To simplify our kriging example further, we consider the following concrete variation with a Gaussian random walk: Suppose that the observation points are at $(0, 20, 40, 60, 80, 100)$ and consider the model

```
ys[0] = normal(0,1)
for i = 1 to 100:
  ys[i] = ys[i−1] + normal(0,1)
for j = 0 to 5:
  ys[20*j] =:= c[j]
```

To illustrate the power of compositional reasoning, we note that exact conditioning here is first-class, and as we will show, it is consistent to reorder programs as long as the dataflow is respected (Proposition 19.10). So this random walk program is equivalent to:

```
ys[0] = normal(0,1)
ys[0] =:= c[0]
for i = 1 to 100:
  ys[i] = ys[i−1] + normal(0,1)
  if i % 20 == 0: ys[i] =:= c[i % 20]
```

We can now use a substitution law and initialization principle to simplify the program to

```
ys[0] = c[0]
for i = 1 to 100:
  if i % 20 == 0:
    ys[i] = c[i % 20]
    (ys[i] − ys[i−1]) =:= normal(0,1)
  else:
    ys[i] = ys[i−1] + normal(0,1)
```

The constraints are now all 'soft', in that they relate an expression with a distribution, and so this last program could be run with a Monte Carlo simulation in Stan or WebPPL. Indeed, the soft-conditioning primitive **observe** can be defined in terms of exact conditioning as

$$\texttt{observe(D,x)} \equiv \texttt{(let y = sample(D) in x =:= y)}$$

as explained in Section 20.1. Our language is by no means restricted to kriging. For example, we can use similar techniques to implement and verify a simple Kálmán filter.

In Section 17, we provide an operational semantics for such a language, in which there are two key commands: drawing from a standard normal distribution (normal()) and exact conditioning ($=:=$). The operational semantics is defined in terms of configurations (t, ψ) where t is a program and ψ is a state, which here is a Gaussian distribution. Each call to normal() introduces a new dimension into the state ψ, and conditioning ($=:=$) alters the state ψ, using a canonical form of conditioning for Gaussian distributions (Section 17.1).

For the program in Figure 3, the operational semantics will first build up the prior distribution shown on the left in Figure 4, and then the second part of the program will condition to yield a distribution as shown on the right. But for the other programs above, the conditioning will be interleaved in the building of the model.

In stateful programming languages, composition of programs is often complicated and local transformations are difficult to reason about. But, as we now explain, we will show that for the Gaussian language, compositionality and local reasoning are straightforward. For example, as we have already illustrated:

- Program lines can be reordered as long as dataflow is respected. That is, the *commutativity equation* (Comm) remains valid for programs with conditioning

$$\begin{array}{l}\mathsf{let}\, x \;=\; u \,\mathsf{in} \\ \mathsf{let}\, y \;=\; v \,\mathsf{in}\, t\end{array} \;\equiv\; \begin{array}{l}\mathsf{let}\, y \;=\; v \,\mathsf{in} \\ \mathsf{let}\, x \;=\; u \,\mathsf{in}\, t\end{array}$$

  where x not free in v and y not free in u.

- We have a *substitution law*: if $t =:= u$ appears in a program, then later occurrences of t may be replaced by u.

$$(t =:= u); v[t/x] \quad \equiv \quad (t =:= u); v[u/x] \tag{59}$$

- As a special base case, if we condition a normal variable on a constant $\underline{c}$, then that variable is *initialized* to this value

$$\mathsf{let}\, x \;=\; \mathrm{normal}() \,\mathsf{in}\, (x =:= \underline{c}); t \quad \equiv \quad t[\underline{c}/x] \tag{60}$$

Denotational Semantics and the Cond Construction In Section 21.1, we show that this compositional reasoning is valid by using a denotational semantics. For a Gaussian language with*out* conditioning, we can easily interpret terms as noisy affine-linear functions, $x \mapsto Ax + c + \mathcal{N}(\Sigma)$. The exact conditioning requires a new construction for building a

semantic model. In fact this construction is not at all specific to Gaussian probability and works generally.

Our conditioning construction starts from a Markov category $\mathbb{C}$, corresponding as a probabilistic programming language without conditioning. We build a new symmetric monoidal category $\mathsf{Cond}(\mathbb{C})$ which is conservative over $\mathbb{C}$ but which contains a conditioning construct. This construction builds on an analysis of conditional probabilities from the Markov category literature, which captures conditioning purely in terms of categorical structure: there is no explicit Radon-Nikodým theorem, limits, reference measures or in fact any measure theory at all. The good properties of the Gaussian language generalize to this abstract setting, as they follow from universal properties alone.

The category $\mathsf{Cond}(\mathbb{C})$ has the same objects as $\mathbb{C}$, but a morphism is reminiscent of the decomposition of the program in Figure 4: a pair of a purely probabilistic morphism together with an observation. These morphisms compose by composing the generative parts and accumulating the observations (for a graphical representation, see Figure 5). The morphisms are considered up-to a natural contextual equivalence. We prove some general properties about $\mathsf{Cond}(\mathbb{C})$:

(i) Proposition 19.12: $\mathsf{Cond}(\mathbb{C})$ is consistent, in that no distinct unconditional distributions from $\mathbb{C}$ are equated in $\mathsf{Cond}(\mathbb{C})$.

(ii) Proposition 19.10: $\mathsf{Cond}(\mathbb{C})$ allows programs to be reordered according to their dataflow graph, i.e. it satisfies the interchange law of monoidal categories.

Returning to the specific case study of Gaussian probability, we show that we have a canonical interpretation of the Gaussian language in $\mathsf{Cond}(\mathsf{Gauss})$, which is fully abstract (Theorem 21.2). In consequence, the principles of reordering and consistency hold for the contextual equivalence induced by the operational semantics.

Equational Presentation Our second semantic analysis (Section 21) has a more syntactic and concrete flavor. We leave the generality of Markov categories and focus again on the Gaussian language. We present an equational theory for programs and derive normal forms.

Our equational theory is surprisingly simple. The first two equations are

$$(\mathsf{let}\, x \;=\; \mathrm{normal}()\,\mathsf{in}\,())\;=\;() \qquad \begin{aligned}\mathsf{let}\, x_1 \;&=\; \mathrm{normal}(), \ldots, x_n = \mathrm{normal}()\,\mathsf{in}\, U\vec{x}\\ &= \mathsf{let}\, x_1 \;=\; \mathrm{normal}(), \ldots, x_n = \mathrm{normal}()\,\mathsf{in}\,\vec{x}\end{aligned}$$

The left equation is discardability. In the right equation, U must be an orthogonal matrix, and we are using shorthand for multiplying a vector by a matrix. These two equations are enough to fully axiomatize the fragment of the Gaussian language without conditioning (Proposition 21.3). In Section 21 we introduce a concise notation, writing the first axiom as $\nu x.\mathrm{r}[] = \mathrm{r}[]$. One instance of the second axiom with a permutation matrix for U is $\nu x.\nu y.\mathrm{r}[x,y] = \nu y.\nu x.\mathrm{r}[x,y]$, reminiscent of name generation in the π-calculus [Milner et al., 1992] or ν-calculus in Chapter V. The remaining axioms focus on conditioning. There are commutativity axioms for reordering parts of programs, as well as the substitution and initialization laws considered above, (59), (60). Finally there are two axioms for eliminating a condition that is tautologous $(a =:= a)$ or impossible $(0 =:= 1)$.

Together, these axioms are consistent, which we can deduce by showing them to hold in the Cond model. To moreover illustrate the strength of the axioms, we show two normal form theorems by merely using the axioms. Here $\text{normal}_n()$ describes the n-dimensional standard normal distribution.

- Theorem 21.8: any closed program is either derivably impossible $(0 =:= 1)$ or derivably equal to a condition-free program of the form $A * \text{normal}_n() + \vec{c}$.
- Theorem 21.10: any program of unit type (with no return value) is either derivably impossible $(0 =:= 1)$ or derivably equal to a soft constraint, i.e. a program of the form $A * \vec{x} =:= B * \text{normal}_n() + \vec{c}$. We also give a uniqueness criterion on A, B and $\vec{c}$.

16.1 Outline

In Section 17, we present a minimalist language with exact conditioning for Gaussian probability, with the purpose of studying the abstract properties of conditioning. Despite its simplicity, the language can express interesting models such as Gaussian processes or Kálmán filters. An implementation is available under [Stein, 2021].

In Section 18, we develop an abstract account of *inference problems* purely in terms of Markov categories. We then introduce *conditioning channels* in Section 19, which extend a Markov category $\mathbb{C}$ to a CD category $\mathsf{Cond}(\mathbb{C})$ in which conditioning is internalized as a morphism. The Gaussian language is recovered as the internal language of $\mathsf{Cond}(\mathsf{Gauss})$ where the Markov category Gauss captures conditioning-free Gaussian probability.

We give three semantics for the language – operational (Section 17), denotational (Section 21.1) and axiomatic (Section 21). We show that the denotational semantics is fully abstract (Theorem 21.2) and that the axiomatic semantics is strong enough to derive normal forms (Theorem 21.10). This justifies properties like commutativity and substitutivity for the language. Thus probabilistic programming with exact conditioning can serve as a practical foundation for compositional statistical modelling.

17 A Language for Gaussian Probability

In Section 17.2, we formally introduce a typed language (Section 3) for Gaussian probability, and provide an operational semantics for it (Section 17.3).

17.1 Recap of Gaussian Probability

We briefly recall *Gaussian probability*, by which we mean the treatment of multivariate Gaussian distributions and affine-linear maps (e.g. [Lauritzen and Jensen, 1999]). A *(multivariate) Gaussian distribution* is the law of a random vector $X \in \mathbb{R}^n$ of the form $X = AZ + \mu$ where $A \in \mathbb{R}^{n \times m}$, $\mu \in \mathbb{R}^n$ and the random vector Z has components $Z_1, \dots, Z_m \sim \mathcal{N}(0,1)$ which are independent and standard normally distributed with density function

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

The distribution of X is fully characterized by its *mean* μ and the positive semidefinite *covariance matrix* Σ. Conversely, for any μ and positive semidefinite matrix Σ there is a unique Gaussian distribution of that mean and covariance denoted $\mathcal{N}(\mu, \Sigma)$. The vector X takes values precisely in the affine subspace $S = \mu + \text{col}(\Sigma)$ where $\text{col}(\Sigma)$ denotes the column space of Σ. We call S the *support* of the distribution.

This defines a small convenient fragment of probability theory: Affine transformations of Gaussians remain Gaussian. Furthermore, conditional distributions of Gaussians are again Gaussian. This is known as self-conjugacy. If we decompose an $(m+n)$-dimensional Gaussian vector $X \sim \mathcal{N}(\mu, \Sigma)$ into components X_1, X_2 with

$$X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}, \mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \text{ where } \Sigma_{21} = \Sigma_{12}^T$$

then the conditional distribution $X_1|(X_2 = a)$ of X_1 conditional on $X_2 = a$ is $\mathcal{N}(\mu', \Sigma')$ where

$$\mu' = \mu_1 + \Sigma_{12}\Sigma_{22}^+(a - \mu_2) \quad \Sigma' = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^+\Sigma_{21} \tag{61}$$

and Σ_{22}^+ denotes the Moore-Penrose pseudoinverse.

This formula becomes particular simple if we condition on a single real-valued variable: Let $X \sim \mathcal{N}(\mu, \Sigma)$ and let $Z = uX$ for some $u \in \mathbb{R}^{n\times 1}$, then the covariance of (X, Z) decomposes with entries

$$\Sigma_{12} = \Sigma u^T, \sigma_{22} = u\Sigma u^T$$

and the conditional distribution of $X|(X = a)$ is $\mathcal{N}(\mu', \Sigma')$ with

$$\mu' = \mu + \frac{a}{\sigma_{22}}\Sigma u^T, \quad \Sigma' = \Sigma - \sigma_{22}^{-1}\Sigma u^T u\Sigma \tag{62}$$

whenever $\sigma_{22} > 0$. If $\sigma_{22} = 0$ and $u\mu = a$, the condition is tautologous $0 = 0$, so we let $\mu' = \mu, \Sigma' = \Sigma$, and otherwise the condition is infeasible.

Example 17.1 Let $X, Y \sim \mathcal{N}(0, 1)$ be independent and $Z = X - Y$. The joint distribution of (X, Y, Z) is $\mathcal{N}(0, \Sigma)$ with covariance matrix

$$\Sigma = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \\ 1 & -1 & 2 \end{pmatrix}$$

The conditional distribution of (X, Y) given $Z = 0$ has covariance matrix

$$\Sigma' = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \frac{1}{2}\begin{pmatrix} 1 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & -1 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

Note that the posterior distribution is equivalent to the model

$$X \sim \mathcal{N}(0, 0.5), Y = X$$

Borel's paradox is an important subtlety that occurs when conditioning on the equality of random variables $X = Y$. The original formulation involves conditioning a uniform point on a sphere to lie on a great circle, but we will use Borel's paradox to refer to any situation

where conditioning on equivalent equations leads to different outcomes (e.g. [Shan and Ramsey, 2017]). For example, if instead of the condition $X - Y = 0$ in Example 17.1 we had chosen the equivalent equations $X/Y = 1$ or even $[X = Y] = 1$, we would have obtained different posteriors:

Example 17.2 (Borel's paradox) If $X, Y \sim \mathcal{N}(0,1)$, then conditioned on $(X/Y = 1)$, the variable X can be shown to have density $|x|e^{-x^2}$ [Proschan and Presnell, 1998]. Under the boolean condition $[X = Y] = 1$, the inference problem is considered infeasible because the model $X, Y \sim \mathcal{N}(0,1), Z = [X = Y]$ is measure-theoretically equal to $X, Y \sim \mathcal{N}(0,1), Z = 0$ and conditioning on $0 = 1$ is inconsistent.

We will address Borel's paradox repeatedly (Section 20), and see that careful type-theoretic phrasing helps alleviate some of its seemingly paradoxical nature (Section 22.1).

17.2 Types and Terms of the Gaussian language

We now describe a language for Gaussian probability and conditioning. The core language resembles first-order OCaml with a construct normal() to sample from a standard Gaussian, and conditioning denoted as $(=:=)$. Types τ are generated from a basic type R denoting *real* or *random variable*, pair types and unit type I.

$$\tau ::= \mathrm{R} \mid \mathrm{I} \mid \tau * \tau$$

Terms of the language are

$$\begin{aligned} e ::= {} & x \mid e + e \mid \alpha \cdot e \mid \underline{\beta} \mid (e,e) \mid () \\ & \mid \mathsf{let}\, x \;=\; e \,\mathsf{in}\, e \mid \mathsf{let}\, (x,y) \;=\; e \,\mathsf{in}\, e \\ & \mid \mathrm{normal}() \mid e =:= e \end{aligned}$$

where α, β range over real numbers. Typing judgements are

$$\frac{}{\Gamma, x : \tau, \Gamma' \vdash x : \tau} \qquad \frac{}{\Gamma \vdash () : \mathrm{I}} \qquad \frac{\Gamma \vdash s : \sigma \quad \Gamma \vdash t : \tau}{\Gamma \vdash (s,t) : \sigma * \tau}$$

$$\frac{\Gamma \vdash s : \mathrm{R} \quad \Gamma \vdash t : \mathrm{R}}{\Gamma \vdash s + t : \mathrm{R}} \qquad \frac{\Gamma \vdash t : \mathrm{R}}{\Gamma \vdash \alpha \cdot t : \mathrm{R}} \qquad \frac{}{\Gamma \vdash \underline{\beta} : \mathrm{R}}$$

$$\frac{}{\Gamma \vdash \mathrm{normal}() : \mathrm{R}} \qquad \frac{\Gamma \vdash s : \mathrm{R} \quad \Gamma \vdash t : \mathrm{R}}{\Gamma \vdash (s =:= t) : \mathrm{I}}$$

$$\frac{\Gamma \vdash s : \sigma \quad \Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \mathsf{let}\, x \;=\; s \,\mathsf{in}\, t : \tau}$$

$$\frac{\Gamma \vdash s : \sigma * \sigma' \quad \Gamma, x : \sigma, y : \sigma' \vdash t : \tau}{\Gamma \vdash \mathsf{let}\, (x,y) \;=\; s \,\mathsf{in}\, t : \tau}$$

Our language is precisely the CD-calculus (Section 7), with base type R and signature

$$(+) : \mathrm{R} * \mathrm{R} \to \mathrm{R}, \quad \alpha \cdot (-) : \mathrm{R} \to \mathrm{R}, \quad \underline{\beta} : \mathrm{I} \to \mathrm{R}, \quad \mathrm{normal} : \mathrm{I} \to \mathrm{R}, \quad (=:=) : \mathrm{R} * \mathrm{R} \to \mathrm{I} \quad (63)$$

This will give us a clear path to denotational semantics: In Section 21.1, we will indeed identify our language as the internal language of an appropriate CD category with an exact conditioning morphism.

We use standard syntactic sugar for sequencing $s; t$, identifying the type $\mathrm{R}^n = \mathrm{R} * (\mathrm{R} * \dots)$ with vectors and for matrix-vector multiplication $A \cdot \vec{x}$. For $\sigma \in \mathbb{R}$ and $e : \mathrm{R}$, we define $\mathrm{normal}(x, \sigma^2) \equiv x + \sigma \cdot \mathrm{normal}()$. More generally, for a covariance matrix Σ, we write $\mathrm{normal}(\vec{x}, \Sigma) = \vec{x} + A \cdot (\mathrm{normal}(), \dots, \mathrm{normal}())$ where A is any matrix such that $\Sigma = AA^T$. We can identify any context and type with R^n for suitable n.

17.3 Operational Semantics

Our operational semantics is call-by-value. Calling $\mathrm{normal}()$ allocates a latent random variable, and a prior distribution over all latent variables is maintained. Calling $(=:=)$ updates this prior by symbolic inference according to the formula (61).

Values v and redexes ρ are defined as

$$\begin{aligned} v &::= x \mid (v, v) \mid v + v \mid \alpha \cdot v \mid \underline{\beta} \mid () \\ \rho &::= \mathrm{normal}() \mid v =:= v \mid \mathsf{let}\, x = v \,\mathsf{in}\, e \mid \mathsf{let}\, (x, y) = v \,\mathsf{in}\, e \end{aligned}$$

A reduction context C with hole $[-]$ is of the form

$$\begin{aligned} C ::= [-] \mid C + e \mid v + C \mid r \cdot C \mid C =:= e \mid v =:= C \\ \mid \mathsf{let}\, x = C \,\mathsf{in}\, e \mid \mathsf{let}\, (x, y) = C \,\mathsf{in}\, e \end{aligned}$$

Every term is either a value or decomposes uniquely as $C[\rho]$. We define a reduction relation for terms: A *configuration* is either a dedicated failure symbol $\bot$ or a pair (e, ψ) where ψ is a Gaussian distribution on $\mathbb{R}^r$ and $z_1 : \mathrm{R}, \dots, z_r : \mathrm{R} \vdash e$. The *latent variables* are taken from a distinct supply of variable names $\{z_i : i \in \mathbb{N}\}$. We first define reduction on redexes:

(i) Calling $\mathrm{normal}()$ allocates a fresh latent variable and adds an independent dimension to the prior

$$(\mathrm{normal}(), \psi) \rhd (z_{\mathrm{r}+1}, \psi \otimes \mathcal{N}(0, 1))$$

(ii) To define conditioning, note that every value $z_1 : \mathrm{R}, \dots, z_r : \mathrm{R} \vdash v : R$ defines an affine function $\mathbb{R}^r \to \mathbb{R}$. In order to reduce $(v =:= w, \psi)$, we $X \sim \psi$ and define the auxiliary variable $Z = v(X) - w(X)$. If 0 lies in the support of Z, we denote by $\psi|_{v=w}$ the outcome of conditioning X on $Z = 0$, and reduce

$$(v =:= w, \psi) \rhd ((), \psi|_{v=w})$$

Otherwise $(v =:= w, \psi) \rhd \bot$, indicating that the inference problem has no solution. To be completely, we find $u \in \mathbb{R}^{1 \times r}$ and $b \in \mathbb{R}$ such that $Z = uX + b$ and condition on $Z = 0$ using formula (62).

(iii) Let bindings are standard

$$\begin{aligned} (\mathsf{let}\, x = v \,\mathsf{in}\, e, \psi) &\rhd (e[v/x], \psi) \\ (\mathsf{let}\, (x, y) = (v, w) \,\mathsf{in}\, e, \psi) &\rhd (e[v/x, w/y], \psi) \end{aligned}$$

(iv) Lastly, under reduction contexts, if $(\rho, \psi) \rhd (e, \psi')$ we define $(C[\rho], \psi) \rhd (C[e], \psi')$. If $(\rho, \psi) \rhd \bot$ then $(C[e], \psi) \rhd \bot$.

Proposition 17.3 *Every closed program $\vdash e : R^n$, together with the empty prior '!', deterministically reduces to either a configuration (v, ψ) or $\bot$.*

We consider the observable result of this execution either failure, or the pushforward distribution $v_*\psi$ on $\mathbb{R}^n$, as this distribution could be sampled from empirically.

Example 17.4 The program

$$\mathsf{let}\,(x, y) \;=\; (\mathrm{normal}(), \mathrm{normal}())\,\mathsf{in}\,x \mathrel{=:=} y; x + y$$

reduces to $(z_1 + z_2, \psi)$ where

$$\psi = \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}0.5 & 0.5\\0.5 & 0.5\end{pmatrix}\right)$$

The observable outcome of the run is the pushforward distribution $(1\,1)_*\psi = \mathcal{N}(0, 2)$ on $\mathbb{R}$.

One goal of this chapter is to study properties of this language compositionally, and abstractly, without relying on any specific properties of Gaussians. The crucial notion to investigate is contextual equivalence.

Definition 17.5 We say $\Gamma \vdash e_1, e_2 : \tau$ are *contextually equivalent*, written $e_1 \approx e_2$, if for all closed contexts $K[-]$ and $i, j \in \{1, 2\}$

(i) when $(K[e_i], !) \rhd^* (v_i, \psi_i)$ then $(K[e_j], !) \rhd^* (v_j, \psi_j)$ and $(v_i)_*\psi_i = (v_j)_*\psi_j$

(ii) when $(K[e_i], !) \rhd^* \bot$ then $(K[e_j], !) \rhd^* \bot$

We study contextual equivalence by developing a denotational semantics for the Gaussian language (Section 21.1), and proving it fully abstract (Theorem 21.2). We furthermore show that these semantics can be axiomatized completely by a set of program equations (Section 21). We also note nothing conceptually limits our language to only Gaussians. We are running with this example for concreteness, but any family of distributions which can be sampled and conditioned can be used. So we will take care to establish properties of the semantics in a general setting.

18 Synthetic Foundations of Conditioning

We'll now generalize the conditioning procedure from Gaussians to arbitrary Markov categories; the key synthetic ingredients are conditionals (Section 8.4), almost-sure equality and absolute continuity (Section 8.2). Following Convention 8.9, we will use the relation $x \ll \mu$ when we speak of supports and avoid the stronger but less widely applicable notion of Section 8.3.

Let $\mathbb{C}$ be a Markov category with all conditionals. In order to describe statistical inference categorically, we introduce the following terminology: An *observation* is a fixed piece of data, that is a *deterministic* state $o : I \to K$. An *inference problem* (K, ψ, o) is given by a joint distribution $\psi : I \to X \otimes K$ called the model and an observation $o : I \to K$. The problem is then to infer the posterior distribution over X conditioned on the observation o.

An inference problem can either succeed, or fail if the observation o is inconsistent with the model. We say (K, ψ, o) *succeeds* if the observation lies in the support of the model, i.e. $o \ll \psi_K$. In that case, a *solution* to the inference problem is the composite $\psi|_K \circ o : I \to X$ where $\psi|_K : K \to X$ is a conditional to ψ with respect to K. The solution is also referred to as a posterior. If $o \not\ll \psi_K$, we say that the inference problem *fails*.

Proposition 18.1 *Solutions to inference problems are unique, i.e. if (K, ψ, o) succeeds and $\psi|_K, \psi|'_K$ are two conditionals then $\psi|_K(o) = \psi|'_K(o)$.*

PROOF Because conditionals are not unique, we need to make use of the assumption $o \ll \psi_K$. It is however immediate from the definitions that conditionals are almost surely unique, i.e. $\psi|_K =_{\psi_K} \psi|'_K$. From the definition of absolute continuity, we derive $\psi|_K =_o \psi|'_K$, which implies $\psi|_K(o) = \psi|'_K(o)$. ∎

We call two inference problems *observationally equivalent* if they both fail, or they both succeed with equal posteriors. We remark that our treatment is *necessary* if suitable program equations like (59) and (60) are to hold: To a programmer, the inference problem (K, ψ, o) represents a closed program of the form

$$\mathsf{let}\,(x, k) \;=\; \psi\,\mathsf{in}\,(k{:=}\,o); x \tag{64}$$

where $k{:=}\,o$ is an exact conditioning operator. Finding a conditional for ψ amounts to restructuring the dataflow of (64) as

$$\mathsf{let}\,k \;=\; \psi_K\,\mathsf{in}\,\mathsf{let}\,x \;=\; \psi|_K(k)\,\mathsf{in}\,(k{:=}\,o); x$$

From commutativity and the initialization principle, we can simplify the problem as

$$(\mathsf{let}\,k \;=\; o\,\mathsf{in}\,\mathsf{let}\,x \;=\; \psi|_K(k)\,\mathsf{in}\,x) = \psi|_K(o)$$

if $k \ll \psi_K$, or obtain failure otherwise. This mirrors the symbolic approach of Shan and Ramsey [2017].

For the rest of this section, we will give concrete instances of this abstract machinery and show that it matches the conditioning procedure from Section 17. We begin by formalizing Gaussian probability in a Markov category:

Definition 18.2 (Fritz [2020, §6]) The symmetric monoidal category Gauss has objects $n \in \mathbb{N}$, which represent the affine space $\mathbb{R}^n$, and $m \otimes n = m + n$. Morphisms $m \to n$ are tuples (A, b, Σ) where $A \in \mathbb{R}^{n\times m}, b \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n\times n}$ is a positive semidefinite matrix. The tuple represents a stochastic map $f : \mathbb{R}^m \to \mathbb{R}^n$ that is affine-linear, perturbed with multivariate Gaussian noise of covariance Σ, informally written

$$f(x) = Ax + b + \mathcal{N}(\Sigma) \text{ or } Ax + \mathcal{N}(b, \Sigma)$$

Such morphisms compose sequentially and in parallel in the expected way, with noise accumulating independently

$$(A, b, \Sigma) \circ (C, d, \Xi) = (AC, Ad + b, A\Xi A^T + \Sigma)$$
$$(A, b, \Sigma) \otimes (C, d, \Xi) = \left(\begin{pmatrix} A & 0 \\ 0 & C \end{pmatrix}, \begin{pmatrix} b \\ d \end{pmatrix}, \begin{pmatrix} \Sigma & 0 \\ 0 & \Xi \end{pmatrix} \right)$$

In Gauss, we furthermore have ability to introduce correlations and discard values by means of the following affine maps, giving it Markov structure

$$\mathrm{copy}_n : \mathbb{R}^n \to \mathbb{R}^{n+n}, x \mapsto (x, x) \quad \mathrm{del}_n : \mathbb{R}^n \to \mathbb{R}^0, x \mapsto ()$$

Like BetaBern in Section 14, Gauss is a compact combinatorial characterization of a Markov subcategory of BorelStoch.

Proposition 18.3 *A morphism (A, b, Σ) in* Gauss *is deterministic iff $\Sigma = 0$, i.e. there is no randomness involved.*

PROOF Write $f = (A, b, \Sigma)$, then the covariance matrices of $f \circ \mathrm{copy}$ and $\mathrm{copy} \circ f$ are

$$\begin{pmatrix} \Sigma & 0 \\ 0 & \Sigma \end{pmatrix} \text{ and } \begin{pmatrix} \Sigma & \Sigma \\ \Sigma & \Sigma \end{pmatrix}$$

respectively. Thus f is copyable iff $\Sigma = 0$. ∎

It follows that the deterministic subcategory $\mathsf{Gauss}_{\mathrm{det}}$ is the category Aff consisting of the spaces $\mathbb{R}^n$ and affine maps between them.

Proposition 18.4 (Fritz [2020, 11.8]) Gauss *has all conditionals. This is known as the self-conjugacy of Gaussians [Jacobs, 2020]. The conditions can be computed by an explicit formula such as (61).*

Recall the notion of support for a Gaussian random variable. If $\mu = \mathcal{N}(b, \Sigma)$, we write $\mathsf{supp}(\mu)$ for the affine subspace $b + \mathsf{col}(\Sigma)$. This agrees with the categorical phrasing of support (Convention 8.9) and is in fact a representable support in the sense of Section 8.3, as we show now:

Proposition 18.5 *For a distribution $\mu : 0 \to m$ in* Gauss*, let $S = \mathsf{supp}(\mu)$ be its support. Then*

(i) If $f, g : m \to n$ are morphisms, then $f =_\mu g$ iff $fx = gx$ for all $x \in S$, seen as deterministic states $x : 0 \to m$.

(ii) If $\nu : 0 \to m$ then $\mu \ll \nu$ iff the support of μ is contained in the support of ν

(iii) In particular for $x : 0 \to m$ deterministic, $x \ll \mu$ iff $x \in S$.

PROOF The characterization (i) of μ-almost sure equality is a strengthening of Example 8.6: The maps $f, g : m \to n$ can be faithfully considered BorelStoch maps $f, g : \mathbb{R}^m \to \mathcal{G}(\mathbb{R}^n)$, so we have $f(x) = g(x)$ for μ-almost all x. Because f, g are continuous kernels and μ is equivalent to the Lebesgue measure on the support S, the equality almost everywhere can be strengthened to equality on all of S.

This immediately implies that the support condition is sufficient in (ii). To see that it is necessary, let $x \in \mathsf{supp}(\mu) \setminus \mathsf{supp}(\nu)$. Then we can find two affine functions f, g which agree on $\mathsf{supp}(\nu)$ but $f(x) \neq g(x)$. Now $f =_\nu g$ but not $f =_\mu g$, hence $\mu \not\ll \nu$. ∎

Supports also work as expected in FinStoch but not in BorelStoch.

Proposition 18.6 *In* FinStoch, *we have* $x \ll \mu$ *iff* $\mu(x) > 0$. *In* BorelStoch, *we have* $x \ll \mu$ *iff* $\mu(\{x\}) > 0$.

PROOF The arguments follow from [Fritz, 2020, 13.2,13.3]. For BorelStoch, if $\mu(\{x_0\}) = 0$ we consider the measurable functions $f(x) = [x = x_0], g(x) = 0$ and obtain $f =_\mu g$ but $f(x_0) \neq g(x_0)$ showing $x_0 \not\ll \mu$. This argument also shows that representable supports don't exist in BorelStoch. For that reason, we'll use $\ll$ the rest of this chapter. ∎

This means that in BorelStoch-probability, we can only condition on observations of positive probability; this agrees with the classical definition of conditional probability

$$P(A|B) \overset{\text{def}}{=} \frac{P(A \cap B)}{P(B)} \text{ if } P(B) > 0$$

In the smaller category Gauss, we can also condition on observations of probability zero. The dependence on the surrounding category is condensed in the following example.

Example 18.7 Let $\mu = \mathcal{N}(0,1)$ be the standard normal distribution. When considered in Gauss, its support is $\mathbb{R}$ and in particular for all $x_0 \in \mathbb{R}$ we have $x_0 \ll \mu$. In BorelStoch, there is *no* x_0 with $x_0 \ll \mu$.

We give an example of how to use the categorical conditioning machinery in practice.

Example 18.8 The statistical model from Example 17.1

$$X \sim \mathcal{N}(0,1);\ Y \sim \mathcal{N}(0,1);\ Z = X - Y$$

corresponds to the inference problem $\mu : 0 \to 3$ with covariance matrix

$$\Sigma = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \\ 1 & -1 & 2 \end{pmatrix}$$

and observation $Z = 0$. A conditional with respect to Z is

$$\mu|_Z(z) = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} z + \mathcal{N}\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

which can be verified by calculating (38). The marginal $\mu_Z = \mathcal{N}(2)$ is supported on all of $\mathbb{R}$, hence $0 \ll \mu_Z$ and by Proposition 18.1 the composite

$$\mu|_Z(0) = \mathcal{N}\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

is uniquely defined and represents the posterior distribution over (X, Y).

We close with the observation that the fragment of the Gaussian language without conditioning (=:=) can be interpreted in the internal language (Section 7) of the category Gauss. That is to say, there is a canonical denotational semantics of the Gaussian language where we interpret types and contexts as objects of Gauss, e.g. $[\![\mathrm{R}]\!] = 1$ and $[\![(x : \mathrm{R}, y : \mathrm{R} \otimes \mathrm{R})]\!] = 3$. Terms $\Gamma \vdash t : A$ are interpreted as stochastic maps $Ax + b + \mathcal{N}(\Sigma)$. This is all automatic once we recognize that addition $(+) : 2 \to 1$, scaling $\alpha \cdot (-) : 1 \to 1$, constants $\underline{\beta} : 0 \to 1$ and sampling $\mathcal{N}(1) : 0 \to 1$ are morphisms in Gauss which interpret the conditioning-free part of the signature (63).

In the next section, we will show that the full Gaussian language *with* conditioning (=:=) is the internal language of a CD category. The fact that commutativity holds is non-trivial. We note that the Gaussian language cannot reasonably be the internal language of a *Markov* category, because conditioning (=:=) is not discardable.

19 Compositional Conditioning – The Cond Construction

In the last section, we have seen that Markov categories with conditionals allow a general recipe for conditioning. In order to give compositional semantics to a language with conditioning, we need to internalize conditioning as a morphism, that is talk about *open inference problems* or *conditioning channels*.

Let $\mathbb{C}$ be a Markov category, then a conditioning channel $X \rightsquigarrow Y$ is given by a morphism $X \to Y \otimes K$ together with an observation (i.e. deterministic state) $o : I \to K$. This represents an intensional open program of the form

$$x : X \vdash \mathsf{let}\,(y, k) : Y \otimes K \;=\; f(x)\;\mathsf{in}\;(k{:=}\,o); y \tag{65}$$

We think of K as an additional hidden output wire, to which we attach the observation o. Such programs compose in the obvious way, by aggregating observations (Figure 5). Two representations (65) are deemed equivalent if they contextually equivalent, that is roughly they compute the same posteriors in all contexts.

For modularity, we present the construction in two stages: In the first stage (Section 19.1) we form a category $\mathsf{Obs}(\mathbb{C})$ on the same objects as $\mathbb{C}$ consisting of the data (65) but without any quotienting. This adds, purely formally, for every observation $o : I \to X$ a conditioning effect $(:= o) : X \rightsquigarrow I$. In the second stage (Section 19.2) – this is the core of the construction – we relate these morphisms to the conditionals present in $\mathbb{C}$, that is we quotient by contextual equivalence. The resulting quotient is called $\mathsf{Cond}(\mathbb{C})$. Under mild assumptions, this will have the good properties of a CD category, showing that conditioning stays commutative.

19.1 Obs – Open Programs with Observations

For ease of notation, we will assume $\mathbb{C}$ is a strictly monoidal category, that is all associators and unitors are identities (this poses no restriction by [Fritz, 2020, 10.17]). We note that all constructions can instead be performed purely string-diagrammatically.

Definition 19.1 The following data define a symmetric premonoidal category (Section 3.4) called $\mathsf{Obs}(\mathbb{C})$:

- the object part of $\mathsf{Obs}(\mathbb{C})$ is the same as $\mathbb{C}$
- morphisms $X \rightsquigarrow Y$ are tuples (K, f, o) where $K \in \mathrm{ob}(\mathbb{C})$, $f \in \mathbb{C}(X, Y \otimes K)$ and $o \in \mathbb{C}_{\mathrm{det}}(I, K)$, representing (65)
- The identity on X is $\mathrm{Id}_X = (I, \mathsf{id}_X, !)$ where $! = \mathsf{id}_I$.
- Composition is defined by

$$(K', f', o') \bullet (K, f, o) = (K' \otimes K, (f' \otimes \mathsf{id}_K) f, o' \otimes o).$$

- if $(K, f, o) : X \rightsquigarrow Y$ and $(K', f', o') : X' \rightsquigarrow Y'$, their (premonoidal) tensor product is defined as

$$(K' \otimes K, (\mathsf{id}_{Y'} \otimes \mathsf{swap}_{K',Y} \otimes \mathsf{id}_K)(f' \otimes f), o' \otimes o)$$

- There is an identity-on-objects functor $J : \mathbb{C} \to \mathsf{Obs}(\mathbb{C})$ that sends $f : X \to Y$ to $(I, f, !) : X \rightsquigarrow Y$. This functor is strict premonoidal and its image central
- $\mathsf{Obs}(\mathbb{C})$ inherits symmetry and comonoid structure

Recall that a symmetric premonoidal category is like a symmetric monoidal category where the interchange law (4) need not hold. This is the case because $\mathsf{Obs}(\mathbb{C})$ does not yet identify observations arriving in different order[16]. This will be remedied automatically later when passing to the quotient $\mathsf{Cond}(\mathbb{C})$. Composition and tensor can be depicted graphically as in Figure 5, where dashed wires indicate condition wires K and their attached observations o. For an observation $o : I \to K$, the conditioning effect $(:= o) : K \rightsquigarrow I$ is given by (I, id_K, o).

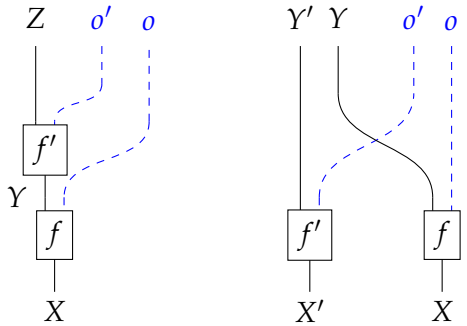


Figure 5: Composition and tensoring of morphisms in Obs

[16] see (101) in Section 30.5 for a graphical representation

19.2 Cond – Contextual Equivalence of Open Programs

Let us now assume that $\mathbb{C}$ has all conditionals. We wish to quotient Obs-morphisms, relating them to the conditionals which can be computed in $\mathbb{C}$. We know how to interpret *closed* programs, because a state $(K, \psi, o) : I \rightsquigarrow X$ is precisely an inference problem as in Section 18: If $o \not\ll \psi_K$, the observation does not lie in the support of the model and conditioning fails. If not, we form the conditional $\psi|_K$ in $\mathbb{C}$ and obtain a well-defined posterior $\mu|_K \circ o$.

This notion of observational equivalence defines an equivalence relation on states $I \rightsquigarrow X$ in $\mathsf{Cond}(\mathbb{C})$. We will extend this relation to a congruence on arbitrary morphisms $X \rightsquigarrow Y$ by a general categorical construction.

Definition 19.2 Given two states $I \rightsquigarrow X$ we define $(K, \psi, o) \sim (K', \psi', o')$ if they are observationally equivalent as inference problems, that is either

(i) $o \ll \psi_K$ and $o' \ll \psi'_{K'}$ and $\psi|_K(o) = \psi'|_{K'}(o')$.

(ii) $o \not\ll \psi_K$ and $o' \not\ll \psi'_{K'}$

That is, both conditioning problems either fail, or both succeed with equal posterior.

Figure 6 reformulates Example 18.8 as an equivalence in $\mathsf{Obs}(\mathsf{Gauss})$.

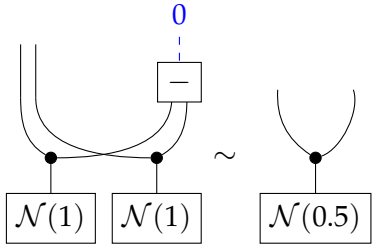


Figure 6: Example 18.8 describes observationally equivalent states $0 \rightsquigarrow 2$

We now give a general recipe to extend an equivalence relation on states to a congruence on arbitrary morphisms $f : X \to Y$.

Definition 19.3 Let $\mathbb{X}$ be a symmetric premonoidal category. An equivalence relation $\sim$ on states $\mathbb{X}(I, -)$ is called *functorial* if $\psi \sim \psi'$ implies $f\psi \sim f\psi'$. We can extend such a relation to a congruence $\approx$ on all morphisms $X \to Y$ via

$$f \approx g \Leftrightarrow \forall A, \psi : I \to A \otimes X, (\mathsf{id}_A \otimes f)\psi \sim (\mathsf{id}_A \otimes g)\psi.$$

The quotient category $\mathbb{X}/\approx$ is symmetric premonoidal.

We show now that under good assumptions, the quotient by conditioning (Definition 19.2) on $\mathbb{X} = \mathsf{Obs}(\mathbb{C})$ is functorial, and induces a quotient category $\mathsf{Cond}(\mathbb{C})$. The technical condition is that supports interact well with dataflow

Definition 19.4 A Markov category $\mathbb{C}$ has *precise supports* if the following are equivalent for all deterministic $x : I \to X$, $y : I \to Y$, and arbitrary $f : X \to Y$ and $\mu : I \to X$.

(i) $x \otimes y \ll \langle \mathsf{id}_X, f \rangle \mu$

(ii) $x \ll \mu$ and $y \ll fx$

Caveat: Precise supports refers to the relation $x \ll \mu$ according to Convention 8.9, and does not presuppose the existence of representable supports (Section 8.3). The category BorelStoch does not have representable supports, but still satisfies Definition 19.4.

Proposition 19.5 Gauss, FinStoch *and* BorelStoch *have precise supports.*

PROOF For Gauss, this follows from the characterization of $\ll$ in Proposition 18.5. Let μ have support S and $f(x) = Ax + \mathcal{N}(b, \Sigma)$. Let T be the support of $\mathcal{N}(b, \Sigma)$. The support of $\langle \mathsf{id}, f \rangle \mu$ is the image space $\{(x, Ax + c) : x \in S, c \in T\}$. Hence $(x, y) \ll \langle \mathsf{id}, f \rangle \mu$ iff $x \ll \mu$ and $y \ll fx$.

For FinStoch, an outcome (x, y) has positive probability under $\langle \mathsf{id}, f \rangle \mu$ iff x has positive probability under μ, and y has positive probability under $f(-|x)$.

For BorelStoch, the measure $\psi = \langle \mathsf{id}, f \rangle \mu$ is given by

$$\psi(A \times B) = \int_{x \in A} f(B|x) \mu(\mathrm{d}x)$$

Hence $\psi(\{(x_0, y_0)\}) = f(\{y_0\}|x)\mu(\{x\})$, which is positive exactly if $\mu(\{x_0\}) > 0$ and $f(\{y_0\}|x) > 0$. ∎

Theorem 19.6 *Let $\mathbb{C}$ be a Markov category that has conditionals and precise supports. Then $\sim$ is a functorial equivalence relation on* $\mathsf{Obs}(\mathbb{C})$.

PROOF Let $(K, \psi, o) \sim (K', \psi', o') : I \rightsquigarrow X$ be equivalent states and $(H, f, v) : X \rightsquigarrow Y$ be any morphism. We need to show that the composites

$$(H \otimes K, (f \otimes \mathsf{id}_K)\psi, v \otimes o) \sim (H \otimes K', (f \otimes \mathsf{id}_{K'})\psi', v \otimes o') \tag{66}$$

are equivalent. We analyze different cases.

The states fail If a state (K, ψ, o) fails because $o \not\ll \psi_K$, then any composite must fail too, because we can apply Proposition 9.11 to the marginalization map recovering o. So both sides of (66) fail and are thus equivalent. Note that the causality assumption is automatic in this chapter, because $\mathbb{C}$ is assumed to have conditionals (Proposition 9.13).

The composite fails Assume from now that the states succeed and thus also have equal posteriors

$$\psi|_K(o) = \psi'|_K(o') \tag{67}$$

We first show that the success conditions on both sides of (66) are the same, so if the LHS fails so does the RHS. The "precise supports" axiom lets us split the success condition into two statements; that is the following are equivalent (and analogous for ψ', o'):

(i) $v \otimes o \ll (f_H \otimes \mathsf{id}_K)\psi$

(ii) $o \ll \psi_K$ and $v \ll f_H \psi|_K(o)$

To see this, we instantiate Definition 19.4 with the morphisms $\mu = \psi_K$ and $g = f_H \circ \psi|_K$, because the definition of the conditional $\psi|_K$ lets us recover

$$\langle g, \mathsf{id}_K \rangle \mu = (f_H \otimes \mathsf{id}_K)\psi.$$

It is clear that condition (ii) agrees for both sides of (66). Hence so does (i).

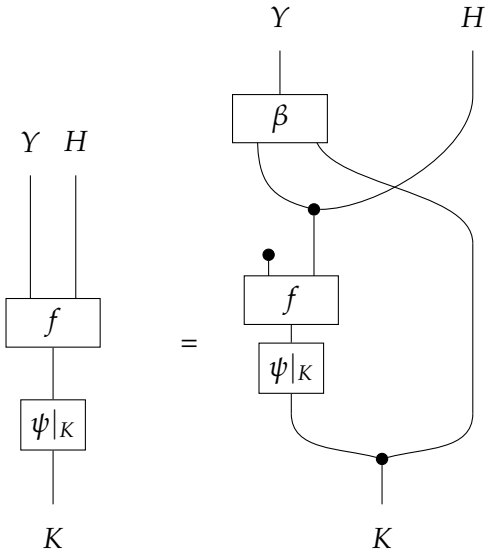
The composite succeeds We are left with the case that both sides of (66) succeed, and need to show that the composite posteriors agree

$$[(f \otimes \mathsf{id}_K)\psi]|_{H\otimes K}(v \otimes o) = [(f \otimes \mathsf{id}_{K'})\psi']|_{H\otimes K'}(v \otimes o') \tag{68}$$

We use a variant of the argument from [Fritz, 2020, 11.11] that double conditionals can be replaced by iterated conditionals. Consider the parameterized conditional

$$\beta \stackrel{\text{def}}{=} (f \circ \psi|_K)|_H : H \otimes K \to Y$$

with universal property

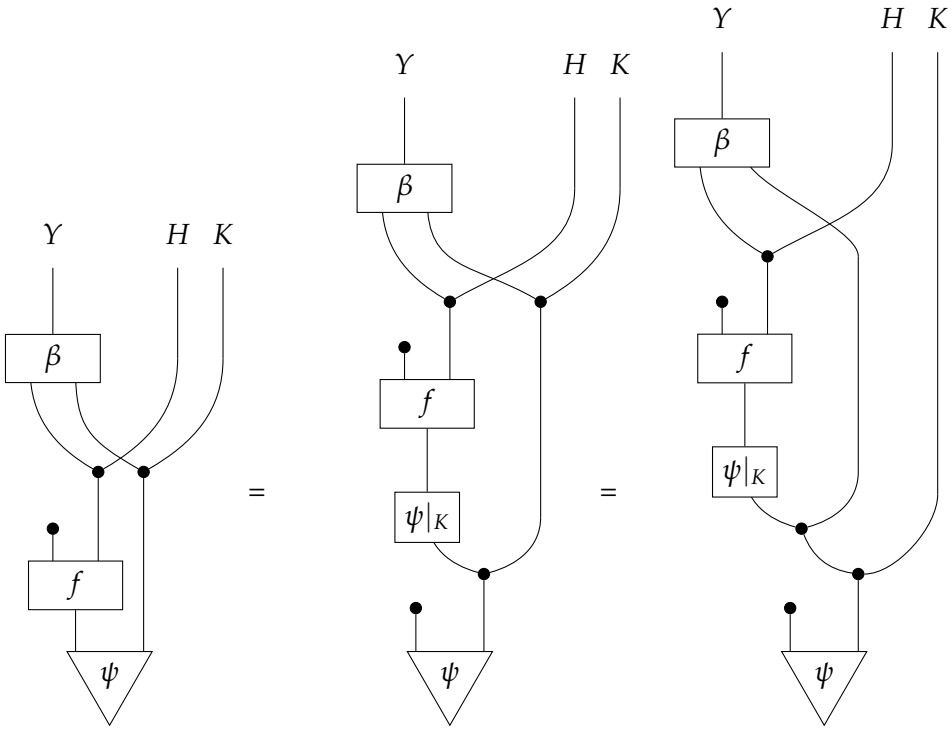


(69)

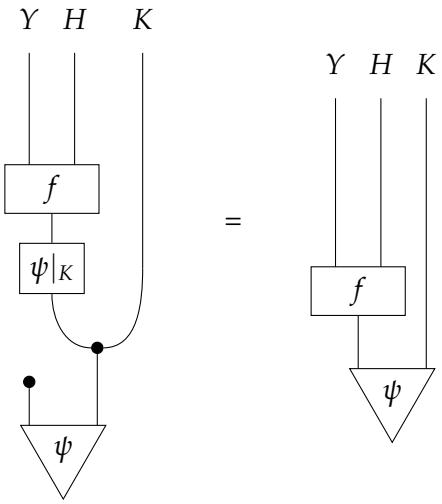
Some string diagram manipulation shows that β too has the universal property of the double conditional

$$\beta = [(f \otimes \mathsf{id}_K)\psi]|_{H\otimes K}$$

We check



which further reduces using (69) to the desired



By specialization (Proposition 8.14), we can fix one observation o in β to obtain a conditional

$$\beta(\mathsf{id}_H \otimes o) = (f \circ \psi|_K(o))|_H \tag{70}$$

But this conditional agrees with $(f \circ \psi'|_K(o'))|_H$ by assumption (67). Hence we can evaluate the joint posterior successively,

$$\begin{aligned}
[(f \otimes \mathsf{id}_K)\psi]|_{H\otimes K}(v \otimes o) &= \beta(\mathsf{id}_H \otimes o) \circ v \\
&\overset{(70)}{=} (f \circ \psi|_K(o))|_H \circ v \\
&\overset{(67)}{=} (f \circ \psi'|_{K'}(o'))|_H \circ v \\
&\overset{\text{symmetric}}{=} [(f \otimes \mathsf{id}_{K'})\psi']|_{H\otimes K'}(v \otimes o)
\end{aligned}$$

establishing (68). ∎

We can spell out the equivalence $\approx$ as follows:

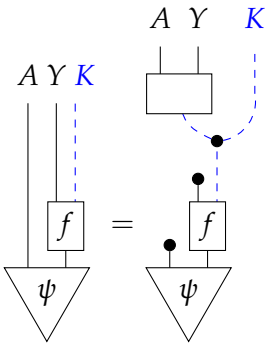
Proposition 19.7 *We have $(K, f, o) \approx (K', f', o') : X \rightsquigarrow Y$ if and only if for all $\psi : I \to A \otimes X$, either*

(i) $o \ll f_K\psi_X$ and $o' \ll f'_{K'}\psi'_X$ and $[(\mathsf{id}_A \otimes f)\psi]|_K(o) = [(\mathsf{id}_A \otimes f')\psi']|_{K'}(o')$

(ii) $o \not\ll f_K\psi_X$ and $o' \not\ll f'_{K'}\psi'_X$

Furthermore, because $\mathbb{C}$ has conditionals, it is sufficient to check these conditions for $A = X$ and ψ of the form $\mathrm{copy}_X \circ \phi$.

The universal property of the conditional in question is



We can show that isomorphic conditions are equivalent under the relation $\approx$.

Proposition 19.8 (Isomorphic conditions) *Let $(K, f, o) : X \rightsquigarrow Y$ and $\alpha : K \cong K'$ be an isomorphism. Then*

$$(K, f, o) \approx (K', (\mathsf{id}_Y \otimes \alpha)f, \alpha o).$$

In programming terms $(k := o) \approx (\alpha k := \alpha o)$.

PROOF Let $\psi : I \to A \otimes X$. We first notice that $o \ll \psi_K$ if and only if $\alpha o \ll \alpha\psi_K$, so the success conditions coincide. It is now straightforward to check the universal property

$$(\mathsf{id}_A \otimes f)\psi|_K = (\mathsf{id}_A \otimes ((\mathsf{id}_X \otimes \alpha)f))\psi|_{K'} \circ \alpha.$$

This requires the fact that isomorphisms are deterministic (Proposition 9.9). The proof works more generally if α is deterministic and split monic. ∎

We can now give the Cond construction:

Definition 19.9 Let $\mathbb{C}$ be a Markov category that has conditionals and precise supports. We define $\mathsf{Cond}(\mathbb{C})$ as the quotient

$$\mathsf{Cond}(\mathbb{C}) = \mathsf{Obs}(\mathbb{C})/\approx$$

This quotient is a CD category, and the functor $J : \mathbb{C} \to \mathsf{Cond}(\mathbb{C})$ preserves CD structure.

PROOF We have checked functoriality of $\sim$ in Theorem 19.6, so by Definition 19.3, the quotient is symmetric premonoidal. It remains to show that the interchange laws holds, i.e. observations can be reordered. But this follows from Proposition 19.8 because swapping is an isomorphism. ■

Proposition 19.10 *By virtue of being a well-defined CD category, the commutativity equation hold in the internal language of* $\mathsf{Cond}(\mathbb{C})$.

We have two diagrammatic languages at our disposal. One is string diagrams in the Markov category $\mathbb{C}$ with blue condition wires around. The other one takes place in the CD category $\mathsf{Cond}(\mathbb{C})$, where conditions are present as actual effects. We will tacitly invoke J to lift diagrams in $\mathbb{C}$ to (conditioning-free) diagrams in $\mathsf{Cond}(\mathbb{C})$.

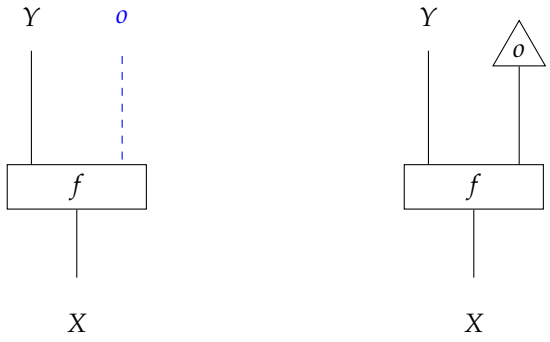
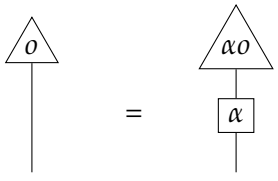


Figure 7: Two diagrammatic representations of an open conditioning program. On the left using conditioning wires, on the right using the conditioning effects in $\mathsf{Cond}(\mathbb{C})$

Example 19.11 Proposition 19.8 states diagrammatically that for all isomorphisms α and observations o, we have



19.3 Laws for Conditioning

We derive some properties of $\mathsf{Cond}(\mathbb{C})$. We firstly notice that J is faithful for common Markov categories.

Proposition 19.12 *Morphisms $f, g : X \to Y$ are equated via $J(f) \approx J(g)$ if and only if*

$$\forall \psi : I \to A \otimes X, (\mathsf{id}_A \otimes f)\psi = (\mathsf{id}_A \otimes g)\psi \tag{71}$$

In particular, J is faithful for Gauss, FinStoch *and* BorelStoch.

PROOF Directly from the definition of $\approx$. ∎

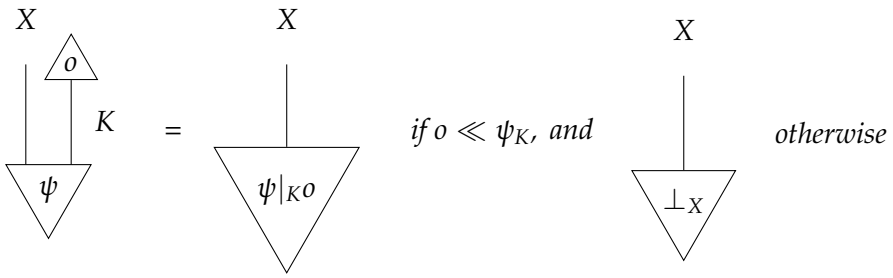
Note that equation (71) is stronger than equality on points: If $J(f) \approx J(g)$ then f and g are almost surely equal with respect to all distributions. In particular $fx = gx$ for all $x : I \to X$. This means J is faithful in all Markov categories $\mathbb{C}$ where I is a separator.

We can classify the states in $\mathsf{Cond}(\mathbb{C})$, which correspond precisely to inference problems. Any such problem either fails or computes a well-defined posterior.

Proposition 19.13 (States in Cond) *The states $I \rightsquigarrow X$ in $\mathsf{Cond}(\mathbb{C})$ are of the following form:*

(i) There is a unique failure state $\bot_X : I \rightsquigarrow X$ given by any (K, ψ, o) with $o \not\ll \psi_K$.[17]

(ii) Any other state is equal to a conditioning-free posterior, namely $(K, \psi, o) \approx J(\psi|_K \circ o)$. That is diagrammatically

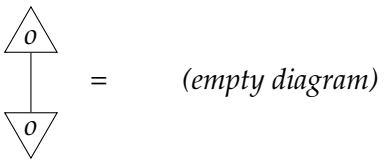


(iii) Failure is "strict" in the sense that any composite or tensor with $\bot$ gives $\bot$.

(iv) The only scalars $I \rightsquigarrow I$ are id_I and $\bot_I$. Both are copyable, but $\bot_I$ is not discardable.

PROOF By definition of $\sim$. ∎

Corollary 19.14 *If $o \ll \psi$ then $(\psi := o) \approx \mathsf{id}_I$ succeeds without any effect; in particular, because $o \ll o$, we can eliminate tautological conditions*



The most important law states that after we enforce a condition, it will hold with exactness. In programming terms, this is the substitution principle (59). Categorically, we are asking how the conditioning effect interacts with copying:

[17] it is a minor extra assumption that there exists a non-instance $o \not\ll \mu$ in $\mathbb{C}$; this should be the case in any Markov category of practical interest

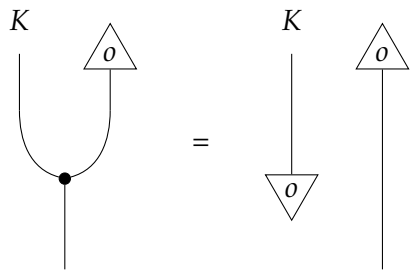
Proposition 19.15 (Enforcing conditions) *We have*

$$(X, \mathrm{copy}_X, o) \approx (X, o \otimes \mathsf{id}_X, o)$$

In programming notation

$$(x{:=}\, o); x \approx (x{:=}\, o); o$$

and expressed in $\mathsf{Cond}(\mathbb{C})$



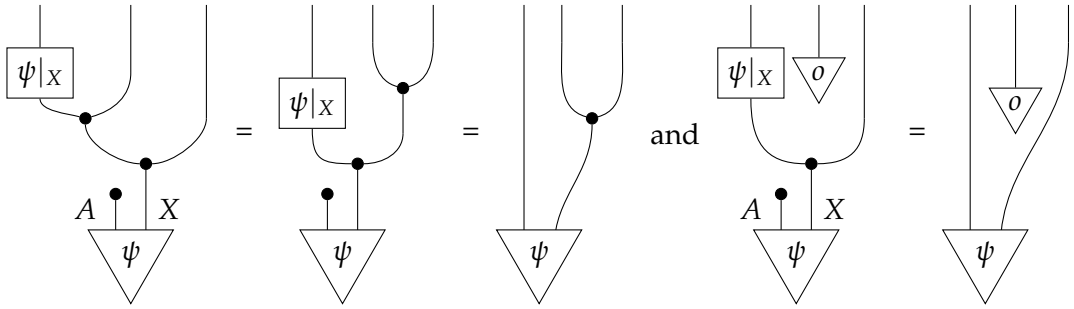
(72)

Note that the conditioning effect *cannot* be eliminated; however after the condition takes place, the other wire can be assumed to contain o.

PROOF Let $\psi : I \to A \otimes X$; the success condition reads $o \ll \psi_X$ both cases. Now let $o \ll \psi_X$ and let $\psi|_X$ be a conditional distribution for ψ. The following maps give the required conditionals

$$[(\mathsf{id}_A \otimes \mathrm{copy}_X)\psi]|_X = \langle \psi|_X, \mathsf{id}_X \rangle \quad [(\mathsf{id}_A \otimes o \otimes \mathsf{id}_X)\psi]|_X = \psi|_X \otimes o$$

as evidenced by the following string diagrams



Composing with o, we obtain the desired equal posteriors

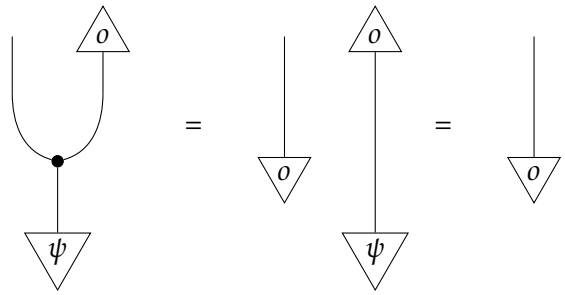
$$\langle \psi|_X, \mathsf{id}_X \rangle o = \psi|_X(o) \otimes o = (\psi|_X \otimes o)(o)$$

from determinism of o. ∎

Corollary 19.16 (Initialization) *Conditioning a fresh variable on a feasible observation makes it assume that observation. Formally, if $o \ll \psi$ then*

$$(\mathsf{let}\, x \;=\; \psi\, \mathsf{in}\, (x{:=}\, o); x) \approx o$$

PROOF Combining Proposition 19.15 and Corollary 19.14, we have



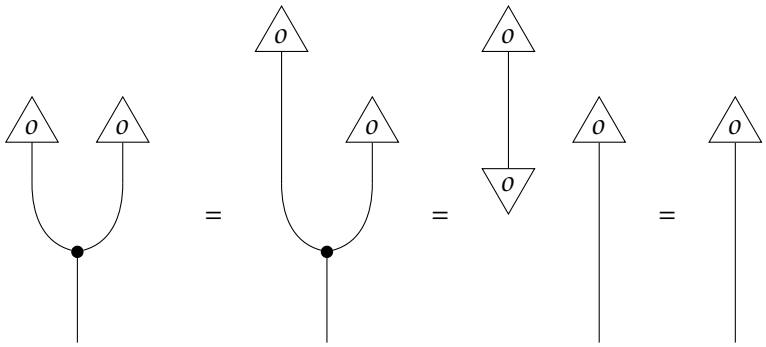
■

Corollary 19.17 (Idempotence) *Conditioning is idempotent, that is*

$$(x := o); (x := o) \approx (x := o)$$

In other words, the conditioning effect is copyable (but not discardable).

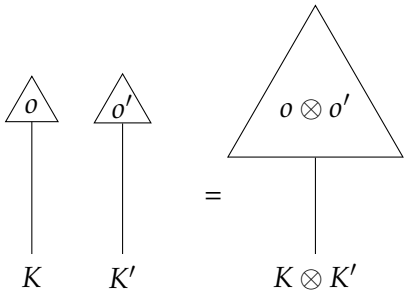
PROOF Again by Proposition 19.15 and Corollary 19.14 we obtain



■

We note that this does not imply that *every* effect in $\mathsf{Cond}(\mathbb{C})$ is copyable, e.g. **observe** statements are are not (Section 20.1).

Proposition 19.18 (Aggregation) *Conditions can be aggregated*

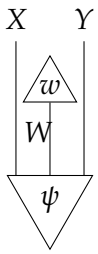


PROOF By definition of the monoidal structure of Obs. ■

We demonstrate the power of our conditioning laws by showing that conditioning behaves as expected for observed variables in a statistical model.

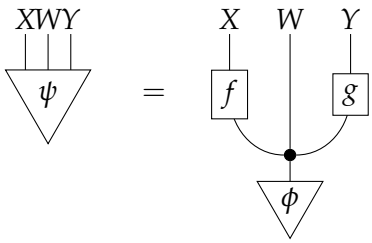
Example 19.19 Let $\psi : I \to X \otimes W \otimes Y$ display the conditional independence $X \perp Y \mid W$, then once $W := w$ is observed, X and Y become independent.

PROOF The conditioned distribution is given by the $\mathsf{Cond}(\mathbb{C})$-state

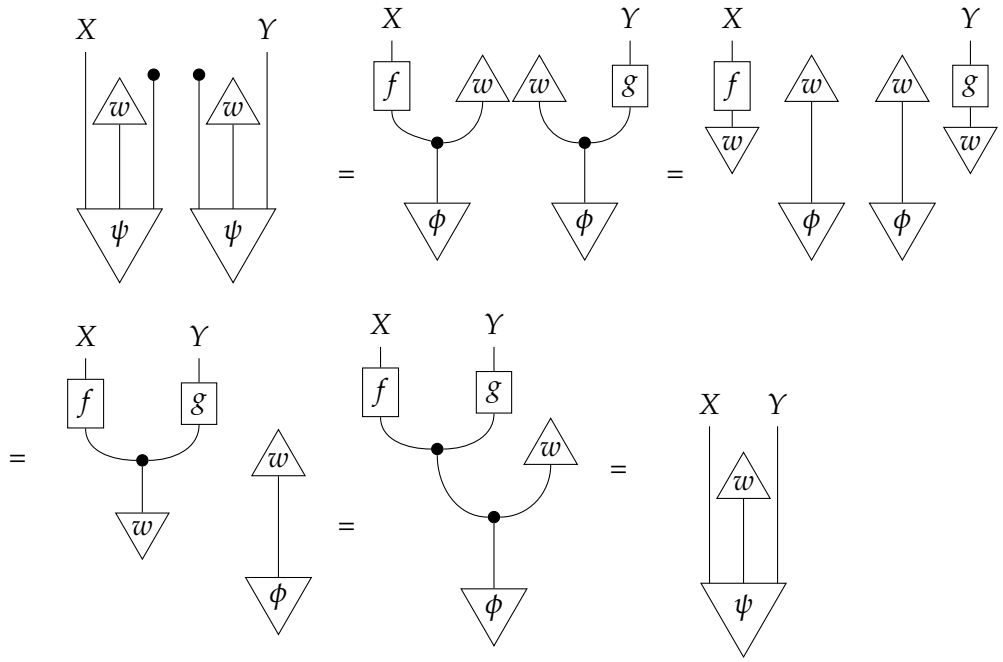


(73)

By the assumption of conditional independence, we can find maps factorizing ψ as follows



We are now able to derive that (73) decomposes as the product of its marginals:



where we repeatedly apply Proposition 19.15, idempotence of scalars and determinism of w. ∎

Case study: Finite probability The Cond construction allows us to work with conditioning channels in any well-behaved setting. We conclude with an important consistency check, namely that the Cond construction reduces to a well-known concept for finite probability:

A *subprobability distribution* on a set X is a formal linear combination $\sum_i p_i[x_i]$ with $p_i \in [0,1]$ and $\sum_i p_i \leq 1$. Similarly a *subprobability kernel* $X \rightsquigarrow Y$ (or *substochastic matrix*) is a kernel $p(y|x)$ such that $\forall x, \sum_y p(y|x) \leq 1$. Subprobability kernels form a CD category that is a natural domain for probabilistic computation with scoring (of scores ≤ 1). We'll now show $\mathsf{Cond}(\mathsf{FinStoch})$ essentially consists of subprobability kernels plus automatic normalization.

A conditioning channel $Q : X \rightsquigarrow Y$ is presented by a finite set Z, a probability kernel $q(y,z|x)$ and an observation $z_0 \in Z$. We associate to Q the subprobability kernel $\rho_Q : X \to Y$ given by the likelihood

$$\rho_Q(y|x) = q(y, z_0|x)$$

Conversely, we can associate to every subprobability kernel $\rho : X \to Y$ a conditioning channel Q_ρ with two possible observations $b \in \{0,1\}$ corresponding to $\rho(y|x)$ and its complementary probability, i.e.

$$q_\rho(y, b|x) = b \cdot \rho(y|x) + (1-b) \cdot (1 - \rho(y|x)).$$

Every conditioning channel Q can be recovered from the subprobability kernel ρ_Q in this way: Given any distribution $p(a,x)$, the relevant posterior in Proposition 19.7 is given by

$$\frac{\sum_x p(a,x) q(y, z_0|x)}{\sum_{a,x,y} p(a,x) q(y, z_0|x)}$$

where the denominator is nonzero iff the support condition is satisfied. We can see that using q_ρ computes the same expression as q, because

$$\sum_x p(a,x) q_\rho(y, 1|x) = \sum_x p(a,x) \cdot \rho(y|x) = \sum_x p(a,x) q(y, z_0|x).$$

However $\mathsf{Cond}(\mathsf{FinStoch})$ is *not quite* equivalent to the category of subprobability kernels, because subprobability computation needs not be normalized, while the Cond-construction takes care of normalization automatically. Formally, two subprobability kernels $\rho, \rho' : X \to Y$ may give rise to the same conditioning channel. By the characterization in Proposition 19.7, they do if and only if for all distributions $p(x)$ we have

$$\forall x \forall y, \quad \frac{p(x)\rho(y|x)}{\sum_{x,y} p(x)\rho(y|x)} = \frac{p(x)\rho'(y|x)}{\sum_{x,y} p(x)\rho'(y|x)}$$

Choosing p to be a uniform distribution, we obtain

$$\forall x \forall y, \quad \frac{\rho(y|x)}{\sum_{x,y} \rho(y|x)} = \frac{\rho'(y|x)}{\sum_{x,y} \rho'(y|x)}$$

Either both support conditions fail, that is $\rho = \rho' = 0$, or both denominators are positive numbers. In either case, ρ, ρ' are proportional to each other, i.e. there exists a nonzero constant λ such that $\rho(y|x) = \lambda \rho'(y|x)$. Up to the constant λ, it doesn't matter that the kernels are substochastic, because any nonnegative kernel can be rescaled; we arrive at the following characterization:

Proposition 19.20 $\mathsf{Cond}(\mathsf{FinStoch})$ *is equivalent to the CD category of* projectivized nonnegative matrices, *that is*

(i) objects are finite sets X

(ii) morphisms $X \to Y$ are equivalence classes $[A]$ of nonnegative matrices $A \in [0,\infty)^{Y \times X}$ where $[A] = [B]$ iff $A = \lambda B$ for some $\lambda \neq 0$.

The idea of considering distributions up to a constant is not novel, see for example [Gromov, 2014]. We illustrate a concrete instance of Proposition 19.13:

Example 19.21 A projectivized subprobability kernel $p : X \to Y$ is discardable if and only if there exists a constant $\lambda \neq 0$ such that

$$\forall x, \sum_y p(y|x) = \lambda$$

In particular, the states $1 \to X$ are either a normalizable distribution $p \in \mathsf{FinStoch}(1, X)$ or the failure kernel $\bot_X = 0$.

20 Conditioning on Equality

Cond gives a canonical solution to the problem of conditioning variables on constant observations $(x{:=}\, o)$. In some cases such as the Gaussian language, we want to do more and condition arbitrary expressions on equality *with each other* $(s \;=\!:\!=\; t)$. For the Gaussian case, there is an obvious way to reduce this to the previous case by setting the difference $s - t$ equal to zero. This is already a morphism $(=\!:\!=) : 2 \rightsquigarrow 0$ in $\mathsf{Cond}(\mathsf{Gauss})$, written

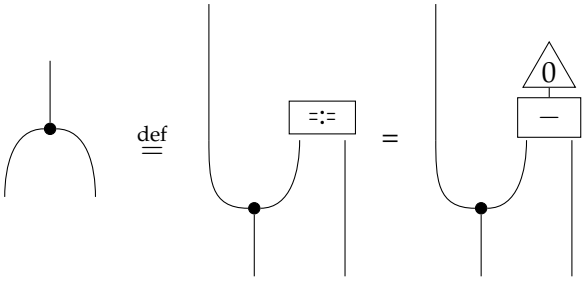
$$(x \;=\!:\!=\; y) \overset{\text{def}}{=} (x - y{:=}\, 0)$$

Similarly in finite probability, we can explain $x \;=\!:\!=\; y$ by setting their boolean equality test to true,

$$(x \;=\!:\!=\; y) \overset{\text{def}}{=} ([x = y]{:=}\, \text{true})$$

Unlike the Cond construction, which is completely canonical, it seems to us that the way of reducing binary conditioning to a constant observation requires an additional choice. After all, explaining $x \;=\!:\!=\; y$ by $x/y \;=\!:\!=\; 1$ famously leads to different results (Example 17.2).

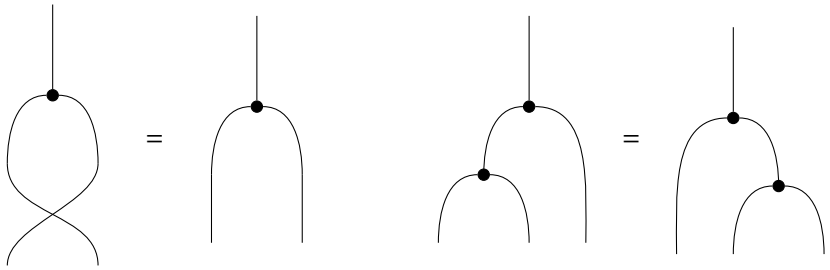
In any case, a choice of binary conditioning effect gives rise additional rich structure on $\mathsf{Cond}(\mathbb{C})$ which we'll sketch now, using Gaussian probability as an example: First, we introduce another operation $\epsilon : 1 \otimes 1 \rightsquigarrow 1$ by setting two wires equal and returning either output.



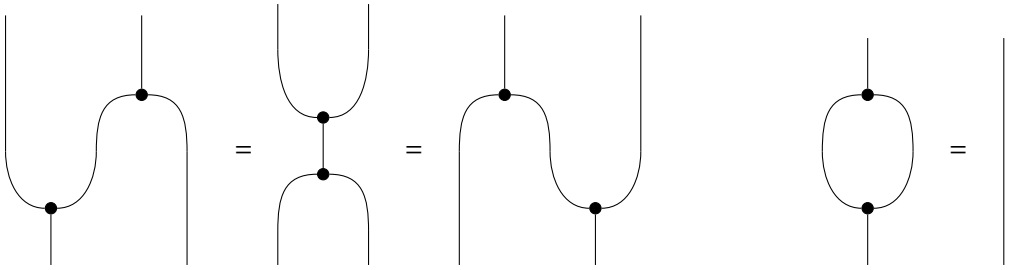
We chose to return a copy of the left input wire, but this does not matter because $(=:=)$ is symmetric (by Proposition 19.8). We can now show the following:

Proposition 20.1 *The conditioning operation has the structure of a commutative special Frobenius semigroup. That is*

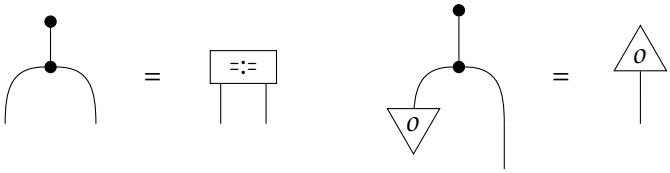
Associativity, Symmetry



Frobenius Law + Specialness condition

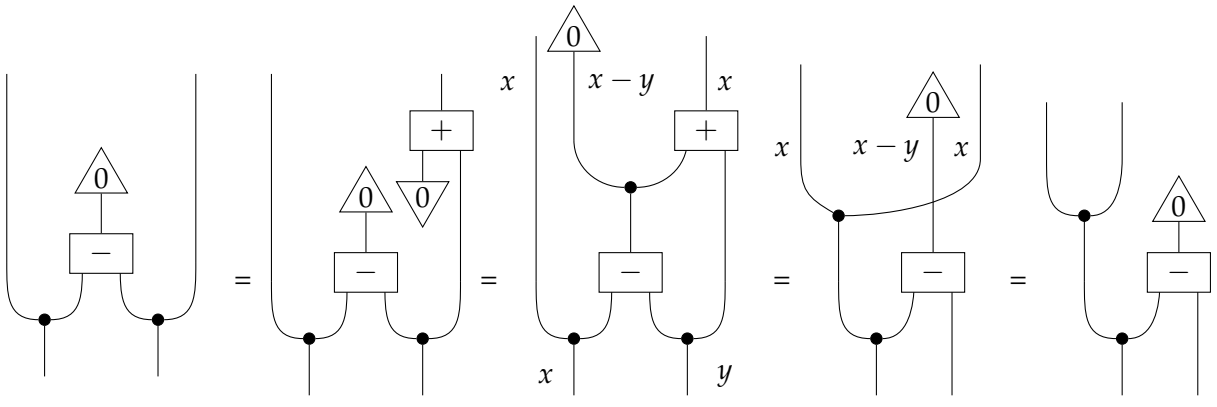


Furthermore we have the following identities where o is any deterministic observation.

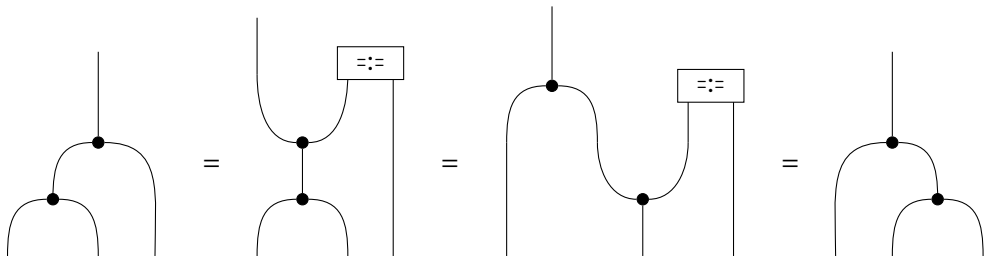


(74)

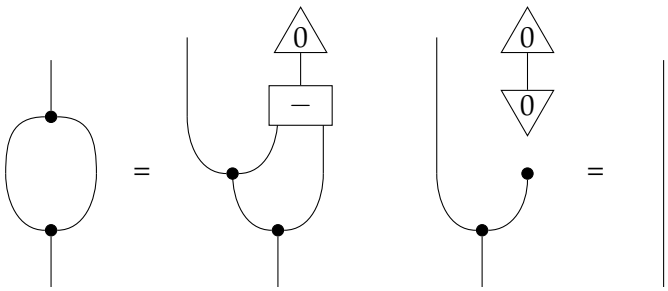
PROOF Symmetry is immediate because $(=:=)$ is symmetric and (74) follow from the comonoid laws and Proposition 19.8. The left equation of the Frobenius law holds by construction while the right equation requires some thought: The idea is that we can write $x = (x - y) + y$ and apply Proposition 19.15 to the condition $(x - y) := 0$ to obtain $x = y$. In string diagrams (where we've added variables to keep track of the arithmetic)



From the Frobenius law, we can easily derive the associativity axiom



The special law amounts to removing the tautological condition $0 := 0$ (Corollary 19.14)



■

The proofs can be adapted to $\mathsf{Cond}(\mathsf{FinStoch})$ by using if-then-else in place of addition and subtraction. The Frobenius structure on each object X is given by the subprobability kernel $\epsilon_X : X \times X \rightsquigarrow X$ defined as

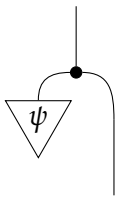
$$\epsilon_X(z|x,y) = [x = y = z] \tag{75}$$

In $\mathsf{Cond}(\mathsf{Gauss})$, we can similarly extend ϵ to every object by conditioning componentwise.

Frobenius algebras feature prominently in the analysis of *classical information flow* in quantum computation [Heunen and Vicary, 2019] and have been linked to Bayesian inference before [Coecke and Spekkens, 2012]. Our analysis of conditioning in Markov categories recovers this structure from first principles and produces a novel class of examples such as $\mathsf{Cond}(\mathsf{Gauss})$. Frobenius algebras also appear as equality tests in categories of relations [Baez and Erbele, 2015; Baez et al., 2018] and unification in logic programming (Section 20.2). We conjecture that the Frobenius structures in probability simplify to these examples under the possibilistic collapse (Proposition 5.4).

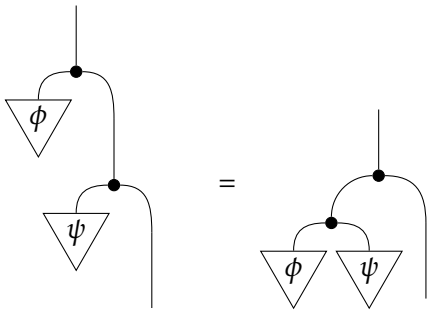
20.1 Scoring

We now have a formal framework to talk about soft conditions, i.e. `score` statements. A *synthetic soft condition* is any morphism of the form



(76)

that is we condition on equality with an independent variable. In programs, this amounts to writing $x =:= \psi()$. By associativity, soft conditions combine as follows



(77)

Soft conditions are the synthetic analogues of scoring. In $\mathsf{Cond}(\mathsf{FinStoch})$, they are given by actual scoring: Given a subprobability distribution $\psi(x)$, the morphism $\rho : X \to X$ defined by (76) is given by multiplication with $\psi(x)$, i.e.

$$\rho(y|x) = \psi(x)[y = x]$$

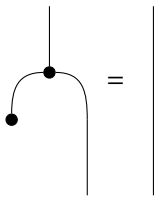
One can also show that every effect $X \to 1$ in $\mathsf{Cond}(\mathsf{FinStoch})$ is actually scoring with a unique subdistribution $1 \to X$ (this is an automatic consequence of the duality in Section 20.2).

In this way, soft conditions generalize working with likelihoods and densities. If a distribution is already presented by a density then the associativity formula (77) means these densities can be multiplied. However we notice that the Cond-construction generalizes to categories like Gauss which have no explicit representations of densities. We purely work with the structure of the underlying Markov category and don't presuppose a surrounding CD category like SfKer. This is reminiscent of the treatment of likelihoods and 'channels represented by an effect' in [Cho and Jacobs, 2019] and more recently [St Clere Smithe, 2020].

20.2 Aside on Uninformative Priors and Frobenius Units

> *If there is not enough symmetry and one can not* postulate *equiprobability (and/or something of this kind such as* independence*) of certain "events", then the advance of the classical calculus stalls [...]*
>
> MIKHAIL GROMOV, Six Lectures on Probability, Symmetry, Linearity

An interesting point is that our binary condition operation is almost a Frobenius algebra, except there need not be a *unit* for conditioning which satisfies



According to the reading of soft conditions, observing from the unit doesn't update the prior and thus introduces no new information. Conversely, starting from a unit prior, we will immediately accept new information without bias. In this way, the unit forms an ideal uniform distribution or a unbiased prior.

Because conditioning on the unit always succeeds, its support must be the whole space. One can show that in $\mathsf{Cond}(\mathsf{FinStoch})$, the unit on X is given by the uniform distribution

$$u(x) = \frac{1}{|X|}$$

In $\mathsf{Cond}(\mathsf{Gauss})$, there is no unit: Every Gaussian distribution has a possibly slight bias towards its mean, and scoring it with a sufficiently flat Gaussian will recover that mean in the limit. There is no uniform probability distribution on $\mathbb{R}$.

In statistics, the Lebesgue measure is sometimes employed as an *improper prior* even though it is unnormalized. This problem disappears in the calculation as soon as we condition on another distribution. A more formally appealing approach to improper priors would be to find an extension of say Gauss by limits to add "formal uniform distributions", which then act as units in the Cond construction. An interesting example of such a synthetic uniform prior is given by the allocation of a fresh logic variable in unification [Staton, 2013a]. Here, the type of terms has a Frobenius algebra structure given by unification

$$\begin{aligned} \epsilon &: \mathsf{term} \times \mathsf{term} \to \mathsf{term} \\ \epsilon(u, v) &= (u =:= v); u \end{aligned}$$

and the following equation, which is derivable from the axioms in Staton's paper, states that $\exists$ is a unit for this structure

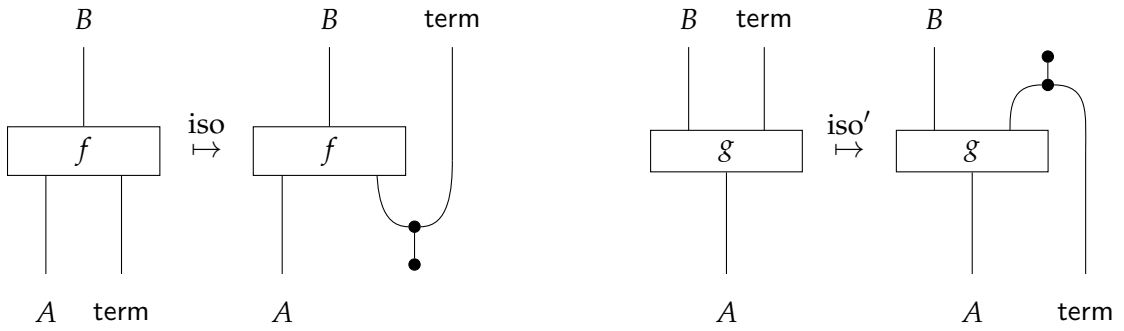
$$b \mid \varphi : 1 \vdash \exists a.(a =:= b)\varphi[a] \equiv \varphi[b] \tag{78}$$

The type isomorphisms from Section 3 of that paper

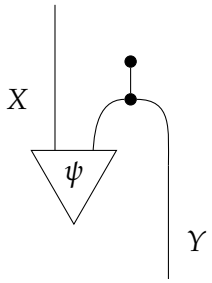
```
val iso : (α * term -> β) -> (α -> β * term)
let iso f a = let u = free() in (f(a,u),u)

val iso′ : (α -> β * term) -> (α * term -> β)
val iso′ g (a,u) = let (b,v) = g(a) in u =:= v; b
```

translate to the well-known bending of wires for the self-duality (e.g. [Heunen and Vicary,

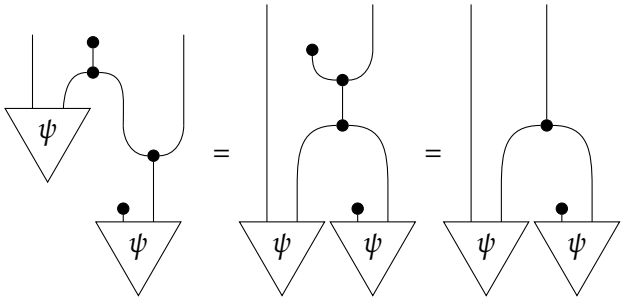
2019]) of term induced by its Frobenius structure



Perhaps surprisingly, that we cannot naively use the ability to bend wires in Cond to symbolically express conditionals. For a distribution $\psi : I \to X \otimes Y$, the morphism



is in general not a conditional $\psi|_Y$ as the defining equation (38) takes the form



which does not equal ψ unless ψ_Y is uniform. However we can extract $\psi|_Y$ if ψ_Y happens to be invertible with respect to ϵ.

Explicit unification lets us revisit Fritz's notation (31), which denotes morphisms $f : A \otimes B \to X \otimes Y$ as $f(x, y|a, b)$. Taking a page out of Prolog's book, we can treat nonlinear use of variables in output position using explicit unification: For example, given a binary predicate `p`, the query `p(X,X)` can be understood as `p(X,Y), X =:= Y`. Similarly for a morphism $p : A \to X \otimes X$, we can explain the expression $p(x, x|a)$ using explicit conditioning as $p(x, y|a), x =:= y$ which is of course just $\epsilon_X \circ p$. Fresh variables are initialized using the unit, and private variables are automatically discarded (marginalized). In fact, by making use of the Frobenius induced duality, we need not distinguish between input and output wires at all.

We can thus conceive of a language which lets us phrase probabilistic queries in a Prolog-like style (cf. [De Raedt and Kimming, 2015]). For example, let the predicate `disease(D)`

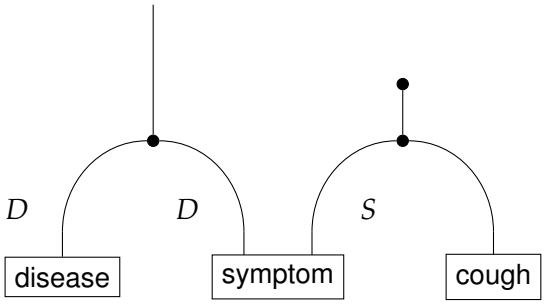
sample possible diseases `D` with their baseline frequencies in a population, and `symptom(D,S)` produce the symptoms statistically associated with each disease. Then the compound query

```
disease(D), symptom(D, cough).
```

expresses a posterior over diseases D after observing a cough. Note that the nonlinear use of the variable `D` triggers explicit conditioning. Spelled out fully, the computation to perform is

```
let d = disease() in
let (d',s) = symptom() in
d' =:= d; s =:= 'cough';
return d
```

or concisely written using string diagrams



21 Equational Presentation of the Gaussian Language

We make use of $\mathsf{Cond}(\mathsf{Gauss})$ to give denotational semantics to our Gaussian language from Section 17 and show this semantics is fully abstract (Theorem 21.2). The result of this section is then dedicated to understanding $\mathsf{Cond}(\mathsf{Gauss})$ in more detail, by deriving an equational presentation and normal forms.

21.1 Denotational Semantics

The Gaussian language embeds into the internal language of $\mathsf{Cond}(\mathsf{Gauss})$, where $x =:= y$ is translated as $(x - y) := 0$ as in Section 20. A term $\vec{x} : R^m \vdash e : R^n$ denotes a conditioning channel $[\![e]\!] : m \rightsquigarrow n$.

Proposition 21.1 (Correctness) *If* $(e, \psi) \triangleright (e', \psi')$ *then* $[\![e]\!]\psi = [\![e']\!]\psi'$. *If* $(e, \psi) \triangleright \bot$ *then* $[\![e]\!] = \bot$.

PROOF We can faithfully interpret ψ as a state in both Gauss and $\mathsf{Cond}(\mathsf{Gauss})$. If $x \vdash e$ and $(e, \psi) \triangleright (e', \psi')$ then e' has potentially allocated some fresh latent variables x'. We show that

$$\mathsf{let}\, x \;=\; \psi\, \mathsf{in}\, (x, [\![e]\!]) = \mathsf{let}\, (x, x') \;=\; \psi'\, \mathsf{in}\, (x, [\![e']\!]). \tag{79}$$

This notion is stable under reduction contexts.

Let C be a reduction context. Then

$$\begin{aligned}
&\mathsf{let}\, x \;=\; \psi \,\mathsf{in}\, (x, [\![C[e]]\!](x)) \\
&= \mathsf{let}\, x \;=\; \psi \,\mathsf{in}\, \mathsf{let}\, y \;=\; [\![e]\!](x) \,\mathsf{in}\, (x, [\![C]\!](x, y)) \\
&= \mathsf{let}\, (x, x') \;=\; \psi' \,\mathsf{in}\, \mathsf{let}\, y \;=\; [\![e']\!](x, x') \,\mathsf{in}\, (x, [\![C]\!](x, y)) \\
&= \mathsf{let}\, (x, x') \;=\; \psi' \,\mathsf{in}\, (x, [\![C[e']]\!])
\end{aligned}$$

Now for the redexes

(i) The rules for let follow from the general axioms of value substitution in the internal language

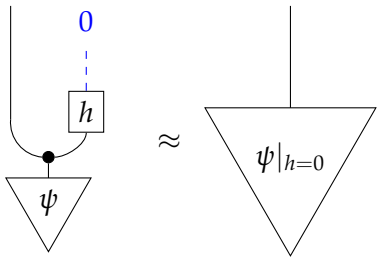
(ii) For normal() we have $(\mathrm{normal}(), \psi) \rhd (x', \psi \otimes \mathcal{N}(0,1))$ and verify

$$\begin{aligned}
&\mathsf{let}\, x \;=\; \psi \,\mathsf{in}\, (x, [\![\mathrm{normal}()]\!]) \\
&= \psi \otimes \mathcal{N}(0,1) \\
&= \mathsf{let}\, (x, x') \;=\; \psi \otimes \mathcal{N}(0,1) \,\mathsf{in}\, (x, [\![x']\!])
\end{aligned}$$

(iii) For conditioning, we have $(v =:= w, \psi) \rhd ((), \psi|_{v=w})$. We need to show

$$\mathsf{let}\, x \;=\; \psi \,\mathsf{in}\, (x, [\![v =:= w]\!]) = \mathsf{let}\, x \;=\; \psi|_{v=w} \,\mathsf{in}\, (x, ())$$

Let $h = v - w$, then we need to the following morphisms are equivalent in $\mathsf{Cond}(\mathsf{Gauss})$:



Applying Proposition 19.13 to the left-hand side requires us to compute the conditional $\langle \mathsf{id}, h \rangle \psi|_2 \circ 0$, which is exactly how $\psi|_{h=0}$ is defined. ∎

Theorem 21.2 (Full abstraction) *$[\![e_1]\!] = [\![e_2]\!]$ if and only if $e_1 \approx e_2$ (where $\approx$ is contextual equivalence, Definition 17.5).*

PROOF For $\Rightarrow$, let $K[-]$ be a closed context. Because $[\![-]\!]$ is compositional, we obtain $[\![K[e_1]]\!] = [\![K[e_2]]\!]$. If both succeed, we have reductions $(K[e_i], !) \rhd^* (v_i, \psi_i)$ and by correctness $v_1 \psi_1 = [\![K[e_1]]\!] = [\![K[e_2]]\!] = v_2 \psi_2$ as desired. If $[\![K[e_1]]\!] = [\![K[e_2]]\!] = \bot$ then both $(K[e_i], !) \rhd^* \bot$.

For $\Leftarrow$, we note that Cond quotients by contextual equivalence, but all Gaussian contexts are definable in the language. ∎

21.2 Equational Theory

We now give an explicit presentation of the equality between programs in the Gaussian language in the style of Section 11.1. We demonstrate the strength of the axioms by using them to characterize normal forms for various fragments of the language (Section 21.3). Besides an axiomatization of program equality, this can also be regarded in other equivalent ways, such as a presentation of a PROP by generators and relations [Staton et al., 2017a], or as a presentation of a strong monad by algebraic effects, or as a presentation of a Freyd category (Section 3.7). But we approach from the programming language perspective.

Similarly to the shorthand for Beta-Bernoulli (Section 10.2), we transform the Gaussian language into an algebraic form that fits the framework of second-order algebra (Section 3.7). The reader may find it helpful to think of this as a normal form for the language modulo associativity of 'let'. We begin by considering a fragment of the language with the following modifications: only variables of type R are allowed in the typing context Γ; we have an explicit command for failure ($\bot$); we separate the typing judgement in two: judgements for expressions of affine algebra $\vdash_{\mathsf{a}}$ and for general computational expressions $\vdash_{\mathsf{c}}$; we have an explicit coercion 'return' between them for clarity.

$$\frac{}{\Gamma, x : \mathrm{R}, \Gamma' \vdash_{\mathsf{a}} x : \mathrm{R}} \qquad \frac{\Gamma \vdash_{\mathsf{a}} s : \mathrm{R} \quad \Gamma \vdash_{\mathsf{a}} t : \mathrm{R}}{\Gamma \vdash_{\mathsf{a}} s + t : \mathrm{R}} \qquad \frac{\Gamma \vdash_{\mathsf{a}} t : \mathrm{R}}{\Gamma \vdash_{\mathsf{a}} \alpha \cdot t : \mathrm{R}}$$

$$\frac{}{\Gamma \vdash_{\mathsf{a}} \underline{\beta} : \mathrm{R}} \qquad \frac{\Gamma, x : \mathrm{R} \vdash t : \mathrm{R}^n}{\Gamma \vdash_{\mathsf{c}} \mathsf{let}\, x \;=\; \mathrm{normal}()\, \mathsf{in}\, t : \mathrm{R}^n}$$

$$\frac{\Gamma \vdash_{\mathsf{a}} s : \mathrm{R} \quad \Gamma \vdash_{\mathsf{a}} t : \mathrm{R} \quad \Gamma \vdash_{\mathsf{c}} u : \mathrm{R}^n}{\Gamma \vdash_{\mathsf{c}} (s \mathrel{=:=} t); u : \mathrm{R}^n}$$

$$\frac{\Gamma \vdash_{\mathsf{a}} t_1 : \mathrm{R} \quad \dots \quad \Gamma \vdash_{\mathsf{a}} t_n : \mathrm{R}}{\Gamma \vdash_{\mathsf{c}} \mathrm{return}(t_1, \dots, t_n) : \mathrm{R}^n} \qquad \frac{}{\Gamma \vdash_{\mathsf{c}} \bot : \mathrm{R}^n}$$

There is no general sequencing construct, but we can combine expressions using the following substitution constructions, whose well-typedness is derivable.

$$\frac{\Gamma, x : \mathrm{R}, \Gamma' \vdash_{\mathsf{c}} t : \mathrm{R}^n \quad \Gamma, \Gamma' \vdash_{\mathsf{a}} s : \mathrm{R}}{\Gamma, \Gamma' \vdash_{\mathsf{c}} t[s/x] : \mathrm{R}^n} \qquad \frac{\Gamma \vdash_{\mathsf{c}} t : \mathrm{R}^m \quad \Gamma, x_1, \dots, x_m : \mathrm{R} \vdash_{\mathsf{c}} u : \mathrm{R}^n}{\Gamma \vdash_{\mathsf{c}} t[x_1, \dots, x_n.u/\mathrm{return}] : \mathrm{R}^n}$$

In the second form we replace the return statement of an expression with another expression, capturing variables appropriately. The precise definition of this hereditary substitution is standard in logical frameworks (e.g. [Adams, 2009], [Staton, 2013a]), for example:

$$\begin{aligned}\big(\mathsf{let}\, x \;=\; \mathrm{normal}()\, \mathsf{in}\, \mathrm{return}(x+3)\big)\big[(a.a \mathrel{=:=} 4; \mathrm{return}(a))/\mathrm{return}\big] \\ = \;\mathsf{let}\, x \;=\; \mathrm{normal}()\, \mathsf{in}\, (x+3) \mathrel{=:=} 4; \mathrm{return}(x+3)\end{aligned}$$

For brevity, we introduce shorthands $\nu x.t$ for $\mathsf{let}\, x \;=\; \mathrm{normal}()\, \mathsf{in}\, t$, r for return and drop ';' when unambiguous. Now the syntax matches the formalism of a second-order algebraic theory (Section 3.7) of the signature

$$\nu : (0 \,|\, 1) \qquad \bot : (0 \,|\, -) \qquad (\mathrel{=:=}) : (2 \,|\, 0)$$

Note that r is really the name of a unique suitably typed metavariable. Because the signature has no operation taking more than one computation, no more than one metavariable is necessary.

The theory is parameterized over an underlying theory of values, which is affine algebra: The type R has the structure of a *pointed vector space*, which obeys the usual axioms of vector spaces plus constant symbols $(\underline{\beta})_{\beta\in\mathbb{R}}$ subject to

$$\alpha \cdot \underline{\beta} = \underline{\alpha\beta}, \underline{\alpha} + \underline{\beta} = \underline{\alpha + \beta}$$

Terms modulo equations are affine functions. The category theorist will recognize the category $\mathsf{Aff} = \mathsf{Gauss}_{\mathrm{det}}$ as the Lawvere theory of pointed vector spaces.

We axiomatize equality by closing the following axioms under the two forms of substitution and also congruence. The following axioms characterize the conditioning-free fragment of the language, that is, Gaussian probability

$$\vdash_{\mathsf{c}} \nu x.\mathrm{r}[] \equiv \mathrm{r}[] : \mathrm{R}^0 \tag{DISC}$$
$$\vdash_{\mathsf{c}} \nu \vec{x}.\mathrm{r}[U\vec{x}] \equiv \nu \vec{x}.\mathrm{r}[\vec{x}] : \mathrm{R}^n \text{ if } U \text{ orthogonal} \tag{ORTH}$$

The following are commutativity axioms for conditioning

$$a, b, c, d \vdash_{\mathsf{c}} (a =:= b)(c =:= d)\mathrm{r}[] \equiv (c =:= d)(a =:= b)\mathrm{r}[] : \mathrm{R}^0 \tag{C1}$$
$$a, b \vdash_{\mathsf{c}} (a =:= b); \nu x.\mathrm{r}[x] \equiv \nu x.(a =:= b)\mathrm{r}[x] : \mathrm{R}^1 \tag{C2}$$
$$a, b \vdash_{\mathsf{c}} (a =:= b)\bot \equiv \bot : \mathrm{R}^n \tag{C3}$$

while the following encode specific properties of $(=:=)$

$$a \vdash_{\mathsf{c}} (a =:= a)\mathrm{r}[] \equiv \mathrm{r}[] : \mathrm{R}^0 \tag{TAUT}$$
$$\vdash_{\mathsf{c}} (\underline{0} =:= \underline{1})\mathrm{r}[] \equiv \bot : \mathrm{R}^0 \tag{FAIL}$$
$$a, b \vdash_{\mathsf{c}} (a =:= b)\mathrm{r}[a] \equiv (a =:= b)\mathrm{r}[b] : \mathrm{R}^1 \tag{SUBS}$$
$$\vdash_{\mathsf{c}} \nu x.(x =:= \underline{c})\mathrm{r}[x] \equiv \mathrm{r}[\underline{c}] : \mathrm{R}^1 \tag{INIT}$$

Lastly, we add the special congruence scheme

$$\Gamma \vdash_{\mathsf{c}} (s =:= t)\mathrm{r}[] \equiv (s' =:= t')\mathrm{r}[] : \mathrm{R}^0 \tag{CONG}$$

whenever $(s = t)$ and $(s' = t')$ are interderivable equations over Γ in the theory of pointed vector spaces.

Axioms (DISC) and (ORTH) completely axiomatize the fragment of the language without conditioning (Proposition 21.3). Axioms (C1)-(C3) describe dataflow – all the operations distribute over each other. The reader should focus on the remaining five axioms (TAUT)-(CONG), which are specific to conditioning.

21.3 Normal forms

Proposition 21.3 *Axioms* (DISC)-(ORTH) *are complete for* Gauss. *That is, conditioning-free terms* $\vec{x} : \mathrm{R}^n \vdash u, v : \mathrm{R}^n$ *denote the same morphism in* Gauss *if and only if* $\vec{x} \vdash u \equiv v$ *is derivable from the axioms.*

PROOF The axioms are clearly validated in Gauss; probability is discardable and independent standard normal Gaussians are invariant under orthogonal transformations. Note that ν commutes with itself because permutation matrices are orthogonal.

It is curious that these laws completely characterize Gaussians: Any term normalizes to the form $\nu\vec{z}.\mathrm{r}[A\vec{x} + B\vec{z} + \vec{c}]$, denoting the map $(A, \vec{c}, BB^T)$ in Gauss. Consider some other term $\nu\vec{w}.\varphi[A'\vec{x} + B'\vec{w} + \vec{c}']$ that has the same denotation. By (DISC), we can without loss of generality assume that $\vec{z}$ and $\vec{w}$ have the same dimension. The condition $(A, c, BB^T) = (A', c', B'(B')^T)$ implies $A = A', \vec{c} = \vec{c}'$. By Proposition 33.5 there is an orthogonal matrix U such that $B' = BU$. So the two terms are equated under (ORTH). ∎

Example 21.4

$$\nu x.\nu y.\mathrm{r}[x + y] \equiv \nu y.\mathrm{r}[\sqrt{2} \cdot y]$$

PROOF Let $s = 1/\sqrt{2}$, then the matrix $U = \begin{pmatrix} s & s \\ -s & s \end{pmatrix}$ is orthogonal. Thus

$$\begin{aligned} \nu x.\nu y.\mathrm{r}[x + y] &\equiv \nu x.\nu y.\mathrm{r}[(sx + sy) + (-sx + sy)] \\ &\equiv \nu x.\nu y.\mathrm{r}[\sqrt{2}y] \\ &\equiv \nu y.\mathrm{r}[\sqrt{2}y] \end{aligned}$$

where we apply (ORTH), affine algebra and (DISC). ∎

We proceed to showing the consistency of the axioms for conditioning.

Proposition 21.5 *Axioms* (DISC)-(CONG) *are valid in* $\mathsf{Cond}(\mathsf{Gauss})$

PROOF Sketch. The commutation properties are straightforward from string diagram manipulation.

(SUBS) Write $a = b + (a - b)$; by Proposition 19.15, once we condition $a - b := 0$, we have $a = b$.

(INIT) By Proposition 19.13, noting that $c \ll \mathcal{N}(0, 1)$

(FAIL) By Proposition 19.13, because $0 \not\ll 1$

(CONG) This follows from Proposition 19.8, because over Aff, equivalent scalar equations are nonzero multiples of each other. Still, this is very surprising axiom scheme, which is substantially generalized in Corollary 21.7. ∎

For the remainder of this section, we will show how to use the theory to derive normal forms for conditioning programs.

Proposition 21.6 *Elementary row operations are valid on systems of conditions. In particular, if S is an invertible matrix then*

$$(A\vec{x} =:= \vec{b})\mathrm{r}[] \equiv (SA\vec{x} =:= S\vec{b})\mathrm{r}[]$$

PROOF Reordering and scaling of equations is (C1), (CONG). For summation, i.e.

$$(s =:= t)(u =:= v)\mathrm{r}[] \equiv (s =:= t)(u + s =:= v + t)\mathrm{r}[]$$

instantiate (SUBS) with $(u + x =:= v + t)\mathrm{r}[]/\mathrm{r}[x]$. Now use the fact that applying any invertible matrix on the left can be decomposed into elementary row operations. ∎

Corollary 21.7 *If $A\vec{x} = \vec{c}$ and $B\vec{x} = \vec{d}$ are linear systems of equations with the same solution space, then*

$$(A\vec{x} =:= \vec{c})\mathrm{r}[] \equiv (B\vec{x} =:= \vec{d})\mathrm{r}[]$$

is derivable.

This generalizes (CONG) to systems of conditions.

PROOF If consistent systems are equivalent, then they must be isomorphic by Corollary 33.3 and we use the previous proposition. If they are inconsistent, we can derive $(0 =:= 1)$ and use (FAIL),(C3) to equate them to $\bot$. ∎

We give a normal form for closed terms.

Theorem 21.8 *Any closed term can be brought into the form $\nu\vec{z}.\mathrm{r}[A\vec{z} + \vec{c}]$ or $\bot$. The matrix AA^T is uniquely determined.*

This is the algebraic analogue of Proposition 19.13.

PROOF By commutativity, we bring the term into the form

$$\nu\vec{z}.(A\vec{z} =:= \vec{b})\mathrm{r}[D\vec{z} + \vec{d}]$$

By Proposition 33.1, we can find invertible matrices S, T such that

$$SAT^{-1} = \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix}$$

and T is orthogonal. Using the orthogonal coordinate change $\vec{w} = T\vec{z}$ and Corollary 21.7, the equations take the form

$$\nu\vec{w}.(SAT^{-1}\vec{w} =:= S\vec{b})\mathrm{r}[DT^{-1}\vec{w} + \vec{d}]$$

This simplifies to

$$\nu\vec{w}.(\vec{w}_{1:r} =:= \vec{c}_{1:r})(0 =:= \vec{c}_{r+1:n})\mathrm{r}[DT^{-1}\vec{w} + \vec{d}]$$

where $\vec{c} = S\vec{b}$. We can process the first block of conditions with (INIT). The conditions $(\underline{0} =:= c_i)$ can either be discarded by (TAUT) if $c_i = \underline{0}$ for all $i = r+1, \ldots, n$, or fail by (FAIL) otherwise. We arrive at a conditioning-free term. ∎

Example 21.9

$$\nu x.\nu y.(x =:= y)\mathrm{r}[x, y] \equiv \nu x.\mathrm{r}[sx, sx]$$

where $s = 1/\sqrt{2}$.

PROOF We use again the unitary matrix U from Example 21.4

$$\begin{aligned}\nu x.\nu y.(x =:= y); \mathrm{r}[x,y] &\equiv \nu x.\nu y.(sx + sy =:= -sx + sy);\\ &\qquad \mathrm{r}[sx + sy, -sx + sy]\\ &\equiv \nu x.\nu y.(x =:= 0)\mathrm{r}[sx + sy, -sx + sy]\\ &\equiv \nu y.\mathrm{r}[sy, sy]\end{aligned}$$

where we apply (ORTH), affine algebra and (INIT). ∎

Lastly, we give a normal form for conditioning effects: Every effect in $\mathsf{Cond}(\mathsf{Gauss})$ can be expressed as $A\vec{x} =:= \psi$, that is a suitable transformation followed by a soft constraint (Section 20.1). However, unlike in a situation where there exists a Frobenius unit, not every effect is of the form $\vec{x} =:= \psi$ (for example $x_1 - x_2 =:= 0$ is not of that form).

Theorem 21.10 (Normal forms) *Every term $\vec{x} : \mathrm{R}^n \vdash u : \mathrm{R}^0$ can either be brought into the form $\bot$ or*

$$\nu \vec{z}.A\vec{x} =:= B\vec{z} + \vec{c} \tag{80}$$

where $A \in \mathbb{R}^{r\times n}$ is in reduced echelon form with no zero rows. The values of A, $\vec{c}$ and BB^T are uniquely determined.

PROOF Through the commutativity axioms, we can bring u into the form $\nu\vec{z}.A\vec{x} =:= B\vec{z} + \vec{c}$ for some general A. Find an invertible matrix S that turns A into reduced row echelon form, and apply it to the condition via Proposition 21.6. The zero columns don't involve $\vec{x}$, so we use Theorem 21.8 to evaluate the condition involving $\vec{z}$ separately. We either obtain $\bot$ or the desired form (80). For uniqueness, we consider the term's denotation $(A\vec{x} =:= \eta) : n \rightsquigarrow 0$ in $\mathsf{Cond}(\mathsf{Gauss})$, where $\eta = \mathcal{N}(\vec{c}, BB^T)$. We must show that A and η can be reconstructed from the observational behavior of the denotation. The proof is given in the appendix (Proposition 33.8). ∎

22 Context, Related Work and Outlook

22.1 Symbolic Disintegration and Paradoxes

Our line of work can be regarded as a synthetic and axiomatic counterpart of the symbolic disintegration of Shan and Ramsey [2017]. (See also [Gehr et al., 2016; Murray et al., 2018; Narayanan and Shan, 2019; Walia et al., 2019]) That work provides in particular verified program transformations to convert an arbitrary probabilistic program of type $\mathrm{R} \otimes \tau$ to an equivalent one that is of the form

$$\mathsf{let}\, x \;=\; \mathsf{lebesgue}()\, \mathsf{in}\, \mathsf{let}\, y \;=\; M\, \mathsf{in}\, (x, y)$$

Now the exact conditioning $x{:=}o$ can be carried out by substituting o for x in M. We emphasize the similarity to our treatment of *inference problems* in Section 18, as well as the role that coordinate transformations play in both our work (Section 21) and [Shan and Ramsey, 2017].

One language novelty in our work is that exact conditioning is a first-class construct, as opposed to a whole-program transformation, in our language, which makes the consistency of exact conditioning more apparent.

Consistency is a fundamental concern for exact conditioning. *Borel's paradox* (Example 17.2) is an example of an inconsistency that arises if one is careless with exact conditioning ([Jaynes, 2003, Ch. 15], [Jacobs, 2021b, §3.3]): It arises when naively substituting equivalent equations within $(=:=)$. Recall that for example, the equation $x - y = 0$ is equivalent to $x/y = 1$ over the (nonzero) real numbers. Yet, in a hypothetical extension of our language which allows division, the following programs would not contextually equivalent:

```
x = normal(0,1)        x = normal(0,1)
y = normal(0,1)   ≢    y = normal(0,1)
x-y =:= 0              x/y =:= 1
```

In our work, Borel's paradox finds a type-theoretic resolution: Conditioning is presented abstractly as an algebraic effect, so the expressions $(s =:= t) : \mathrm{I}$ and $(s == t) : \mathrm{bool}$ have a different formal status and can no longer be confused. They must be related explicitly through axioms like (SUBS), and special laws for simplifying conditions are given in (CONG) or Corollary 21.7. By Proposition 19.8, we can always substitute conditions which are formally isomorphic, but $x - y =:= 0$ and $x/y =:= 1$ are not isomorphic conditions in this sense. For the special case of Gaussian probability, we proved that equivalent affine equations are automatically isomorphic, making it very easy to avoid Borel's paradox in this restricted setting (Corollary 21.7). To include the non-example above, our language needs a nonlinear operation like $(/)$. If beyond that we introduced equality testing to the language, difference between equations and conditions would become even more apparent. The *equation* $x - y = 0$ is obviously equivalent to the equation $(x == y) = \mathrm{true}$, but the *condition* $(x == y) =:= \mathrm{true}$ would cause the whole program to fail, since measure-theoretically, $(x == y)$ is the same as false.

This also suggests a tradeoff between expressivity of the language and well-behavedness of conditioning. On this subject, Shan and Ramsey [2017] wrote:

> *The [measure-theoretic] definition of disintegration allows latitude that our disintegrator does not take: When we disintegrate $\xi = \Lambda \otimes \kappa$, the output κ is unique only almost everywhere — κx may return an arbitrary measure at, for example, any finite set of x's. But our disintegrator never invents an arbitrary measure at any point. The mathematical definition of disintegration is therefore a bit too loose to describe what our disintegrator actually does. How to describe our disintegrator by a tighter class of "well-behaved disintegrations" is a question for future research. In particular, the notion of continuous disintegrations [Ackerman et al., 2016b] is too tight, because depending on the input term, our disintegrator does not always return a continuous disintegration, even if one exists.*

By our definitions, we have tackled this research problem: a notion of "well-behaved disintegrations" is given by a Markov category with precise supports. The more comprehensive category BorelStoch admits conditioning only on events of positive probability. The smaller

category Gauss however features a better notion of support and an interesting theory of conditioning. Studying Markov categories of different degrees of specialization helps navigating the tradeoff. Once in the synthetic setting of a Markov category $\mathbb{C}$ with precise supports, the program transformations of Shan and Ramsey [2017] are all valid in $\mathsf{Cond}(\mathbb{C})$, and the Markov conditioning property (Definition 8.13) exactly matches the correctness criterion for symbolic disintegration.

22.2 Other Directions

Once a foundation is in algebraic or categorical form, it is easy to make connections to and draw inspiration from a variety of other work: The Obs construction (Definition 19.1) that we considered here is reminiscent of lenses [Clarke et al., 2020] and the Oles construction [Hermida and Tennent, 2012]. These have recently been applied to probability theory [St Clere Smithe, 2020], quantum theory [Huot and Staton, 2018] and reversible computing [Heunen and Kaarsgaard, 2021]. The details and intuitions are different, but a deeper connection or generalization may be profitable in the future.

We have explored algebraic presentations of probability theories and conjugate-prior relationships in Chapter III. Furthermore, the concept of exact conditioning is reminiscent of unification in Prolog-style logic programming. Our presentation in Section 21 is partly inspired by the algebraic presentation of predicate logic of Staton [2013a], which has a similar signature and axioms. The analogy has been explored in the form of Frobenius structures given by conditioning and unification (Section 20) and their units (Section 20.2). Logic programming is also closely related to relational programming, and we note that our presentation is reminiscent of presentations of categories of linear relations [Baez and Erbele, 2015; Bonchi et al., 2017, 2019] and their string-diagrammatic presentation.

On the semantic side, we recall that presheaf categories have been used as a foundation for logic programming [Kinoshita and Power, 1996]. Our axiomatization can be regarded as the presentation of a generalized measure monad P on the category $[\mathsf{Aff}^{\mathrm{op}}, \mathsf{Set}]$, via [Staton, 2013a], where Aff is the category of finite dimensional affine spaces discussed in Section 21.

Probabilistic logic programming. ProbLog [De Raedt and Kimming, 2015] supports both logic variables as well as random variables within a common formalism. We have not considered logic variables in conjunction with the Gaussian language, but a challenge for future work is to bring the ideas of exact conditioning closer to the ideas of unification, both practically and in terms of the semantics. We wonder if it is possible to extend Gauss by a synthetic uniform prior which acts as a Frobenius unit (Section 20.2).

Implementation. The purpose of our Gaussian language was to give a minimalistic calculus in which to study the novel effect of conditioning in isolation. The close fit of the denotational semantics to the language was thus expected, and can be seen as an instance of letting semantics inspire language design. To extend our calculus to a full-blown programming language, one can make use of the general framework of algebraic effects to combine conditioning with other effects like memory or recursion. For example, we can treat higher-order functions by modelling the language on a presheaf category, which is cartesian closed. The operational semantics easily extend to a full language, for which we have given implementations in Python and F# [Stein, 2021].

Chapter V
Name Generation and Probability on Function Spaces

There are only two hard things in Computer Science: cache invalidation and naming things.

— PHIL KARLTON

SUMMARY: We present fresh name generation as a synthetic probabilistic effect which captures independence or perfect correlation. At higher-order types, names exhibit subtle phenomena such as privacy and information-hiding which have a profound impact on the dataflow properties of the theory and make name generation the canonical example of a non-positive effect (see Section 9.1).

We then use quasi-Borel spaces to give a probabilistic model of fresh name generation as random sampling. We show that this model is fully abstract up to first-order function types, which is surprising for an 'off-the-shelf' model, and requires a novel analysis of probability on function types. Our tools include descriptive set theory as well as syntactic methods such as logical relations and normal forms. A more detailed overview over the results of this chapter can be found in Section 23.1.

This chapter is based on joint work with Marcin Sabok, Sam Staton and Michael Wolman [Sabok et al., 2021].

23 Introduction

Name generation is a ubiquitous phenomenon in computer science and one of the simplest instances of generativity. A *name* is an abstract entity which has no properties other than its identity, that is whether or not it is equal to other names. We can generate *fresh names*, that is create a new name which is distinct from all other names. Examples of names are *GUIDs*, *database IDs* or *URLs*, but also *variable names* in metaprogramming (`gensym`), *locations* for memory allocation (`new`) and *cluster names* in statistics.

We study name generation using the ν-calculus [Pitts and Stark, 1993], which is a minimalistic call-by-value λ-calculus with a special construct $\nu a.M$ which reads as "generate a fresh name, bind it to a and continue with M". For example, the program equation

$$\nu a.\nu b.(a = b) \approx \text{false}$$

means that two fresh names are always distinct. Because the ν-calculus satisfies exchangeability and discardability equations, we can consider it a particularly minimalistic probabilistic programming language, which can only express perfect correlation or independence.

In all the examples above, we can choose some number or string to represent the name, but this information should be treated opaquely. One way of making this formal is that we

can always consistently rename, as long as we don't introduce collisions between existing names: This is what happens in capture-avoiding substitution, when for example the λ-term $(\lambda x.\lambda y.x)\, y$ reduces to $\lambda z.y$ where the bound variable y has been α-renamed to z. The renaming approach is formalized in the category of nominal sets [Pitts, 2013b] which we'll revisit from a probabilistic viewpoint in Section 25. Nominal sets form the basis of the language FRESH O'CAML [Shinwell and Pitts, 2005].

Name generation as randomness: Our goal is to give a genuinely *probabilistic model* of name generation, where fresh name generation is random sampling. The idea is not far-fetched, as some real-world implementations of `gensym` return a random string, which works as intended if the probability of a collision is low enough. If we sample instead from a continuous distribution such as a Gaussian, the probability of a collision becomes zero. The analogy between name generation and sampling has been recognized in statistics, for example in random graph models or when `gensym` is used in place of a probability distribution in cluster analysis (more under Section 30.1). By subsuming name generation under random sampling, the two effects are unified in probabilistic programming.

We show that *quasi-Borel spaces*, a model for higher-order probabilistic programming, soundly model the ν-calculus. It is crucial here that we use a model of higher-order probability, because measurable spaces cannot interpret λ-abstraction (Section 4.3).

A theory of random functions: The probabilistic reading of name generation leads to interesting questions about the nature of probability on function spaces. For example, the higher-order function

$$\nu x.\lambda y.(y = x) : \mathsf{name} \to \mathsf{bool} \tag{81}$$

is observationally equivalent to $\lambda y.\mathrm{false}$; this is because the fresh name x remains *private* in the body of the function and is never revealed to the caller – the function can intuitively never return true. We dub this phenomenon "privacy".

We show that the same is true for the random model: By identifying sets and characteristic functions, the analogue of (81) is a *random singleton set* $\{X\}$ where X is a random real number. We prove that in quasi-Borel spaces, the set-valued random variable $\{X\}$ is equal in distribution (Section 4) to the empty set

$$\{X\} \overset{\mathrm{d}}{=} \emptyset \tag{82}$$

This result uses methods from descriptive set theory; its applicability however is broad: By considering a certain pathological Borel set $B \subseteq \mathbb{R}^2$ (Theorem 28.8), we can show that it is inconsistent to distinguish $\{X\}$ from $\emptyset$ with positive probability in any higher-order model that conservatively extends standard Borel probability. Thus, every higher-order probabilistic programming language will exhibit name generation-like phenomena like Privacy.

To give an example of our method, we demonstrate that it is impossible to define a functional $\exists : 2^{\mathbb{R}} \to 2$ which tests if a given Borel set is nonempty. Such a functional would easily distinguish $\{X\}$ and $\emptyset$. However, given any Borel subset $B : \mathbb{R}^2 \to 2$ of the plane, we could then define its projection $\pi(B) : \mathbb{R} \to 2$ via currying as

$$\pi(B) = \lambda x.\exists(\lambda y.B(x, y))$$

In the setting of probability theory, we require the invariant that all definable expressions are measurable so that integration can be used. However, it is not true that for every Borel set B, the projection $\pi(B)$ is again Borel. Thus $\exists$ cannot exist in the model. Our probabilistic interpretation of the ν-calculus gives a new intuition for such definability results. The question of the closedness of Borel sets under projection has an interesting history: Originally believed to be true by Lebesgue, Suslin proved that this is not the case. This result became the starting point of the notion of 'analytic sets' and the discipline of descriptive set theory. We refer to [Kechris, 1987] for an exposition to the field.

Name generation itself is commutative and discardable, and can thus be considered a synthetic probabilistic effect. However, it has some unusual properties which set it apart from ordinary probability, and synthetic probability is precisely the right language to discuss these differences: For example, we have seen in (82) that the variable $\{X\}$ is equal in distribution to the constant $\emptyset$, but is not independent the variable X! Similarly, the variable X cannot admit a conditional distribution given $\{X\}$, because that would mean extracting information that is private. Higher-order probability requires a refined analysis in terms of the "dataflow axioms" of Section 9.

Full abstraction: Our random model of the ν-calculus is a surprisingly good fit. Extending the privacy result, we can show that two name-generating expressions $\Gamma \vdash M_1, M_2 : \tau$ are observationally equivalent if and only if their denotations are equal in quasi-Borel spaces. This holds for all τ which are first-order function types, that is non-nested function types of the form $\tau_1 \to \cdots \to \tau_n$ with $\tau_i \in \{\mathsf{name}, \mathsf{bool}\}$.

Informal Theorem (Full abstraction – Theorem 29.10) *Quasi-Borel spaces are a fully abstract model for the ν-calculus up to first-order function types.*

This is surprising for an 'off-the-shelf' model such as quasi-Borel spaces; simple models of name generation such as nominal sets don't even validate the privacy equation (Proposition 26.2). Note that because λ and ν don't commute in the call-by-value ν-calculus, first-order function types can encode a considerable amount of complexity. For example, we have an observational equivalence

$$\nu a.\, \nu b.\, \lambda x.\, \mathsf{if}\ (x = a)\ \mathsf{then}\ a\ \mathsf{else}\ b \approx \nu b.\lambda x.b \tag{83}$$

because the name a is private on the left. On the other hand, the similar function

$$\nu a.\, \nu b.\, \lambda x.\, \mathsf{if}\ (x = a)\ \mathsf{then}\ b\ \mathsf{else}\ a \tag{84}$$

does not simplify, because calling it *twice* reveals both a and b. Our random semantics takes care of this, as we explain – the concept of higher-order measurability alone gives rise to sophisticated information hiding.

23.1 Outline

In Section 24, we review the ν-calculus, its semantics and contextual equivalence. In Section 25, we review nominal sets, a traditional model for name generation, from a synthetic probabilistic viewpoint. This section is illustrative of the concepts of name generation and

helps contrasts the equivariance of nominal sets with the group action in Section 29, but is not a prerequisite for what follows. In Section 26, we discuss the phenomenon of private names in detail and show that validating privacy means violating the positivity axiom of Section 9.

In Section 27, we review quasi-Borel spaces and show that they form a probabilistic model of the ν-calculus (Theorem 27.15). In Section 28, we show that quasi-Borel space semantics validates the privacy equation, that is a random singleton set is equal to the emptyset (Theorem 28.1). Our proof makes use of a construction from descriptive set theory. In Section 29, we prove our main full abstraction result: Two ν-calculus terms at a first-order type are observationally equivalent if and only if their denotation in quasi-Borel spaces is equal.

Our proof of full abstraction proceeds in two steps

(i) On the syntactic side, we give a normalization algorithm for observational equivalence at first order. Our algorithm, which appears to be novel, refines a logical relations argument by Pitts and Stark [1993], by identifying and eliminating all private names as in (83). Our construction simplifies the analysis of observational equivalence at first order. This also provides a general strategy for proving full abstraction.

(ii) Returning to the semantic side, we show that the normalization steps are validated in the quasi-Borel space model. The key idea here is that atomless measures such as the normal and uniform distributions are invariant under certain translations. We use this translation invariance to reduce our problem to the privacy equation, and use this to prove full abstraction at first order. Our use of an invariant action on the space of names is similar to but distinct from nominal techniques of Section 25; our action is internal to the model, and does not feature in its construction.

24 Name Generation and the ν-Calculus

We introduce name generation as a computational phenomenon, which will lay the foundations for the mathematical models of name generation we are considering. The *ν-calculus* [Pitts and Stark, 1993; Stark, 1994] is a computational λ-calculus with a construct for name generation. It has higher-order functions and follows a call-by-value evaluation strategy. We will start by recalling its syntax, operational semantics and observational equivalence:

The ν-calculus has two ground types bool for booleans and name for names, as well as function types

$$\tau ::= \mathsf{bool} \mid \mathsf{name} \mid \tau \to \tau$$

Terms M are formed as follows

$$M ::= x \mid \mathsf{true} \mid \mathsf{false} \mid M = M \mid MM \mid \lambda x.M \mid \nu n.M \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M$$

with typing rules given in Figure 8. The signature construct $\nu n.M$ is read as "create a new name, bind it to the variable n and continue with M". Names can be compared for equality using the boolean equality test $M = N$. The generic effect (see Section 3.6) of name generation is sometimes written $\mathsf{new} \stackrel{\text{def}}{=} \nu a.a$. Note that Stark's work formally distinguishes

$$\frac{}{\Gamma \vdash x : \tau}\ ((x:\tau) \in \Gamma) \qquad \frac{}{\Gamma \vdash b : \mathsf{bool}}\ (b = \mathrm{true}, \mathrm{false})$$

$$\frac{\Gamma \vdash M : \mathsf{bool} \quad \Gamma \vdash N_1 : \tau \quad \Gamma \vdash N_2 : \tau}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : \tau} \qquad \frac{\Gamma \vdash M : \mathsf{name} \quad \Gamma \vdash N : \mathsf{name}}{\Gamma \vdash (M = N) : \mathsf{bool}}$$

$$\frac{\Gamma, x\colon \mathsf{name} \vdash M : \tau}{\Gamma \vdash \nu x.M : \tau} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}$$

Figure 8: Grammar and typing rules for the ν-calculus [Pitts and Stark, 1993, Table 1].

between variables of type name and 'free names'. We will avoid this distinction here, but take explicit care in Sections 24.1 and 24.2 that all names bound to variables corresponding to free names must be distinct.

Convention (Ground, First-Order, Higher-Order): In this thesis, we distinguish ground types and higher-order types. Ground types are bool, name while higher-order types are all function types. By first-order types, we mean *the simplest higher-order types* consisting of non-nested functions $\tau_1 \to \tau_2 \to \ldots \to \tau_n$ with each τ_i ground. A typical first-order type is $\mathsf{name} \to \mathsf{name} \to \mathsf{bool}$, a typical second-order type is $(\mathsf{name} \to \mathsf{bool}) \to \mathsf{bool}$. To be precise, the order of a type can be defined recursively following Stark [1994]

$$\mathrm{order}(\mathsf{bool}) = \mathrm{order}(\mathsf{name}) = 0 \quad \mathrm{order}(\sigma \to \tau) = \max(\mathrm{order}(\sigma) + 1, \mathrm{order}(\tau)).$$

This is contrasted with other conventions which say 'first-order' for what we call 'ground'. Much focus in this chapter lies on first-order types. Because the ν-calculus is an effectful call-by-value language, those types already encode a considerable amount of complexity and cannot be simplified by currying, e.g.

$$\mathsf{name} \to \mathsf{name} \to \mathsf{bool} \ncong (\mathsf{name} \times \mathsf{name}) \to \mathsf{bool}$$

even if we extended the language with product types. This will be evident by the translation into the monadic metalanguage, where $\mathsf{N} \to T(\mathsf{N} \to T2) \ncong \mathsf{N} \times \mathsf{N} \to T2$.

24.1 Operational Semantics and Observational Equivalence

The evaluation relation of the ν-calculus is defined on *expressions*, those are terms with free variables of type name, and no other free variables. In this operational semantics, these variables are understood to be names that are generated in the course of running a program, and so they are assumed to be distinct, and we tend to use m or n for them. If $s = \{n_1, \ldots, n_k\}$ is a set of names and τ is a type, we define the set

$$\mathsf{Exp}_\tau(s) \stackrel{\mathrm{def}}{=} \left\{ M \mid n_1\colon \mathsf{name}, \ldots n_k\colon \mathsf{name} \vdash M : \tau \right\}$$

of expressions of type τ only involving the names s, and we define the subset $\mathsf{Val}_\tau(s) \subseteq \mathsf{Exp}_\tau(s)$ to consist of the *values* of the appropriate type, i.e. the expressions $\lambda x.M$, true, false, n.

$$\frac{-}{s \vdash V \Downarrow_\tau ()V}$$

$$\frac{s \vdash M \Downarrow_{\mathsf{name}} (s_1)m \quad s \vdash N \Downarrow_{\mathsf{name}} (s_2)n}{s \vdash (M = N) \Downarrow_{\mathsf{bool}} (s_1 \oplus s_2)\mathsf{false}}\ m \neq n \qquad \frac{s \vdash M \Downarrow_{\mathsf{name}} (s_1)m \quad s \vdash N \Downarrow_{\mathsf{name}} (s_2)m}{s \vdash (M = N) \Downarrow_{\mathsf{bool}} (s_1 \oplus s_2)\mathsf{true}}$$

$$\frac{s \vdash M \Downarrow_{\mathsf{bool}} (s_1)V \quad (s \oplus s_1) \vdash N_V \Downarrow_\tau (s_2)V'}{s \vdash \mathsf{if}\ M\ \mathsf{then}\ N_{\mathsf{true}}\ \mathsf{else}\ N_{\mathsf{false}} \Downarrow_\tau (s_1 \oplus s_2)V'} \qquad \frac{s \oplus \{n\} \vdash M \Downarrow_\tau (s')V}{s \vdash \nu n.\, M \Downarrow_\tau (\{n\} \oplus s')V}\ n \notin s$$

$$\frac{s \vdash M \Downarrow_{\sigma\to\tau} (s_1)\lambda x.M' \quad (s \oplus s_1) \vdash N \Downarrow_\sigma (s_2)V \quad (s \oplus s_1 \oplus s_2) \vdash M'[V/x] \Downarrow_\tau (s_3)V'}{s \vdash M\,N \Downarrow_\tau (s_1 \oplus s_2 \oplus s_3)V'}$$

Figure 9: Evaluation relation for the ν-calculus [Pitts and Stark, 1993, Table 2].

If s, t are sets of names, we write $s \oplus t$ to denote the union $s \cup t$ with the additional assumption that s and t must be disjoint (this can always be achieved by renaming free names if necessary).

The big-step evaluation relation $s \vdash M \Downarrow_\tau (s')V$ is given in Figure 9, where $M \in \mathsf{Exp}_\tau(s)$ and $V \in \mathsf{Val}_\tau(s \oplus s')$, meaning M evaluates to V generating fresh names s'. This is reminiscent of the way the Gaussian language in Section 17 allocates fresh variables. Crucially however, those variables are not comparable for boolean equality while names are.

Evaluation is terminating and deterministic up to choice of free names. We will not need to work directly with this evaluation relation very much, because we will build on existing methods for observational equivalence [Pitts and Stark, 1993; Stark, 1996], but we include it for completeness.

Example 24.1 We have

$$\emptyset \vdash \nu m.\nu n.(m = n) \Downarrow_{\mathsf{bool}} (\{a, b\})\mathsf{false}$$

That is the program generates two names a, b during its execution and evaluates to the value false, because a and b are distinct.

Observational equivalence is defined in a standard way. A *boolean context* $\mathcal{C}[\cdot]$ *for type* τ is an expression $\mathcal{C}$ where some subexpressions are replaced by a placeholder, such that if $M \in \mathsf{Exp}_\tau(s)$ then $\mathcal{C}[M] \in \mathsf{Exp}_{\mathsf{bool}}(s)$. Two terms $M_1, M_2 \in \mathsf{Exp}_\tau(s)$ are *observationally equivalent*, written $M_1 \approx_\tau M_2$, if for every boolean context $\mathcal{C}[\cdot]$ we have $\exists s'(s \vdash \mathcal{C}[M_1] \Downarrow_{\mathsf{bool}} (s')\mathsf{true})$ if and only if $\exists s'(s \vdash \mathcal{C}[M_2] \Downarrow_{\mathsf{bool}} (s')\mathsf{true})$. We omit annotating types where they can be inferred.

We will discuss some important observational equivalences and distinctions to showcase the intricacies of name generation. Notice that the creation of fresh names is not directly observable, only their effects when compared with other names.

Example 24.2 (Stark [1994, Section 2.5.(2)-(3)]) The ν-calculus satisfies the commutativity and discardability equations up to observational equivalence.

$$\nu a.\nu b.M \approx \nu b.\nu a.M \qquad \nu c.M \approx M \quad (c \notin \mathsf{fv}(M))$$

This makes the ν-calculus a model of synthetic probability according to Soft Definition 1.0. We will obtain a precise instance through the definition of a categorical model (Definition 24.7).

Example 24.3 (λ and ν don't commute)

$$\nu a.\lambda x.a \not\approx_{\mathsf{bool}\to\mathsf{name}} \lambda x.\nu a.a \tag{85}$$

PROOF This is typical of call-by-value semantics. Calling the function on the left-hand side *twice* will return the same name, but will produce different names on the right-hand side. That is, the context

$$C[-] = (\lambda f.(f\,\mathsf{true}) = (f\,\mathsf{true}))(-)$$

distinguishes the two sides of the would-be equivalence. We have

$$\emptyset \vdash C[\nu a.\lambda x.a] \Downarrow_{\mathsf{bool}} (\{a\})\mathsf{true} \qquad \emptyset \vdash C[\lambda x.\nu a.a] \Downarrow_{\mathsf{bool}} (\{a,b\})\mathsf{false}$$

Note that in call-by-*name* semantics, λ *does* commute with effects. For example, in the $\lambda\nu$-calculus of Odersky [1994], (85) *is* an observational equivalence. ∎

While an observational distinction can be demonstrated by a suitable distinguishing context, observational equivalences are harder to establish, as they quantify over all contexts. This may require more elaborate methods such as logical relations or denotational semantics. One such equivalence is of crucial importance for the rest of this chapter:

Example 24.4 (Privacy)

$$\nu a.\lambda x.(x = a) \approx_{\mathsf{name}\to\mathsf{bool}} \lambda x.\mathsf{false} \tag{86}$$

This equivalence is proved in Pitts and Stark [1993, Example 5] using logical relations. The intuition is that while the function $\lambda x.(x = a)$ is not constant (it returns true for the input $x = a$), there is no programmatic way of extracting the value of a from the function and supplying it as an argument. The name a is *private* or hidden in the closure $\lambda x.(x = a)$, which becomes observationally indistinguishable from $\lambda x.\mathsf{false}$. We therefore dub (86) the Privacy equation.

The privacy phenomenon yields an array of subtle equivalences at first-order function types, for example

Example 24.5 At type $\mathsf{name} \to \mathsf{name}$ we have

$$\begin{aligned} & \nu a.\nu b.\lambda x.\mathsf{if}\ (x = a)\ \mathsf{then}\ a\ \mathsf{else}\ b \approx \nu b.\lambda x.b \\ \text{however}\quad & \nu a.\nu b.\lambda x.\mathsf{if}\ (x = a)\ \mathsf{then}\ b\ \mathsf{else}\ a \not\approx \nu a.\lambda x.a \end{aligned}$$

In the first line, a remains private and the true-branch can be removed. In the second equation, none of the names remain private! Calling the function with a fresh name returns a, which can then be used to reveal b *on a second call*. That is, the context

$$C[-] = (\lambda f.\nu c.(f\,c) = (f\,(f\,c)))(-)$$

distinguishes the two sides.

In Section 29, we will show that the privacy equation is under certain assumptions the only source of complication at first-order types, and our random model will validate it and thus all first-order observational equivalences .

We finish this subsection with an example of an interesting observational equivalence at second-order types, which remains outside the methods of this thesis. It is neither validated by the simple logical relation nor probabilistic semantics, as we explain in Section 30.3.

Example 24.6 (Stark [1994, Section 2.5.(13)])

$$\nu a.\nu b.\lambda f.((f\,a) \Leftrightarrow (f\,b)) \approx_{(\mathsf{name}\to\mathsf{bool})\to\mathsf{bool}} \lambda f.\mathrm{true}$$

The idea is that any definable function f must treat all fresh names equivalently. Here $c_1 \Leftrightarrow c_2$ stands for the boolean biconditional, which is definable in the language through nested if-then-else. This result is proved using a refined logical relation.

24.2 Categorical Semantics

Stark [1996] gives denotational semantics to the ν-calculus by translating it to Moggi's monadic metalanguage with an if-then-else construct. The translation is standard for computational λ-calculi (see Section 3.3), though care is needed to model distinctness assumptions on free names.

Definition 24.7 (Stark [1996, Section 4]) A *categorical model of the ν-calculus* comprises

(i) a cartesian closed category $\mathbb{C}$ with finite limits

(ii) a commutative and affine monad T on $\mathbb{C}$

(iii) a disjoint coproduct $2 \stackrel{\text{def}}{=} 1 + 1$ of the terminal object with itself

(iv) a distinguished object of names N with a decidable equality test $(=) : N \times N \to 2$

(v) a distinguished morphism $\mathrm{new} : 1 \to TN$ satisfying the following equation

$$m : N \vdash \mathsf{let}\, n \leftarrow \mathrm{new}\,\mathsf{in}\,[(n, m = n)] = \mathsf{let}\, n \leftarrow \mathrm{new}\,\mathsf{in}\,[(n, \mathrm{false})] \qquad \text{(FRESH)}$$

This definition references 'disjoint coproducts' and 'decidable equality', concepts from categorical logic, but we will not assume familiarity with these in the rest of the thesis. We refer to Stark's paper and [Carboni et al., 1993] for reference.

Because the monad T is assumed affine and commutative, categorical models of the ν-calculus immediately fit the definition of a categorical model of probability from Section 5.

In any categorical model, we can interpret ν-calculus types as objects, using the standard call-by-value translation into the monadic metalanguage:

$$[\![\mathsf{bool}]\!] \stackrel{\text{def}}{=} 2 \quad [\![\mathsf{name}]\!] \stackrel{\text{def}}{=} N \quad [\![\sigma \to \tau]\!] \stackrel{\text{def}}{=} [\![\sigma]\!] \to T[\![\tau]\!]$$

This is extended to contexts via products. A ν-calculus term $\Gamma \vdash M : \tau$ is routinely interpreted as a term of the metalanguage and hence a morphism $[\![\Gamma]\!] \to T[\![\tau]\!]$ by induction on the structure of M (Figure 10).

$$[\![\lambda x.M]\!] \stackrel{\text{def}}{=} [\lambda x.[\![M]\!]] \qquad [\![x]\!] \stackrel{\text{def}}{=} [x] \qquad [\![\text{true}]\!] \stackrel{\text{def}}{=} [\text{true}] \qquad [\![\text{false}]\!] \stackrel{\text{def}}{=} [\text{false}]$$

$$[\![M = N]\!] \stackrel{\text{def}}{=} \text{let } m = [\![M]\!] \text{ in let } n = [\![N]\!] \text{ in } [m = n] \quad [\![M\,N]\!] \stackrel{\text{def}}{=} \text{let } f = [\![M]\!] \text{ in let } x = [\![N]\!] \text{ in } f(x)$$

$$[\![\nu x.M]\!] \stackrel{\text{def}}{=} \text{let } x = \text{new in } [\![M]\!] \quad [\![\text{if } M \text{ then } N_1 \text{ else } N_2]\!] \stackrel{\text{def}}{=} \text{let } b = [\![M]\!] \text{ in if } b \text{ then } [\![N_1]\!] \text{ else } [\![N_2]\!]$$

Figure 10: Interpretation of ν-calculus expressions in a categorical model, using its metalanguage [Stark, 1996, Figure 5].

Using the categorical limits and the equality test on N, we can build a subobject $N^{\neq s} \rightarrowtail N^s$ for all finite sets s, modelling the assumption $(\neq s)$ of distinct names. Formally, $N^{\neq s}$ is the equalizer of $(n : N^s \vdash \bigvee_{i \neq j}(n_i = n_j) : 2)$ and $(n : N^s \vdash \text{false} : 2)$. For expressions $M \in \mathsf{Exp}_\tau(s)$, we will typically use the restricted interpretation $[\![M]\!]_{\neq s} : N^{\neq s} \rightarrowtail N^s \xrightarrow{[\![M]\!]} T[\![\tau]\!]$.

We note that values $V \in \mathsf{Val}_\tau(s)$ factor through the unit of the monad $[-]_{[\![\tau]\!]} : [\![\tau]\!] \to T[\![\tau]\!]$, and we will sometimes write $|V| : N^{\neq s} \to [\![\tau]\!]$ for the pure semantics. Any categorical model according to Definition 24.7 is sound and, under mild assumptions, adequate:

Theorem 24.8 (Stark [1994, Theorems 3.10, 3.11]) *For any categorical model of the ν-calculus:*

- *The big-step semantics is sound with respect to the denotational semantics: If $s \vdash M \Downarrow_\tau (s')V$ then $[\![M]\!]_{\neq s} = [\![\nu s'.V]\!]_{\neq s}$.*
- *If 1 is not an initial object and $[-]_2 : 2 \to T(2)$ is monic, then the denotational semantics is adequate for observational equivalence: If $[\![M_1]\!]_{\neq s} = [\![M_2]\!]_{\neq s}$ then $M_1 \approx_\tau M_2$, for all expressions $M_1, M_2 \in \mathsf{Exp}_\tau(s)$.*

Aside on the 'Mono' requirement: When working with the monadic metalanguage, several authors [Stark, 1996; Moggi, 1991] ask that the monad T satisfy the requirement

$$[-]_X : X \to TX \text{ is monic for all } X \qquad \text{(MONO)}$$

While this is certainly a natural requirement, the adequacy theorem only needs the Mono requirement at the object 2.

Categorical models are axiomatized to be sound and correct for ground computation. However they need not identify observationally equivalent terms at higher types: We will see that simple models do not validate the privacy equation (86). Much of our study will be devoted to understanding which higher observational equivalences a model validates.

25 Aside on Traditional Models of Name Generation

We give a digression on *nominal sets* [Pitts, 2013b], which are a traditional model of name generation. This model is explicitly constructed with the concept of α-equivalence in mind, and captures many name generation phenomena precisely. Nominal techniques are widespread in computer science and underlie languages such as FRESH O'CAML [Shinwell and Pitts,

2005].

While nominal sets aren't required for understanding our probabilistic model, they provide relevant intuitions for name generation, especially the failure of the Privacy equation (Proposition 26.2). It is also interesting to contrast the notion of equivariance in nominal sets with our use of group actions in Section 29.2.

In this section, we give a brief introduction to nominal sets with an eye towards a probabilistic reading: Name generation can be seen as a simple generalized probability theory which only knows perfect correlation and independence. Our treatment of nominal sets is purely expository, and fully based on Pitts' book. The phrasing in terms of synthetic probability is novel; we give a combinatorial definition of a Markov category for name generation in (Proposition 25.19) reminiscent of Section 14. Propositions 25.20 and 25.21 are new.

25.1 Nominal Sets

> *What's in a name? That which we call a rose*
> *By any other name would smell as sweet;*
>
> – WILLIAM SHAKESPEARE, Romeo and Juliet

The following propositions can be found in [Pitts, 2013b, Chapters 1-3].

Definition 25.1 (Nominal set) Fix a countably infinite set $\mathbb{A}$ whose elements we call *atoms* or *names* (written $a, b, c, \ldots$) and let $\mathsf{Perm}(\mathbb{A})$ denote the group of finite permutations of $\mathbb{A}$. A *nominal set* is a set X with a $\mathsf{Perm}(\mathbb{A})$-action $(\pi, x) \mapsto \pi \cdot x$ such that every $x \in X$ has a *finite support*, that is there exists a finite set $A \subseteq \mathbb{A}$ such that

$$\forall \pi \in \mathsf{Perm}(\mathbb{A})((\forall a \in A(\pi(a) = a)) \Rightarrow \pi \cdot x = x) \tag{87}$$

A morphism of nominal sets is an equivariant function $f : X \to Y$, i.e. for all permutations π we have $f(\pi \cdot x) = \pi \cdot f(x)$. Nominal sets and equivariant functions form the category Nom.

We write $\mathsf{supp}(x)$ for the least set A satisfying (87), which exists when x comes from a nominal set, and write $x \# y$ if $\mathsf{supp}(x) \cap \mathsf{supp}(y) = \emptyset$. Note that to give a point $1 \to X$ of a nominal set is to give an invariant element $x \in X$. The most important nominal set is the set of atoms $\mathbb{A}$ itself, endowed with the permutation action $\pi \cdot a = \pi(a)$.

Finite products and coproducts in Nom are computed like in Set with pointwise actions. If X, Y are two $\mathsf{Perm}(\mathbb{A})$-sets, the set of all functions $\mathsf{Set}(X, Y)$ has an action defined via

$$(\pi \cdot f)(x) = \pi \cdot f(\pi^{-1} \cdot x)$$

We call a function $f : X \to Y$ *finitely supported* if it has a finite support under this action.

Proposition 25.2 *If X, Y are nominal sets, then the nominal set*

$$(X \to_{\mathsf{fs}} Y) \stackrel{def}{=} \{ f : X \to Y \mid f \textit{ finitely supported} \}$$

is an exponential object in Nom*, along with the equivariant evaluation map*

$$\mathrm{ev}_{X,Y} : (X \to_{\mathsf{fs}} Y) \times X \to Y, (f, x) \mapsto f(x)$$

A function is equivariant if and only if its support is $\emptyset$, that is

$$\mathsf{Nom}(1, X \to_{\mathsf{fs}} Y) \cong \mathsf{Nom}(X, Y)$$

Proposition 25.3 Nom *forms a complete and cocomplete topos. Subobjects $A \rightarrowtail X$ can be identified with* equivariant subsets $A \subseteq X$*, that is a subsets such that $x \in A \Rightarrow \pi \cdot x \in A$. The topos* Nom *is two-valued, its subobject classifier is the discrete nominal set* 2*. We have a natural bijection*

$$\{ \textit{ equivariant subsets } A \subseteq X \} \cong \mathsf{Nom}(X, 2)$$

Example 25.4 (Internal powerset) A subset $A \subseteq \mathbb{A}$ is finitely supported if and only if A is finite or cofinite, in which case

$$\mathsf{supp}(\{a_1, \ldots, a_n\}) = \{a_1, \ldots, a_n\} = \mathsf{supp}(\mathbb{A} \setminus \{a_1, \ldots, a_n\})$$

The internal powerset of $\mathbb{A}$ consists of the finite-cofinite subsets,

$$\mathcal{P}(\mathbb{A}) \cong (\mathbb{A} \to_{\mathsf{fs}} 2) \cong \{A \subseteq \mathbb{A} \text{ finitely supported } \}$$

The invariant subsets of $\mathbb{A}$ are in bijection with the invariant elements of $\mathcal{P}(\mathbb{A})$; those are only $\emptyset$ and $\mathbb{A}$.

Definition 25.5 (Separated product) For nominal sets X, Y, define their *separated product* as

$$X * Y = \{(x, y) \in X \times Y : x \# y\}$$

The separated product defines a symmetric monoidal structure on Nom. This structure is in fact monoidal closed.

Definition 25.6 For $n \in \mathbb{N}$, we define the nominal set $\mathbb{A}^{\#n}$ as

$$\mathbb{A}^{\#n} = \{(a_1, \ldots, a_n) \in \mathbb{A}^n \text{ all distinct } \}.$$

This is isomorphic to the n-fold separated product of $\mathbb{A}$ with itself.

Example 25.7 (Failure of choice) Nom violates various formulations of the axiom of choice. Cardinality is an equivariant map, in particular let

$$\mathbb{A}^{[2]} = \{A \subseteq \mathbb{A} : |A| = 2\}$$

then the map $p : \mathbb{A}^{\#2} \to \mathbb{A}^{[2]}, (a, b) \mapsto \{a, b\}$ is an equivariant surjection. Yet p admits no section, and in fact there is no equivariant map $f : \mathbb{A}^{[2]} \to \mathbb{A}$ at all (hence no map to $\mathbb{A}^{\#2}$ by extension).

Nominal sets can in fact be identified with a Fraenkel-Mostowski model of set theory with atoms, which provides a counterexample to the axiom of choice. The importance of the sets $\mathbb{A}^{\#n}$ is underlined by the fact that Nom can be presented as a Grothendieck topos where the objects $\mathbb{A}^{\#n}$ are the representables.

Proposition 25.8 (Pitts [2013b, 6.3]) Nom *is equivalent to the Schanuel topos, that is to covariant sheaves* $[\mathsf{Inj}, \mathsf{Set}]$ *for the atomic topology. Here* Inj *denotes the category of finite sets and injections. The Yoneda lemma reads*

$$\mathsf{Nom}(\mathbb{A}^{\#m}, \mathbb{A}^{\#n}) \cong \mathsf{Inj}(n, m)$$

Concretely, every equivariant map $\mathbb{A}^{\#m} \to \mathbb{A}^{\#n}$ is of the form

$$(a_1, \ldots, a_m) \mapsto (a_{f(1)}, \ldots, a_{f(n)})$$

where $f : n \to m$ is injective.

25.2 Name Generation Monad

We now recall the monad T on nominal sets which is commonly used to model name generation. T is formally introduced as the free restriction set monad in [Pitts, 2013b, Chapter 9.5] but we'll simply refer to it as *the* name-generation monad. The construction of T is reminiscent of the writer monad (Section 5.2) for the monoid $(\mathcal{P}_\mathsf{f}(\mathbb{A}), \cup, \emptyset)$ of finite subsets of $\mathbb{A}$. That is, an execution produces a result together with a finite set of generated names. The permutation action of nominal sets is then used to consider those names up to α-equivalence as well as discard unused ones, which lets them behave correctly like bound names.

Definition 25.9 (α-equivalence) Let X be a nominal set, then define an equivariant equivalence relation $\sim$ on $\mathcal{P}_\mathsf{f}(\mathbb{A}) \times X$ by $(A, x) \sim (A', x')$ if there exists $\pi \in \mathsf{Perm}(\mathbb{A})$ such that

(i) $\pi \cdot x = x'$

(ii) $\mathsf{supp}(x) \setminus A = \mathsf{supp}(x') \setminus A'$

(iii) $\forall a \in (\mathsf{supp}(x) \setminus A)(\pi(a) = a)$

We call the quotient TX and write $(A)x$ or $\{a_1, \ldots, a_n\}x$ for the equivalence class of (A, x).

Note that $\mathsf{supp}((A)x) = \mathsf{supp}(x) \setminus A$; the names listed in A are bound and considered up to α-equivalence.

Example 25.10 Every element of $T(\mathbb{A})$ is equal to $\{\}a$ for some $a \in A$, or to the element $\nu = \{a\}a$. In particular $T(\mathbb{A}) \cong \mathbb{A} + 1$. We call the unique point $\nu : 1 \to T(\mathbb{A})$ the *fresh name distribution*.

The construction T can be given the structure of a strong monad as

$$Tf((A)x) = (A)f(x), \quad [x] = \{\}x, \quad \mathsf{join}((A)(B)x) = (A \cup B)x$$

when A, B are fresh enough sets of names.

Remark 25.11 Unlike for all the monads we have seen previously, the strength for T has to do nontrivial amounts of work: This is where all the capture-avoiding renaming happens. For example, in

$$\mathsf{st}_{\mathbb{A},\mathbb{A}}(a, \{a\}a) = \mathsf{st}_{\mathbb{A},\mathbb{A}}(a, \{b\}b) = \{b\}(a, b)$$

the bound name a had to be renamed to b to avoid collision. Formally, giving a strength corresponds to an enrichment

$$T : (X \to_{\mathsf{fs}} Y) \to (TX \to_{\mathsf{fs}} TY)$$

that correctly extends the functorial action from equivariant to finitely supported functions. The tensor in the Kleisli category crucially involves the strength and hence renaming, e.g.

$$\begin{aligned} \nu \otimes \nu &= \mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, y \leftarrow \nu\, \mathsf{in}\, [(x,y)] \\ &= \mathsf{let}\, x \leftarrow \{a\}a\, \mathsf{in}\, \mathsf{let}\, \{a\}a \leftarrow \nu\, \mathsf{in}\, [(x,y)] \\ &= \mathsf{let}\, x \leftarrow \{a\}a\, \mathsf{in}\, \mathsf{let}\, \{b\}b \leftarrow \nu\, \mathsf{in}\, [(x,y)] \\ &= \{a,b\}(a,b) \end{aligned}$$

Proposition 25.12 *(i) For any discrete nominal set X, $TX \cong X$*

(ii) The monad T is affine and commutative.

(iii) Like the writer monad, T preserves all colimits

Proposition 25.13 Nom *forms a categorical model of the ν-calculus with*

(i) $\mathbb{A}$ as object of names

(ii) T as the name generation monad

(iii) $\mathsf{new} = \nu : 1 \to T(\mathbb{A})$.

PROOF We verify (FRESH). Indeed, the morphism

$$[\![m : A \vdash \mathsf{let}\, n \leftarrow \mathsf{new}\, \mathsf{in}\, [(n, m = n)]]\!] : \mathbb{A} \to T(\mathbb{A} \times 2)$$

sends the name a to $\{b\}(a, a = b) = \{b\}(a, \mathsf{false})$, so it equals

$$[\![m : A \vdash \mathsf{let}\, n \leftarrow \mathsf{new}\, \mathsf{in}\, [(n, \mathsf{false})]]\!].$$

■

Aside on zero-one laws: The isomorphism $T(2) \cong 2$ can be interpreted as stating that in Nom, the only probabilities are 0 and 1. Given a distribution $1 \to T(X)$, every property $X \to_{\mathsf{fs}} 2$ will hold almost surely or almost never. This gives further credibility to our statement that names are about perfect correlation and independence.

Results in probability theory indicating that certain properties must hold almost surely or almost never are known as *zero-one laws*. Famous such laws are *Kolmogorov's zero-one law* for tail events and the *Hewitt-Savage zero-one law* for symmetric events. Zero-one laws are generally typical of ergodic theory, which deals with measure-preserving transformations. Permutations and the fresh-name measure exhibit similar ideas. Recently, synthetic versions of these laws have been formulated [Fritz and Rischel, 2020]. In more elementary terms, many statements about name generation are 'some/any theorems', meaning that if they hold for some fresh enough name, they will hold for any fresh enough name [Pitts,

2013b, Theorem 3.9]. Again, this is reminiscent of zero-one laws.

Before we address the higher-order properties of name generation in Section 26, we outline some of the ground behavior of name generation, for which Nom is fully abstract. Here, the name generation monad admits a compact combinatorial description.

Proposition 25.14 *(i) The name generation monad satisfies*

$$T(A * B) \cong T(A) * T(B) \tag{88}$$

(ii) In particular

$$T(\mathbb{A}^{\#n}) \cong \sum_k \binom{n}{k} \mathbb{A}^{\#k}.$$

PROOF The first statement says that for a separated a pair (x, y) with $x \# y$, all bound names in $(A)(x, y)$ can be split to affect either only x or y; there is no correlation. The second statement follows from the binomial theorem

$$T(\mathbb{A} * \cdots * \mathbb{A}) \cong (\mathbb{A} + 1) * \cdots * (\mathbb{A} + 1)$$

and the fact that $X * (-)$ is a left adjoint and thus commutes with coproducts. Concretely, every element of $T(\mathbb{A}^{\#n})$ looks like $\{a_{i_1}, \ldots, a_{i_k}\}(a_1, \ldots, a_n)$ where $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$.■

We proceed to classify the joint distributions on $\mathbb{A}^n$. We do this by splitting $\mathbb{A}^n$ into orbits depending depending on which entries of a tuple of names $(a_1, \ldots, a_n)$ are equal to one another.

Proposition 25.15 *Let B_n (for Bell number) denote the set of equivalence relations on n. The map $E : \mathbb{A}^n \to B_n$ given by*

$$E(a_1, \ldots, a_n) = \{(i, j) \in n \times n : a_i = a_j\}$$

is equivariant, and decomposes $\mathbb{A}^n$ into orbits

$$\mathbb{A}^n \cong \sum_{R \in B_n} E^{-1}(R)$$

If the equivalence relation R has k blocks, then $E^{-1}(R) \cong \mathbb{A}^{\#k}$ by choosing k distinct names (one for each block), and repeating that name inside the block. In conclusion

$$\mathbb{A}^n \cong \sum_k S_{n,k} \mathbb{A}^{\#k}$$

where $S_{n,k}$ denotes the Stirling number of the second kind.

Corollary 25.16 *The nominal set $\mathbb{A}^n$ has precisely 2^{B_n} equivariant subsets.*

Example 25.17 The space $\mathbb{A}^2$ decomposes into orbits

$$\mathbb{A}^2 = \{(a, b) : a \neq b\} \cup \{(a, b) : a = b\}$$

and is thus isomorphic to $\mathbb{A}^{\#2} + \mathbb{A}$. We have

$$T(\mathbb{A}^2) \cong T(\mathbb{A}^{\#2}) + T(\mathbb{A}) \cong (\mathbb{A}^{\#2} + 2\mathbb{A} + 1) + (\mathbb{A} + 1)$$

corresponding to all different shapes of elements

$$\{\}(a,b), \{a\}(a,b), \{b\}(a,b), \{a,b\}(a,b), \{\}(a,a), \{a\}(a,a)$$

In fact

$$T(\mathbb{A}^2) \cong T(\mathbb{A}) \times T(\mathbb{A}) + 1$$

which shows that T does not preserve products; all elements of $T(\mathbb{A}^2)$ decompose into product distributions except $\{a\}(a,a)$. The only two global states $1 \to T(\mathbb{A}^2)$ are $\nu \otimes \nu$ (independence) and $\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [(x,x)]$ (perfect correlation).

Example 25.18 The double distribution space

$$T(T(\mathbb{A})) \cong \mathbb{A} + 1 + 1$$

has the two points $[\nu] = \{\}(\{a\}a)$ and $\mathsf{let}\, a \leftarrow \nu\, \mathsf{in}\, [[a]] = \{a\}(\{\}a)$.

We finish with a combinatorial account of a Kleisli category for ground name generation. Its construction is reminiscent of the Markov category BetaBern in Section 14.

Proposition 25.19 *Consider the full subcategory* Atoms *of nominal sets isomorphic to finite sums of representables*

$$\sum_i n_i \mathbb{A}^{\#k_i}$$

Then Atoms *is closed under products, coproducts, separated products, and the name generation monad. Furthermore,* Atoms *is equivalent to* $\mathsf{FinFam}(\mathsf{Inj}^{\mathsf{op}})$*, that is*

(i) objects are finite lists $(A_1, \ldots, A_n)$ of finite sets

(ii) morphisms $(A_1, \ldots, A_m) \to (B_1, \ldots, B_n)$ consist of a function $f : m \to n$ and injections $g_i : B_{f(i)} \to A_i$ for all $i \in m$.

PROOF It remains to see that the products of Atoms-objects lie again in Atoms. This is not automatic because Inj does not have coproducts. However, we can still decompose the product

$$\mathbb{A}^{\#m} \times \mathbb{A}^{\#n}$$

into sums of representables. Alternatively, we notice that Atoms consists, up to isomorphism, of the equivariant subsets of $\mathbb{A}^n$ for $n \in \mathbb{N}$. ∎

We conclude with some novel structural remarks about the Kleisli category of the name generation monad:

Proposition 25.20 *The monad T on* Nom *satisfies the representability condition (Definition 6.8).*

PROOF Let $\xi = (A)x$ be an element of $T(X)$. Without loss of generality, we can assume that $A \subseteq \mathsf{supp}(x)$ and write $A + B = \mathsf{supp}(x)$. Assume that ξ is deterministic (Section 6.3), i.e.

$$(A)(x, x) = (A + A')(x, \sigma \cdot x)$$

where $A' \cap \mathsf{supp}(x) = \emptyset$ consists of fresh names and σ is a bijection $A \cong A'$ fixing B. By definition of α-equivalence, there is a permutation π such that

$$\pi \cdot (x, x) = (x, \sigma \cdot x)$$

We conclude that $x = \pi \cdot x = \sigma \cdot x$. Therefore

$$A + B = \mathsf{supp}(x) = \mathsf{supp}(\sigma \cdot x) = A' + B$$

from which we obtain $A = A' = \emptyset$, that is $\xi = \{\}x$ is pure. ∎

Proposition 25.21 *The Kleisli category* Nom_T *does not have all conditionals (see Definition 8.13).*

PROOF Consider the joint distribution $\mu \in T(2^{\mathbb{A}} \times \mathbb{A})$ given by

$$\mu = \{a, b\}(\{a, b\}, a)$$

Then μ admits no conditional $f = \mu|_1 : 2^{\mathbb{A}} \to T(\mathbb{A})$. Indeed, by equivariance, $f(\{a, b\})$ cannot take the value $\{\}c$ for any $c \in \mathbb{A}$. The remaining alternative is that $f(\{a, b\}) = \{c\}c$, but the composite

$$\mathsf{let}\ (A, \_) \leftarrow \mu\ \mathsf{in}\ \mathsf{let}\ c \leftarrow f(A)\ \mathsf{in}\ [(A, c)] = \{a, b, c\}(\{a, b\}, c)$$

fails to reconstruct μ, as evidenced by pushing forward under

$$(\ni) : 2^{\mathbb{A}} \times \mathbb{A} \to 2$$

∎

It is the same statement as Proposition 26.4, though the proof idea differs because Nom invalidates Privacy (Proposition 26.2). We remark that all objects of Atoms are *strong nominal sets* admitting a tighter notion of 'strong support' [Tzevelekos, 2008a], while the space $2^{\mathbb{A}}$ used as a counterexample in Proposition 25.21 does not. We wonder if the Kleisli category of T restricted to Atoms *does* have all conditionals.

We now return to our study of the ν-calculus and observational equivalences at higher types.

26 Name Generation at Higher Types

As we have seen, ground computation with fresh names has a straightforward combinatorial account (Proposition 25.19). The real interest involves the higher-order features of the ν-calculus, that is the interaction of name generation with function types: This is where interesting phenomena like privacy and information hiding happen.

We recall that categorical models of the ν-calculus need not validate observational equivalences at higher types. An example is given in Proposition 26.2: The nominal sets model does not validate the privacy equation (86). We thus make the following definition:

Definition 26.1 (Full abstraction) A categorical model of the ν-calculus is *fully abstract* at type τ if

$$M_1 \approx_\tau M_2 \Rightarrow [\![M_1]\!] = [\![M_2]\!]$$

All nondegenerate categorical models are fully abstract at ground types (Theorem 24.8). We will be interested in full abstraction at first-order function types.

26.1 The Privacy Equation

The privacy equation (86) is arguably the simplest nontrivial observational equivalence at a higher type. Its categorical intepretation is

$$\mathsf{let}\, a \leftarrow \mathsf{new}\, \mathsf{in}\, [\lambda x.[x = a]] = [\lambda x.[\mathsf{false}]] : T[\![\mathsf{name} \to \mathsf{bool}]\!] = T(N \to T2) \tag{89}$$

The metalanguage has extra types such as $N \to 2$ which are not the interpretation of ν-calculus types, so it is possible to consider the following stronger and slightly simpler variation of (89) by getting rid of one layer of the monad:

$$\mathsf{let}\, a \leftarrow \mathsf{new}\, \mathsf{in}\, [\lambda x.(x = a)] = [\lambda x.\mathsf{false}] : T(N \to 2). \tag{PRIV}$$

Equation (PRIV) implies (89) by postcomposing with $[-]_2 : 2 \to T2$. Under common assumptions (which we will always meet), (89) also implies (PRIV). This is for example the case if $2 \to T2$ is *split* monic (a strengthening of (MONO)). We will thus say that a categorical model verifies the Privacy equation if (PRIV) holds. Every model which is fully abstract at first order types has to validate the Privacy equation, and a converse holds under certain assumptions (Section 29.2).

We begin by observing that the nominal sets model does not validate Privacy, hence it is not fully abstract at first order [Stark, 1994, 3.6].

Proposition 26.2 (Failure of Privacy) Nom *does not validate the Privacy equation* (PRIV).

PROOF Identifying $2^{\mathbb{A}} \cong \mathcal{P}(\mathbb{A})$, the Privacy equation asks whether

$$(\{a\})\{a\} = (\{\})\emptyset \in T(2^{\mathbb{A}}) \tag{90}$$

that is, 'is the fresh singleton equal to the emptyset'? But these elements are not identified in the name-generation monad, because we cannot α-covert $\{a\}$ into $\emptyset$. More formally, we have seen that cardinality is an equivariant notion, so in particular the nonemptiness-check

$$\exists : 2^{\mathbb{A}} \to 2, \quad \exists(A) = [A \neq \emptyset]$$

is a morphism in Nom and can be used to distinguish the two sides of (90). ∎

We will contrast this with the situation for Qbs (which has a weak subobject classifier $\Omega \ncong 2$) in Proposition 28.4 and Attempt 28.5.

Stark has given a logical relation R which captures observational equivalence up to first-order types, that is

$$M_1 \approx_\tau M_2 \Leftrightarrow M_1 \, R_\tau \, M_2$$

He has categorified this relation to give a variant of the nominal sets model which achieves full abstraction at first-order types [Stark, 1994, 4.4]. We will not use his model further, except to argue that the notion of a fully abstract categorical model is not vacuous. Our main result is that the random model of name generation in quasi-Borel spaces achieves the same level of abstraction out of the box. Stark's logical relation is not part of the construction of this model, however we will make use of it in the proof (Section 29).

We proceed with some structural implications of the Privacy equation.

26.2 Privacy contradicts Positivity

The Privacy equation states that a fresh singleton set is indistinguishable from the empty set. This is very intuitive in the context of name generation, yet strange from the viewpoint of synthetic probability: For any given name A, the set $X = \{A\}$ is nonempty, but if we sample A freshly, the resulting "random set" X becomes empty. Note that this is only the case if information about A does not leak, because otherwise we can use the membership check $A \in (-)$ to distinguish X from $\emptyset$. This sort of information-hiding behavior makes name generation the canonical example of a non-positive probability theory (Section 9).

Recall that in statistical terms, the 'positivity' axiom (Section 9.1) says that constants are independent of everything. In the model $A \sim \text{new}$ and $X = \{A\}$, Privacy implies that $X \overset{\text{d}}{=} \emptyset$ is deterministic. However X is not independent of A! The precise categorical phrasing is as follows:

Proposition 26.3 *Any categorical model of the ν-calculus which validates the Privacy equation must violate positivity.*

PROOF Consider the joint distribution

$$\mu = \text{let } a \leftarrow \text{new in } [(\{a\}, a)] \in T(2^N \times N) \tag{91}$$

Then by the Privacy equation, the first marginal $\text{let } a \leftarrow \text{new in } [\{a\}] = [\emptyset]$ is deterministic, yet μ is not the product of its marginals $[\emptyset] \otimes \text{new}$ because the map

$$(\ni) : 2^N \times N \to 2$$

distinguishes them. This contradicts the deterministic marginal property, which is equivalent to positivity (Proposition 9.3). ∎

From this example, we conclude that the failure of positivity has no inherent connection to negative probabilities. Proposition 26.3 applies to Stark's logical relations model (which only has 'probabilities' $\{0, 1\}$) as well as to quasi-Borel spaces (Theorem 28.1), which feature standard $[0, 1]$-valued probabilities. Rather, Positivity is a dataflow property which prevents hiding of information in the sense of Section 9. One could, with some artistic license, consider Privacy a form of limited *destructive interference* where the nonempty set $X = \{A\}$

becomes empty after "blurring out", i.e. randomizing, the value of A. We will briefly revisit the idea of information leaking in Section 30.5.

Proposition 9.4 states that the existence of conditionals implies positivity. Hence the privacy equation must be at odds with conditioning. We can construct the following concrete counterexample:

Proposition 26.4 *Any categorical model of the ν-calculus which validates the Privacy equation cannot have all conditionals.*

PROOF Consider again the distribution $(X, A) \sim \mu$ from (91). Then intuitively, once we throw away the copy of A, we cannot recover its value from $X = \{A\}$ alone, because that information has been hidden. Formally, let $\mu|_1 : 2^N \to T(N)$ be an attempted conditional, then the instantiation of (38)

$$\mathsf{let}\, x \leftarrow [\emptyset]\, \mathsf{in}\, \mathsf{let}\, a \leftarrow \mu|_1(x)\, \mathsf{in}\, [(x, a)] = [\emptyset] \otimes \mu|_1(\emptyset)$$

fails to reconstruct μ, because $(\ni) : 2^N \times N \to 2$ distinguishes both distributions. ∎

Conditionals cannot exist because they would require the extraction of private information.

26.3 Towards Probabilistic Semantics for Name Generation

We have so far given an analysis of name generation as a synthetic probabilistic effect. For the remainder of this chapter, we will interpret name generation using *actual* probability. As explained in Section 23, this is well motivated by practice, as for example many implementations of `gensym` return a random symbol. Such implementations work well if *collisions* of names are sufficiently unlikely. In fact, if we draw names from a continuous distribution ν such as a Gaussian, collisions will have probability 0 and the freshness equation (FRESH) holds.

We begin by recalling atomless distributions on standard Borel spaces and argue that they interpret ground name-generating programs. The challenge for interpreting the full ν-calculus is then to find a cartesian closed model of probability which supports higher-order functions, equality tests and atomless distributions. This will be achieved in Section 27.

Definition 26.5 (Atomless distribution) The following are equivalent for a probability measure ν on a standard Borel space X

(i) if $X, Y \sim \nu$ are independent, then $\Pr(X = Y) = 0$

(ii) ν satisfies (FRESH), i.e. $\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [(x, x = y)] = \mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [(x, \mathsf{false})]$.

(iii) $\nu(\{x\}) = 0$ for all $x \in X$

We call a measure satisfying these conditions *atomless*[18].

[18] this is sometimes called *continuous measure* in the literature

PROOF We have (i) $\Rightarrow$ (iii) because if $\nu(\{x_0\}) = p > 0$ then $P(X = Y) \geq p^2$. (iii) means $\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [x = y] = [\mathrm{false}]$ which implies (ii) by the deterministic marginal property for Meas (Proposition 9.3). (ii) implies (i) by integrating over y and marginalizing. ∎

It is straightforward to argue that this probabilistic semantics is adequate for ground name-generating computation. We interpret names in a standard Borel space N and sample fresh names through an atomless measure $\nu : 1 \to \mathcal{G}(N)$. This satisfies all requirements of Definition 24.7 except cartesian closure.

It remains to wonder which measurable space N to choose as space of names: It is clear that N must be uncountable for an atomless measure measure to exist. Apart from that, we can think of a multitude of examples:

(i) The reals carry plenty of atomless probability measures, like the Gaussian distributions, but none of them is particularly canonical.

(ii) The interval $[0, 1]$ carries the uniform distribution, which is just the restricted Lebesgue measure.

(iii) So does the circle $\mathbb{S}^1 \cong [0, 1)$. This space furthermore carries a group structure (addition modulo 1) under which the measure is invariant. This will come in handy in Section 29.2.

(iv) Cantor space 2^ω carries a measure which can be interpreted as an infinite sequence of fair coin flips. This would correspond to an idealized `gensym` returning an infinitely long random bitstring.

It turns out that all of these choices are equivalent:

Proposition 26.6 (e.g. Kechris [1987, 17.41]) *Let u denote the uniform distribution on $[0, 1]$. If ν is any atomless probability measure on a standard Borel space X, then there is a measurable isomorphism $f : [0, 1] \to X$ such that $\nu = f_* u$.*

Intuitively, for the purposes of name generation, it should never matter which particular atomless measure we employ. Recall that questions about name generation are $\{0, 1\}$-valued, so intermediate values that ν assigns should never come up. This is of course an abstraction that can be broken (Section 30.3), but surprisingly, the abstraction will be perfect up to first-order types. For now, we can frame this as a *meta-principle* to guide our intuitions: For example, we can *expect* certain sets to be countable or cocountable based on the following lemma (which led us to conjecture Lemma 28.9):

Lemma 26.7 *Let X be an uncountable standard Borel space and $A \subseteq X$ be a measurable subset such that all atomless measures ν assign A the same value c. Then $c \in \{0, 1\}$, and A is countable or cocountable.*

PROOF Towards a contradiction, assume that both A and A^c are uncountable. By Theorem 4.4, we can find a Borel isomorphism $f : [0, 1] \to X$ such that $f^{-1}(A) = [0, 1/2]$. By pushing forward appropriate atomless measures on $[0, 1]$ along f, we obtain atomless measures on X which assign A arbitrary values. ∎

As an aside, it is instructive to recall the measurable space X from Example 6.7, which fails to interpret name generation in an interesting way: If X is uncountable and equipped with the countable-cocountable σ-algebra, then the measure $\nu(A) = [A \text{ cocountable}]$ satisfies condition (iii) of Definition 26.5. However, unlike fresh name generation, ν is deterministic and the equality test is not measurable. The issue here is that X is not a standard Borel space.

27 Quasi-Borel spaces and Higher-Order Probability

The question for the remainder of the chapter is: Can we find a genuinely probabilistic higher-order model of the ν-calculus? If so, how abstract is it?

Quasi-Borel spaces [Heunen et al., 2017] are a convenient setting including both measure theory and higher-order functions, which are increasingly widely used (e.g. [Lew et al., 2019; Sato et al., 2019; Ścibior et al., 2017; Vandenbroucke and Schrijvers, 2020]). They work by first restricting probability theory to the well-behaved domain of standard Borel spaces, and then provide a conservative extension to function spaces, achieving cartesian closure. We have surveyed other models of higher-order probability in Section 4.3.

In this chapter, we recall the theory of quasi-Borel spaces and show that they give a categorical model of the ν-calculus (Theorem 27.15). In the consecutive sections, we will then take an in-depth look at the theory of random functions obtained that way, and show that quasi-Borel space semantics is fully abstract up to first-order types (Theorem 29.10). All definitions are taken from Heunen et al. [2017] unless otherwise stated.

Definition 27.1 A *quasi-Borel space* is a set X together with a collection M_X of distinguished functions $\alpha : \mathbb{R} \to X$ called *random elements*. The collection M_X must satisfy

(i) for every $x \in X$, the constant map $\lambda r.x$ lies in M_X

(ii) if $\alpha \in M_X$ and $\varphi : \mathbb{R} \to \mathbb{R}$ is Borel measurable, then $\alpha \circ \phi \in M_X$

(iii) if $\{A_i\}_{i=1}^{\infty}$ is a countable Borel partition of $\mathbb{R}$ and $\alpha_i \in M_X$ are given, then the case-split $\alpha(r) = \alpha_i(r)$ for $r \in A_i$ lies in M_X

A map $f : X \to Y$ between quasi-Borel spaces is a *morphism* if for all $\alpha \in M_X$ we have $f \circ \alpha \in M_Y$. This defines a category Qbs. As usual, we will sometimes write (X, M_X) to emphasize the particular quasi-Borel structure.

We consider the reals with their canonical quasi-Borel structure $M_{\mathbb{R}} = \mathsf{Meas}(\mathbb{R}, \mathbb{R})$. Under that definition, we can recover the random elements of any other quasi-Borel space X as $M_X = \mathsf{Qbs}(\mathbb{R}, X)$.

Definition 27.2 (i) A *discrete* quasi-Borel space carries the *minimal* quasi-Borel structure consisting only of *simple functions*, that is $\alpha \in M_X$ if there exists a countable Borel partition $\mathbb{R} = \sum_{i=1}^{\infty} A_i$ such that f is constant on each A_i.

(ii) An *indiscrete* quasi-Borel space carries the *maximal* quasi-Borel structure in which *every function* $\mathbb{R} \to X$ is a random element.

We consider the booleans 2 as a discrete quasi-Borel space. Equivalently, $M_2 = \mathsf{Meas}(\mathbb{R}, 2)$ where 2 is a discrete measurable space. This is an instance of a general construction to pass between quasi-Borel structures and σ-algebras:

Definition 27.3 (i) any measurable space X has an induced quasi-Borel structure M_X whose random elements are precisely the measurable maps $\mathbb{R} \to X$

$$M_X \stackrel{\text{def}}{=} \mathsf{Meas}(\mathbb{R}, X)$$

(ii) a subset $A \subseteq X$ of a quasi-Borel space is deemed *measurable* if its characteristic function is a morphism $X \to 2$. We obtain a σ-algebra on X by

$$\Sigma_X \stackrel{\text{def}}{=} \mathsf{Qbs}(X, 2)$$

Equivalently, Σ_X is the greatest σ-algebra making all random elements measurable:

$$A \in \Sigma_X \Leftrightarrow \forall \alpha \in M_X, \alpha^{-1}(A) \text{ Borel.} \tag{92}$$

It is important to note that if we begin with a measurable space (X, Σ_X), the induced σ-algebra Σ_{MX} on the quasi-Borel space (X, M_X) will in general *not* equal Σ_X. The categorical relationship between these constructions is clarified as follows

Proposition 27.4 (Heunen et al. [2017, Prop. 15]) *The assignments*

$$M : (X, M_X) \mapsto (X, M_X) \quad \Sigma : (X, M_X) \mapsto (X, \Sigma_X)$$

are functorial and form an adjunction

$$\mathsf{Qbs} \underset{M}{\overset{\Sigma}{\rightleftarrows}} \mathsf{Meas} \quad (\Sigma \dashv M)$$

The adjunction is idempotent and we have $\Sigma M \Sigma = \Sigma$ *and* $M \Sigma M = M$.

As a small digression, it is instructive to consider the following "overlap" between Meas and Qbs that is fixed by the adjunction.

Definition 27.5 (Standardly generated spaces) For a quasi-Borel space X, the following are equivalent

(i) $X \cong MY$ for some measurable space Y

(ii) $M_X = M_{\Sigma X}$ for some σ-algebra Σ_X on X

(iii) $X = M\Sigma X$.

The analogous conditions are equivalent for measurable spaces Y and quasi-Borel structures M_Y. We call spaces satisfying the equivalent conditions *standardly generated*[19]. We denote the full subcategories of standardly generated spaces as $\mathsf{SQbs} \subseteq \mathsf{Qbs}$, $\mathsf{SMeas} \subseteq \mathsf{Meas}$ respectively.

[19] in analogy with *compactly generated* Hausdorff spaces, also known as k-spaces, in topology. Terminology due to Heunen, Kammar, Staton, Yang (unpublished communication)

Proposition 27.6 *The following spaces are standardly generated*

(i) *Standard Borel spaces*

(ii) *Discrete spaces*

(iii) *Indiscrete spaces*

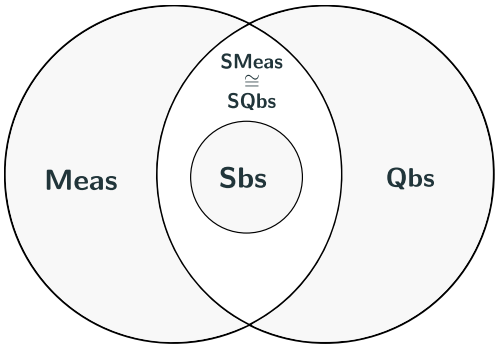
Furthermore

(iv) *The adjunction 27.4 restricts to an equivalence of categories* $\mathsf{SQbs} \cong \mathsf{SMeas}$

(v) $\mathsf{SQbs} \hookrightarrow \mathsf{Qbs}$ *is a reflective subcategory with reflector* $M\Sigma$. $\mathsf{SMeas} \hookrightarrow \mathsf{Meas}$ *is a coreflective subcategory with coreflector* ΣM

PROOF Standard Borel spaces fixed by the adjunction due to Heunen et al. [2017, Prop. 15]. We defer the proof that (in)discrete measurable spaces are mapped to (in)discrete quasi-Borel spaces and vice versa to the appendix (Propositions 34.1 and 34.2). The last two bullet points are simple categorical corollaries. ∎

The functor M restricts to a full and faithful embedding $\mathsf{Sbs} \to \mathsf{Qbs}$. We will generally identify standard Borel spaces with their images in Qbs and write say $\mathbb{R}$ or 2 for the quasi-Borel space and measurable space alike. This is formally justified by proposition 27.14. Note that while standard Borel spaces are standardly generated, the converse is not true. The relationships of the different categories are shown in the following Venn diagram:



We will later show that the function space $2^{\mathbb{R}}$ is an example of a space that is *not* standardly generated Proposition 28.12.

27.1 Cartesian closure

Unlike Meas, quasi-Borel spaces do admit a well-defined notion of function space:

Proposition 27.7 (Heunen et al. [2017, Prop. 18]) Qbs *is cartesian closed, with* $Y^X = \mathsf{Qbs}(X, Y)$. *A morphism* $\mathbb{R} \to Y^X$ *is a random element if and only if its uncurrying* $\mathbb{R} \times X \to Y$ *is a morphism.*

For example, the space $2^{\mathbb{R}}$ comprises the characteristic functions of Borel subsets of $\mathbb{R}$, and the random elements $\mathbb{R} \to 2^{\mathbb{R}}$ are the curried characteristic functions of Borel subsets of $\mathbb{R}^2$. Identifying subsets with their characteristic functions, the following notation corresponds to currying:

Notation 27.8 For a subset $A \subseteq X \times Y$ of a product and $x \in X$, we write $A_x = \{y \in Y \mid (x, y) \in A\}$ for the (vertical) *slice* of A at x.

Under this notation, all random elements $\mathbb{R} \to 2^{\mathbb{R}}$ are of the form $x \mapsto A_x$ where $A \subseteq \mathbb{R}^2$ is a Borel subset of the plane. The quasi-Borel structure on $2^{\mathbb{R}}$ induces a σ-algebra $\Sigma_{2^{\mathbb{R}}}$ on the set $2^{\mathbb{R}}$, which is well-known in the literature as 'Borel-on-Borel' (we will analyze this further in Section 28). For now, we can use Aumann's result to illustrate the crucial fact that Σ does not preserve products:

Example 27.9 By cartesian closure, the evaluation map $(\ni) : 2^{\mathbb{R}} \times \mathbb{R} \to 2$ is a morphism of quasi-Borel spaces, hence $(\ni) \in \Sigma_{2^{\mathbb{R}} \times \mathbb{R}}$. However by Aumann's result (Theorem 4.11), $(\ni)$ is not measurable in the product-σ-algebra $\Sigma_{2^{\mathbb{R}}} \otimes \Sigma_{\mathbb{R}}$. Hence the canonical map

$$\Sigma(2^{\mathbb{R}} \times \mathbb{R}) \to \Sigma(2^{\mathbb{R}}) \times \Sigma(\mathbb{R})$$

is *not* an isomorphism. We'll see a purely probabilistic proof of this statement in Proposition 28.10, which doesn't presuppose Aumann's result.

In general, the σ-algebra $\Sigma_{X \times Y}$ is strictly bigger than $\Sigma_X \otimes \Sigma_Y$. Another instance of this is the following: It is easy to see that finite products of discrete quasi-Borel spaces are again discrete, but the same is not true for uncountable measurable spaces, as $\mathcal{P}(X) \otimes \mathcal{P}(Y) \subsetneq \mathcal{P}(X \times Y)$. We finish with a categorical remark:

Proposition 27.10 (Heunen et al. [2018]) Qbs *is a category of concrete sheaves on a concrete site. In particular, it is a Grothendieck quasitopos.*

This brings it on a similar footing such as diffeological spaces [Baez and Hoffnung, 2008], which are of interest to semantics [Huot et al., 2020]. There is an object Ω which classifies strong subobjects[20], given by the space $\{\text{true}, \text{false}\}$ carrying the *indiscrete structure*. Given any quasi-Borel space X and a subset $A \subseteq X$, its characteristic function $\chi_A : X \to \Omega$ is a morphism. However note that χ_A does not factor through $2 \to \Omega$ unless A is a measurable subset of X.

27.2 Probability on Quasi-Borel Spaces

In order to define how probability works in the quasi-Borel setting, we consider the source of all randomness to come from some probability distribution μ on $\mathbb{R}$. The random elements $\mathbb{R} \to X$ tell us how this randomness may be pushed forward onto X. Two such random elements are identified if their *laws* agree as measures on the induced σ-algebra. The definition of Σ_X makes sure the pushforward is well-defined.

Definition 27.11 A probability distribution on a quasi-Borel space X is an equivalence class $[\alpha, \mu]$ where $\alpha \in M_X$, $\mu \in \mathcal{G}(\mathbb{R})$ and we let

$$[\alpha, \mu] = [\alpha', \mu'] \Leftrightarrow \alpha_* \mu = (\alpha')_* \mu' \in \mathcal{G}(X, \Sigma_{M_X})$$

[20] making Ω a weak subobject-classifier!

We note that the significance of the induced σ-algebra on a quasi-Borel space X is to give a notion of *equality of distributions* on X, which is simply extensional equality of the pushforward measures. Unlike in Meas, the σ-algebra does not determine the maps into X unless X is standardly generated.

There is a Giry-like strong monad P on Qbs which sends X to the space of probability distributions on X, endowed with the quasi-Borel structure

$$M_{P(X)} = \{\beta : \mathbb{R} \to P(X) \mid \exists \alpha \in M_X, \kappa \in \mathsf{Meas}(\mathbb{R}, \mathcal{G}(\mathbb{R})), \beta(r) = [\alpha, \kappa(r)]\}$$

For $x \in X$, one can form the Dirac distribution δ_x on X by taking $\delta_x = [\lambda r.x, \mu]$ for any $\mu \in \mathcal{G}(\mathbb{R})$. This forms the unit of the monad. We reduce the bind of P to the bind of the Giry monad: Given $f : X \to P(Y)$ and $[\alpha, \mu] \in P(X)$, we have $f\alpha \in M_{P(Y)}$ so there is some $\beta \in M_Y$ and $\kappa : \mathbb{R} \to \mathcal{G}(\mathbb{R})$ such that $f(\alpha(r)) = [\beta, \kappa(r)]$. We define a distribution on Y by taking $f^+([\alpha, \mu]) = [\beta, g^+(\mu)]$. The monad obtained that way satisfies the axioms of a probability monad.

Example 27.12 For a discrete space X, to give a distribution in $P(X)$ is to give a countable discrete probability distribution on X. For an indiscrete space X, we have $P(X) \cong 1$.

PROOF For the discrete case, all probability must be pushed forward along a simple function, therefore the resulting distribution has countable support. This contrasts with the case of Meas where measures on $\mathcal{P}(X)$ are difficult to classify. If X is indiscrete, we have $\Sigma_X = \{0, X\}$ hence any two probability measures on X are equal. ∎

Proposition 27.13 (Heunen et al. [2017, Prop. 22]) *The monad P is strong, affine and commutative.*

Probability theory over standard Borel spaces is the same whether done in Meas or Qbs

Proposition 27.14 (Heunen et al. [2017, Prop. 19, 22]) *The embedding* Sbs $\to$ Qbs *preserves countable products, countable coproducts and probability monads.*

Nonpreservation of products: Example 27.9 shows that joint distributions in Qbs more subtle than in Meas. If $\mu \in P(X), \nu \in P(Y)$ are two distributions with laws defined on Σ_X, Σ_Y respectively, then their product distribution $\mu \otimes \nu \in P(X \times Y)$ will have a law defined on $\Sigma_{X \times Y}$ which may be strictly larger than the product σ-algebra $\Sigma_X \otimes \Sigma_Y$. Because μ, ν are of the restricted form of Definition 27.11, their product measure can be unambiguously extended beyond the product-σ-algebra on which product measures are usually defined. Note that membership in $\Sigma_{X \times Y}$ is not given inductively but via universal quantification (92). This gives the theory much needed flexibility for Section 28, while also making it more difficult to work with.

27.3 Quasi-Borel Spaces model the ν-Calculus

We can now give probabilistic semantics to the ν-calculus (Definition 24.7) by interpreting names as elements of a quasi-Borel space and name generation as random sampling.

Theorem 27.15 Qbs *is a categorical model of the ν-calculus under the following assignment:*

(i) *the object of names is N is any uncountable standard Borel space*

(ii) *the name-generation monad is P*

(iii) new *is given by any atomless distribution $\nu \in P(N)$.*

PROOF Because all spaces involved are standard Borel, this conservatively (Proposition 27.14) extends the ground semantics of Section 26.3. Moreover, Qbs semantics is adequate (Theorem 24.8) because $0 \ncong 1$ and the unit $[-]_2 : 2 \to P(2)$ is split monic just like in measure theory (Example 4.10). ∎

As explained in Section 26.3, we have many different isomorphic choices for the space of names N, such as $\mathbb{R}$ or Cantor space. We will take advantage of this in section 29.2, where it will be convenient to work with the circle $\mathbb{S}^1$.

Aside: In Stark's adequacy proof (Theorem 24.8), it is only required that $[-]_2 : 2 \to P(2)$ be monic. By using 'separated' quasi-Borel spaces we can support the full (MONO) requirement. We mention this for completeness with respect to the literature, and will not require this notion later in this chapter.

Definition 27.16 A quasi-Borel space X is *separated* if the maps $X \to 2$ separate points, meaning that for all $x \neq x' \in X$ there is some morphism $f : X \to 2$ such that $f(x) \neq f(x')$.

This is equivalent to saying that the induced σ-algebra Σ_{M_X} on X separates points.

Proposition 27.17 *A quasi-Borel space X is separated if and only if it satisfies the (MONO) rule, i.e. $[-]_X : X \to P(X)$ is injective. Additionally, we have: standard Borel spaces are separated; if X, Y are separated, so is $X \times Y$; if Y is separated, so is Y^X; and for every X, $P(X)$ is separated.*

PROOF Assume X is separated and $[x] = [x']$. Then for all $f : X \to 2$, we have $f(x) = \int f(y)[x](\mathrm{d}y) = \int f(y)[x'](\mathrm{d}y) = f(x')$; by separatedness this implies $x = x'$. Conversely let X satisfy (MONO) and assume that for all $f : X \to 2$ we have $f(x) = f(x')$. Then $[x] = [x']$ because $[x](A) = \chi_A(x) = \chi_A(x') = [x'](A)$ for all $A \in \Sigma_X$, so by injectivity $x = x'$. For products and function spaces, the required separating maps can be easily constructed from projections and evaluations. If $\mu \neq \mu' \in P(X)$, then $\mu(A) \neq \mu'(A)$ for some $A \in \Sigma_X$, hence $f(\psi) = [\mathrm{ev}_A(\psi) = \mu(A)]$ separates μ and μ'. ∎

Therefore all quasi-Borel spaces interpreting ν-calculus types are automatically separated.

28 The Privacy Equation in Qbs

We now turn to the proof that Qbs is a fully abstract model of the ν-calculus at first-order types: In this section, we prove an important stepping stone, namely that the privacy equation holds in Qbs. This is instructive as the simplest nontrivial observational equivalence at a higher type. Furthermore, in Section 29.2, we manage to reduce all other first-order observational equivalences to Privacy using syntactic methods.

The proof of the privacy equation on the other hand requires an analysis of the σ-algebra $\Sigma_{2^{\mathbb{R}}}$ with tools from descriptive set theory.

Theorem 28.1 (Privacy for Qbs) Qbs *validates the privacy equation* (PRIV). *This means that the random singleton is indistinguishable from the empty set:*

$$\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [\{x\}] = [\emptyset] : P(2^{\mathbb{R}}) \tag{93}$$

In statistical notation, we would consider a Borel set-valued random variable $\{X\}$ where $X \sim \nu$. Privacy states that $\{X\} \overset{d}{=} \emptyset$ in distribution. Before presenting the proof of the theorem on page 183, we will try out several natural attempts at *distinguishing* $\emptyset$ from $\{X\}$, and see why those attempt *fail*.

To be completely formal, let us call the distribution on the left hand side of (93) $\mathsf{rs} \overset{\text{def}}{=} (\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [\{x\}])$ for 'random singleton'. In each of the attempts, we will consider some form of predicate $\rho : 2^{\mathbb{R}} \to Y$ into a test space Y, and compare the pushforwards $\rho_* \mathsf{rs}$ and $\rho_* \delta_\emptyset$. If those were different, we would have shown $\mathsf{rs} \neq \delta_\emptyset$; but in each case, we'll see that the pushforwards agree.

Attempt 28.2 (Membership test) Fix a number $x_0 \in \mathbb{R}$. We try and distinguish the two random sets by testing membership of x_0. However, this fails to distinguish $\{X\}$ from $\emptyset$, because X is sampled from an atomless distribution, that is

$$\Pr(x_0 \in \{X\}) = \Pr(X = x_0) = 0 = \Pr(x_0 \in \emptyset).$$

Formally, we're considering the evaluation morphism $\mathsf{ev}_{x_0} : 2^{\mathbb{R}} \to 2$ and find that its pushforward fails to distinguish rs from $\delta_\emptyset$, because

$$(\mathsf{ev}_{x_0})_* \mathsf{rs} = \delta_{\text{false}} = (\mathsf{ev}_{x_0})_* \delta_\emptyset$$

Note that the failure of this attempt only uses freshness. As discussed in (85), this is a strictly weaker statement than Privacy because λ and ν don't commute (and freshness *does* hold in the nominal set model, while Privacy does not).

Attempt 28.3 (Applying measures) We try to distinguish $\{X\}$ from $\emptyset$ by considering the value assigned to these sets by some test measure μ on the reals. Of course we always have $\mu(\emptyset) = 0$, so now we have to analyze the distribution of the number $\mu(\{X\})$.

If μ is an s-finite measure on $\mathbb{R}$, then evaluating it is morphism of quasi-Borel spaces

$$\mu : 2^{\mathbb{R}} \to [0, \infty]$$

as shown in [Ścibior et al., 2017, §4.3]. We can however show that $\mu(\{X\}) = 0$ almost surely, hence $\mu_* \mathsf{rs} = \delta_0 = \mu_* \delta_\emptyset$. This is because by s-finiteness, the set $S = \{x : \mu(\{x\}) > 0\}$ of atoms of μ must be countable, and $\mu(\{X\}) \neq 0$ only if $X \in S$, which has probability zero under the atomless measure ν.

The condition that μ is s-finite is crucial for the attempt above: For example, the counting measure $|-|$ *would* distinguish singletons from the emptyset, but as we'll now see, the function

$$|-| : 2^{\mathbb{R}} \to \mathbb{N} \cup \{\infty\}$$

is not a morphism of quasi-Borel spaces and thus cannot serve as a valid predicate in the attempt. The use of the counting measure is similar to the idea of using the nonemptyness check $2^{\mathbb{A}} \to 2$ to break privacy in nominal sets Section 26.1. We now show that this function is incompatible with Borel-based probability:

Proposition 28.4 *The nonemptiness check $\exists : 2^{\mathbb{R}} \to 2$ is not a morphism of quasi-Borel spaces.*

PROOF We repeat the argument from page 156 in quasi-Borel spaces: Assume towards a contradiction that $\exists : 2^{\mathbb{R}} \to 2$ were a morphism. There exists a Borel subset $B \subseteq \mathbb{R}^2$ of the plane whose projection $\pi(B) \subseteq \mathbb{R}$ is not Borel (e.g. [Kechris, 1987, 14.2]). The characteristic function $\chi_B : \mathbb{R} \times \mathbb{R} \to 2$ is a morphism, and so is its currying $\beta : \mathbb{R} \to 2^{\mathbb{R}}$. We obtain that the characteristic function of the projection $\chi_{\pi(B)} = \exists \circ \beta$ is a morphism, because it is the composite of two morphisms. But $\pi(B)$ is not Borel, resulting in a contradiction. ∎

An immediate corollary is that the singleton set $\{\emptyset\} \subseteq 2^{\mathbb{R}}$ is *not* measurable. Furthermore, the equality check between sets $2^{\mathbb{R}} \times 2^{\mathbb{R}} \to 2$ is not a morphism in Qbs.

Attempt 28.5 (Nonemptiness check II) Recall from Proposition 27.10 that Qbs is a quasitopos; we *can* define a nonemptiness check

$$\exists : 2^{\mathbb{R}} \to \Omega$$

whose codomain is the weak subobject classifier Ω. Ω is the space $\{\text{true}, \text{false}\}$ with the *indiscrete* structure. This poses no contradiction to Privacy because Ω is an indiscrete space and hence $P(\Omega) \cong 1$. Applying $\exists_*$ to the Privacy equation results in the peculiar but valid identity $[\text{true}] = [\text{false}] \in P(\Omega)$. The space Ω violates (MONO) requirement.

Proof of the privacy equation: Generalizing the case of the nonemptiness predicate, the strategy is to show that any predicate $2^{\mathbb{R}} \to 2$ which can distinguish $\{X\}$ and $\emptyset$ *cannot be measurable*. For this we must analyze the σ-algebra $\Sigma_{2^{\mathbb{R}}}$ on $2^{\mathbb{R}}$ in detail. Recall the notation $A_x = \{y : (x, y) \in A\}$ for the slice of a subset. If we identify the space $2^{\mathbb{R}} = \mathsf{Qbs}(\mathbb{R}, 2)$ with the Borel subsets of $\mathbb{R}$ then every morphism $\alpha : \mathbb{R} \to 2^{\mathbb{R}}$ is of the form $x \mapsto A_x$ for $A \subseteq \mathbb{R}^2$ Borel. If $\mathcal{U} \subseteq 2^{\mathbb{R}}$ is a collection of Borel sets, then $\alpha^{-1}(\mathcal{U}) = \{x : A_x \in \mathcal{U}\}$. This yields the following characterization of the induced σ-algebra $\Sigma_{2^{\mathbb{R}}}$:

Definition 28.6 (e.g. Kechris [1987]) A collection $\mathcal{U} \subseteq 2^{\mathbb{R}}$ of Borel sets is *Borel-on-Borel* if for all Borel $A \subseteq \mathbb{R}^2$, the set $\{x \in \mathbb{R} \mid A_x \in \mathcal{U}\}$ is Borel.

A family $\mathcal{U}$ is measurable in $2^{\mathbb{R}}$ if and only if it is Borel-on-Borel. Examples of such families include the examples from before, like the family of null sets with respect to a Borel probability measure, but also plenty further notions like the family of meager sets (e.g. [Kechris, 1987, §18.B]). The collection $\{\emptyset\}$ is not Borel-on-Borel. No general classification of Borel-on-Borel families seems known.

We will now proceed to prove a key lemma (Lemma 28.9) about Borel-on-Borel families which can be used to derive Privacy. A first proof this lemma was communicated to the authors by Alexander Kechris. We have since developed an independent proof based on the notion of Borel inseparability (Definition 28.7):

Definition 28.7 (e.g. Kechris [1987]) Let X be a standard Borel space. Two disjoint sets $A, A' \subseteq X$ are said to be *Borel separable* if there is a Borel set $B \subseteq X$ such that $A \subseteq B$ and $A' \cap B = \emptyset$. A, A' are *Borel inseparable* if no such set exists.

We will now exhibit a pathological Borel set B whose existence renders distinguishing $\{X\}$ from $\emptyset$ inconsistent:

Theorem 28.8 (Becker [Kechris, 1987, 35.2]) *There exists a Borel set $B \subseteq \mathbb{R}^2$ such that the sets*

$$B^0 \overset{def}{=} \{x \in \mathbb{R} \mid B_x = \emptyset\} \quad \textit{and} \quad B^1 \overset{def}{=} \{x \in \mathbb{R} \mid B_x \textit{ is a singleton}\}$$

are Borel inseparable.

Lemma 28.9 *Let $\mathcal{U} \subseteq 2^{\mathbb{R}}$ be Borel on Borel. If $\emptyset \in \mathcal{U}$ then $\{r\} \in \mathcal{U}$ for all but countably many $r \in \mathbb{R}$.*

PROOF Let $A = \{r \in \mathbb{R} \mid \{r\} \notin \mathcal{U}\}$. This is a Borel set because $\mathcal{U}$ is Borel-on-Borel.

Now suppose for the sake of contradiction that A were uncountable. Because A is an uncountable Borel subset of a standard Borel space, it must be isomorphic to $\mathbb{R}$ (Section 4.2). Fixing such an isomorphism, we find by Theorem 28.8 a Borel set $B \subseteq \mathbb{R} \times A$ such that B^0, B^1 are Borel inseparable.

Now, if $r \in B^0$ then $B_r = \emptyset \in \mathcal{U}$. On the other hand, if $r \in B^1$ then $B_r = \{a\}$ for some $a \in A$, and so $B_r = \{a\} \notin \mathcal{U}$. It follows that $B^0 \subseteq \{r \in \mathbb{R} \mid B_r \in \mathcal{U}\}$ and $B^1 \subseteq \{r \in \mathbb{R} \mid B_r \notin \mathcal{U}\}$. As $\mathcal{U}$ is Borel on Borel, $\{r \in \mathbb{R} \mid B_r \in \mathcal{U}\}$ provides a Borel separation of B^0, B^1, a contradiction. ∎

We obtain the Privacy equation as a corollary of this lemma:

PROOF (PROOF OF THEOREM 28.1) To show that the two quasi-Borel measures in (93) are equal, we must check that their pushforward measures agree on the measurable space $(2^{\mathbb{R}}, \Sigma_{2^{\mathbb{R}}})$, meaning that for all $\mathcal{U} \in \Sigma_{2^{\mathbb{R}}}$ we have

$$\emptyset \in \mathcal{U} \iff \nu(\{r \in \mathbb{R} \mid \{r\} \in \mathcal{U}\}) = 1.$$

Every such $\mathcal{U}$ is Borel-on-Borel, and by possibly taking complements we can assume that $\emptyset \in \mathcal{U}$. By Lemma 28.9, the set $\{r \in \mathbb{R} \mid \{r\} \in \mathcal{U}\}$ is co-countable, and because ν is atomless this must have ν-measure 1. ∎

Generalizations: We offer some comments about this proof. The strategy we employed is surprisingly general and extends beyond the category of quasi-Borel spaces. Take any model of higher-order probability which agrees with standard Borel spaces on ground types, or at least satisfies the following minimum requirements

(i) all morphisms $\mathbb{R} \to 2$ are Borel measurable

(ii) all Borel maps $\mathbb{R}^2 \to 2$ are available as morphisms

Then the Borel-on-Borel property is a necessary constraint on second-order functions $2^{\mathbb{R}} \to 2$, arising from cartesian closure alone. Also Becker's set is present in the model, so Lemma 28.9 applies and it is inconsistent for any predicate to tell apart the empty set from a random singleton with positive probability.

It is now merely an extensionality aspect of the Qbs that these constraints are sufficient for Privacy, that is the inability to distinguish the empty set from singletons implies equality in distribution. The category of sheaves in [Staton et al., 2016] features a more intensional probability monad, where the two sides of the privacy equation presumably cannot be identified.

28.1 Consequences

Having proved the privacy equation, we can immediately apply the results from Section 26 to quasi-Borel spaces. In particular

(i) the probability theory on function spaces like $2^{\mathbb{R}}$ is non-positive (Proposition 26.3)

(ii) conditionals on $2^{\mathbb{R}}$ cannot exist in general, because that would mean extracting private information (Proposition 26.4)

We can derive further interesting corollaries: For example, the probability theory on Meas *is* positive (Proposition 9.5). This mismatch with Qbs allows us to re-derive Example 27.9 in a purely probabilistic way:

Proposition 28.10 *The functor Σ does not preserve the product $2^{\mathbb{R}} \times \mathbb{R}$.*

PROOF The distribution $\psi = \mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [(\{x\}, x)]$ on the measurable space $\Sigma(2^{\mathbb{R}}) \times \Sigma(\mathbb{R})$ still has a deterministic first marginal by Privacy. If $(\ni) : \Sigma(2^{\mathbb{R}}) \times \Sigma(\mathbb{R}) \to 2$ were measurable, we could use it to show that ψ is not the product of its marginals, contradicting the positivity of Meas. This is a special case of Aumann's result that $(\ni)$ is not measurable in any product-σ-algebra. ∎

Similar arguments can be constructed to show that other products are not preserved: For example, the Lebesgue measure $\ell : 2^{[0,1]} \to [0,1]$ is a morphism [Ścibior et al., 2017, §4.3] which has a section given by sending the value $a \in [0,1]$ to the interval $[0,a]$ of length a. The distribution

$$(\mathsf{let}\, a \leftarrow \nu\, \mathsf{in}\, [(\{a\}, [0,a])]) \in P(2^{[0,1]} \times 2^{[0,1]})$$

violates the deterministic marginal property as in Proposition 28.10, showing that Σ does not preserve the product $2^{[0,1]} \times 2^{[0,1]}$. Because $\mathbb{R} \cong [0,1]$ in Qbs, the same is true for $2^{\mathbb{R}} \times 2^{\mathbb{R}}$.

Another interesting corollary concerns the difference between random functions and their graphs:

Example 28.11 Let the injection $\Gamma : \mathbb{R}^{\mathbb{R}} \to 2^{\mathbb{R}\times\mathbb{R}}$ send a function to its graph $\Gamma f(x,y) = [y = f(x)]$. There exist distinct random functions with identical random graphs.

PROOF Consider the following random functions in $P(\mathbb{R}^{\mathbb{R}})$ from Example 24.5

$$f = \mathsf{let}\, a \leftarrow \nu\, \mathsf{in}\, [\lambda x.a], \quad g = \mathsf{let}\, a \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, b \leftarrow \nu\, \mathsf{in}\, [\lambda x.\mathsf{if}\ x = a\ \mathsf{then}\ b\ \mathsf{else}\ a]$$

Then $f \neq g$ can be witnessed by calling f *twice*, i.e. if $p : \mathbb{R}^{\mathbb{R}} \to 2$ is defined by

$$p(h) \stackrel{\text{def}}{=} [h(0) = h(h(0))]$$

then $p_* f = [\text{true}]$ while $p_* g = [\text{false}]$. The graphs of f, g are the following random subsets of $\mathbb{R}^2$

$$\Gamma f = \mathsf{let}\, a \leftarrow \nu\, \mathsf{in}\, [\lambda(x,y).(y = a)]]$$
$$\Gamma g = \mathsf{let}\, a \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, b \leftarrow \nu\, \mathsf{in}\, [\lambda(x,y).\mathsf{if}\ x = a\ \mathsf{then}\ (y = b)\ \mathsf{else}\ (y = a)]$$

Using Privacy, we obtain that both graphs are in fact equal to the empty set $[\lambda(x,y).\text{false}]$. ∎

Using Lemma 28.9, we can also show that $2^{\mathbb{R}}$ is not standardly generated (Definition 27.5).

Proposition 28.12 *The space $2^{\mathbb{R}}$ is not isomorphic to $M(X)$ for any measurable space X.*

PROOF By Definition 27.5 it is sufficient to show that $M_{2^{\mathbb{R}}}$ is strictly smaller than $M_{\Sigma_{2^{\mathbb{R}}}}$. Let $f : \mathbb{R} \to \mathbb{R}$ be a bijective function that is not measurable, and let $A \subseteq \mathbb{R}^2$ be the graph of f. By [Srivastava, 1998, Theorem 4.5.2], the set A is not Borel and hence the map $\alpha : \mathbb{R} \to 2^{\mathbb{R}}, x \mapsto A_x = \{f(x)\}$ does not lie in $M_{2^{\mathbb{R}}}$. However $\alpha \in M_{\Sigma_{2^{\mathbb{R}}}}$, that is α is a measurable map from $\mathbb{R}$ to $(2^{\mathbb{R}}, \Sigma_{2^{\mathbb{R}}})$. Namely, for every $\mathcal{U} \in \Sigma_{2^{\mathbb{R}}}$, we have $\alpha^{-1}(\mathcal{U}) = \{x : \{f(x)\} \in \mathcal{U}\}$. By Lemma 28.9, the set $S = \{x : \{x\} \in \mathcal{U}\}$ is always countable or cocountable, and so is $\alpha^{-1}(\mathcal{U}) = f^{-1}(S)$ by bijectivity of f. So the preimage is a Borel set as desired. ∎

29 Full Abstraction at First-Order Types

Building on the Privacy equation, we will now prove that Qbs semantics is fully abstract for first-order types. The proof proceeds in two stages:

- In Section 29.1, we construct a normal form for observational equivalence at first-order types which eliminates the use of private names. The normalization algorithm is of interest on its own right, and builds on a logical relation due to Pitts and Stark [1993].
- In Section 29.2, we show that Qbs validates our normalization procedure and is therefore fully abstract at first-order types. We make use of a measure-invariant group structure on the space of names to reduce this problem to the privacy equation: Privacy is in this sense the prototypical observational equivalence at first-order types. The group-theoretic argument amounts to a novel way of treating private names as interchangeable, which is distinct but related to the idea of equivariance in nominal sets (Section 25).

29.1 A Normal Form for Observational Equivalence

We define a normalization procedure for first-order expressions which lets us decide their observational equivalence. The key idea is to identify and eliminate *private names* in such expressions, as those are not observable. At first-order types, this is the only obstruction, unlike for higher types where names can be partially revealed in more complicated ways (e.g. Example 15 in [Stark, 1994]).

Convention: In order to save space, we will abbreviate the types name and bool as N and B respectively.

η-normal form: Let τ be a first-order type. By the deterministic reduction relation of the ν-calculus, it is possible to expand every expression $M \in \mathsf{Exp}_\tau(s)$ into an η-normal form. Ground values are already in normal form, and in a λ-abstraction, we must immediately and exhaustively case-analyze its argument. If $s = \{n_1, \ldots, n_k\}$ is a set of names, we introduce the notation

$$\mathsf{if}\ x = n \in s\ \mathsf{then}\ M_n\ \mathsf{else}\ M_0$$

as shorthand for the case-split

$$\mathsf{if}\ x = n_1\ \mathsf{then}\ M_{n_1}\ \mathsf{else if}\ \cdots\ \mathsf{else if}\ x = n_k\ \mathsf{then}\ M_{n_k}\ \mathsf{else}\ M_0$$

Note that in M_0, the variable x is known to be distinct from all $n \in s$ and can thus be treated as a fresh name. We also write $\nu s.M$ as shorthand for $\nu n_1 \ldots . \nu n_k . M$. The η-normal forms can be formally defined as follows

$$\begin{aligned}
\mathsf{NF}^{\mathsf{val}}_{\mathsf{B}}(s) &= \{\mathsf{true}, \mathsf{false}\} \qquad \mathsf{NF}^{\mathsf{val}}_{\mathsf{N}}(s) = \{n \mid n \in s\} \\
\mathsf{NF}^{\mathsf{val}}_{\mathsf{B}\to\tau}(s) &= \{\lambda b.\mathsf{if}\ b\ \mathsf{then}\ M_{\mathrm{true}}\ \mathsf{else}\ M_{\mathrm{false}} \mid M_{\mathrm{true}}, M_{\mathrm{false}} \in \mathsf{NF}^{\mathsf{exp}}_\tau(s)\} \\
\mathsf{NF}^{\mathsf{val}}_{\mathsf{N}\to\tau}(s) &= \{\lambda x.\mathsf{if}\ x = n \in s\ \mathsf{then}\ M_n\ \mathsf{else}\ M_\nu \mid (\forall n \in s) M_n \in \mathsf{NF}^{\mathsf{exp}}_\tau(s), M_0 \in \mathsf{NF}^{\mathsf{exp}}_\tau(s \oplus \{x\})\} \\
\mathsf{NF}^{\mathsf{exp}}_\tau(s) &= \{\nu t.M \mid M \in \mathsf{NF}^{\mathsf{val}}_\tau(s \oplus t)\}
\end{aligned}$$

Every $M \in \mathsf{Exp}_\tau(s)$ is provably (in the metalanguage) equal to an η-normal form.

Public normal form: Different η-normal forms can still be observationally equivalent; the Privacy equation implies that the following normal forms behave the same:

$$\nu a.\lambda x.\mathsf{if}\ x = a\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false} \approx \lambda x.\mathsf{false}$$

The creation of the private name a prevents the normal forms from being unique. We therefore aim to restrict η-normal forms to *public normal forms* where, whenever new names are created in an expression $\nu t.M$, all $n \in t$ must be *public*, i.e. observable in some appropriate sense. Every η-normal form can be refined to a public normal form by identifying which names are private and eliminating them, as well all case-statements associated to them. This is reminiscent of escape analysis and garbage collection (more in Section 30.2).

To understand this process better, consider the transposition function $(a\,b) : \mathsf{N} \to \mathsf{N}$ which swaps a and b

$$(a\,b) \stackrel{\mathrm{def}}{=} \lambda x.\mathsf{if}\ x = a\ \mathsf{then}\ b\ \mathsf{else if}\ x = b\ \mathsf{then}\ a\ \mathsf{else}\ x \tag{94}$$

One can show that the fresh transposition is observationally equivalent to the identity function, because both names a and b remain private

$$\nu a.\nu b.\lambda x.(a\,b)\;x \approx \lambda x.x \tag{95}$$

Removing the two unreachable if–branches $x = a$ and $x = b$ makes both sides equal

$$\lambda x.\,\text{~~if } x = a \text{ then } b \text{ else if } x = b \text{ then } a \text{ else~~}\; x = \lambda x.x \tag{96}$$

This example reveals another subtlety: Privacy of names is not an absolute concept, but must be defined relative to what other names are publicly known. a and b are *jointly* private, but if say a were public, then we could use the transposition to also reveal b.

In principle, given an η-normal form $M \in \mathsf{NF}_\tau(t)$ and a subset of initially known public names $s \subseteq t$, one can give an inductive definition of the set $\mathsf{Pub}(M,s) \subseteq t$ of names which can be revealed by accessing M. We have $s \subseteq \mathsf{Pub}(M,s)$ because names from s are already public. More generally $\mathsf{Pub}(M,-)$ is a closure operator, i.e. also monotonic and idempotent. From the previous examples, we have

$$\mathsf{Pub}((a\,b),\emptyset) = \emptyset \qquad \mathsf{Pub}((a\,b),\{a\}) = \{a,b\} \tag{97}$$

We call an expression M s-safe (written $M \in \mathsf{Safe}^s_\tau$) if $\mathsf{Pub}(M,s) = s$, that is no names beyond s get leaked. Consequently, the complement $t \setminus s$ contains the *private names* of M. We give formal definitions of these concepts in Definition 29.2.

Given $M \in \mathsf{Safe}^s_\tau$, we define a normalization procedure that turns it into a public normal form $\mathsf{pnf}(M,s) \in \mathsf{NF}_\tau(s)$, eliminating all private names recursively. That normal form is shown observationally equivalent to M if only names from s may be used, and is unique among equivalent terms.

Instead of reinventing all the required combinatorics, we will now express the ideas of public and private names using a logical relation by Pitts and Stark [1993], which we also use to prove uniqueness and observational equivalence of our normal forms. This approach is in particular due to Michael Wolman.

Logical relation at first-order types: Let s_1, s_2 be sets of names; we write $R\colon s_1 \leftrightharpoons s_2$ for a partial bijection or *span* between s_1 and s_2. We write $R \oplus R'$ for the disjoint union of spans between disjoint sets of names, and we write $\mathsf{id}_s\colon s \oplus t_1 \leftrightharpoons s \oplus t_2$ to denote the partial bijection defined that is the identity on s and undefined on t_1, t_2.

Pitts and Stark [1993] define two families of relations $R^{\mathsf{val}}_\tau \subseteq \mathsf{Val}_\tau(s_1) \times \mathsf{Val}_\tau(s_2)$ and $R^{\mathsf{exp}}_\tau \subseteq \mathsf{Exp}_\tau(s_1) \times \mathsf{Exp}_\tau(s_2)$ by mutual induction, given in Figure 11. We note that R^{val}_τ and R^{exp}_τ coincide at values, so we will simply write the relations as R_τ. Additionally, by renaming related names we can without loss of generality reduce any span R to a subdiagonal $R = \mathsf{id}_s$ between sets $s_i = s \oplus t_i$.

The logical relation coincides with observational equivalence ($\approx$) at first-order types:

Theorem 29.1 (Pitts and Stark [1993, Theorem 22]) *Let τ be a first-order type. Then for $M_1, M_2 \in \mathsf{Exp}_\tau(s)$ we have*

$$M_1 \approx_\tau M_2 \Leftrightarrow M_1\,(\mathsf{id}_s)_\tau\,M_2$$

$$b_1 \, R_{\mathsf{B}}^{\mathsf{val}} \, b_2 \Leftrightarrow b_1 = b_2 \qquad n_1 \, R_{\mathsf{N}}^{\mathsf{val}} \, n_2 \Leftrightarrow n_1 \, R \, n_2$$

$$\begin{aligned}(\lambda x.M_1) \, R_{\sigma\to\tau}^{\mathsf{val}} \, (\lambda x.M_2) \Leftrightarrow \forall R' \colon s_1' \rightleftharpoons s_2', V_1 \in \mathsf{Val}_\sigma(s_1 \oplus s_1'), V_2 \in \mathsf{Val}_\sigma(s_2 \oplus s_2'),\\ V_1 \, (R \oplus R')_\sigma^{\mathsf{val}} \, V_2 \Rightarrow M_1[V_1/x] \, (R \oplus R')_\tau^{\mathsf{exp}} \, M_2[V_2/x]\end{aligned}$$

$$\begin{aligned}M_1 \, R_\tau^{\mathsf{exp}} \, M_2 \Leftrightarrow \exists R' \colon s_1' \rightleftharpoons s_2', V_1 \in \mathsf{Val}_\tau(s_1 \oplus s_1'), V_2 \in \mathsf{Val}_\tau(s_2 \oplus s_2'),\\ s_1 \vdash M_1 \Downarrow_\tau (s_1')V_1 \,\&\, s_2 \vdash M_2 \Downarrow_\tau (s_2')V_2 \,\&\, V_1 \, (R \oplus R')_\tau^{\mathsf{val}} \, V_2\end{aligned}$$

Figure 11: Stark's logical relation

We will see that the logical relation (id_s) has the following specific interpretation at first-order types τ:

(i) the relation $(\mathsf{id}_s)_\tau$ is a partial equivalence relation, in particular it is transitive.

(ii) a term $M \in \mathsf{Exp}_\tau(t)$ is related to itself as $M \, (\mathsf{id}_s)_\tau \, M$ if and only if M does not leak any further names if the names s are known

(iii) $(\mathsf{id}_s)_\tau$ relates expressions which are observationally equivalent given names s are public

This motivates the following definitions

Definition 29.2 (Public and Leaked Names) Let $M \in \mathsf{Exp}_\tau(t)$ and $s \subseteq t$. We define the set of public names in M given s, denoted $\mathsf{Pub}(M, s)$, to be the least set u such that $s \subseteq u$ and $M \, (\mathsf{id}_u)_\tau \, M$. We define the leaked names $\mathsf{Leak}(M, s) \stackrel{\text{def}}{=} \mathsf{Pub}(M, s) \setminus s$ as the names that are revealed beyond s. A term is s-safe if it satisfies any of the equivalent conditions

$$\mathsf{Pub}(M, s) = s \quad \Leftrightarrow \quad \mathsf{Leak}(M, s) = \emptyset \quad \Leftrightarrow \quad M \, (\mathsf{id}_s) \, M$$

We write Safe_τ^s for the set of s-safe expressions.

We show these definitions are well-defined in Proposition 34.4. The following example shows how to reason with the logical relation, and identify private and public names

Example 29.3 The privacy equation for the ν-calculus can be established by means of the logical relation. Because $\{a, x\} \vdash (x = a) \Downarrow_{\mathsf{B}} \text{false}$ whenever a, x are distinct names, the following terms are in relation

$$\begin{aligned}\lambda x.(x = a) \, (\mathsf{id}_\emptyset)_{\mathsf{N}\to\mathsf{B}} \, \lambda x.\text{false}\\ \nu a.\lambda x.(x = a) \, (\mathsf{id}_\emptyset)_{\mathsf{N}\to\mathsf{B}} \, \nu a.\lambda x.\text{false},\end{aligned}$$

By Theorem 29.1 this proves the observational equivalence Equation (86). According to Definition 29.2, the unmatched name a is private. A similar analysis for the transposition $(a\,b)$ shows that our definitions validate our examples (95), (97) as desired.

Given $M \in \mathsf{Exp}_\tau(s \oplus t)$ such that $M \in \mathsf{Safe}^s_\tau$ holds, we wish to define a normal form which eliminates all unnecessary private names t. We do this by η-expanding M into a big case analysis, and dropping all branches pertaining to private names $n \in t$, because those branches can never be called.

Definition 29.4 (Public normal form) Let σ be a first-order type and $M \in \mathsf{Exp}_\sigma(s \oplus t)$ such that $M \in \mathsf{Safe}^s_\sigma$ holds. We define the normal form $\mathsf{pnf}(M, s)$ by induction on σ as follows:

Ground case: If σ is a ground type and M is a value, then we let $\mathsf{pnf}(M, s) \stackrel{\text{def}}{=} M$.

Function case $\mathsf{B} \to \tau$: If M is a value of type $\mathsf{B} \to \tau$, expand it into its η-normal form

$$M = \lambda x.\mathsf{if}\ x = \mathsf{true}\ \mathsf{then}\ M_{\mathrm{true}}\ \mathsf{else}\ M_{\mathrm{false}}$$

for some $M_{\mathrm{true}}, M_{\mathrm{false}} \in \mathsf{Exp}_\tau(s \oplus t)$. We know $M\ (\mathsf{id}_s)_\sigma\ M$, so by definition of the logical relation, we also have $M_{\mathrm{true}}\ (\mathsf{id}_s)_\tau\ M_{\mathrm{true}}$ and $M_{\mathrm{false}}\ (\mathsf{id}_s)_\tau\ M_{\mathrm{false}}$. We thus define

$$\mathsf{pnf}(M, s) \stackrel{\text{def}}{=} \lambda x.\mathsf{if}\ x = \mathsf{true}\ \mathsf{then}\ \mathsf{pnf}(M_{\mathrm{true}}, s)\ \mathsf{else}\ \mathsf{pnf}(M_{\mathrm{false}}, s).$$

Function case $\mathsf{N} \to \tau$: If M is a value of type $\mathsf{N} \to \tau$, expand it into its η-normal form

$$M = \lambda x.\mathsf{if}\ x = n \in s \oplus t\ \mathsf{then}\ M_n\ \mathsf{else}\ M_0$$

for some $M_n \in \mathsf{Exp}_\tau(s \oplus t)$ and $M_0 \in \mathsf{Exp}_\tau(s \oplus t \oplus \{x\})$. In this case, $M\ (\mathsf{id}_s)_\sigma\ M$ implies that $M_0\ (\mathsf{id}_{s \oplus \{x\}})_\tau\ M_0$ and $M_n\ (\mathsf{id}_s)_\tau\ M_n$ for all $n \in s$; nothing can be said for $n' \in t$ because such $M_{n'}$ are not observable. We define the normal form as a case split over s only, while the cases for the private names t are dropped like in (96):

$$\mathsf{pnf}(M, s) \stackrel{\text{def}}{=} \lambda x.\mathsf{if}\ x = n \in s\ \mathsf{then}\ \mathsf{pnf}(M_n, s)\ \mathsf{else}\ \mathsf{pnf}(M_0, s \oplus \{x\}).$$

Expression case: Because $M\ (\mathsf{id}_s)_\sigma\ M$, there is some $V \in \mathsf{Val}_\sigma(s \oplus t \oplus w)$ such that $s \oplus t \vdash M \Downarrow_\sigma (w)V$ and $V\ (\mathsf{id}_{s \oplus w'})_\sigma\ V$ for some $w' \subseteq w$. Let $u = \mathsf{Leak}(M, s) \subseteq w'$ consist of those names which can actually leak from V. Then $V\ (\mathsf{id}_{s \oplus u})_\tau\ V$, so we can define

$$\mathsf{pnf}(M, s) \stackrel{\text{def}}{=} \nu u.\mathsf{pnf}(V, s \oplus u).$$

That is, we need only create fresh names u which become public given s. All other private names get eliminated by the normal form.

We verify the following desired properties of the normal form construction.

Proposition 29.5 *Let τ be a first-order type and $M \in \mathsf{Exp}_\tau(s \oplus t)$ with $M \in \mathsf{Safe}^s_\tau$. Then*

(i) $\mathsf{pnf}(M, s)$ *is well-defined up to renaming bound variables and names*

(ii) if M is a value, so is $\mathsf{pnf}(M, s)$

(iii) $\mathsf{pnf}(M, s) \in \mathsf{Exp}_\tau(s)$, *that is* $\mathsf{pnf}(M, s)$ *eliminates the private names t*

(iv) $\mathsf{pnf}(M,s) \in \mathsf{Safe}^s_\tau$

(v) $M\ (\mathsf{id}_s)_\tau\ \mathsf{pnf}(M,s)$

PROOF In the appendix (Proposition 34.6). ∎

We can now reduce the question of observationally equivalence to the equality of normal forms:

Theorem 29.6 *Let σ be a first-order type and let $M_i \in \mathsf{Exp}_\sigma(s \oplus t_i)$ for $i = 1,2$. The following are equivalent:*

(i) $M_1\ (\mathsf{id}_s)_\sigma\ M_2$*;*

(ii) $M_i \in \mathsf{Safe}^s_\sigma$ *and* $\mathsf{pnf}(M_1,s) = \mathsf{pnf}(M_2,s)$ *after possibly renaming bound variables and names.*

PROOF In the appendix (Theorem 34.7). ∎

This shows that at first-order types, the elimination of private names is sufficient for full abstraction.

29.2 Proof of Full Abstraction

We give a semantic corollary of Theorem 29.6: A model is fully abstract if and only if it validates passing to public normal forms.

Theorem 29.7 *Let $(\mathbb{C}, T)$ be a categorical model of the ν-calculus. $\mathbb{C}$ is fully abstract at first-order types if and only if for all first-order types τ and all $M \in \mathsf{Exp}_\tau(s)$, we have*

$$[\![M]\!]_{\neq s} = [\![\mathsf{pnf}(M,s)]\!]_{\neq s}. \tag{98}$$

PROOF That this is necessary is clear, as by Proposition 29.5 normal forms preserve logical relations and therefore (by Theorem 29.1) observational equivalence. To see that it is sufficient, suppose that $\mathbb{C}$ satisfies (98) and let $M_1, M_2 \in \mathsf{Exp}_\tau(s)$ for a first-order type τ. If $M_1 \approx_\tau M_2$, then by Theorems 29.1 and 29.6 $\mathsf{pnf}(M_1,s) = \mathsf{pnf}(M_2,s)$, and so

$$[\![M_1]\!]_{\neq s} = [\![\mathsf{pnf}(M_1,s)]\!]_{\neq s} = [\![\mathsf{pnf}(M_2,s)]\!]_{\neq s} = [\![M_2]\!]_{\neq s}$$ ∎

The goal of this section is to show that quasi-Borel spaces validate the normalization procedure. It will be convenient to pick the unit circle $\mathbb{T} = \mathbb{S}^1$ as the space of names, and let ν be the uniform measure on $\mathbb{T}$ (we may assume this is the case by Section 26.3). We choose to work with the circle as there is a canonical group structure $(\mathbb{T}, +)$ on $\mathbb{T}$ (addition modulo 1) that is *ν-invariant*. This means that the structure maps $+ \colon \mathbb{T} \times \mathbb{T} \to \mathbb{T}$ and $- \colon \mathbb{T} \times \mathbb{T} \to \mathbb{T}$ are morphisms and for all $g \in \mathbb{T}$, the shift map $x \mapsto g + x$ preserves the measure ν, i.e. $\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, [g + x] = \nu$. More generally this implies that for all $f : \mathbb{T} \to P(X)$ and $g \in \mathbb{T}$,

$$\mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, f(g+x) = \int f(g+x)\nu(\mathrm{d}x) = \int f(x)\nu(\mathrm{d}x) = \mathsf{let}\, x \leftarrow \nu\, \mathsf{in}\, f(x)$$

The idea of ν-invariance will be used to treat private names as interchangeable in Qbs. This seems to us a different but related idea to the equivariance in nominal sets (Section 25). Note

that in contrast to the space $\mathbb{T}$, the nominal set $\mathbb{A}$ of atoms does not admit any group structure.

The trick is that in an η-normal form, if a private name a appears in an if-branch $x = b$, then b must itself be private, because otherwise we could leak a. This gives private names a form of dependency on another, which we can use to shift them jointly using the group action. Names shifted that way can be eliminated using a clever combination of a set-parameter, invariance and the privacy equation. We consider the following instructive example:

Example 29.8 We have seen that the fresh transposition normalizes to the identity function,

$$\mathsf{pnf}(\nu a.\nu b.\lambda x.(a\,b)x, \emptyset) = \lambda x.x.$$

We wish to show that their semantics are equated in Qbs, i.e. $[\![\nu a.\nu b.\lambda x.(a\,b)x]\!] = [\![\lambda x.x]\!]$ as elements of $P(P(\mathbb{T})^{\mathbb{T}})$. To do this, we define a helper function $f : 2^{\mathbb{T}} \times \mathbb{T}^3 \to \mathbb{T}$ as follows:

$$f(G,a,b,x) = \begin{cases} (x-a)+b & \text{if } x-a \in G, \\ (x-b)+a & \text{else if } x-b \in G, \\ x & \text{otherwise.} \end{cases}$$

This function behaves like a generalized transposition, with an extra set-argument G. If $G = \emptyset$, then $f(\emptyset,a,b,x) = x$ is just the identity on x. If $G = \{g\}$ is a singleton, then

$$f(\{g\},a,b,x) = \begin{cases} g+b & \text{if } x = g+a, \\ g+a & \text{else if } x = g+b, \\ x & \text{otherwise,} \end{cases}$$

which is a transposition whose parameters have been shifted by g. We curry this to give a map $f' : 2^{\mathbb{T}} \times \mathbb{T}^2 \to P(P(\mathbb{T})^{\mathbb{T}})$ by $f'(G,a,b) = [\lambda x.[f(G,a,b,x)]]$, so that now

$$f'(\emptyset,a,b) = [\![\lambda x.x]\!] \quad \text{and} \quad f'(\{g\},a,b) = [\![\lambda x.(a\,b)x]\!](g+a, g+b),$$

and define $h : 2^{\mathbb{T}} \to P(P(\mathbb{T})^{\mathbb{T}})$ by integrating out the names a, b:

$$h(G) = \mathsf{let}\, a \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, b \leftarrow \nu\, \mathsf{in}\, f'(G,a,b)$$

It is clear that $h(\emptyset) = [\![\lambda x.x]\!]$. On the other hand, by the ν-invariance of the action we have

$$\begin{aligned} h(\{g\}) &= \mathsf{let}\, a \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, b \leftarrow \nu\, \mathsf{in}\, [\![\lambda x.(a\,b)x]\!](g+a, g+b) \\ &= \mathsf{let}\, a \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, b \leftarrow \nu\, \mathsf{in}\, [\![\lambda x.(a\,b)x]\!])(a,b) \\ &= [\![\nu a.\nu b.\lambda x.(a\,b)x]\!], \end{aligned}$$

independently of $g \in \mathbb{T}$.

Our problem can now be reduced to the privacy equation. We obtain

$$\begin{aligned}
[\![\lambda x.x]\!] &= \mathsf{let}\, G \leftarrow [\emptyset]\, \mathsf{in}\, h(G) \\
&= \mathsf{let}\, G \leftarrow (\mathsf{let}\, g \leftarrow \nu\, \mathsf{in}\, [\{g\}])\ \mathsf{in}\, h(G) \\
&= \mathsf{let}\, g \leftarrow \nu\, \mathsf{in}\, h(\{g\}) \\
&= \mathsf{let}\, g \leftarrow \nu\, \mathsf{in}\, [\![\nu a.\nu b.\lambda x.(a\, b)x]\!] \\
&= [\![\nu a.\nu b.\lambda x.(a\, b)x]\!],
\end{aligned}$$

where the second equality is Privacy and the final one discardability.

The crucial idea is that we can define the shifted transposition map using only $G = \{g\}$ but *without extracting* g (which is impossible by Privacy). The map h, which is parameterized by G, acts as an interpolant between the original denotation and the normal form. Setting $G = \emptyset$ removes the unreachable branches just like the normalization algorithm.

To prove the general full abstraction result, we need to construct such interpolants in a systematic way: **Notation:** If $\vec{t} = (t_1, \ldots, t_n)$ is a vector in $\mathbb{T}^n$ and $g \in \mathbb{T}$, we write $g + \vec{t} = (g + t_1, \ldots, g + t_n)$. Additionally, we write let $t \leftarrow \nu$ to be shorthand for drawing t samples in a sequence:

$$\mathsf{let}\, t_1 \leftarrow \nu\, \mathsf{in}\, \cdots \mathsf{let}\, t_k \leftarrow \nu.$$

Proposition 29.9 *Let τ be a first-order type and let $M \in \mathsf{Exp}_\tau(s \oplus t)$. If $M \in \mathsf{Safe}^s_\tau$, then there is a quasi-Borel map*

$$f : 2^{\mathbb{T}} \times \mathbb{T}^{\neq s \oplus t} \to P([\![\tau]\!])$$

such that

$$f(\emptyset, \vec{s}, \vec{t}) = [\![\mathsf{pnf}(M, s)]\!]_{\neq s}(\vec{s}) \ \text{ and } \ f(\{g\}, \vec{s}, \vec{t}) = [\![M]\!]_{\neq s \oplus t}(\vec{s}, g + \vec{t})$$

whenever $(\vec{s}, g + \vec{t}) \in \mathbb{T}^{\neq s \oplus t}$. In the case that $M = V$ is a value, f factors through the unit of the monad.

PROOF We construct f inductively, in parallel to the construction of the normal forms.

Ground case: If τ is a ground type and V is a value, then $\mathsf{pnf}(V, s) = V$ so we simply let

$$f(G, \vec{s}, \vec{t}) = [\![\mathsf{pnf}(V, s)]\!](\vec{s})$$

Function case $\mathsf{B} \to \tau$: Suppose that V is a value of type $\mathsf{B} \to \tau$. We η-expand V, so that

$$V = \lambda x.\mathsf{if}\ x = \mathsf{true}\ \mathsf{then}\ M_{\mathsf{true}}\ \mathsf{else}\ M_{\mathsf{false}}.$$

By definition of the logical relation and the normal form we have $M_{\mathsf{true}}, M_{\mathsf{false}} \in \mathsf{Safe}^s_\tau$ and

$$\mathsf{pnf}(V, s) = \lambda x.\mathsf{if}\ x = \mathsf{true}\ \mathsf{then}\ \mathsf{pnf}(M_{\mathsf{true}}, s)\ \mathsf{else}\ \mathsf{pnf}(M_{\mathsf{false}}, s).$$

Per inductive hypothesis, we find functions $f_{\mathsf{true}}, f_{\mathsf{false}} : 2^{\mathbb{T}} \times \mathbb{T}^{\neq s \oplus t} \to P([\![\tau]\!])$ satisfying the conditions of Proposition 29.9 for M_{true} and M_{false}. We then define $f : 2^{\mathbb{T}} \times \mathbb{T}^{\neq s \oplus t} \to P([\![\tau]\!])^2$ by

$$f(G, \vec{s}, \vec{t}) = \lambda x. \begin{cases} f_{\mathsf{true}}(G, \vec{s}, \vec{t}) & \text{if } x = \mathsf{true}, \\ f_{\mathsf{false}}(G, \vec{s}, \vec{t}) & \text{otherwise.} \end{cases}$$

It is clear that $f(\emptyset, \vec{s}, \vec{t}) = |\mathsf{pnf}(V,s)|(\vec{s})$ and $f(\{g\}, \vec{s}, \vec{t}) = |V|(\vec{s}, g + \vec{t})$ when $(\vec{s}, g + \vec{t}) \in \mathbb{T}^{\neq s \oplus t}$ from the properties of $f_{\mathrm{true}}, f_{\mathrm{false}}$.

Function case $\mathsf{N} \to \tau$: Suppose that V is a value of type $\mathsf{N} \to \tau$. We η-expand V, so that

$$V = \lambda x.\mathsf{if}\ x = n \in s \oplus t\ \mathsf{then}\ M_n\ \mathsf{else}\ M_0.$$

By definition of the logical relation and the normal form we have $M_0 \in \mathsf{Safe}_\tau^{s \oplus \{x\}}$, $M_n \in \mathsf{Safe}_\tau^s$ for $n \in s$ and

$$\mathsf{pnf}(V,s) = \lambda x.\mathsf{if}\ x = n \in s\ \mathsf{then}\ \mathsf{pnf}(M_n, s)\ \mathsf{else}\ \mathsf{pnf}(M_0, s \oplus \{x\}).$$

Inductively we find functions $f_n : 2^{\mathbb{T}} \times \mathbb{T}^{\neq s \oplus t} \to P(\llbracket \tau \rrbracket)$ for $n \in s$ and $f_0 : 2^{\mathbb{T}} \times \mathbb{T}^{\neq s \oplus t \oplus \{x\}} \to P(\llbracket \tau \rrbracket)$ satisfying the conditions of 29.9 for M_n and M_0. Writing $t = (t_1, \ldots, t_k)$, we define $f : 2^{\mathbb{T}} \times \mathbb{T}^{\neq s \oplus t} \to P(\llbracket \tau \rrbracket)^{\mathbb{T}}$ similar to Example 29.8 by

$$f(G, \vec{s}, \vec{t}) = \lambda x. \begin{cases} f_n(G, \vec{s}, \vec{t}) & \text{if } x = n \in \vec{s}, \\ \llbracket M_{t_1} \rrbracket(\vec{s}, (x - t_1) + \vec{t}) & \text{else if } (x - t_1) \in B, \\ \ldots & \\ \llbracket M_{t_k} \rrbracket(\vec{s}, (x - t_k) + \vec{t}) & \text{else if } (x - t_k) \in B, \\ f_0(G, \vec{s}, \vec{t}, x) & \text{otherwise.} \end{cases}$$

If $G = \emptyset$, then

$$f(\emptyset, \vec{s}, \vec{t}) = \lambda x. \begin{cases} \llbracket \mathsf{pnf}(M_n, s) \rrbracket(\vec{s}) & \text{if } x = n \in \vec{s}, \\ \llbracket \mathsf{pnf}(M_0, s) \rrbracket(\vec{s}) & \text{otherwise} \end{cases}$$

so that $f(\emptyset, \vec{s}, \vec{t}) = |\mathsf{pnf}(V,s)|(\vec{s})$. On the other hand, if $G = \{g\}$ is a singleton, we obtain

$$f(\{g\}, \vec{s}, \vec{t}) = \lambda x. \begin{cases} \llbracket M_n \rrbracket(\vec{s}, g + \vec{t}) & \text{if } x = n \in \vec{s}, \\ \llbracket M_{t_1} \rrbracket(\vec{s}, g + \vec{t}) & \text{else if } x = g + t_1, \\ \ldots & \\ \llbracket M_{t_k} \rrbracket(\vec{s}, g + \vec{t}) & \text{else if } x = g + t_k, \\ \llbracket M_0 \rrbracket(\vec{s}, g + \vec{t}, x) & \text{otherwise} \end{cases}$$

so that $f(\{g\}, \vec{s}, \vec{t}) = |V|(\vec{s}, g + \vec{t})$ when $(\vec{s}, g + \vec{t}) \in \mathbb{T}^{\neq s \oplus t}$. Thus f satisfies Proposition 29.9 for V.

Expression case: Suppose that we have constructed these reductions for values of type τ. Let M be an expression with $M\,(\mathsf{id}_s)_\tau\,M$, so by definition of the logical relation and the normal form there is some $V \in \mathsf{Val}_\tau(s \oplus t \oplus u \oplus w)$ such that $s \oplus t \vdash M \Downarrow_\tau (u \oplus w)V$ and $u = \mathsf{Leak}(V,s)$. Therefore $V \in \mathsf{Safe}_\tau^{s \oplus u}$ and $\mathsf{pnf}(M,s) = \nu u.\mathsf{pnf}(V, s \oplus u)$.

By inductive hypothesis, there is a function $f_V : 2^{\mathbb{T}} \times \mathbb{T}^{\neq s \oplus t \oplus u \oplus w} \to P(\llbracket \tau \rrbracket)$ satisfying the conditions of Proposition 29.9 for V and $\mathsf{pnf}(V, s \oplus u)$. We then define $f : 2^{\mathbb{T}} \times \mathbb{T}^{\neq s \oplus t} \to P(\llbracket \tau \rrbracket)$ by

$$f(G, \vec{s}, \vec{t}) = \mathsf{let}\ u \leftarrow \nu\ \mathsf{in}\ \mathsf{let}\ w \leftarrow \nu\ \mathsf{in}\ f_V(B, \vec{s}, \vec{t}, \vec{u}, \vec{w}).$$

It follows that

$$\begin{aligned} f(\{g\},\vec{s},\vec{t}) &= \mathsf{let}\, u \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, w \leftarrow \nu\, \mathsf{in}\, f_V(\{g\},\vec{s},\vec{t},\vec{u},\vec{w}) \\ &= \mathsf{let}\, u \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, w \leftarrow \nu\, \mathsf{in}\, [\![V]\!]_{\neq s\oplus t\oplus u\oplus w}(\vec{s},g+\vec{t},\vec{u},g+\vec{w}) \\ &= \mathsf{let}\, u \leftarrow \nu\, \mathsf{in}\, \mathsf{let}\, w \leftarrow \nu\, \mathsf{in}\, [\![V]\!]_{\neq s\oplus t\oplus u\oplus w}(\vec{s},g+\vec{t},\vec{u},\vec{w}) \\ &= [\![\nu u.\nu w.V]\!]_{\neq s\oplus t}(\vec{s},g+\vec{t}) \\ &= [\![M]\!]_{\neq s\oplus t}(\vec{s},g+\vec{t}) \end{aligned}$$

whenever $(\vec{s},g+\vec{t}) \in \mathbb{R}^{\neq s\oplus t}$, where the third equality follows by ν-invariance. Similarly, we verify that $f(\emptyset,\vec{s},\vec{t}) = [\![\mathsf{pnf}(M,s)]\!]_{\neq s}(\vec{s})$. ∎

Again, the construction described here is not specific to quasi-Borel spaces; it can be performed completely syntactically in the metalanguage once the object of names carries a ν-invariant group structure.

It follows that passing to normal forms preserves Qbs semantics, and therefore that Qbs is fully abstract at first-order types:

Theorem 29.10 Qbs *is fully abstract at first-order types.*

PROOF By Theorem 29.7 it is enough to show that Qbs validates passing to normal forms. Let τ be a first-order type and let $M \in \mathsf{Exp}_\tau(s)$. By Proposition 29.9 there is a quasi-Borel map $f : 2^{\mathbb{T}} \times \mathbb{T}^{\neq s} \to P([\![\tau]\!])$ such that

$$f(\emptyset,\vec{s}) = [\![\mathsf{pnf}(M,s)]\!]_{\neq s}(\vec{s}) \quad \text{and} \quad f(\{g\},\vec{s}) = [\![M]\!]_{\neq s}(\vec{s}).$$

Currying, we get a map $h : 2^{\mathbb{T}} \to P([\![\tau]\!])^{\mathbb{T}^{\neq s}}$ such that

$$h(\emptyset) = [\![\mathsf{pnf}(M,s)]\!]_{\neq s} \quad \text{and} \quad h(\{g\}) = [\![M]\!]_{\neq s}.$$

for all $g \in \mathbb{T}$. It follows from the privacy equation that

$$\begin{aligned} [\![\mathsf{pnf}(M,s)]\!]_{\neq s} &= \mathsf{let}\, G \leftarrow [\emptyset]\, \mathsf{in}\, h(G) = \mathsf{let}\, G \leftarrow (\mathsf{let}\, g \leftarrow \nu\, \mathsf{in}\, [\{g\}])\, \mathsf{in}\, h(G) \\ &= \mathsf{let}\, g \leftarrow \nu\, \mathsf{in}\, h(\{g\}) = \mathsf{let}\, n \leftarrow \nu\, \mathsf{in}\, [\![M]\!]_{\neq s} = [\![M]\!]_{\neq s}. \end{aligned}$$ ∎

30 Related Work and Context

30.1 Names in Computer Science and Statistics

Names are important in almost every area of practical computer science. There are two main ways to implement name generation: the first is to have one or more servers that deterministically supply fresh names as requested, and the second is to pick them randomly. Our work has emphasized the surprising effectiveness of the latter approach for programming semantics, in that it provides a model that is fully abstract up to first order, not by construction, but by general properties of the real numbers.

Names might be server names in distributed systems, nonces in cryptography, object names in object oriented programming, `gensym` in Lisp, or abstract memory locations in heap-based programming. Beyond computer science, names play a vital role in logic and set theory. In the context of probabilistic programming, we emphasize in particular two ways that names are used in statistics, and the way that name generation is already understood in terms of randomness there:

- The Dirichlet process can be used as a method for clustering data points where the number of clusters is unknown. The 'base distribution' of a Dirichlet process allocates a label to each cluster that is discovered. It is common to use an atomless distribution such as a Gaussian for this, so that the labels are in effect fresh names for the clusters. In the CHURCH probabilistic programming language, it is common to actually use Lisp's `gensym` as the base distribution for the Dirichlet process [Roy et al., 2008].

- A *graphon* is a measurable function $g\colon [0,1]^2 \to [0,1]$, which determines a countably infinite random graph in the following way: we label nodes in the graph with numbers drawn uniformly from $[0,1]$, and there is an edge between two nodes r, s with probability $g(r,s)$. Thus when building a graph node-by-node, the name of each fresh node is, in effect, a real number [Orbanz and Roy, 2015].

While many programming languages support name generation directly or through libraries, we have here focused on the ν-calculus, which is stripped down so that the relationship between name generation and functions can be investigated. There are many other calculi for names, including $\lambda\nu$, which is a call-by-name analogue of the ν-calculus [Odersky, 1994], and the π-calculus, for concurrency [Milner, 1999]. Moreover, research on the ν-calculus has led to significant developments in different directions, including memory references (e.g. [Jeffrey and Rathke, 1999; Laird, 2004; Murawski and Tzevelekos, 2016]) and cryptographic protocols (e.g. [Sumii and Pierce, 2003]). It may well be informative to pursue quasi-Borel based analyses of these applications in the future.

30.2 Compiler Optimization, Memory and Garbage Collection

One major application of semantic models is in validating observational equivalences that may be used for compiler optimizations. In probabilistic programming, optimizations are performed as part of statistical inference algorithms. For instance, discardability and exchangeability are simple but useful translations in practical inference [Murray and Schön, 2018; Nori et al., 2014], and partial evaluation and normalization are used in several systems [Gehr et al., 2020; Shan and Ramsey, 2017]. Our work in this chapter is primarily foundational, but one application is that a statistical inference algorithm could legitimately involve our normalization algorithm (Section 29) to simplify certain higher-order functions. Note that in order to show that the privacy equation and its corollaries are contextual equivalences in real-world probabilistic languages (with recursion), one would invoke the adequacy theorem for $\omega\mathsf{Qbs}$ from [Vákár et al., 2019] to transfer our denotational results.

Equations such as the privacy equation are interesting in that they allow us to analyze variables captured in *closures*. The private name a in $\nu a.\lambda x.(x = a)$ can be eliminated altogether, and never needs to be sampled or captured. In this sense, our normalization proce-

dure is a very simple form of compile-time garbage collection or escape analysis [Kotzmann and Mossenbock, 2007].

Outlook: Name generation and memory Taking the connection to memory further, [Stark, 1994, Chapter 5.9] shows that every model of name generation gives rise to a model of ground local store, as we recall now: Let $(\mathbb{C}, T)$ be a categorical model of name generation with object of names N, which we now consider as storage locations. If V denotes the object of storable values, we define the *memory* as the space of functions $\mathsf{Mem} \stackrel{\text{def}}{=} N \Rightarrow V$ from locations to values. We can now define a monad for local state as a monad transformer with global state Mem,

$$\mathsf{St}(X) \stackrel{\text{def}}{=} \mathsf{Mem} \Rightarrow T(X \times \mathsf{Mem}).$$

This comes with morphisms

$$\mathsf{ref} : V \to \mathsf{St}(N) \quad \mathsf{get} : N \to \mathsf{St}(V) \quad \mathsf{set} : N \times V \to \mathsf{St}(1) \tag{99}$$

defined as

$$\begin{aligned} \mathsf{ref}(v) &\stackrel{\text{def}}{=} \lambda m.\mathsf{let}\, a \leftarrow \mathsf{new}\, \mathsf{in}\, [(a, \lambda b.\mathsf{if}\ b = a\ \mathsf{then}\ v\ \mathsf{else}\ m(b)) \\ \mathsf{get}(a) &\stackrel{\text{def}}{=} \lambda m.[(m(a), m)] \\ \mathsf{set}(a, v) &\stackrel{\text{def}}{=} \lambda m.[((), \lambda b.\mathsf{if}\ b = a\ \mathsf{then}\ v\ \mathsf{else}\ m(b)) \end{aligned}$$

For $\mathsf{ref}(v)$ we allocate a fresh location a through new and return it, along with an updated memory at that location. As Stark says, *"consideration of the nu-calculus has isolated the problematic parts of ML references; actual storage of values is straightforward, if a little tedious in detail"*. Note that local state for $V = 1$ is mere name generation, as references contain no extra information, so $\mathsf{Mem} \cong 1$ and $\mathsf{St} \cong T$.

Stark doesn't quite make explicit what is required of a model of name generation to be considered adequate for local store. We sketch a program for future work to make this connection precise (going back to discussions with Martín Abadi, Gordon Plotkin, and Paul Levy). Staton has published a list of 14 axioms for local state [Staton, 2010] which are syntactically complete in the sense of Section 13.3. Verifying the axioms makes use of the Privacy equation, leading to the following proposition:

Proposition 30.1 *A categorical model of name generation gives rise to an adequate model of local store if it satisfies the Privacy equation, in the sense that is satisfies Staton's axioms.*

PROOF (SKETCH) It is tedious but straightforward to verify that every adequate model of name generation satisfies 13 out of the 14 axioms. The remaining axiom is (B1), which is the discardability of ref:[21]

$$\mathsf{let}\, a \leftarrow_{\mathsf{St}} \mathsf{ref}(v)\, \mathsf{in}\, [()]_{\mathsf{St}} = [()]_{\mathsf{St}} \tag{B1}$$

[21] here, we annotate the metalanguage syntax with the monad to distinguish the operations of St from T

We can derive (B1) from Privacy as follows:

$$
\begin{aligned}
&(\mathsf{let}\, a \leftarrow_{\mathsf{St}} \mathsf{ref}(v)\, \mathsf{in}\, [()]_{\mathsf{St}})(m) \\
&= \mathsf{let}\, a \leftarrow_T \mathsf{new}\, \mathsf{in}\, [((), \lambda b.\mathsf{if}\ b = a\ \mathsf{then}\ v\ \mathsf{else}\ m(b)) \\
&= \mathsf{let}\, e \leftarrow_T (\mathsf{let}\, a \leftarrow_T \mathsf{new}\, \mathsf{in}\, [\lambda x.(x = a)])\, \mathsf{in}\, [(), \lambda b.\mathsf{if}\ e(b)\ \mathsf{then}\ v\ \mathsf{else}\ m(b)] \\
(93) \quad &= \mathsf{let}\, e \leftarrow_T [\lambda x.\mathsf{false}]\, \mathsf{in}\, [(), \lambda b.\mathsf{if}\ e(b)\ \mathsf{then}\ v\ \mathsf{else}\ m(b)] \\
&= [((), \lambda b.\mathsf{if}\ \mathsf{false}\ \mathsf{then}\ v\ \mathsf{else}\ m(b)) \\
&= [((), m)) \\
&= ([()]_{\mathsf{St}})(m)
\end{aligned}
$$

■

Thus, every model of name generation that is fully abstract up to first-order gives a model of local state fully abstract at ground types. Note that because of the higher-order nature of Mem, (B1) is an example of an equation at the second-order type

$$V \Rightarrow (N \Rightarrow V) \Rightarrow T(1 \times (N \Rightarrow V))$$

We see no issue extending this approach to 'full-ground memory' (references to references etc.), for example letting $V = N$.

Memory layout randomization: Quasi-Borel spaces are fully abstract up to first-order, so Proposition 30.1 applies to them: We obtain an adequate *probabilistic model of local state* where real numbers are used for locations. In fact, every higher-order probabilistic language is capable of encoding the complexities of local state that way. This is an interesting source of random higher-order functions, like for example

$$\mathsf{ref} : V \to V^{\mathbb{R}} \Rightarrow P(\mathbb{R} \times V^{\mathbb{R}})$$

Randomizing heap layouts is frequently considered in practice for security purposes (e.g. [Abadi and Plotkin, 2012]). While the interest in those works is more quantitative, it is interesting to see aspects of this reflected in our idealized setting of real numbers.

30.3 Full Abstraction at Higher Types

To deal with this incompleteness of nominal sets, Stark [Stark, 1994, §4.4] proposed a semantic version of the logical relations that we have recalled in Section 29. This model, based on functors between double categories, is fully abstract at first order, as is ours. Subsequently an alternative logical relations model was proposed by Zhang and Nowak [2003], by working with logical relations over a functor category that more clearly distinguishes between public and private names. Qbs is different in spirit to these models, as it is a general purpose model of probability theory rather than a model purpose-built for full abstraction. A quasi-Borel space can be regarded as an $\mathbb{R}$-indexed logical relation (in the sense of Plotkin [1973]), but it also has a basic role motivated by probability theory.

Recall that none of the models we presented validate the observational equivalence at second-order from Example 24.6,

$$\nu a.\nu b.\lambda f.(fa \Leftrightarrow fb) \approx_{(\mathsf{N}\to\mathsf{B})\to\mathsf{B}} \lambda f.\mathrm{true}$$

where $\Leftrightarrow$ denotes the biconditional of booleans. To see that this equation fails in the quasi-Borel space model, notice that the order relation on the space of names $N = [0,1]$ lets us define a a Qbs morphism $\mathsf{half} : [0,1] \to P2$ given by $\mathsf{half}(x) = [x > 0.5]$. Such a function is not definable in the ν-calculus because it breaks the abstraction of names: We evaluate $[\![(\lambda f.\mathsf{true})]\!](\mathsf{half}) = [\mathsf{true}]$, while

$$[\![(\nu a.\nu b.\lambda f.(fa \Leftrightarrow fb))]\!](\mathsf{half}) = \int_{[0,1]} \int_{[0,1]} [a > 0.5 \Leftrightarrow b > 0.5] \mathrm{d}a\mathrm{d}b$$

returns true with probability 0.5. It is surprising that the abstraction of names as random numbers is *not* broken if only first-order functions are involved.

Higher-order observational equivalences such as this one are only validated by Stark's refined logical relation [Stark, 1994, Chapter 6], game-semantic models [Abramsky et al., 2004; Tzevelekos, 2008b] and bisimulation models [Benton and Koutavas, 2008]. In common with our work, normal forms play an implicit role in those models, but those models are very different from ours at higher types. In the future it may be interesting to impose further invariance properties on quasi-Borel spaces to bridge the gap.

30.4 Other Models of Higher-Order Probability

In this chapter, we have focused on quasi-Borel spaces, but it is interesting to compare our results to other models of higher-order probability have been proposed (Section 4.3). There are two essential requirements for a model to interpret the ν-calculus with name generation as randomness

(i) it must support an atomless distribution, such as the normal distribution, on some uncountable space N;

(ii) it must support equality checking on that space, as a function $N \times N \to 2$.

Some models, such as probabilistic coherence spaces [Ehrhard et al., 2014], do not seem to support atomless distributions, which makes it unclear how to use them for this purpose. Other models are based on the idea that all functions are continuous or computable, e.g. [Escardo, 2009; Huang et al., 2018] and then it is impossible to have equality checking for $N = \mathbb{R}$. This leaves plenty of models to explore from Section 4.3.

There are also recent logics for higher-order probability [Sato et al., 2019] and an operational bisimulation [Lago and Gavazzo, 2019]. We understand from the authors that operational bisimulation violates the Privacy law, for an interesting reason, and that the boolean topos model [Simpson, 2017] violates it because of booleanness (as in Proposition 26.2). It remains to be seen how abstract the other recent models are for interpreting the ν-calculus. We note that [Dahlqvist and Kozen, 2020; Ehrhard et al., 2018] are currently focused on call-by-*name* semantics and so it is not obvious how to use them with the call-by-*value* ν-calculus in our setup (see (85)).

Finally, we note the similarity of fresh name generation with urn creation from Chapter III. The Beta distribution on $[0,1]$ is atomless, but its synthetic analogue $\nu_{i,j}$ on the type I is not, purely because equality tests of urns are not expressible in the model. We could allow them in a refined, linear language where finite sets and injections Inj replace the role of Fin.

The new construction would be reminiscent of the construction of nominal sets as sheaves on Inj.

30.5 Outlook: A Categorical Theory of Information Leaking

In this section, we propose some ideas for looking at positivity and other dataflow axioms (Section 9) from the perspective of information leaking.

Fix a Markov category $\mathbb{C}$ and consider a morphism of the form $f : A \to X \otimes E$, where we are interested in X as the relevant output and view E as some additional environment. Taking the marginal $f_X : A \to X$ is a destructive operation, which gets rid of the information leaked into the environment altogether. A less destructive operation would be to *hide* the environment E, giving us a morphism $f_{(X)} : A \rightsquigarrow X$ in a different category where E has merely been removed from sight, but less information is lost.

The formalization presented here is somewhat experimental: We think that the intuitions of leaking might be more naturally expressed in a higher-categorical language with less quotienting, such as bicategories or double categories. A unification of these concepts with CD categories remains to be developed.

We include the definition because it has interesting connections to Privacy, the Cond construction (Chapter IV) and the notion of dilation in [Houghton-Larsen, 2021]. Our construction is similar to the Oles construction [Hermida and Tennent, 2012] and the affine reflection [Huot and Staton, 2018]. A bicategorical account of privacy has been given in [Stay and Vicary, 2013]. Section 30.5 is based on joint work with Paolo Perrone.

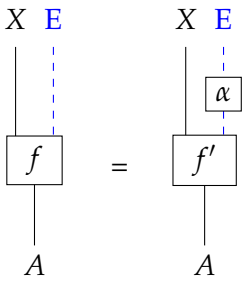
Definition 30.2 Let $\mathbb{C}$ be a symmetric monoidal category. We define the data of a category $\mathsf{Leak}(\mathbb{C})$ as follows

(i) objects are the objects of $\mathbb{C}$

(ii) morphisms $A \rightsquigarrow X$ are equivalence classes $[E, f]$ where E is an object and $f : A \to X \otimes E$ is a morphism in $\mathbb{C}$. We identify $[E, f]$ with $[E', f']$ iff there exists an isomorphism $\alpha : E \to E'$ such that

$$f' = (\mathsf{id}_X \otimes \alpha) f \tag{100}$$

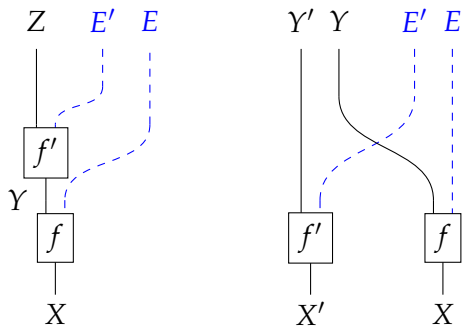
We draw (representatives of) morphisms in $\mathsf{Leak}(\mathbb{C})$ using string diagrams in $\mathbb{C}$ where wires into E are highlighted as *leaking wires*. The equivalence relation of depicted as follows



Morphisms can be composed as before while treating the environment as hidden.

Proposition 30.3 $\mathsf{Leak}(\mathbb{C})$ *can be given the structure of a symmetric monoidal category as follows:*

(i) Morphisms are composed and tensored as follows, while accumulating environments



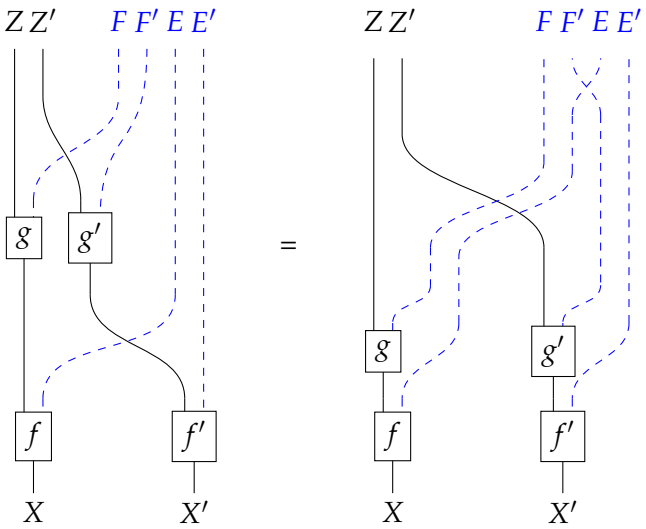
(ii) Morphisms $f : X \to Y$ in $\mathbb{C}$ give rise to morphisms in $\mathsf{Leak}(\mathbb{C})$ via

$$J(f) \overset{def}{=} [I, X \xrightarrow{f} Y \cong Y \otimes I] : X \rightsquigarrow Y$$

Identities and symmetric monoidal structure are inherited from $\mathbb{C}$

Furthermore, if $\mathbb{C}$ has the structure of a CD category, so does $\mathsf{Leak}(\mathbb{C})$ and the assignment $J : \mathbb{C} \to \mathsf{Leak}(\mathbb{C})$ gives rise to an identity-on-objects CD functor

PROOF Well-definedness of composition and tensor are easily verified; verifying coherence in $\mathsf{Leak}(\mathbb{C})$ is tedious but straightforwardly reduces to $\mathbb{C}$, because the J-construction introduces no change in the environment up to coherence isomorphisms. The only interesting part of the computation is the verification of the interchange law for $\mathsf{Leak}(\mathbb{C})$: The environments produced by $(g \otimes g') \circ (f \otimes f')$ and $(g \circ f) \otimes (g' \circ f')$ differ in the order of the middle two wires. They get identified under (100) by applying the swap isomorphism, i.e.



(101)

■

In the following, we take $\mathbb{C}$ to be a Markov category. Note that $\mathsf{Leak}(\mathbb{C})$ is not a Markov category, because the unit I is not terminal: There are plenty of effects $A \rightsquigarrow I$ which return no output but leak nontrivial information.

Because $\mathsf{Leak}(\mathbb{C})$ is a CD category, we have access to two types of string diagrams, reminiscent of Figure 7: In $\mathsf{Leak}(\mathbb{C})$-diagrams, the environments are completely hidden. Every such diagram has a representative in $\mathbb{C}$ where environments are made visible as leaking wires. Given $f : A \to X \otimes E$, its *pseudo-marginal* $f_{(X)} : A \to X$ is defined as the equivalence class $[E, f]$. Note that the environment E of a Leak-morphism can only be recovered up to isomorphism; we only care about the information content that's being leaked, while the information itself is being abstracted away.

Marginalization is a projection from $\mathsf{Leak}(\mathbb{C})$ back to $\mathbb{C}$

Proposition 30.4 *Marginalization gives rise to an identity-on-objects CD functor*

$$\Pi : \mathsf{Leak}(\mathbb{C}) \to \mathbb{C}, [E, f : A \to X \otimes E] \mapsto f_X$$

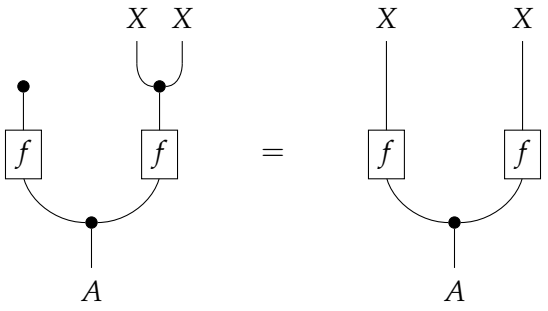
satisfying $\Pi J = \mathsf{Id}_{\mathbb{C}}$. *We have* $f_X = \Pi(f_{(X)})$.

PROOF Marginalization is clearly well-defined, and all categorical operations after marginalization are just those of $\mathbb{C}$. ∎

We can now have a look at the dataflow axioms in terms of leaking: The deterministic marginal property says that if a marginal X is deterministic, then it is independent from whatever environment. This suggests that positivity is about *strengthening* the notion of determinism from marginals to pseudo-marginals: That is, determinism remains valid even in the presence of information leaks.

To make this precise, we will propose a weaker notion of determinism (called repeatability) for CD categories which gives some leeway for unnormalized (non-discardable) morphisms to still be considered deterministic.

Definition 30.5 A morphism $f : A \to X$ in a CD category is called *repeatable* if



(102)

This notion simplifies to the usual definition of determinism whenever f is discardable, e.g. in Markov categories.

Example 30.6 The scoring kernel in SfKer

$$\mathsf{score} : \mathbb{R} \rightsquigarrow 1, \mathsf{score}(r, 1) = r$$

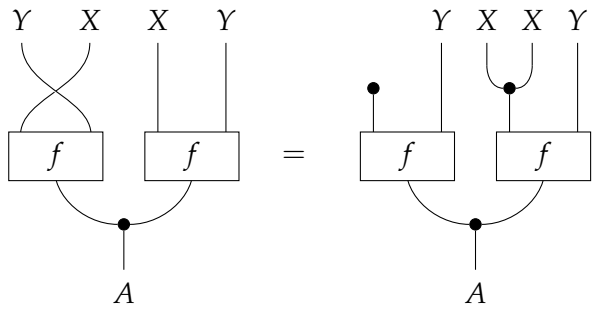
is repeatable according to (102). However, it is neither copyable nor discardable, and the effect $\mathsf{score}(a) : 1 \rightsquigarrow 1$ is copyable only if $a \in \{0, 1\}$.

In order to relate the positivity axiom to determinism under leaking, we first establish a reformulation of the deterministic marginal property:

Lemma 30.7 *For a Markov category $\mathbb{C}$, the following are equivalent*

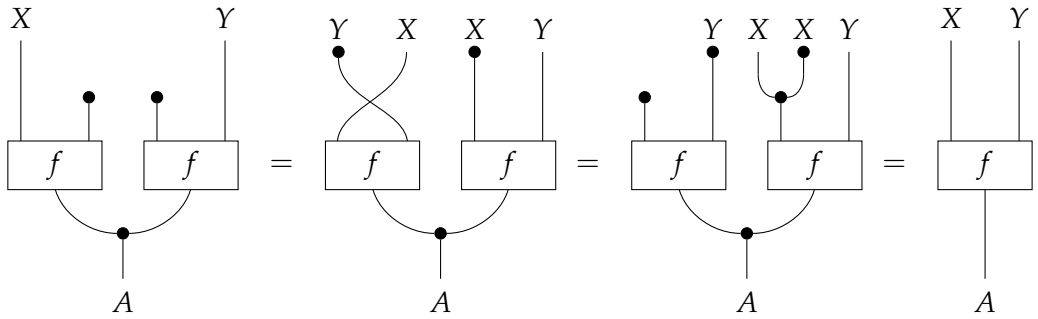
(i) $\mathbb{C}$ *is positive*

(ii) for every $f : A \to X \otimes Y$, if f_X is deterministic then

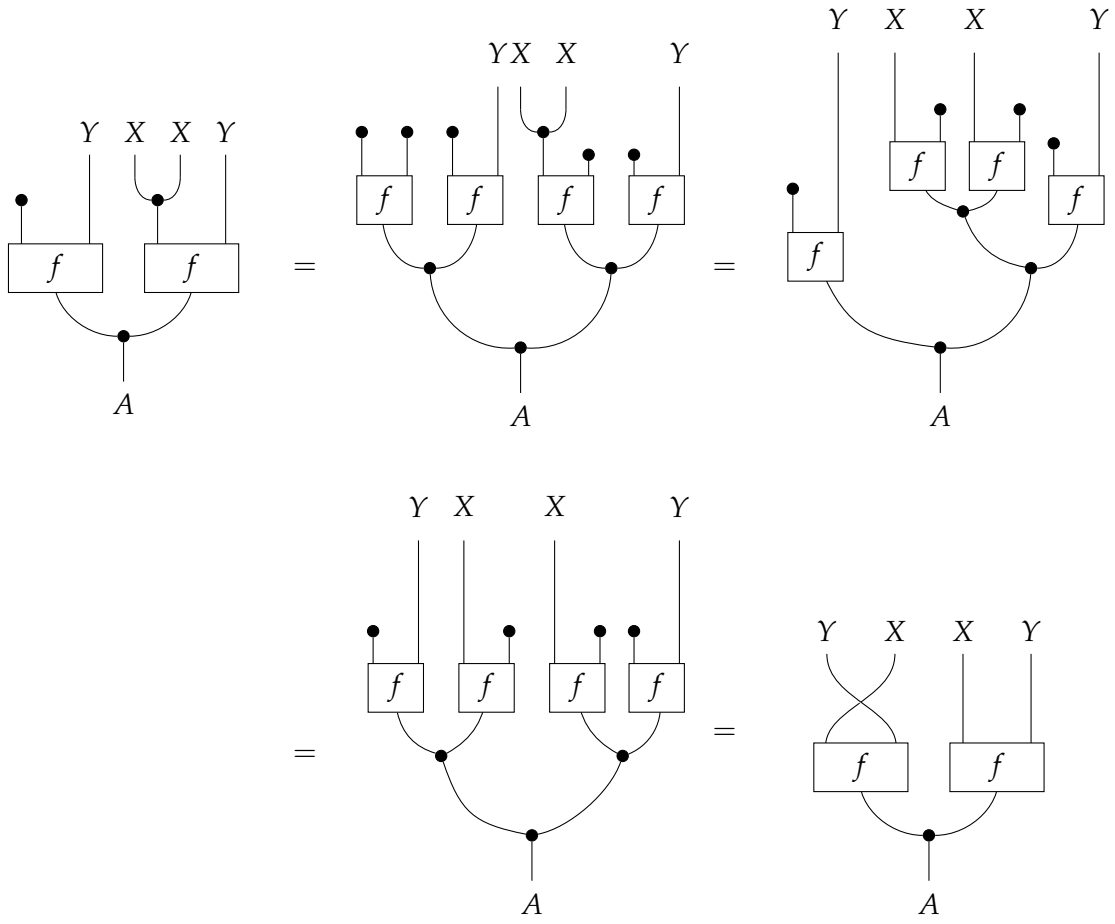


(103)

PROOF (ii) implies the deterministic marginal property by marginalizing the first and third wires from the left.



On the other hand, if f_X is deterministic and $f = \langle f_X, f_Y \rangle$ holds, we obtain



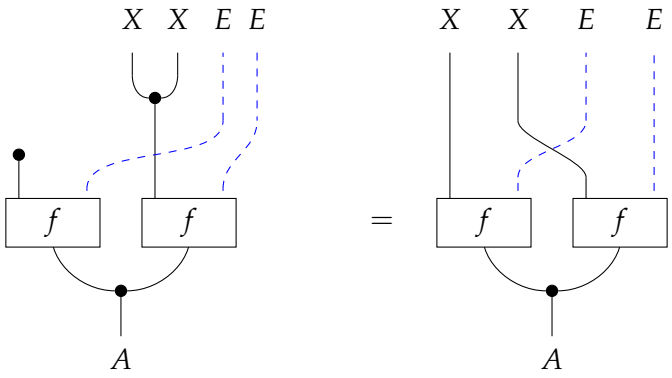
■

The condition (103) is obtained by instantiating repeatability in the CD category $\mathsf{Leak}(\mathbb{C})$: We arrive at the following result, which states that positivity lets us strength determinism from marginals to pseudomarginals.

Proposition 30.8 (Determinism strengthening) *Let $\mathbb{C}$ be a positive Markov category and $f : A \to X \otimes E$ a morphism in $\mathbb{C}$. If the marginal $f_X : A \to X$ is deterministic, then the pseudomarginal $f_{(X)} : A \rightsquigarrow X$ in $\mathsf{Leak}(\mathbb{C})$ is repeatable.*

PROOF Repeatability (Definition 30.5) instantiated for the morphism $f_{(X)}$ gives rise to the

following string diagram in $\mathbb{C}$, where E is treated as a leaking wire



Bringing the first E-wire to the left, this is precisely equation (103). ∎

The gist is that in positive Markov categories, determinism of a random variable can be considered in isolation of its environment. If positivity fails, this is no longer possible. We revisit the pathological example of a leaking name (Proposition 26.3).

Example 30.9 If we consider the joint distribution $\mu = \mathsf{let}\, x \leftarrow \mathsf{new}\, \mathsf{in}\, [(\{x\}, x)]$, then its first marginal μ_1 is deterministic, yet the pseudo-marginal $\mu_{(1)}$ is not repeatable; because the actual names get leaked into the environment, we can distinguish the two sides of (103).

We think that Proposition 30.8 *should be* an if-and-only-if characterization of positivity, because determinism in the presence of leaking wires is exactly the notion (103). We can't formally prove this from our setup because of the quotienting by the isomorphisms in (100). This should be a starting point for replacing the quotient by higher structure. We also conjecture that Leak allows a similar characterization of causality (Section 9.2) related to the equivalent notion of *parameterized equality strengthening* [Cho and Jacobs, 2019], which is about strengthening μ-almost sure equality in the presence of leaking wires.

Chapter VI
Conclusion

We conclude by summarizing the contributions of the thesis and possibilities for future work:

We have given an introduction to synthetic probability theory as a way of organizing probabilistic thinking, abstracting away and comparing the underlying mathematical formalisms (Chapter I), and opening them up to generalizations. We believe that synthetic reasoning provides powerful and elegant definitions, which enable greater conceptual insight into probability theory and statistics (Chapter II). This is perhaps best exemplified in our synthetic theory of conditioning (Chapter IV).

Mathematical theories shine in conjunction with a compelling formal language. For synthetic probability theory, this is the language of string diagrams, which is already widely applied in areas from physics to control theory. By showing that the internal language of CD categories is the prototypical ground probabilistic programming language (Section 7), we make a connection between synthetic probability theory and computer science: Probabilistic programming languages not only have a mathematical foundation in categorical models of probability, they also act as a convenient interface to the models themselves. The interplay between language and model can inspire the addition of novel features such as the exact conditioning construct $(=:=)$ in Chapter IV. This all substantiates the main slogan from the introduction that probabilistic programming languages should be thought of as the internal languages of appropriate categorical models.

Commutative effects are abundant in computer science. We consider them to be of a distinct flavor which makes them amenable to a probabilistic reading. This makes synthetic probability theory a unifying language for various phenomena in computer science, such as nondeterminism, name generation, logic programming, generativity and local state (Section 5). We have seen that even phenomena like information hiding (Section 26) and garbage collection (Section 30.2) admit a probabilistic discussion. Interesting examples which blend probabilistic concepts and features of effectful programming are double distributions in hierarchical models (Section 14) or memory as a source of nontrivial random higher-order functions (Proposition 30.1).

In the main chapters of the thesis, we have have developed three synthetic accounts of particular situations:

(i) We have studied the interaction of the Beta and Bernoulli distributions in an algebraic fashion and proved our theory complete with respect to measure-theoretic semantics (Theorem 12.10) as well as syntactically complete (Theorem 13.5).

(ii) We have developed a synthetic theory of conditioning kernels which allow measure-zero observations (Section 19.2) and resolve Borel's paradox in a type-theoretic fashion (Section 22.1). We have used this to explore the addition of an exact conditioning operator $(=:=)$ to a language for Gaussian probability. We have established convenient properties of exact conditioning in general (Section 19.3), given an algebraic axiomati-

zation of Gaussian probability with conditioning and shown a normal form result for the effects in that language (Theorem 21.8).

(iii) We have interpreted name generation as an interesting instance of synthetic probability (Section 25). We proceeded to show that name generation can be soundly interpreted using random sampling, namely that quasi-Borel spaces form a semantic model of the ν-calculus (Theorem 27.15). We used the connection with name generation to investigate the behavior of random higher-order functions, which we prove exhibit the same phenomena of Privacy and information hiding as name generation. Crucially, the a random singleton set is indistinguishable from the empty set (Theorem 28.1). This has a profound impact on the dataflow properties in higher-order probability and prevents the existence of certain conditional distributions (Section 26) by a Privacy argument. We extended the Privacy result to show that the quasi-Borel spaces model is fully abstract for the ν-calculus up to first-order types (Theorem 29.10). The proof requires a novel normalization procedure for the first-order terms, which identifies and eliminates private names.

Recurring themes of this thesis are the relation between generativity and probability, abstract types, presheaves, second-order algebraic theories, normal forms and the study of information leaks. The methods exemplified in the main chapters can be seen as general tools of presenting synthetic probability theories, such as the Cond construction (Section 19.2) or the categories in Section 14 and proposition 25.19.

The thesis contains many starting points for future work, found in the conclusions of the individual chapters and in the 'outlook' sections scattered across the thesis. We summarize some points here:

It would be interesting to elaborate on probabilistic models of local state (Section 30.2), as well as develop a categorical account of Privacy and information leaking (Section 30.5).

Section 20 suggests a connection between Frobenius algebras, exact conditioning and PROLOG-style logic programming. A logical next step to Section 21.2 would be to extend the category Gauss with synthetic improper priors which act as Frobenius unit for conditioning (Section 20.2), or prove that such an extension is impossible. By compact closure, the extension would lead to an improved characterization of Theorem 21.10, because effects and states are put in bijective correspondence. Another area of research is to extend the Cond-construction to branching programs, which are known to have subtle interactions with conditioning [Jacobs, 2021b]. There is an ongoing search to find large but well-behaved Markov categories, for example a well-behaved Markov category of measurable spaces [Fritz, 2020, Conjecture 11.10] or one that has 'Kolmogorov products' [Fritz and Rischel, 2020, Problem 6.7] or conditionals and representable supports (Section 8.3). Another direction would be to find smaller categories with a more combinatorial flavor, for example extending BetaBern to deeper hierarchical processes (Section 15).

References

Abadi, M. and Plotkin, G. D. (2012). On protection by layout randomization. *ACM Trans. Inf. Syst. Secur.*, 15(2).

Abelson, H., Sussman, G. J., and with Julie Sussman (1996). *Structure and Interpretation of Computer Programs*. MIT Press/McGraw-Hill, Cambridge, 2nd edition.

Abramsky, S., Ghica, D. R., Murawski, A. S., Ong, C.-H. L., and Stark, I. D. B. (2004). Nominal games and full abstraction for the nu-calculus. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, LICS '04, pages 150 – 159, USA. IEEE Computer Society.

Ackerman, N. L., Freer, C. E., and Roy, D. M. (2016a). Exchangeable random primitives. Workshop on Probabilistic Programming Semantics (PPS 2016).

Ackerman, N. L., Freer, C. E., and Roy, D. M. (2016b). On computability and disintegration. *Math. Struct. Comput. Sci.*, 27(8).

Adams, R. (2009). Lambda free logical frameworks. *Ann. Pure. Appl. Logic*. to appear.

Ahman, D. and Staton, S. (2013). Normalization by evaluation and algebraic effects. In *Proc. MFPS 2013*, volume 298 of *Electron. Notes Theor. Comput. Sci*, pages 51–69.

Alves Diniz, M., Salasar, L. E., and Bassi Stern, R. (2016). Positive polynomials on closed boxes. *arXiv e-print 1610.01437*.

Antoy, S. and Hanus, M. (2010). Functional logic programming. *Commun. ACM*, 53(4):74–85.

Aumann, R. J. (1961). Borel structures for function spaces. *Illinois Journal of Mathematics*, 5.

Bacci, G., Furber, R., Kozen, D., Mardare, R., Panangaden, P., and Scott, D. (2018). Boolean-valued semantics for stochastic lambda-calculus. In *Proc. LICS 2018*.

Baez, J. and Hoffnung, A. (2008). Convenient categories of smooth spaces. *Transactions of the American Mathematical Society*, 363.

Baez, J. C., Coya, B., and Rebro, F. (2018). Props in network theory.

Baez, J. C. and Erbele, J. (2015). Categories in control. *Theory Appl. Categ.*, 30:836–881.

Bauer, A. and Pretnar, M. (2015). Programming with algebraic effects and handlers. *Journal of Logical and Algebraic Methods in Programming*, 84(1):108–123.

Bell, J. L. (2012). Types, sets, and categories. *Handbook of the History of Logic*, 6.

Benton, N. and Koutavas, V. (2008). A mechanized bisimulation for the nu-calculus. Technical Report MSR-TR-2008-129, Microsoft Research.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2018). Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*.

Bizjak, A. and Birkedal, L. (2015). Step-indexed logical relations for probability. In *Proc. FOSSACS 2015*, pages 279–294.

Bonchi, F., Piedeleu, R., Sobocinski, P., and Zanasi, F. (2019). Graphical affine algebra. In *Proc. LICS 2019*.

Bonchi, F., Sobocinski, P., and Zanasi, F. (2017). The calculus of signal flow diagrams I: linear relations on streams. *Inform. Comput.*, 252.

Borgström, J., Dal Lago, U., Gordon, A. D., and Szymczak, M. (2016). A lambda-calculus foundation for universal probabilistic programming. *SIGPLAN Not.*, 51(9):33–46.

Breugel, F. (2005). The metric monad for probabilistic nondeterminism. *unpublished (arXiv)*.

Carboni, A., Lack, S., and Walters, R. (1993). Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84(2):145–158.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, 76(1).

Cho, K. and Jacobs, B. (2019). Disintegration and Bayesian inversion via string diagrams. *Mathematical Structures in Computer Science*, 29:938 – 971.

Clarke, B., Elkins, D., Gibbons, J., Loregian, F., Milewski, B., Pillmore, E., and Roman, M. (2020). Profunctor optics, a categorical update. arxiv:2001.07488.

Coecke, B. and Spekkens, R. W. (2012). Picturing classical and quantum Bayesian inference. *Synthese*, 186(3):651–696.

Cusumano-Towner, M. F., Saad, F. A., Lew, A. K., and Mansinghka, V. K. (2019). Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 221–236. ACM.

Dahlqvist, F. and Kozen, D. (2020). Semantics of higher-order probabilistic programs with conditioning. In *Proc. POPL 2020*.

Dal Lago, U. and Hoshino, N. (2019). The geometry of Bayesian programming. In *Proc. LICS 2019*.

Dash, S. and Staton, S. (2020). A monad for probabilistic point processes. In *ACT*.

Dash, S. and Staton, S. (2021). Monads for measurable queries in probabilistic databases. In *To Appear in Proceedings of Mathematical Foundations of Programming Semantics (MFPS)*.

De Raedt, L. and Kimming, A. (2015). Probabilistic (logic) programming concepts. *Mach. Learn.*, 100.

Doberkat, E. (2004). Characterizing the Eilenberg-Moore algebras for a monad of stochastic relations. *Information and Computation*.

Ehrhard, T., Pagani, M., and Tasson, C. (2018). Measurable cones and stable, measurable functions. In *Proc. POPL 2018*.

Ehrhard, T., Tasson, C., and Pagani, M. (2014). Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *Proc. POPL 2014*, pages 309–320.

Escardo, M. (2009). Semi-decidability of May, Must and probabilistic testing in a higher-type setting. In *Proc. MFPS 2009*.

Farouki, R. T. (2012). The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 29(6):379–419.

Feynman, R. P. (1987). Negative probability. In Hiley, B. J. and Peat, D., editors, *Quantum Implications: Essays in Honour of David Bohm*, pages 235–248. Methuen.

Fiore, M. and Hur, C.-K. (2010). Second-order equational logic (extended abstract). In Dawar, A. and Veith, H., editors, *Computer Science Logic*, pages 320–335, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fiore, M. and Mahmoud, O. (2010). Second-order algebraic theories. In *Mathematical Foundations of Computer Science 2010*, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fong, B. (2012). *Causal Theories: A Categorical Perspective on Bayesian Networks (MSc thesis)*. PhD thesis, University of Oxford.

Fritz, T. (2020). A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Adv. Math.*, 370.

Fritz, T., Gonda, T., and Perrone, P. (2021). De Finetti's theorem in categorical probability.

Fritz, T., Gonda, T., Perrone, P., and Rischel, E. F. (2020). Representable Markov categories and comparison of statistical experiments in categorical probability.

Fritz, T. and Perrone, P. (2017). A probability monad as the colimit of spaces of finite samples. *arXiv: Probability*.

Fritz, T., Perrone, P., and Rezagholi, S. (2019). Probability, valuations, hyperspace: Three monads on top and the support as a morphism. *ArXiv*, abs/1910.03752.

Fritz, T. and Rischel, E. F. (2020). Infinite products and zero-one laws in categorical probability. *Compositionality*, 2.

Führmann, C. (2002). Varieties of effects. In *Proc. FOSSACS 2002*, pages 144–159.

Furber, R. and Jacobs, B. (2015). From Kleisli categories to commutative C*-algebras: Probabilistic Gelfand duality. *Log. Methods Comput. Sci.*, 11(2).

Gehr, T., Misailovic, S., and Vechev, M. (2016). PSI: Exact symbolic inference for probabilistic programs. In *Proc. CAV 2016*.

Gehr, T., Steffen, S., and Vechev, M. T. (2020). λPSI: exact inference for higher-order probabilistic programs. In *Proc. PLDI 2020*.

Goldblatt, R. (2006). *Topoi: The Categorical Analysis of Logic*. Dover Publications.

Goodman, N. D., Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. B. (2008). Church: A language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, UAI'08, pages 220 – 229, Arlington, Virginia, USA. AUAI Press.

Goodman, N. D. and Stuhlmüller, A. (2014). The Design and Implementation of Probabilistic Programming Languages. http://dippl.org. Accessed: 2021-8-3.

Goodman, N. D., Tenenbaum, J. B., and Contributors, T. P. (2016). Probabilistic Models of Cognition. http://probmods.org. Accessed: 2021-3-26.

Gromov, M. (2014). Six lectures on probabiliy, symmetry, linearity.

Hanus, M., Kuchen, H., Aachen, R., and Ii, I. (2000). Curry: A truly functional logic language.

Hermida, C. and Tennent, R. D. (2012). Monoidal indeterminates and categories of possible worlds. *Theoret. Comput. Sci.*, 430.

Heunen, C. and Kaarsgaard, R. (2021). Bennett and Stinespring, together at last. In *Proceedings of the 18th International Conference on Quantum Physics and Logic (QPL 2021)*, number 343 in Electronic Proceedings in Theoretical Computer Science, pages 102–118. EPTCS.

Heunen, C., Kammar, O., Staton, S., Moss, S., Vákár, M., Ścibior, A., and Yang, H. (2018). The semantic structure of quasi-Borel spaces. https://pps2018.luddy.indiana.edu/files/2018/01/pps18-qbs-semantic-structure.pdf.

Heunen, C., Kammar, O., Staton, S., and Yang, H. (2017). A convenient category for higher-order probability theory. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '17. IEEE Press.

Heunen, C. and Vicary, J. (2019). *Categories for quantum theory: an introduction*. Oxford University Press, United Kingdom.

Houghton-Larsen, N. G. (2021). *A Mathematical Framework for Causally Structured Dilations and its Relation to Quantum Self-Testing*. PhD thesis, University of Copenhagen.

Huang, D., Morrisett, G., and Spitters, B. (2018). An application of computable distributions to the semantics of probabilistic programs. arxiv:1806.07966.

Huot, M. and Staton, S. (2018). Universal properties in quantum theory. In *Proc. QPL 2018*.

Huot, M., Staton, S., and Vákár, M. (2020). Correctness of automatic differentiation via diffeologies and categorical gluing. In *Foundations of Software Science and Computation Structures*, pages 319–338, Cham. Springer International Publishing.

Hyland, M. and Power, J. (2007). The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electronic Notes in Theoretical Computer Science*, 172:437–458.

Jacobs, B. (2016). Affine monads and side-effect-freeness. In *Coalgebraic Methods in Computer Science*, pages 53–72, Cham. Springer International Publishing.

Jacobs, B. (2020). A channel-based perspective on conjugate priors. *Mathematical Structures in Computer Science*, 30(1):44–61.

Jacobs, B. (2021a). From multisets over distributions to distributions over multisets. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 21)*, pages 1–13.

Jacobs, B. and Staton, S. (2020). De Finetti's construction as a categorical limit. In *To appear In 15th International Workshop on Coalgebraic Methods in Computer Science*. Preprint arxiv:2003.01964.

Jacobs, J. (2021b). Paradoxes of probabilistic programming. In *Proc. POPL 2021*.

Janson, S., Konstantopoulos, T., and Yuan, L. (2016). On a representation theorem for finitely exchangeable random vectors. *Journal of Mathematical Analysis and Applications*, 442(2):703–714.

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. CUP.

Jeffrey, A. (1998). Premonoidal categories and a graphical view of programs.

Jeffrey, A. and Rathke, J. (1999). Towards a theory of bisimulation for local names. In *Proc. LICS 1999*.

Johann, P., Simpson, A., and Voigtländer, J. (2010). A generic operational metatheory for algebraic effects. In *Proc. LICS 2010*, pages 209–218.

Joyal, A. and Street, R. (1991). The geometry of tensor calculus, i. *Advances in Mathematics*, 88:55–112.

Joyal, A. and Street, R. (1993). Braided tensor categories. *Advances in Mathematics*, 102(1):20–78.

Kallenberg, O. (1997). *Foundations of Modern Probability*. Springer, New York.

Kammar, O. and Plotkin, G. D. (2012). Algebraic foundations for effect-dependent optimisations. In *Proc. POPL 2012*, pages 349–360.

Kechris, A. (1987). *Classical Descriptive Set Theory*. Springer.

Kinoshita, Y. and Power, J. (1996). A fibrational semantics for logic programs. In *Proc. ELP 1996*.

Kiselyov, O. and Shan, C.-C. (2010). Probabilistic programming using first-class stores and first-class continuations. In *Proc. 2010 ACM SIGPLAN Workshop on ML*.

Kock, A. (1972). Strong functors and monoidal monads. *Archiv der Mathematik*, 23:113–120.

Kock, A. (2011). Commutative monads as a theory of distributions. *Theory and Applications of Categories*, 26.

Kotzmann, T. and Mossenbock, H. (2007). Run-time support for optimizations based on escape analysis. In *International Symposium on Code Generation and Optimization (CGO'07)*, pages 49–60.

Kozen, D. (1981). Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350.

Lago, U. D. and Gavazzo, F. (2019). On bisimilarity in lambda calculi with continuous probabilistic choice. *Electronic Notes in Theoretical Computer Science*, 347:121 – 141. Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics.

Laird, J. (2004). A game semantics of local names and good variables. In *Proc. FOSSACS 2004*, pages 289–303.

Lauritzen, S. and Jensen, F. (1999). Stable local computation with conditional Gaussian distributions. *Statistics and Computing*, 11.

Lawvere, F. W. (1963). Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences*, 50(5):869–872.

Leijen, D. (2014). Koka: Programming with row polymorphic effect types. *Electronic Proceedings in Theoretical Computer Science*, 153.

Levy, P. B., Power, J., and Thielecke, H. (2003). Modelling environments in call-by-value programming languages. *Information and Computation*, 185(2):182–210.

Lew, A. K., Cusumano-Towner, M. F., Sherman, B., Carbin, M., and Mansinghka, V. K. (2019). Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proc. ACM Program. Lang.*, 4(POPL).

Linton, F. E. J. (1966). Autonomous equational categories. *J. Math. Mech.*, 15:637–642.

Lunn, D., Thomas, A., Best, N., and Spiegelhalter, D. (2000). Winbugs — a Bayesian modelling framework: concepts, structure, and extensibility. *Statistics and Computing*, 10.

MacLane, S. (1971). *Categories for the Working Mathematician*. Springer-Verlag. Graduate Texts in Mathematics, Vol. 5.

Mansinghka, V., Selsam, D., and Perov, Y. (2014). Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv e-print 1404.0099*.

Milner, R. (1999). *Communicating and mobile systems - the Pi-calculus*. CUP.

Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, I. *Inform. Comput.*, 100(1).

Minka, T., Winn, J., Guiver, J., Zaykov, Y., Fabian, D., and Bronskill, J. (2018). Infer.NET 0.3. Microsoft Research Cambridge.

Møgelberg, R. and Staton, S. (2014). Linear usage of state. *Logical Methods in Computer Science [electronic only]*, 10.

Moggi, E. (1989). Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, page 14–23. IEEE Press.

Moggi, E. (1991). Notions of computation and monads. *Information and Computation*, 93(1):55 – 92. Selections from 1989 IEEE Symposium on Logic in Computer Science.

Murawski, A. S. and Tzevelekos, N. (2016). Nominal game semantics. *Found. Trends Program. Lang.*

Murray, L., Lundén, D., Kudlicka, J., Broman, D., and Schön, T. (2018). Delayed sampling and automatic Rao-Blackwellization of probabilistic programs. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, pages 1037–1046.

Murray, L. M. and Schön, T. B. (2018). Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 46:29 – 43.

Narayanan, P., Carette, J., Romano, W., Shan, C., and Zinkov, R. (2016). Probabilistic inference by program transformation in Hakaru (system description). In *Proc. FLOPS 2016*, pages 62–79.

Narayanan, P. and Shan, C. (2019). Applications of a disintegration transformation. In *Workshop on program transformations for machine learning*.

Neumann, W. D. (1970). On the quasivariety of convex subsets of affine space. *Arch. Math.*, 21:11–16.

Nori, A., Hur, C.-K., Rajamani, S., and Samuel, S. (2014). R2: An efficient MCMC sampler for probabilistic programs. In *Proc. AAAI 2014*.

Odersky, M. (1994). A functional theory of local names. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '94, pages 48 – 59, New York, NY, USA. Association for Computing Machinery.

Orbanz, P. and Roy, D. M. (2015). Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE Trans. Pattern Anal. Mach. Intell.*, pages 437–461.

Panangaden, P. (2016). Analysis of probabilistic systems. https://simons.berkeley.edu/talks/analysis-of-probabilistic-systems. Talk series at *Logical Structures in Computation Boot Camp*, Simons Institute.

Paquet, H. and Winskel, G. (2018). Continuous probability distributions in concurrent games. In *Proc. MFPS 2018*, pages 321–344.

Parzygnat, A. J. (2020). Inverses, disintegrations, and Bayesian inversion in quantum Markov categories. *arXiv: Quantum Physics*.

Pitts, A. M. (2013a). *Names and symmetry in Computer science*. CUP.

Pitts, A. M. (2013b). *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Pitts, A. M. and Stark, I. (1993). Observable properties of higher order functions that dynamically create local names, or: What's *new*? In *Mathematical Foundations of Computer Science: Proceedings of the 18th International Symposium MFCS '93*, number 711 in Lecture Notes in Computer Science, pages 122–141. Springer-Verlag.

Plotkin, G. and Power, J. (2003). Algebraic operations and generic effects. *Applied Categorical Structures*, 11:69–94.

Plotkin, G. D. (1973). Lambda-definability and logical relations. Technical Report SAI-RM-4, School of A.I., Univ.of Edinburgh.

Plummer, M. (2003). Jags: A program for analysis of Bayesian graphical models using gibbs sampling.

Power, J. and Robinson, E. (1997). Premonoidal categories and notions of computation. *Math. Struct. Comput. Sci.*, 7:453–468.

Powers, V. and Reznick, B. (2000). Polynomials that are positive on an interval. *Trans. Amer. Math. Soc.*, 352(10):4677–4692.

Pretnar, M. (2010). *The Logic and Handling of Algebraic Effects*. PhD thesis, Univ. Edinburgh.

Proschan, M. A. and Presnell, B. (1998). Expect the unexpected from conditional expectation. *The American Statistician*, 52(3).

Rainforth, T. (2017). *Automatic Inference, Learning, and Design using Probabilistic Programming*. PhD thesis, University of Oxford.

Romanowska, A. B. and Smith, J. D. H. (2002). *Modes*. World Scientific.

Roy, D., Mansinghka, V., Goodman, N., and Tenenbaum, J. (2008). A stochastic programming perspective on nonparametric bayes. In *Proc. ICML Workshop on Nonparametric Bayes*.

Russell, S. J. and Milch, B. (2006). Probabilistic models with unknown objects.

Sabok, M., Staton, S., Stein, D., and Wolman, M. (2021). Probabilistic programming semantics for name generation. In *Proceedings of the ACM on Programming Languages vol 5 (POPL 21)*.

Sato, T., Aguirre, A., Barthe, G., Gaboardi, M., Garg, D., and Hsu, J. (2019). Formal verification of higher-order probabilistic programs: Reasoning about approximation, convergence, Bayesian inference, and optimization. *Proc. ACM Program. Lang.*, 3(POPL).

Schervish, M. J. (1995). *Theory of statistics*. Springer.

Ścibior, A., Kammar, O., Vákár, M., Staton, S., Yang, H., Cai, Y., Ostermann, K., Moss, S., Heunen, C., and Ghahramani, Z. (2017). Denotational validation of higher-order Bayesian inference. *Proceedings of the ACM on Programming Languages*, 2.

Selinger, P. (2011). *A Survey of Graphical Languages for Monoidal Categories*, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg.

Shan, C.-c. and Ramsey, N. (2017). Exact Bayesian inference by symbolic disintegration. In *Proc. POPL 2017*.

Shinwell, M. R. and Pitts, A. (2005). Fresh objective caml user manual. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-621.pdf.

Simpson, A. (2017). Probability sheaves and the Giry monad. In *Proc. CALCO 2017*.

Somogyi, Z., Henderson, F., and Conway, T. (1996). The execution algorithm of mercury, an efficient purely declarative logic programming language. *The Journal of Logic Programming*, 29(1):17–64. High-Performance Implementations of Logic Programming Systems.

Srivastava, S. M. (1998). *A Course on Borel Sets*. Springer, New York.

St Clere Smithe, T. (2020). Bayesian updates compose optically.

Stark, I. (1994). *Names and Higher-Order Functions*. PhD thesis, University of Cambridge. Also available as Technical Report 363, University of Cambridge Computer Laboratory.

Stark, I. (1996). Categorical models for local names. *LISP and Symbolic Computation*, 9(1):77–107.

Staton, S. (2010). Completeness for algebraic theories of local state. In Ong, L., editor, *Foundations of Software Science and Computational Structures*, pages 48–63, Berlin, Heidelberg. Springer Berlin Heidelberg.

Staton, S. (2013a). An algebraic presentation of predicate logic. In Pfenning, F., editor, *Foundations of Software Science and Computation Structures*, pages 401–417, Berlin, Heidelberg. Springer Berlin Heidelberg.

Staton, S. (2013b). Instances of computational effects. In *Proc. LICS 2013*.

Staton, S. (2013c). Instances of computational effects: An algebraic perspective. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '13, page 519, USA. IEEE Computer Society.

Staton, S. (2014). Freyd categories are enriched lawvere theories. *Electronic Notes in Theoretical Computer Science*, 303:197–206.

Staton, S. (2015). Algebraic effects, linearity, and quantum programming languages. *SIGPLAN Not.*, 50(1):395–406.

Staton, S. (2017). Commutative semantics for probabilistic programming. In Yang, H., editor, *Programming Languages and Systems*, pages 855–879, Berlin, Heidelberg. Springer Berlin Heidelberg.

Staton, S., Shulman, M., and Baez, J. (2017a). Describing props using generators and relations. https://golem.ph.utexas.edu/category/2014/07/describing_props_using_generat.html#c052012.

Staton, S., Stein, D., Yang, H., Ackerman, N., Freer, C., and Roy, D. (2018). The beta-bernoulli process and algebraic effects. *Proceedings of 45th International Colloquium on Automata, Languages and Programming (ICALP '18)*.

Staton, S., Yang, H., Ackerman, N., Freer, C., and Roy, D. M. (2017b). Exchangeable random processes and data abstraction. Workshop on Probabilistic Programming Semantics (PPS 2017).

Staton, S., Yang, H., Wood, F., Heunen, C., and Kammar, O. (2016). Semantics for probabilistic programming: Higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 525 – 534, New York, NY, USA. Association for Computing Machinery.

Stay, M. and Vicary, J. (2013). Bicategorical semantics for nondeterministic computation. *Electronic Notes in Theoretical Computer Science*, 298.

Stein, D. (2021). GaussianInfer. https://github.com/damast93/GaussianInfer.

Stein, D. and Staton, S. (2021). Compositional semantics for probabilistic programs with exact conditioning (long version). In *Proceedings of Thirty-Sixth Annual ACM/IEEE Conference on Logic in Computer Science (LICS 2021)*.

Stone, M. H. (1949). Postulates for the barycentric calculus. *Ann. Mat. Pura Appl. (4)*, 29:25–30.

Sumii, E. and Pierce, B. C. (2003). Logical relations for encryption. *J. Comput. Secur.*, 11(4):521–554.

Tao, T. (2010). 254a, notes 0: A review of probability theory. https://terrytao.wordpress.com/2010/01/01/254a-notes-0-a-review-of-probability-theory/. Accessed: 2021-8-03.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2006). Hierarchical Dirichlet processes. *J. Amer. Statist. Assoc.*, 101(476):1566–1581.

Tijms, H. (2007). Negative probabilities at work in the m/d/1 queue. *Probability in the Engineering and Informational Sciences*, 21:67 – 76.

Tolpin, D., van de Meent, J.-W., Yang, H., and Wood, F. (2016). Design and implementation of probabilistic programming language anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages*, IFL 2016, New York, NY, USA. Association for Computing Machinery.

Tsirelson, B. (2012). Measurability and continuity (course).

Tzevelekos, N. (2008a). *Nominal Game Semantics*. PhD thesis, University of Oxford.

Tzevelekos, N. (2008b). *Nominal game semantics*. PhD thesis, Oxford University Computing Laboratory.

Vákár, M., Kammar, O., and Staton, S. (2019). A domain theory for statistical probabilistic programming. *Proceedings of the ACM on Programming Languages vol 3 (POPL 19)*.

van de Meent, J.-W., Paige, B., Yang, H., and Wood, F. (2018). An introduction to probabilistic programming.

Vandenbroucke, A. and Schrijvers, T. (2020). P$\lambda\omega$NK: functional probabilistic NetKAT. In *Proc. POPL 2020*.

von Neumann, J. (1951). Various techniques used in connection with random digits. *Nat. Bur. Stand. Appl. Math. Series*, 12:36–38.

Walia, R., Narayanan, P., Carette, J., Tobin-Hochstadt, S., and Shan, C.-c. (2019). From high-level inference algorithms to efficient code. *Proc. ACM Program. Lang.*, 3(ICFP).

Wu, J. (2013). Reduced traces and JITing in Church. Master's thesis, Mass. Inst. of Tech.

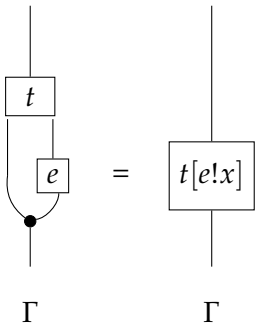
Zhang, Y. and Nowak, D. (2003). Logical relations for dynamic name creation. In *Proc. CSL 2003*, pages 575–588.

Chapter VII
Appendix

31 Appendix to Chapter II

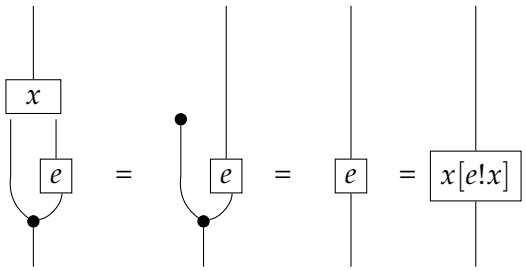
We verify the remaining axioms of the CD calculus.

PROOF (OF PROPOSITION 7.5) (let.ξ) follows from the compositionality of the semantics. ($*.\beta$), ($*.\eta$) and (unit.η) are are immediate. We proceed to prove that (let.lin) is valid, that is if t uses x exactly once, then

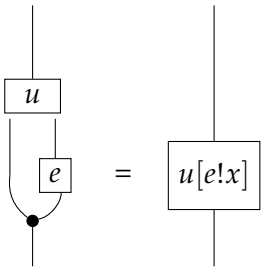


We argue by induction over the term structure of t.

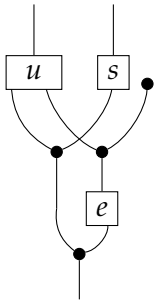
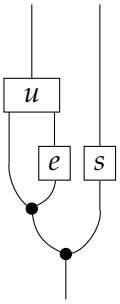
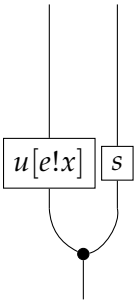
Variable If $t = x$, then



Pairing Let $t = (u, s)$ where wlog x occurs freely exactly once in u and zero times in s. By inductive hypothesis, we have $(\mathsf{let}\, x = e\, \mathsf{in}\, u) = u[e!x]$, i.e.

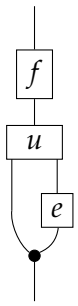


from which we derive by the comonoid laws and weakening of s

$\mathsf{let}\, x \;=\; e \,\mathsf{in}\, (u, s) \quad =$  $=$  $=$  $= \quad (u[e!x], s)$

The case for (s, u) is symmetric.

Function application If $t = f\, u$ we obtain immediately from the inductive hypothesis

$\mathsf{let}\, x \;=\; e \,\mathsf{in}\, f\, u \quad =$  $= \quad u[e!x] \quad = \quad (f\, u)[e!x]$

The proof for the projection case $t = \pi_i\, u$ is analogous.

Let-binding I Let $t = (\mathsf{let}\, y \;=\; u \,\mathsf{in}\, s)$ with u, s as before then

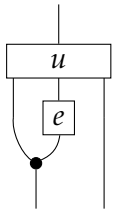
$$(\mathsf{let}\, x \;=\; e \,\mathsf{in}\, \mathsf{let}\, y \;=\; u \,\mathsf{in}\, s) \equiv (\mathsf{let}\, y \;=\; (\mathsf{let}\, x \;=\; e \,\mathsf{in}\, u) \,\mathsf{in}\, s) \equiv (\mathsf{let}\, y \;=\; u[e!x] \,\mathsf{in}\, s)$$

is a special case of (assoc) which was proved in (36).

Let-binding II Let $t = (\mathsf{let}\, y \;=\; s \,\mathsf{in}\, u)$, then

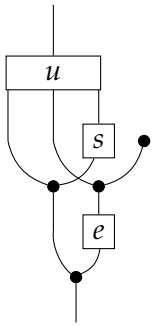
$$(\mathsf{let}\, x \;=\; e \,\mathsf{in}\, \mathsf{let}\, y \;=\; s \,\mathsf{in}\, u) \equiv (\mathsf{let}\, y \;=\; s \,\mathsf{in}\, \mathsf{let}\, x \;=\; e \,\mathsf{in}\, u) \equiv (\mathsf{let}\, y \;=\; s \,\mathsf{in}\, u[e!x])$$

is a special case of (comm). The inductive hypothesis on u involves both weakening and exchange and reads

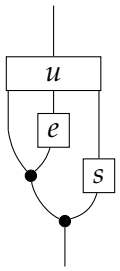
$u[e!x] \quad =$ 

from which we derive

$\mathsf{let}\, x \;=\; e \,\mathsf{in}\, \mathsf{let}\, y \;=\; s \,\mathsf{in}\, u \quad =$



$=$



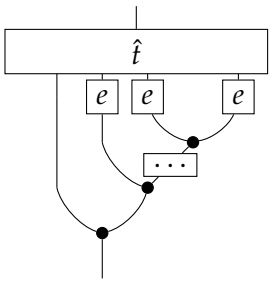
$= \quad \mathsf{let}\, y \;=\; s \,\mathsf{in}\, u[e!x]$

This finishes the validation for linear substitution.

For (let.val), we carry out the semantic analogue of Proposition 7.3 and consider sequences of let-bindings

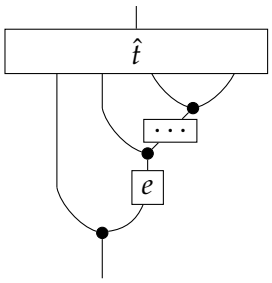
$$\mathsf{let}\, x_1 \;=\; e \,\mathsf{in}\, \cdots \mathsf{let}\, x_n \;=\; e \,\mathsf{in}\, \hat{t}$$

whose denotation is



Γ

This can be replaced by $\mathsf{let}\, x \;=\; e \,\mathsf{in}\, \mathsf{let}\, x_1 \;=\; x \,\mathsf{in}\, \cdots \mathsf{let}\, x_n \;=\; x \,\mathsf{in}\, \hat{t}$, i.e.



Γ

whenever the denotation of e is deterministic in the CD category sense. It remains to note that the denotations of all values of the CD calculus are always deterministic. ∎

32 Appendix to Chapter III

32.1 Example derivations

In this appendix, we give explicit derivations of some equations mentioned in Chapter III. The first equation, as found in Section 11.3, is

$$x +_{1:1} y = ((x +_{1:1} y) ?_p x) ?_p (y ?_p (x +_{1:1} y))$$

which is called von Neumann's trick. Here is the derivation of this equation:

$$\begin{aligned}
((x +_{1:1} y) ?_p x) ?_p (y ?_p (x +_{1:1} y)) &= ((x +_{1:1} y) ?_p (x +_{1:1} x)) ?_p ((y +_{1:1} y) ?_p (x +_{1:1} y)) \\
&= ((x ?_p x) +_{1:1} (y ?_p x)) ?_p ((y ?_p x) +_{1:1} (y ?_p y)) \\
&= ((y ?_p x) +_{1:1} (x ?_p x)) ?_p ((y ?_p x) +_{1:1} (y ?_p y)) \\
&= ((y ?_p x) ?_p (y ?_p x)) +_{1:1} ((x ?_p x) ?_p (y ?_p y)) \\
&= (y ?_p x) +_{1:1} (x ?_p y) \\
&= (y +_{1:1} x) ?_p (x +_{1:1} y) \\
&= (x +_{1:1} y) ?_p (x +_{1:1} y) \\
&= (x +_{1:1} y).
\end{aligned}$$

Using normal forms, we can in fact easily see that $x +_{1:1} y$ is the only normalized term $(- \mid x, y \vdash t)$ that satisfies $t = (t ?_p x) ?_p (y ?_p t)$. Making $(t ?_p x) ?_p (y ?_p t)$ permutation-invariant means computing the average

$$\begin{aligned}
(t ?_p x) ?_p (y ?_p t) &= ((t ?_p x) ?_p (y ?_p t)) +_{1:1} ((t ?_p y) ?_p (x ?_p t)) \\
&= ((t +_{1:1} t) ?_p (x +_{1:1} y)) ?_p ((x +_{1:1} y) ?_p (t +_{1:1} t)) \\
&= (t ?_p (x +_{1:1} y)) ?_p ((x +_{1:1} y) ?_p t) \quad = C_2(t, x +_{1:1} y, t)
\end{aligned}$$

whereas by discardability, the permutation-invariant form of t is simply $C_2(t, t, t)$. As these are normal forms, we can read off that $t = x +_{1:1} y$ is the only fixed point.

The following derivations show the normalization procedure being applied to the programs from Example 13.6:

$$\begin{aligned}
&\nu_{1,1} p.((y ?_p z) ?_p z) \\
&= (\nu_{2,1} p.(y ?_p z)) +_{1:1} (\nu_{1,2} p.z) \\
&= (y +_{2:1} z) +_{1:1} z \\
&= (y +_{2:1} z) +_{3:3} (y +_{0:3} z) \\
&= (y +_{2:0} y) +_{2:4} (z +_{1:3} z) \\
&= y +_{2:4} z \\
&= y +_{1:2} z.
\end{aligned}$$

$$\begin{aligned}
&\nu_{1,1} p.\nu_{1,1} q.((y ?_q z) ?_p z) \\
&= \nu_{1,1} p.((\nu_{1,1} q.(y ?_q z)) ?_p (\nu_{1,1} q.z)) \\
&= \nu_{1,1} p.((y +_{1:1} z) ?_p z) \\
&= (y +_{1:1} z) +_{1:1} z \\
&= (y +_{1:1} z) +_{2:2} (y +_{0:2} z) \\
&= (y +_{1:0} y) +_{1:3} (z +_{1:2} z) \\
&= y +_{1:3} z.
\end{aligned}$$

32.2 Allowing zero hyperparameters

In term formation, we have excluded zero hyperparameters $\nu_{i,0}$ and $\nu_{0,i}$. This is because $\beta_{i,0}$ and $\beta_{0,i}$ are not absolutely continuous distributions on I, but Dirac peaks on 1 and 0 respectively. This makes the geometric reasoning about the dimension of chains more complicated. For example, the equation

$$\nu_{1,0}p.x(p,p) = \nu_{1,0}p.\nu_{1,0}q.x(p,q)$$

holds in the model because Dirac distributions are deterministic. Yet our axioms cannot derive this equality, as it treats both sides as degenerate chains of distinct types! We suggest introducing new constant symbols $\mathbf{0}$ and $\mathbf{1}$ alongside axioms

$$\vdash \nu_{1,0}p.x(p) = x(\mathbf{1}) \quad \vdash \nu_{0,1}p.x(p) = x(\mathbf{0})$$

to solve the problem. This restores the well-defined notion of dimension of chains and allows us to derive

$$\vdash \nu_{1,0}p.x(p,p) = x(\mathbf{1},\mathbf{1}) = \nu_{1,0}p.\nu_{1,0}q.x(p,q)$$

as desired.

33 Appendix to Chapter IV

33.1 Linear Algebra

The following facts from linear algebra are useful to recall and get used throughout.

Proposition 33.1 *Let $A \in \mathbb{R}^{m\times n}$, then there are invertible matrices S, T such that*

$$SAT^{-1} = \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix}$$

where $r = \operatorname{rank}(A)$. Furthermore, T can taken to be orthogonal.

PROOF Take a singular-value decomposition (SVD) $A = UDV^T$, let $T = V$ and use create S from U^T by rescaling the appropriate columns. ∎

Proposition 33.2 (Row equivalence) *Two matrices $A, B \in \mathbb{R}^{m\times n}$ are called* row equivalent *if the following equivalent conditions hold*

(i) for all $x \in \mathbb{R}^n$, $Ax = 0 \Leftrightarrow Bx = 0$

(ii) A and B have the same row space

(iii) there is an invertible matrix S such that $A = SB$

Unique representatives of row equivalence classes are matrices in reduced row echelon form.

Corollary 33.3 *Let $A, B \in \mathbb{R}^{m\times n}$ and let $Ax = c$ and $Bx = d$ be* consistent *systems of linear equations that have the same solution space. There is an invertible matrix S such that $B = SA$ and $d = Sc$.*

Proposition 33.4 (Column equivalence) *For matrices $A, B \in \mathbb{R}^{m\times n}$, the following are equivalent*

(i) A and B have the same column space

(ii) there is an invertible matrix T such that $A = BT$.

Proposition 33.5 *For matrices $A, B \in \mathbb{R}^{m\times n}$, the following are equivalent*

(i) $AA^T = BB^T$

(ii) there is an orthogonal matrix U such that $A = BU$.

PROOF This is a known fact, but we sketch a proof for lack of reference. In the construction of the SVD $A = UDV^T$, we can choose U and D depending on AA^T alone. It follows that the same matrices work for B, giving SVDs $A = UDV^T, B = UDW^T$. Then $A = B(WV^T)$ as claimed. ∎

33.2 Uniqueness of Normal Forms

We now present a proof of the uniqueness of normal forms for conditioning morphisms. Some preliminary facts:

Proposition 33.6 *Let $X \sim \mathcal{N}(\mu_X, \Sigma_X)$ and $Y \sim \mathcal{N}(\mu_Y, \Sigma_Y)$ be independent. Then $X|(X = Y)$ has distribution $\mathcal{N}(\bar{\mu}, \bar{\Sigma})$ given by*

$$\bar{\mu} = \mu_X + \Sigma_X(\Sigma_X + \Sigma_Y)^+(\mu_Y - \mu_X)$$
$$\bar{\Sigma} = \Sigma_X - \Sigma_X(\Sigma_X + \Sigma_Y)^+\Sigma_X$$

In programming terms, this is written

$$\mathsf{let}\, x \;=\; N(\mu_X, \Sigma_X)\, \mathsf{in}\, (x =\!:\!= N(\mu_Y, \Sigma_Y)); \mathsf{return}\, x$$

and corresponds to the **observe** statement from []

$$\texttt{x = normal}(\mu_X, \Sigma_X)\texttt{; observe(normal}(\mu_Y, \Sigma_Y)\texttt{, x)}$$

Corollary 33.7 *No 1-dimensional observe statement leaves the prior $\mathcal{N}(0, 1)$ unchanged.*

PROOF Conditioning decreases variance. If we observe from $\mathcal{N}(\mu, \sigma^2)$, the variance of the posterior is

$$1 - (1 + \sigma^2)^{-1} < 1.$$

■

Proposition 33.8 *Consider a morphism $\kappa : n \rightsquigarrow 0$ in $\mathsf{Cond}(\mathsf{Gauss})$ given by*

$$\kappa(x) = (Ax =\!:\!= \equiv) \tag{104}$$

where $A \in \mathbb{R}^{r \times n}$ is in reduced row echelon form with no zero rows, and $\eta \in \mathsf{Gauss}(0, r)$. Then the matrix A and distribution η are uniquely determined.

PROOF We will probe κ by applying the condition 104 to different priors $\psi \in \mathsf{Gauss}(0, n)$, giving either a result $\psi' \in \mathsf{Gauss}(0, n)$ or $\bot$.

Let $S \subseteq \mathbb{R}^r$ be the support of η and $W = \{x \in R^n : Ax \in S\}$. We can recover W from observational behavior, because for deterministic priors $\psi = x_0$, we have $\psi' \neq \bot$ iff $x_0 \in W$. We have $\kappa = \bot$ iff $W = \emptyset$. Assume W is nonempty now.

Next, we can identify the nullspace K of A by considering subspaces along which no conditioning updates happens. Call an affine subspace $V \subseteq \mathbb{R}^n$ *vacuous* if for all $\psi \ll V$ we have $\psi' = \psi$. Any such V must be contained in W. We claim that every maximal vacuous subspace is of the form $K + x_0$ where $x_0 \in W$:

Every space of the form $K + x_0$ is clearly vacuous: If $\psi \ll K$ then the condition (104) becomes constant as $Ax_0 =\!:\!= \eta$. Because by assumption $Ax_0 \in S$, this condition is vacuous and can be discarded without effect.

Let V be any vacuous subspace and $x_0 \in V$. We show $V \subseteq x_0 + K$: Assume there is any other $x_1 \in W$ such that $x_1 - x_0 \notin K$ and consider the 1-dimensional prior

$$t \sim \mathcal{N}(0, 1), \quad x = x_0 + t(x_1 - x_0)$$

Let $d = A(x_1 - x_0) \neq 0$ and find an invertible matrix T such that $Td = (1, 0, \ldots, 0)^T$. The condition becomes

$$(t, 0, \ldots, 0) \eqqcolon T\eta - TAx_0.$$

All but the first equations do not involve t. By commutativity, they can be computed independently, resulting in an updated right-hand side and a 1-dimensional condition $t =: \eta'$ with η' either a Gaussian or $\bot$. By 33.7, such a condition cannot leave the prior $\mathcal{N}(0,1)$ unchanged, contradicting vacuity of V.

Having reconstructed K, the matrix A in reduced row echelon form is determined uniquely by its nullspace. Group the coordinates $x_1, \ldots, x_n$ into exactly r pivot coordinates x_p and $n - r$ free coordinates x_u. Setting $x_u = 0$ in (104) results in the simplified condition $x_p \eqqcolon \eta$. It remains to show that we can recover the observing distribution μ from observational behavior. Intuitively, if we put a flat enough prior on x_p, the posterior will resemble μ arbitrarily closely:

Let $\mu = \mathcal{N}(b, \Sigma)$ and consider $x_p \sim \mathcal{N}(\lambda I)$ for $\lambda \to \infty$. The matrix $(I + \lambda^{-1}\Sigma)$ is invertible for all large enough λ. By the formulas 33.6, the mean of the posterior is

$$\bar{\mu} = (I + \lambda^{-1}\Sigma)^{-1}\mu \xrightarrow{\lambda\to\infty} \mu$$

For the covariance, we truncate Neumann's series as

$$(I + \lambda^{-1}\Sigma)^{-1} = I - \lambda^{-1}\Sigma + o(\lambda^{-2})$$

to obtain

$$\bar{\Sigma} = \lambda I - \lambda(I + \lambda^{-1}\Sigma)^{-1} = \Sigma + o(\lambda^{-2}) \xrightarrow{\lambda\to\infty} \Sigma$$

■

33.3 Implementation

We have implemented the operational semantics of the Gaussian language of Section 17 in Python and F# [Stein, 2021]. We showcase some simple models beyond those in Section 16. In each example, we plot 100 samples, their mean (red) and standard deviation $\pm 3\sigma$ (blue):

Listing 1: Gaussian regression (Fig. 12)

```
xs = [1.0, 2.0, 2.25, 5.0, 10.0]
ys = [-3.5, -6.4, -4.0, -8.1, -11.0]

a = Gauss.N(0,10)
b = Gauss.N(0,10)
f = lambda x: a*x + b

for (x,y) in zip(xs,ys):
  Gauss.condition(f(x), y + Gauss.N(0,0.1))
```

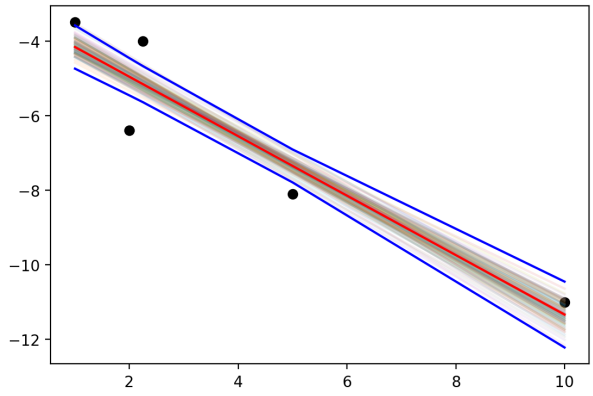


Figure 12: Gaussian regularized regression example (Ridge regression)

Listing 2: 1-dimensional Kálmán filter (Fig. 13)

```
xs = [ 1.0, 3.4, 2.7, 3.2, 5.8, 14.0, 18.0, 11.7, 19.5, 19.2]

x = [0] * len(xs)
v = [0] * len(xs)

# Initial parameters
x[0] = xs[0] + Gauss.N(0,1)
v[0] = 1.0 + Gauss.N(0,10)

for i in range(1,len(xs)):
  # Predict movement
  x[i] = x[i-1] + v[i-1]
  v[i] = v[i-1] + Gauss.N(0,0.75)

# Make noisy observations
Gauss.condition(x[i] + Gauss.N(0,1),xs[i])
```

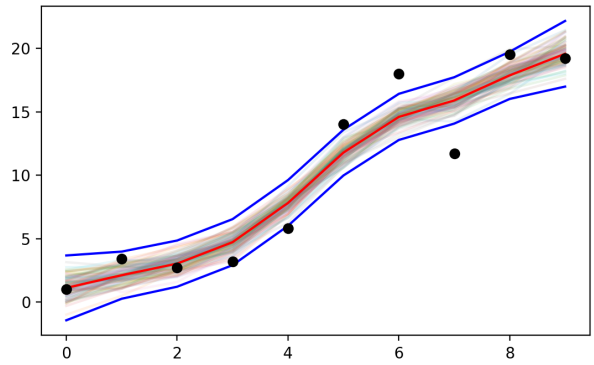


Figure 13: Kálmán filter example

34 Appendix to Chapter V

34.1 Results on quasi-Borel spaces

Proposition 34.1 *The functors* Σ, M *take discrete spaces to discrete spaces.*

PROOF If X is a discrete quasi-Borel space, then every $A \subseteq X$ is measurable, because if $\alpha : \mathbb{R} \to X$ is simple then so is $\chi_A \alpha$. For the converse, we need to show that if X is a discrete measurable space and $f : \mathbb{R} \to X$ is measurable, then f is simple. It suffices to show that the image of f is countable. The following argument is due to Ohad Kammar: Restrict our attention to $f : \mathbb{R} \to J$ where J is the image of f equipped it with the discrete σ-algebra. Then f is still measurable and now also surjective. Pick any set-theoretic section $g : J \to \mathbb{R}$ to f, then g is measurable because J is discrete. So J is a measurable retract of $\mathbb{R}$ and hence standard Borel. A standard Borel space which is also discrete must be countable. ∎

Proposition 34.2 *The functors* Σ, M *take indiscrete spaces to indiscrete spaces.*

PROOF If X is an indiscrete measurable space then MX obtains the indiscrete quasi-Borel structure because every function into X is measurable. Conversely if $M_X = \mathsf{Set}(\mathbb{R}, X)$, we need to show that $\Sigma_X = \{\emptyset, X\}$. For any $A \subseteq X$, if there exist points $x \in A, y \notin A$, pick a non-Borel set $S \subseteq \mathbb{R}$ and let $f : \mathbb{R} \to X$ send S to x and its complement to y. Then $f \in M_X$ but $\chi_A f \notin M_2$, hence χ_A is not a morphism. ∎

34.2 Normalization of First-order Expression

We will only be considering logical relations at first-order types. The crucial advantage over the general definition (Figure 11) is that in the function case $\sigma \to \tau$, we need only consider $\sigma \in \{\mathsf{N}, \mathsf{B}\}$. Arguments of these types are explicitly definable, which reduces the logical relation to a more manageable form. For boolean arguments, we merely need to check the behavior in inputs true, false, which don't involve any new names.

$$(\lambda x.M_1)\, R^{\mathsf{val}}_{\mathsf{B}\to\tau}\, (\lambda x.M_2) \Leftrightarrow \forall b \in \{\mathsf{true}, \mathsf{false}\}, M_1[b/x]\, R^{\mathsf{exp}}_{\tau}\, M_2[b/x] \tag{105}$$

For name arguments, we need to check the cases of names (n_1, n_2) which are either already in relation, or both fresh and related.

$$(\lambda x.M_1)\, R^{\mathsf{val}}_{\mathsf{N}\to\tau}\, (\lambda x.M_2) \Leftrightarrow \quad \forall (n_1,n_2) \in R, M_1[n_1/x]\, R^{\mathsf{exp}}_{\tau}\, M_2[n_2/x] \qquad (106)$$
$$\& \forall (n_1,n_2) \notin s_1 \times s_2, M_1[n_1/x]\, (R \oplus \{(n_1,n_2)\})^{\mathsf{exp}}_{\tau}\, M_2[n_2/x]$$

Notation: If $R\colon s_0 \leftrightharpoons s_1$ and $S\colon s_1 \leftrightharpoons s_2$ are spans, we let RS denote their relational composite in diagrammatic (reverse) order, meaning that $n_1\, RS\, n_3$ if $n_1\, R\, n_2$ and $n_2\, S\, n_3$ for some $n_2 \in s_2$. Note the identity

$$(R \oplus R')(S \oplus S') = RS \oplus R'S' \qquad (107)$$

Lemma 34.3 (Transitivity) *If σ is a first-order type, $M_i \in \mathsf{Exp}_\sigma(s_i)$ for $i = 1,2,3$ and $R\colon s_1 \leftrightharpoons s_2, S\colon s_2 \leftrightharpoons s_3$ are spans such that $M_1\, R_\sigma\, M_2$ and $M_2\, S_\sigma\, M_3$, then $M_1\, RS_\sigma\, M_3$.*

PROOF We argue by induction over σ. The statement is immediate on ground types; the case $\sigma = \mathsf{N}$ precisely matches the definition of relation composition. We show the case for related terms $(\lambda x.M_i)$ of type $\sigma = \mathsf{N} \to \tau$ and $i = 1,2,3$ and with to derive $(\lambda x.M_1)\, RS_{\mathsf{N}\to\tau}\, (\lambda x.M_3)$ according to (106). We distinguish two cases: If $(n_1,n_2) \in R$, $(n_2,n_3) \in S$, we have

$$M_1[n_1/x]\, R_\tau\, M_2[n_2/x] \qquad M_2[n_2/x]\, S_\tau\, M_3[n_3/x]$$

and use the inductive hypothesis to obtain

$$M_1[n_1/x]\, RS_\tau\, M_3[n_3/x]$$

On the other hand, if we have fresh arguments $(n_1,n_3) \notin s_1 \times s_3$, we pick any $n_2 \notin s_2$ and have

$$M_1[n_1/x]\, (R \oplus \{(n_1,n_2)\})_\tau\, M_2[n_2/x] \qquad M_2[n_2/x]\, (S \oplus \{(n_2,n_3)\})_\tau\, M_3[n_3/x]$$

By inductive hypothesis and (107)

$$M_1[n_2/x]\, (RS \oplus \{(n_1,n_3)\})_\tau\, M_3[n_3/x]$$

as desired. The case $\mathsf{B} \to \tau$ is analogous and no fresh names need even be considered. The expression case follows from the inductive hypothesis and (107). ∎

The following lemma shows that the definition of $\mathsf{Pub}(M,s)$ is well-defined

Proposition 34.4 *Let σ be a first-order type. For every $M \in \mathsf{Exp}_\sigma(t)$ and $s \subseteq t$, there exists a least set u with $s \subseteq u$ and $M\, \mathsf{id}_s\, M$.*

PROOF If $s \subseteq u_1, u_2$ have the property $M\, (\mathsf{id}_{u_1})_\sigma\, M$ and $M\, (\mathsf{id}_{u_2})_\sigma\, M$, then by transitivity (Lemma 34.3), so does the composite $\mathsf{id}_{u_1}\mathsf{id}_{u_2} = \mathsf{id}_{(u_1 \cap u_2)}$. We can therefore take u to be the intersection of all such sets. ∎

Lemma 34.5 *Let σ be a first-order type. Let $M_i \in \mathsf{Exp}_\sigma(s \oplus t_i)$ and suppose there is some $R\colon t_1 \leftrightharpoons t_2$ such that $M_1\, (\mathsf{id}_s \oplus R)_\sigma\, M_2$. Then R restricts to a bijection $\mathsf{Leak}(M_1,s) \cong \mathsf{Leak}(M_2,s)$.*

PROOF Write $u_i = \mathsf{Leak}(M_i, s)$. We know that $RR^{-1} = \mathsf{id}_{\mathsf{dom}(R)}$, so $M_1\ (\mathsf{id}_{s \oplus \mathsf{dom}(R)})_\sigma\ M_1$ by transitivity. As u_1 is least with this property, $u_1 \subseteq \mathsf{dom}(R)$.

Now consider the restriction $R|_{u_1}$ of R to u_1. Because $R|_{u_1} = \mathsf{id}_{u_1} R$, transitivity implies $M_1\ (\mathsf{id}_s \oplus R|_{u_1})_\sigma\ M_2$. Repeating the same argument for M_2 shows that $M_2\ (\mathsf{id}_s \oplus \mathsf{cod}(R|_{u_1}))_\sigma\ M_2$ and by minimality $u_2 \subseteq \mathsf{cod}(R|_{u_1})$. The symmetric argument shows that $R|_{u_1}$ is a bijection $u_1 \cong u_2$. ∎

Proposition 34.6 *Let τ be a first-order type and $M \in \mathsf{Exp}_\tau(s \oplus t)$ with $M \in \mathsf{Safe}^s_\tau$. Then*

(i) $\mathsf{pnf}(M, s)$ *is well-defined up to renaming bound variables and names*

(ii) if M is a value, so is $\mathsf{pnf}(M, s)$

(iii) $\mathsf{pnf}(M, s) \in \mathsf{Exp}_\tau(s)$, *that is* $\mathsf{pnf}(M, s)$ *eliminates the private names t*

(iv) $\mathsf{pnf}(M, s) \in \mathsf{Safe}^s_\tau$

(v) $M\ (\mathsf{id}_s)_\tau\ \mathsf{pnf}(M, s)$

PROOF (ii) is clear by construction. (iv) follows trivially from (iii). We prove (iii), (i) and (v) by induction on τ, following the construction of the normal form $\mathsf{pnf}(M, s)$. For (iii), the induction steps are clear and so is the case where M is a value of type B. If M is a value of type N, then $M\ (\mathsf{id}_s)_\mathsf{N}\ M$ implies that $M \in s$, and so $\mathsf{pnf}(M, s) = M \in \mathsf{Exp}_\mathsf{N}(s)$. For (i), the cases where M is a value are clear, and the expression case follows because we made a canonical choice of $u = \mathsf{Leak}(V, s)$ in the construction of $\mathsf{pnf}(M, s)$. For (v), the expression case follows directly from the inductive hypothesis and the definition of logical relations. In the case where M is a value and $\tau = \mathsf{N} \to \sigma$, we η-expand and write

$$M = \lambda x.\mathsf{if}\ x = n \in s \oplus t\ \mathsf{then}\ M_n\ \mathsf{else}\ M_0.$$

We need to verify that $M_0\ (\mathsf{id}_{s \oplus \{x\}})_\sigma\ \mathsf{pnf}(M_0, s \oplus \{x\})$ and that $M_n\ (\mathsf{id}_s)_\sigma\ \mathsf{pnf}(M_n, s)$ for $n \in s$, both of which follow from the inductive hypothesis. The case that M is a value and $\sigma = \mathsf{B} \to \tau$ is handled similarly. ∎

Theorem 34.7 *Let σ be a first-order type and let $M_i \in \mathsf{Exp}_\sigma(s \oplus t_i)$ for $i = 1, 2$. The following are equivalent:*

(i) $M_1\ (\mathsf{id}_s)_\sigma\ M_2$;

(ii) $M_i \in \mathsf{Safe}^s_\sigma$ *and* $\mathsf{pnf}(M_1, s) = \mathsf{pnf}(M_2, s)$ *after possibly renaming bound variables and names.*

PROOF If $M_i \in \mathsf{Safe}^s_\tau$ and $\mathsf{pnf}(M_1, s) = \mathsf{pnf}(M_2, s)$, then $\mathsf{pnf}(M_1, s)\ (\mathsf{id}_s)_\sigma\ \mathsf{pnf}(M_2, s)$ and so by transitivity of logical relations (34.3) and Proposition 34.6 we have $M_1\ (\mathsf{id}_s)_\sigma\ M_2$.

For the converse, suppose that $M_1\ (\mathsf{id}_s)_\sigma\ M_2$. By transitivity, it is clear that $M_i\ (\mathsf{id}_s)_\sigma\ M_i$.

To show that $\mathsf{pnf}(M_1, s) = \mathsf{pnf}(M_2, s)$, we argue by induction, following the construction of the normal forms. The base case is clear. Now consider the inductive step at values. In the case that $\sigma = \mathsf{N} \to \tau$, we η-expand and write

$$M_i = \lambda x.\mathsf{if}\ x = n \in s \oplus t_i\ \mathsf{then}\ M^i_n\ \mathsf{else}\ M^i_0.$$

By (106), we have $M_n^1 \, (\mathsf{id}_s)_\sigma \, M_n^2$ for all $n \in s$ and $M_0^1 \, (\mathsf{id}_{s \oplus \{x\}})_\sigma \, M_0^2$. By our inductive hypothesis, this means $\mathsf{pnf}(M_n^1, s) = \mathsf{pnf}(M_n^2, s)$ and $\mathsf{pnf}(M_0^1, s \oplus \{x\}) = \mathsf{pnf}(M_0^2, s \oplus \{x\})$. It follows that $\mathsf{pnf}(M_1, s) = \mathsf{pnf}(M_2, s)$. The case that $\sigma = \mathsf{B} \to \tau$ is the analogous.

In the case of expressions, let $V_i \in \mathsf{Val}_\sigma(s \oplus t_i \oplus t_i')$ be the values such that $s \oplus t_i \vdash M_i \Downarrow (t_i')V_i$. We know that $M_1 \, (\mathsf{id}_s)_\sigma \, M_2$, so there is some $R \colon t_1' \leftrightharpoons t_2'$ such that $V_1 \, (\mathsf{id}_s \oplus R)_\sigma \, V_2$. Let $u_i = \mathsf{Leak}(V_i, s) \subseteq t_i'$. By Lemma 34.5, R restricts to a bijection $u_1 \cong u_2$. Without loss of generality, rename the names u_2 in V_2 using R so we have $u_1 = u_2 = u$ and $V_1 \, (\mathsf{id}_{s \oplus u})_\sigma \, V_2$. This is possible because all names in u_2 will be bound in a ν-abstraction anyways. Now $V_i \in \mathsf{Safe}_\sigma^{s \oplus u_i}$ hence by inductive hypothesis, $\mathsf{pnf}(V_1, s \oplus u) = \mathsf{pnf}(V_2, s \oplus u)$ and we obtain

$$\mathsf{pnf}(M_1, s) \stackrel{\text{def}}{=} \nu u.\mathsf{pnf}(V_1, s \oplus u) = \nu u.\mathsf{pnf}(V_2, s \oplus u) \stackrel{\text{def}}{=} \mathsf{pnf}(M_2, s). \qquad \blacksquare$$