Probabilistic NetKAT

Nate Foster
Cornell University

Dexter Kozen
Cornell University

Konstantinos Mamouras
Cornell University

Mark Reitblatt
Cornell University

Alexandra Silva
Radboud University Nijmegen

Abstract

This paper develops a new language for programming software-defined networks based on a probabilistic semantics. We extend the NetKAT language with new primitives for expressing probabilistic behaviors and enrich the semantics from one based on deterministic functions to one based on measures and measurable functions on sets of packet histories. We establish fundamental properties of the semantics, prove that it is a conservative extension of the deterministic semantics, and show that it satisfies a number of natural equations. We present case studies that show how the language can be used to model a diverse collection of scenarios drawn from real-world networks.

1. Introduction

Formal specification and verification of networks has become a reality in recent years with the emergence of network-specific programming languages and property-checking tools. Programming languages like Frenetic [14], Pyretic [43], Maple [60], FlowLog [45], and others are enabling programmers to specify the intended behavior of a network in terms of high-level constructs such as boolean predicates and functions on packets. Verification tools like Header Space Analysis [25], VeriFlow [26], and NetKAT [15] are making it possible to check properties such as connectivity, loop freedom, and traffic isolation automatically.

However, despite many notable advances, these frameworks all have a fundamental limitation: they model network behavior in terms of deterministic packet-processing functions. This approach works well enough in settings where the network functionality is simple, or where the properties of interest only concern the forwarding paths used to carry traffic. But it does not provide satisfactory accounts of more complicated situations that often arise in practice:

- **Congestion:** the network operator wishes to calculate the expected degree of congestion on each link given traffic drawn from a statistical model.
- **Failure:** the network operator wishes to calculate the probability that packets will be delivered to their destination, given that devices and links fail with a certain probability.
- **Randomization:** the network operator wishes to configure the switches to forward traffic using randomized schemes such as equal cost multi-path routing (ECMP) or Valiant load balancing (VLB), which balance load across multiple paths.

Overall, there is a significant mismatch between the capabilities of existing reasoning frameworks and the realities of modern networks. This paper presents a new framework, Probabilistic NetKAT (ProbNetKAT), that is designed to bridge this gap.

As its name suggests, ProbNetKAT is based on NetKAT, a network programming language developed in prior work [2]. NetKAT is an extension of Kleene algebra with tests (KAT), an algebraic system for propositional verification of imperative programs that has been extensively studied for nearly two decades [30]. At the level of syntax, NetKAT offers a rich collection of intuitive constructs including: conditional tests; primitives for modifying packet headers and encoding topologies; and sequential, parallel, and iteration operators. The semantics of the language can be understood in terms of a denotational model based on functions from packet histories to sets of packet histories (where a history records the path through the network taken by a packet) or equivalently, using an equational deductive system that is sound and complete with respect to the denotational semantics. NetKAT has a PSPACE decision procedure that exploits the coalgebraic structure of the language and can solve many verification problems automatically [15]. Several practical applications of NetKAT have been developed, including algorithms for testing reachability and non-interference, a syntactic correctness proof for a compiler that translates programs to hardware instructions for SDN switches, and an implementation that handles programs written against virtual topologies [56].

Probabilistic NetKAT enriches the semantics of NetKAT so that programs denote functions that yield probability distributions on sets of packet histories. Although this change is relatively simple at the surface, it enables adding powerful primitives such as probabilistic choice, which makes it possible to handle all of the scenarios involving congestion, failure, and randomized forwarding discussed above. At the same time, it creates significant challenges, because the semantics must be extended to handle probability distributions while preserving the intuitive meaning of NetKAT's existing programming constructs. A number of important questions do not have obvious answers: Should the semantics be based on discrete or continuous distributions? How should it handle operators such as parallel composition that combine multiple distributions into a single distribution? Do suitable fixpoints exist that can be used to provide semantics for iteration?

The development of our semantics for ProbNetKAT follows a classic approach: we first define a suitable mathematical semantic space of objects, and then identify objects in this space that serve as denotations for each of the syntactic constructs in the language. More specifically, our semantics is based on Markov kernels over sets of packet histories. To a first approximation, these can be thought of as functions that produce a probability distribution on sets of packet histories, but the properties of Markov kernels ensure that important operators such as sequential composition behave as

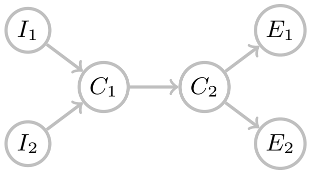

Figure 1. Congestion example: barbell topology.

expected. The parallel composition operator is particularly interesting, since it must combine disjoint and overlapping distributions—the latter models multicast, as is the Kleene star operator, since it requires showing that fixpoints exist.

We prove that the probabilistic semantics of ProbNetKAT is a conservative extension of the standard NetKAT semantics. This is a crucial point of our work: the language developed in this paper is based on NetKAT, which in turn is an extension of KAT, a well-established framework for program verification. Hence, this work can be seen as the next step in the modular development of an expressive network programming language, with increasingly sophisticated set of features, based on a sound and long-standing mathematical framework.

To evaluate our design for ProbNetKAT, we develop a number of case studies that illustrate the use of the semantics on examples inspired by real-world scenarios. Our case studies model congestion, failure, and randomization, as discussed above, as well as a gossip protocol that disseminates information through a network.

Overall, the contributions of this paper are as follows:

- We present the design of ProbNetKAT, the first language-based framework for specifying and verifying probabilistic network behavior.
- We develop a formal semantics for ProbNetKAT based on Markov kernels and prove that it conservatively extends the semantics of NetKAT.
- We discuss a number of case studies that illustrate the use of ProbNetKAT on real-world examples.

The rest of this paper is organized as follows: §2 introduces the basic ideas behind ProbNetKAT through an example; §3 reviews concepts from measure theory needed to define the semantics; §4 and §5 present the syntax and semantics of ProbNetKAT; §6 further illustrates the semantics by proving conservativity and some natural equations; §7 discusses applications of the semantics to real-world examples. We discuss related work in §8 and conclude in §9.

2. Overview

This section introduces the main features of ProbNetKAT using a simple example, and discusses some of the key challenges we experienced in designing the language.

Preliminaries. First, a bit of notation. A *packet* π is a record with fields x_1 to x_k that range over standard header fields (Ethernet addresses, frame type, VLAN, IP addresses and type, TCP ports, etc.) as well as special fields for the switch and port that indicate the location of the packet in the network:

$$\{x_1 = n_1, \ldots, x_k = n_k\}$$

We write $\pi(x)$ for value of π's x field and $\pi[n/x]$ for the packet obtained from π by setting the x field to n. In examples, we often abbreviate the switch field as sw. A *packet history* is a nonempty sequence of packets $\pi_1 : \pi_2 : \cdots : \pi_m$, listed in order of youngest to oldest. The *head packet* is π_1. Operationally, only the head packet exists in the network, but in the language we keep track of the packet's history to enable precise specification of forwarding behavior involving specific paths through the network. We write $\pi : \sigma$ for the history with head π and tail σ and H for the set of all packet histories.

Example. Consider the network shown in Figure 2 with six switches arranged into a "barbell" topology. Suppose the network operator wants to configure the network so it forwards traffic on the two left-to-right paths from I_1 to E_1 and I_2 to E_2. The following ProbNetKAT program implements this behavior:

$$\begin{aligned} p \triangleq (&sw = I_1; \mathtt{dup}; sw \leftarrow C_1; \mathtt{dup}; sw \leftarrow C_2; \mathtt{dup}; sw \leftarrow E_1)\ \&\\ (&sw = I_2; \mathtt{dup}; sw \leftarrow C_1; \mathtt{dup}; sw \leftarrow C_2; \mathtt{dup}; sw \leftarrow E_2) \end{aligned}$$

Because it only uses deterministic constructs, this program can be modeled as a function $f \in 2^H \to 2^H$ on sets of packet histories. In such a function, the input represents the initial set of in-flight packets while the output represents the final set of results produced by the program—the empty set is produced when the input packets are dropped (e.g., in a firewall) and a set with more elements than the input set is produced when some input packets are copied (e.g., in multicast). Our example program consists of tests ($sw = I_1$), which filter the set of input packets, retaining only those whose head packets satisfy the test; modifications ($sw \leftarrow C_1$), which change the value of one of the fields in the head packet; duplication ($\mathtt{dup}$), which archives the current value of the head packet in the history; and sequential (;) and parallel (&) composition operators. In this instance, the tests on each line are mutually exclusive so the parallel composition behaves like a disjoint union operator.

Probabilistic semantics. Now suppose the network operator wants to calculate not just *where* traffic is routed but also *how much* traffic is sent across each link. The deterministic semantics we have seen so far calculates the trajectories that packets take through the network. Hence, for a given set of inputs, we can use the semantics to calculate the set of output histories and then count how many packets traversed each link, yielding an upper bound on congestion. But now suppose we want to *predict* the amount of congestion that could be induced from a model that encodes expectations about the set of possible inputs. Such models, which are usually represented as traffic matrices, can be built from historical monitoring data using a variety of statistical techniques [42]. Unfortunately, even simple calculations of how much congestion is likely to occur on a given link cannot be performed using the deterministic semantics.

Encoding traffic models. Suppose that we wish to represent the following traffic model in ProbNetKAT: in each time period, the number of packets originating at I_1 is either 0, 1 or 2, with equal probability, and likewise for I_2. Let π_1 to π_4 be packets that can be distinguished from each other at any location, and write $\pi_{I_1,i}!$ for the sequence of assignments that produces the packet π_i located at switch I_1. We can express the traffic distributions at I_1 and I_2 in ProbNetKAT as the following terms:[1]

$$\begin{aligned} d_1 &\triangleq \mathtt{drop} \oplus \pi_{I_1,1}! \oplus (\pi_{I_1,1}!\ \&\ \pi_{I_1,2}!)\\ d_2 &\triangleq \mathtt{drop} \oplus \pi_{I_2,3}! \oplus (\pi_{I_2,3}!\ \&\ \pi_{I_2,4}!), \end{aligned}$$

Note that because d_1 and d_2 involve probabilistic choice, they denote functions whose values are *distributions* on sets of histories rather than simply sets of histories as before. However, because they do not contain tests, they are actually constant functions, so we can treat them as distributions. For the full input distribution to the network, we combine d_1 and d_2 independently using parallel composition: $d = d_1\ \&\ d_2$.

[1] An expression $p_1 \oplus \cdots \oplus p_n$ means that one of the p_i should be chosen at random with uniform probability and executed.

Calculating congestion. To calculate a distribution that encodes the amount of congestion on links in the network, we can push the input distribution d through the forwarding policy p using sequential composition: $d; p$. This produces a distribution on sets of trajectories through the network. In this example, there are nine such sets of trajectories, where we write $I_{1,1}$ to indicate that π_1 was processed at I_1, and similarly for the other switches and packets:

$$\begin{array}{l}\{\,\}\\ \{E_{1,1} : C_{2,1} : C_{1,1} : I_{1,1}\}\\ \{E_{1,1} : C_{2,1} : C_{1,1} : I_{1,1},\ E_{1,2} : C_{2,2} : C_{1,2} : I_{1,2}\}\\ \{E_{2,3} : C_{2,3} : C_{1,3} : I_{2,3}\}\\ \{E_{2,3} : C_{2,3} : C_{1,3} : I_{2,3},\ E_{2,4} : C_{2,4} : C_{1,4} : I_{2,4}\}\\ \quad\quad\vdots\end{array}$$

and the output distribution is uniform, each set occurring with probability $1/9$. Now suppose we wish to calculate the expected number of packets traversing the link ℓ from C_1 to C_2. We can filter the output distribution on the set

$$b \triangleq \{\sigma \mid C_{2,i} : C_{1,i} \in \sigma \text{ for some } i\}$$

and ask for the expected size of the resulting set. The filtering is again done by composition, viewing b as a guard. (In this example, all trajectories traverse the link ℓ, so the filter b has no effect.) The expected number of packets crossing ℓ is given by integration:

$$\int_{a \in 2^H} |a| \cdot [\![d; p; b]\!](da) = 2.$$

Hence, even in a simple example where forwarding is deterministic, our semantics for ProbNetKAT is quite useful: it enables making predictions about quantitative properties such as congestion, which can be used to provision capacity, inform traffic engineering algorithms, or calculate the risk that service-level agreements may be violated. More generally, ProbNetKAT can be used to express much richer behaviors such as randomized routing, faulty links, gossip, etc., as shown by the examples presented in Section 7.

Challenges. We faced several challenges in formulating the semantics of ProbNetKAT in a satisfactory way. The deterministic semantics of NetKAT [2, 15] interprets programs as packet-processing functions on sets of packet histories. This is different enough from other probabilistic models in the literature that it was not obvious how to apply standard approaches. On the one hand, we wanted to extend the deterministic semantics conservatively—i.e., a ProbNetKAT program that makes no probabilistic choices should behave the same as under the deterministic NetKAT semantics. This goal was achieved (Theorem 2) using the notion of a *Markov kernel*, well known from previous work in probabilistic semantics [13, 29, 47]. On the other hand, when moving to the probabilistic domain, several properties enjoyed by the deterministic version are lost, and great care was needed to formulate the new semantics correctly. Most notably, it is no longer the case that the meaning of a program on an input set of packet histories is uniquely determined by its action on individual histories (§6.4). The parallel composition operator (&), which supplants the union operator (+) of NetKAT, is no longer idempotent except when applied to deterministic programs (Lemma 1(vi)), and distributivity no longer holds in general (Lemma 4). Nevertheless, the semantics provides a powerful means of reasoning that is sufficient to derive many interesting and useful properties of networks (§7).

Perhaps the most challenging theoretical problem for us was the formulation of the semantics of iteration (*). In the deterministic version, the iteration operator can be defined as a sum of powers. In ProbNetKAT, this approach does not work, as it requires that parallel composition be idempotent. Hence, we formulate the semantics of iteration in terms of an infinite stochastic process. Giving denotational meaning to this operational construction required an intricate application of the Kolmogorov extension theorem (Appendix A). This formulation gives a canonical solution to an appropriate fixpoint equation as desired (Theorem 1). However the solution is not unique, and it is not a least fixpoint in any natural ordering that we are aware of. The correct notion of approximation is still not clear to us and is left as an interesting subject of future investigation.

Another challenge was the observation that in the presence of both duplication (`dup`) and iteration (*), models based on discrete distributions do not suffice, and it is necessary to base the semantics on an uncountable state space with continuous measures and sequential composition defined by integration. Most models in the literature deal with discrete distributions only, with a few exceptions (e.g. [13, 28, 29, 46, 47]). To see why a discrete semantics would suffice in the absence of either duplication or iteration note that H is a countable set. Without iteration, we could limit our attention to distributions on finite subsets of H, which is also a countable set. Similarly, with iteration but without duplication, the set H would be finite, thus a discrete semantics would suffice in that case as well, even though iterative processes do not necessarily converge after finitely many steps as with deterministic processes. However, in the presence of both duplication and iteration, infinite sets and continuous measures are unavoidable (§6.3). Having said that, in many applications, we only need to deal with discrete distributions.

3. Measure Theory Primer

To make this paper as self-contained as possible, this section introduces the background mathematics necessary to understand the semantics of ProbNetKAT. Because ProbNetKAT requires continuous probability distributions, we review some basic measure theory. This is not a matter of discretion (see §6.3). There are many excellent texts on this material; see Halmos [20], Chung [8], or Rao [51] for a more thorough treatment.

Measures are a generalization of the concepts of length or volume of Euclidean geometry to other spaces, and form the basis of continuous probability theory. In this section, we will explain what it means for a space to be *measurable*, say how to construct measurable spaces, and give basic operations and constructions on measurable spaces including Lebesgue integration with respect to a measure, and the construction of product spaces. We will also define the crucial notion of *Markov kernels*, the analog of Markov transition matrices for finite-state stochastic processes, which form the basis of our semantics for ProbNetKAT.

Measurable Spaces & Measurable Functions. A *σ-algebra* $\mathcal{B}$ on a set S is a collection of subsets of S containing $\varnothing$ and closed under complement and countable union (hence also closed under countable intersection). A pair $(S, \mathcal{B})$ where S is a set and $\mathcal{B}$ is a σ-algebra on S is called a *measurable space*. If the σ-algebra is obvious from the context, we simply say that S is a measurable space. For a measurable space $(S, \mathcal{B})$, we say that a subset $A \subseteq S$ is *measurable* if it is in $\mathcal{B}$. For applications in probability theory, elements of S and $\mathcal{B}$ are often called *outcomes* and *events*, respectively.

If $\mathcal{F}$ is a collection of subsets of a set S, then we define $\sigma(\mathcal{F})$ to be the smallest σ-algebra that contains $\mathcal{F}$. That is,

$$\sigma(\mathcal{F}) \triangleq \bigcap\{\mathcal{A} \mid \mathcal{F} \subseteq \mathcal{A} \text{ and } \mathcal{A} \text{ is a } \sigma\text{-algebra}\}.$$

Note that $\sigma(\mathcal{F})$ is well-defined, since the intersection is nonempty (we have that $\mathcal{F} \subseteq \mathcal{P}(S)$, and $\mathcal{P}(S)$ is a σ-algebra). We say that $\sigma(\mathcal{F})$ is the σ-algebra *generated* by $\mathcal{F}$. If $(S, \mathcal{B})$ is a measurable space and $\mathcal{B} = \sigma(\mathcal{F})$, we say that the space is *generated* by $\mathcal{F}$.

Let S and T be measurable spaces. A function $f : S \to T$ is *measurable* if the inverse image $f^{-1}(B) = \{x \in S \mid f(x) \in B\}$ of every measurable subset $B \subseteq T$ is a measurable subset of S.

For the particular case where T is generated by the collection $\mathcal{F}$, we have the following criterion for measurability: f is measurable if and only if $f^{-1}(B)$ is measurable for every $B \in \mathcal{F}$.

Topological Spaces & Continuous Functions. A *topology* $\mathcal{T}$ on a set S is a collection of subsets of S that contains $\varnothing$ and S and is closed under arbitrary union and binary intersection. It follows that $\mathcal{T}$ is closed under finite intersection. A pair $(S, \mathcal{T})$ where S is a set and $\mathcal{T}$ is a topology on S is called a *topological space*. We say that a subset $U \subseteq S$ is an *open set* of S if U belongs to the topology $\mathcal{T}$. If the topology is obvious from the context, we simply say that S is a topological space.

Let $\mathcal{T}$ be a topology on S. A collection $\mathcal{U}$ of open sets in $\mathcal{T}$ is said to be a *basis* for the topology $\mathcal{T}$ if every element of $\mathcal{T}$ can be written as a union of elements of $\mathcal{U}$. A subcollection $\mathcal{S} \subseteq \mathcal{T}$ is said to be a *subbasis* for the topology $\mathcal{T}$ if the collection of all finite intersections of sets in $\mathcal{S}$ is a basis for $\mathcal{T}$.

Let S and T be topological spaces. A function $f : S \to T$ is said to be *continuous* if for every open subset $V \subseteq T$, the inverse image $f^{-1}(V) = \{x \in S \mid f(x) \in V\}$ is an open subset of S. Suppose now that $\mathcal{V}$ is a basis and $\mathcal{S}$ is a subbasis for the topology of T. Then, f is continuous if and only if $f^{-1}(V)$ is open for every $V \in \mathcal{V}$ if and only if $f^{-1}(V)$ is open for every $V \in \mathcal{S}$. These equivalences give us simpler criteria for continuity.

Example 1 (The Cantor Space)**.** Consider the set $2 = \{0, 1\}$ with the *discrete topology*, which consists of the open sets $\varnothing$, $\{0\}$, $\{1\}$, and $\{0, 1\}$. We can think of 2^ω variously as infinite streams of Booleans, as the set of ω-indexed tuples with values in 2, as the set of subsets of ω, or as the set of functions of type $\omega \to 2$. We define the projection mappings $\pi_i : 2^\omega \to 2$ by putting $\pi_i(a) = a(i)$ for all $a \in 2^\omega$. The *product topology* on 2^ω is the coarsest (smallest) topology for which the projections π_i are continuous. So for all indices $i < \omega$, the sets

$$\begin{aligned} \pi_i^{-1}(\varnothing) &= \varnothing & \pi_i^{-1}(\{0\}) &= \{a \in 2^\omega \mid i \notin a\} \\ \pi_i^{-1}(\{0,1\}) &= 2^\omega & \pi_i^{-1}(\{1\}) &= \{a \in 2^\omega \mid i \in a\} \end{aligned}$$

must be open. It suffices to consider the collection $\mathcal{S}$ consisting of the sets B_i and ${\sim}B_i = 2^\omega \setminus B_i$, where

$$B_i \triangleq \{a \in 2^\omega \mid i \in a\}.$$

The *Cantor space* is the set 2^ω together with the product topology, which is the coarsest topology containing $\mathcal{S}$. It follows that $\mathcal{S}$ is a subbasis for the topology.

Consider the set $2^\omega \times 2^\omega$ together with the topology generated by the sets $U \times V$ where U, V are open. The *binary union* function, which sends a pair $(a, b) \in 2^\omega \times 2^\omega$ to $a \cup b \in 2^\omega$, is continuous, because the subsets

$$\begin{aligned} \{(a,b) \mid a \cup b \in B_i\} &= \{(a,b) \mid i \in a \cup b\} \\ &= (B_i \times 2^\omega) \cup (2^\omega \times B_i) \\ \{(a,b) \mid a \cup b \in {\sim}B_i\} &= \{(a,b) \mid i \notin a \cup b\} \\ &= ({\sim}B_i \times 2^\omega) \cap (2^\omega \times {\sim}B_i) \\ &= {\sim}B_i \times {\sim}B_i \end{aligned}$$

are open for all indices $i < \omega$.

Borel Sets & Measurable Real-Valued Functions. Let $\mathcal{T}$ be a topology on the set S. We say that $\sigma(\mathcal{T})$ is the *Borel σ-algebra* generated by the topology $\mathcal{T}$. The sets of this σ-algebra are also called the *Borel sets* of the topology.

Let S and T be topological spaces. If the function $f : S \to T$ is continuous, then it is also measurable with respect to the Borel sets. In this case we say that f is *Borel measurable*.

Example 2 (Borel Sets of $\mathbb{R}$)**.** We say that a subset $U \subseteq \mathbb{R}$ is *open* if for every $x \in U$ there is an open interval (a, b) with $a < b$ such that $x \in (a, b)$ and $(a, b) \subseteq U$. This is the *standard topology* on $\mathbb{R}$. We observe that the collection of open intervals is a basis for the topology. The *Borel σ-algebra* of $\mathbb{R}$ is the σ-algebra generated by the open sets of the standard topology, or equivalently by the open intervals. A set that belongs to the Borel σ-algebra is called a *Borel set* of $\mathbb{R}$.

Let S be a measurable space and $f : S \to \mathbb{R}$. We say that f is *measurable* if it is measurable with respect to the Borel sets of $\mathbb{R}$. This is equivalent to the condition: the inverse image $f^{-1}((a, b))$ of every open interval (a, b) is a measurable subset of S. Consider four different collections of subsets of $\mathbb{R}$ that consist of intervals of the following four forms respectively:

$$(-\infty, b) \qquad (-\infty, b] \qquad (a, \infty) \qquad [a, \infty)$$

The Borel σ-algebra on $\mathbb{R}$ is generated by any of the above collections of intervals. So $f : S \to \mathbb{R}$ being measurable is equivalent to each of the following four conditions, each of which gives a simple criterion for the measurability of f:

For every $b \in \mathbb{R}$, the set $\{x \in S \mid f(x) < b\}$ is measurable.
For every $b \in \mathbb{R}$, the set $\{x \in S \mid f(x) \leq b\}$ is measurable.
For every $a \in \mathbb{R}$, the set $\{x \in S \mid f(x) > a\}$ is measurable.
For every $a \in \mathbb{R}$, the set $\{x \in S \mid f(x) \geq a\}$ is measurable.

Example 3. Let S be a measurable space. The *characteristic function* $\chi_A : S \to \mathbb{R}$ of a subset $A \subseteq S$ is given by

$$\chi_A(x) \triangleq \begin{cases} 1, & \text{if } x \in A; \\ 0, & \text{if } x \notin A. \end{cases}$$

Then A is measurable iff χ_A is measurable. The proof relies on a straightforward use of one of the above criteria.

For the spaces we consider, the set $\mathcal{B}$ will always be the Borel sets of the topology—see [8, 20].

Measures. A *measure* on $(S, \mathcal{B})$ is a countably additive map $\mu : \mathcal{B} \to \mathbb{R}$. The condition that the map be *countably additive* stipulates that if $A_i \in \mathcal{B}$ is a countable set of pairwise disjoint events, then $\mu(\bigcup_i A_i) = \sum_i \mu(A_i)$. Equivalently, if A_i is a countable chain of events, that is, if $A_i \subseteq A_j$ for $i \leq j$, then $\lim_i \mu(A_i)$ exists and is equal to $\mu(\bigcup_i A_i)$. A measure is a *probability measure* if $\mu(A) \geq 0$ for all $A \in \mathcal{B}$ and $\mu(S) = 1$. By convention, $\mu(\varnothing) = 0$.

For every $a \in S$, the Dirac (or point-mass) measure on a is the probability measure:

$$\delta_a(A) = \begin{cases} 1, & a \in A, \\ 0, & a \notin A. \end{cases}$$

A measure is said to be *discrete* if it is a countable weighted sum of Dirac measures.

Product Spaces and Product Measures. Given two measurable spaces $(S, \mathcal{B}_S)$ and $(T, \mathcal{B}_T)$, the *product space* has elements $S \times T$ and measurable sets the Borel sets of the product topology, which is the weakest topology making the two projections $\pi_1 : S \times T \to S$ and $\pi_2 : S \times T \to T$ continuous. This is also the smallest σ-algebra containing the *measurable rectangles* $A \times B$, where $A \in \mathcal{B}_S$ and $B \in \mathcal{B}_T$, and the smallest σ-algebra such that π_1 and π_2 are measurable functions.

Given μ, ν measures on S and T, respectively, the *product measure* $\mu \times \nu$ is a measure on the product space defined by

$$(\mu \times \nu)(A \times B) = \mu(A) \cdot \nu(B)$$

for measurable rectangles $A \times B$. The product measure captures the idea of choosing a pair $(s, t) \in S \times T$ by sampling μ and ν independently.

More generally, given a finite or countable collection of measurable spaces $(S_n, \mathcal{B}_n)$, the *product space* has elements $\prod_n S_n$ and measurable sets the Borel sets of the product topology on $\prod_n S_n$, which is the weakest topology making all projections continuous. Given μ_n measures on S_n, the *product measure* $\prod_n \mu_n$ is a measure on the product space defined by

$$(\textstyle\prod_n \mu_n)(\prod_n A_n) = \prod_n(\mu_n(A_n))$$

for measurable rectangles $\prod_n A_n$.

Integration. A probability measure μ on $(S, \mathcal{B})$ and a bounded measurable function $f : (S, \mathcal{B}) \to \mathbb{R}$ can be combined by the Lebesgue integral:

$$\int_{s\in S} f(s) \cdot \mu(ds) \in \mathbb{R}.$$

We will often make use of the *change-of-variable rule* [20, Theorem 39.C]: If $g : (S, \mathcal{B}_S) \to (T, \mathcal{B}_T)$ is a measurable function and $f : (T, \mathcal{B}_T) \to \mathbb{R}$ is a bounded measurable function, then

$$\int_{s\in S} f(g(s)) \cdot \mu(ds) = \int_{t\in T} f(t) \cdot \mu(g^{-1}(dt)). \tag{3.1}$$

Markov Kernels. Let $(S, \mathcal{B}_S)$ and $(T, \mathcal{B}_T)$ be measurable spaces. A function $P : S \times \mathcal{B}_T \to \mathbb{R}$ is called a *Markov kernel* (and sometimes also called a Markov transition, measurable kernel, stochastic kernel, stochastic relation, etc.) if

- for fixed $A \in \mathcal{B}_T$, the map $\lambda s.P(s, A) : S \to \mathbb{R}$ is a measurable function on $(S, \mathcal{B}_S)$; and
- for fixed $s \in S$, the map $\lambda A.P(s, A) : \mathcal{B}_T \to \mathbb{R}$ is a probability measure on $(T, \mathcal{B}_T)$.

These properties allow integration on the left and right respectively.

The measurable spaces and Markov kernels form a category, the *Kleisli category of the Giry monad*; see [13, 46, 47]. In this context, we occasionally write $P : (S, \mathcal{B}_S) \to (T, \mathcal{B}_T)$ or just $P : S \to T$. Composition is given by integration: for $P : S \to T$ and $Q : T \to U$,

$$(P\,; Q)(s, A) = \int_{t\in T} P(s, dt) \cdot Q(t, A).$$

Associativity of composition is essentially Fubini's theorem (see Chung [8] or Halmos [20]). Markov kernels were first proposed as a model of probabilistic while programs by Kozen [29].

Deterministic Kernels. A Markov kernel $P : S \to T$ is *deterministic* if for every $s \in S$, there is an $f(s) \in T$ such that:

$$P(s, A) = \delta_{f(s)}(A) = \delta_s(f^{-1}(A)) = \chi_A(f(s)).$$

The set function $f : S \to T$ is necessarily Borel measurable. Conversely, every measurable function gives a deterministic kernel. Thus the deterministic kernels and the Borel measurable functions are in one-to-one correspondence.

Naturals	n	$\in$	$0 \mid 1 \mid 2 \mid \dots$	
Fields	x	$::=$	$x_1 \mid \cdots \mid x_k$	
Packets	pk	$::=$	$\{x_1 = n_1, \cdots, x_k = n_k\}$	
Histories	σ	$::=$	$\langle pk \rangle \mid pk{:}\sigma$	
Guards	g	$\subseteq$	2^H	
	`skip`	$\triangleq$	2^H	
	`drop`	$\triangleq$	$\{\}$	
	$x = n$	$\triangleq$	$\{pk{:}h \mid pk(x) = n\}$	
Tests	a	$::=$	g	*Guard*
		$\mid$	$a_1 \mathbin{\&} a_2$	*Disjunction*
		$\mid$	$a_1; a_2$	*Conjunction*
		$\mid$	$\overline{a}$	*Negation*
Actions	p	$::=$	a	*Test*
		$\mid$	$x \leftarrow n$	*Modification*
		$\mid$	$p_1 \mathbin{\&} p_2$	*Parallel Composition*
		$\mid$	$p_1; p_2$	*Sequential Composition*
		$\mid$	$p_1 \oplus_r p_2$	*Probabilistic Choice*
		$\mid$	p^*	*Iteration*
		$\mid$	`dup`	*Duplication*

Figure 2. ProbNetKAT Syntax.

Deterministic kernels compose on the left and right with an arbitrary kernel as follows:

$$\begin{aligned}(f\,; P)(s, A) &= \int_t \delta_{f(s)}(dt) \cdot P(t, A)\\ &= P(f(s), A)\\ (P\,; f)(s, A) &= \int_t P(s, dt) \cdot \chi_A(f(t))\\ &= \int_u P(s, f^{-1}(du)) \cdot \chi_A(u)\\ &= \int_{u\in A} P(s, f^{-1}(du))\\ &= P(s, f^{-1}(A)).\end{aligned}$$

Next we define the syntax and semantics of ProbNetKAT formally, with Markov kernels playing a central role in the semantics.

4. Syntax

ProbNetKAT extends NetKAT [2, 15], which is itself based on Kleene algebra with tests (KAT) [30], a generic equational system for reasoning about partial correctness of programs.

4.1 Kleene Algebra (KA) & Kleene Algebra with Tests (KAT)

A *Kleene algebra* (KA) is an algebraic structure,

$$(K,\, +,\, \cdot,\, ^*,\, 0,\, 1)$$

where K is an idempotent semiring under $(+, \cdot, 0, 1)$, and $p^* \cdot q$ is the least solution of the affine linear inequality $p \cdot r + q \leq r$, where $p \leq q$ is shorthand for $p + q = q$, and similarly for $q \cdot p^*$. A *Kleene algebra with tests* (KAT) is a two-sorted algebraic structure,

$$(K,\, B,\, +,\, \cdot,\, ^*,\, 0,\, 1,\, \neg)$$

where $\neg$ is a unary operator defined only on B, such that

- $(K,\, +,\, \cdot,\, ^*,\, 0,\, 1)$ is a Kleene algebra,

- $(B, +, \cdot, \neg, 0, 1)$ is a Boolean algebra, and
- $(B, +, \cdot, 0, 1)$ is a subalgebra of $(K, +, \cdot, 0, 1)$.

The elements of B and K are usually called *tests* and *actions*.

The axioms of KA and KAT (both elided here) capture natural conditions such as associativity of $\cdot$; see the original paper by Kozen for a complete listing [30]. Note that the KAT axioms do not hold for arbitrary ProbNetKAT programs—e.g., parallel composition is not idempotent—although they do hold for the deterministic fragment of the language.

4.2 NetKAT

NetKAT [2, 15] extends KAT with network-specific primitives for filtering, modifying, and forwarding packets, along with additional axioms for reasoning about programs built using those primitives. Formally, NetKAT is KAT with atomic actions and tests

$$x \leftarrow n \qquad\qquad x = n \qquad\qquad \texttt{dup}$$

with the following intuitive meanings: The assignment $x \leftarrow n$ assigns the value n to the field x in the current packet; the test $x = n$ tests whether field x of the current packet contains the value n; the action dup duplicates the packet in the packet history, which keeps track of the path the packet takes through the network. In NetKAT, we write ; instead of $\cdot$, skip instead of 1, and drop instead of 0, as these names capture their intuitive use as programming constructs. We often use juxtaposition to indicate sequential composition in examples.

For example, the NetKAT expression

$$sw = 6\,; pt = 8\,; dst \leftarrow 10.0.1.5\,; pt \leftarrow 5$$

encodes the command: "For all packets located at port 8 of switch 6, set the destination address to 10.0.1.5 and forward it out on port 5."

4.3 ProbNetKAT

ProbNetKAT extends NetKAT with several new operations, as shown in the grammar in Figure 2:

- A *random choice* operation $p \oplus_r q$, where p and q are expressions and r is a real number in the interval $[0, 1]$. The expression $p \oplus_r q$ intuitively behaves according to p with probability r and q with probability $1 - r$. We frequently omit the subscript r, in which case r is understood to implicitly be $1/2$.
- A *parallel composition* operation $p \;\&\; q$, where p and q are expressions. The expression $p \;\&\; q$ intuitively says to perform both p and q, making any probabilistic choices in p and q independently, and combine the results. The operation & serves the same purpose as $+$ in NetKAT and replaces it syntactically. We use the notation & to distinguish it from $+$, which is used in the semantics to add measures and measurable functions as in [28, 29].
- *Guards* g which generalize NetKAT's tests by allowing them to operate on the entire packet history rather than simply the head packet. Formally a guard g is just an element of 2^H. The guard skip is defined as the set of all packet histories and drop is the empty set. An atomic test $x = n$ is defined as the set of all histories σ where the x field of the head packet of σ is n. As we saw in §2, guards are often useful for reasoning probabilistically about properties such as congestion.

Although ProbNetKAT is based on KAT, it is important to keep in mind that because the semantics is probabilistic, many of the familiar equations of KAT no longer hold. For example, idempotence of parallel composition does not hold in general. We will however prove that ProbNetKAT conservatively extends NetKAT, so it follows that the NetKAT axioms hold on the deterministic fragment.

5. Semantics

The standard semantics of NetKAT interprets expressions as packet-processing functions. As defined in Figure 2, a packet π is a record whose fields assign constant values n to fields x and a packet history is a nonempty sequence of packets $\pi_1 : \pi_2 : \cdots : \pi_k$, listed in order of youngest to oldest. Recall that operationally, only the head packet π_1 exists in the network, but we keep track of the packet's history to enable precise specification of behavior involving forwarding along different paths.

Formally, a NetKAT term p denotes a function

$$[\![p]\!] : H \to 2^H,$$

where H is the set of all packet histories. Intuitively, the function $[\![p]\!]$ takes an input packet history σ and produces a set of output packet histories $[\![p]\!](\sigma)$.

The semantics of the primitive actions and tests in NetKAT are as follows. For a packet history $\pi : \sigma$ with head packet π,

$$\begin{aligned}
[\![x \leftarrow n]\!](\pi : \sigma) &= \{\pi[n/x] : \sigma\} \\
[\![x = n]\!](\pi : \sigma) &= \begin{cases} \{\pi : \sigma\}, & \pi(x) = n \\ \varnothing, & \pi(x) \neq n \end{cases} \\
[\![\texttt{dup}]\!](\pi : \sigma) &= \{\pi : \pi : \sigma\} \\
[\![\texttt{skip}]\!](\sigma) &= \{\sigma\} \\
[\![\texttt{drop}]\!](\sigma) &= \varnothing.
\end{aligned}$$

where $\pi[n/x]$ denotes the packet π with the field x rebound to the value n. A test $x = n$ drops the packet if the test is not satisfied and passes it through unaltered if it is satisfied—that is, tests behave as filters on packets. The dup construct duplicates the head packet π, yielding a fresh copy that can be modified by other constructs. Hence, in this standard model, the dup construct can be used to encode paths through the network, with each occurrence of dup marking an intermediate hop.

The KAT operations are interpreted as follows:

$$\begin{aligned}
[\![p + q]\!](\sigma) &= [\![p]\!](\sigma) \cup [\![q]\!](\sigma) \\
[\![p; q]\!](\sigma) &= \bigcup_{\tau \in [\![p]\!](\sigma)} [\![q]\!](\tau) \\
[\![p^*]\!](\sigma) &= \bigcup_n [\![p^n]\!](\sigma) \\
[\![\overline{a}]\!](\sigma) &= \begin{cases} \{\sigma\}, & \text{if } [\![a]\!](\sigma) = \varnothing \\ \varnothing, & \text{if } [\![a]\!](\sigma) = \{\sigma\} \end{cases}
\end{aligned}$$

Note that $+$ behaves like a disjunction operation when applied to tests and like a union operation when applied to actions. Similarly, ; behaves like a conjunction operation when applied to tests and like a sequential composition when applied to actions. Negation is only ever applied to tests, as is enforced by the syntax of the langauge.

5.1 Sets of Packet Histories as a Measurable Space

To give a denotational semantics to ProbNetKAT, we must first identify a suitable space of mathematical objects. Because we want to reason about probability distributions over sets of network paths, we construct a *measurable space* (as defined in §3) from sets of packet histories, and then define the semantics using Markov kernels on this space.

The powerset 2^H of packet histories H forms a topological space with topology generated by basic clopen sets,

$$B_\tau = \{a \in 2^H \mid \tau \in a\},\ \tau \in H.$$

This space is homeomorphic to the *Cantor space*, the topological product of countably many copies of the discrete two-element space. Let $\mathcal{B} \subseteq 2^{2^H}$ be the Borel sets of this topology. This is the smallest σ-algebra containing the B_τ. The measurable space $(2^H, \mathcal{B})$ with outcomes 2^H and events $\mathcal{B}$ provides a foundation for interpreting ProbNetKAT programs as Markov kernels $2^H \to 2^H$.

5.2 The Operation &

Next, we define an operation on measures that will be needed to define the semantics of ProbNetKAT's parallel composition operator. Parallel composition differs in some important ways from NetKAT's union operator—intuitively, union merely combines the sets of packet histories generated by its arguments, whereas parallel composition must somehow combine measures on sets of packet histories, which is a more intricate operation. For example, while union is idempotent, parallel composition will not be in general.

Operationally, the & operation on measures can be understood as follows: given measures μ and ν, to compute the measure $\mu \mathbin{\&} \nu$, we sample μ and ν independently to get two subsets of H, then take their union. The probability of an event $A \in \mathcal{B}$ is the probability that this union is in A.

Formally, given $\mu, \nu \in \mathcal{M}$, let $\mu \times \nu$ be the product measure on the product space $2^H \times 2^H$. The union operation $\bigcup : 2^H \times 2^H \to 2^H$ is continuous and therefore measurable, so we can define

$$(\mu \mathbin{\&} \nu)(A) \triangleq (\mu \times \nu)(\{(a, b) \mid a \cup b \in A\}). \qquad (5.1)$$

Intuitively, this is the probability that the union $a \cup b$ of two independent samples taken with respect to μ and ν lies in A.

The & operation enjoys a number of useful properties, as captured by the following lemma:

Lemma 1.

(i) & *is associative and commutative* (*however, it is not idempotent in general—see* (vi)).
(ii) & *is linear in both arguments.*
(iii) $(\delta_a \mathbin{\&} \mu)(A) = \mu(\{b \mid a \cup b \in A\})$.
(iv) $\delta_a \mathbin{\&} \delta_b = \delta_{a \cup b}$.
(v) $\delta_\varnothing$ *is a two-sided identity for* &.
(vi) $\mu \mathbin{\&} \mu = \mu$ *iff* $\mu = \delta_a$ *for some* $a \in 2^H$.

The proof of these facts is given in Appendix B.

There is also an infinitary version of & that operates on finite or countable multisets of measures, but we will not need this for our development.

5.3 Semantics of ProbNetKAT

Now we are ready to define the semantics of ProbNetKAT itself. Every ProbNetKAT term p will denote a Markov kernel

$$[\![p]\!] : 2^H \times \mathcal{B} \to \mathbb{R}$$

which can be curried variously as

$$[\![p]\!] : 2^H \to \mathcal{B} \to \mathbb{R} \qquad [\![p]\!] : \mathcal{B} \to 2^H \to \mathbb{R}.$$

Intuitively, the term p, given an input $a \in 2^H$, produces an output according to the distribution $[\![p]\!](a)$. We can think of running the program p with input a as a probabilistic experiment, and the value $[\![p]\!](a, A) \in \mathbb{R}$ is the probability that the outcome of the experiment lies in $A \in \mathcal{B}$. The measure $[\![p]\!](a)$ is not necessarily discrete (§6.3): its total weight is always 1, although the probability of any given singleton may be 0.

The semantics of the atomic operations are defined as follows. For $a \in 2^H$,

$$\begin{aligned}
[\![x \leftarrow n]\!](a) &= \delta_{\{\pi[n/x]:\sigma \,\mid\, \pi:\sigma \in a\}}\\
[\![x = n]\!](a) &= \delta_{\{\pi:\sigma \,\mid\, \pi:\sigma \in a,\ \pi(x)=n\}}\\
[\![\mathtt{dup}]\!](a) &= \delta_{\{\pi:\pi:\sigma \,\mid\, \pi:\sigma \in a\}}\\
[\![\mathtt{skip}]\!](a) &= \delta_a\\
[\![\mathtt{drop}]\!](a) &= \delta_\varnothing,
\end{aligned}$$

where $\pi[n/x]$ denotes packet π with the field x rebound to the value n. Note that if no elements of a satisfy the test $x = n$, the result is $\delta_\varnothing$, which is the Dirac measure on the emptyset, not the constant 0 measure.

These are all deterministic terms, and as such, they correspond to measurable functions $f : 2^H \to 2^H$. In each of these cases, the function f is completely determined by its action on singletons, and indeed by its action on the head packet of the unique element of each of those singletons.

The semantics of the remaining ProbNetKAT terms, except for Kleene star, is defined as follows:

$$\begin{aligned}
[\![p \oplus_r q]\!](a) &= r[\![p]\!](a) + (1-r)[\![q]\!](a)\\
[\![p\,;q]\!](a) &= [\![q]\!]([\![p]\!](a))\\
[\![p \mathbin{\&} q]\!](a) &= [\![p]\!](a) \mathbin{\&} [\![q]\!](a)
\end{aligned}$$

Note that the semantics of composition requires us to extend $[\![q]\!]$ to allow measures as inputs. This is done by integration as described in §3. For μ a measure on 2^H,

$$[\![q]\!](\mu) \triangleq \lambda A. \int_{a \in 2^H} [\![q]\!](a, A) \cdot \mu(da).$$

It is not surprising that this extension is needed. In NetKAT, the semantics had to be similarly extended to take sets of histories as input to define the semantics of sequential composition. Both phenomena are consequences of sequential composition taking place in the Kleisli category of a monad: the powerset monad for NetKAT and the Giry monad for ProbNetKAT.

5.4 Semantics of Iteration

To complete the semantics, we must define the semantics of the Kleene star operator. This turns out to be quite challenging, because the usual definition of star as a sum of powers does not work with ProbNetKAT. Instead, we define an infinite stochastic process and show that it satisfies the essential fixpoint equation that Kleene star is expected to obey (Theorem 1).

To define the measure $[\![p^*]\!](c_0)$, consider the following infinite stochastic process. Starting with $c_0 \in 2^H$, create a sequence $c_0, c_1, c_2, \ldots$ inductively. After n steps, say we have constructed $c_0, \ldots, c_n$. Let c_{n+1} be the outcome obtained by sampling 2^H according to the distribution $[\![p]\!](c_n)$. Continue this process forever to get an infinite sequence $c_0, c_1, c_2, \ldots \in (2^H)^\omega$. Take the union of the resulting sequence $\bigcup_n c_n$ and ask whether it is in A. The probability of this event is taken to be $[\![p^*]\!](c_0, A)$.

This intuitive operational view can also be justified denotationally. However, the formal development is quite technical and depends on an application of the Kolmogorov extension theorem. For the details, see Appendix A.

The following theorem shows that the iteration operator satisfies a certain desired fixpoint equation. This property is in fact the original motivating force behind the definition as we have given it. It can be used to describe the iterated behavior of a network (§7) and to define the semantics of while loops (§5.5).

Theorem 1. $[\![p^*]\!] = [\![\mathtt{skip} \mathbin{\&} pp^*]\!]$.

Proof. To determine the probability $[\![p^*]\!](c_0, A)$, we sample $[\![p]\!](c_0)$ to get an outcome c_1, then run the protocol $[\![p^*]\!]$ on c_1 to obtain a set c, then ask whether $c_0 \cup c \in A$. Thus

$$\begin{aligned}
[\![p^*]\!](c_0, A) &= \int_{c_1} [\![p]\!](c_0, dc_1) \cdot [\![p^*]\!](c_1, \{c \mid c_0 \cup c \in A\}) \\
&= [\![p^*]\!]([\![p]\!](c_0))(\{c \mid c_0 \cup c \in A\}) \\
&= (\delta_{c_0} \;\&\; [\![p^*]\!]([\![p]\!](c_0)))(A) \qquad \text{by Lemma 1(iii)} \\
&= ([\![\mathtt{skip}]\!](c_0) \;\&\; [\![pp^*]\!](c_0))(A) \\
&= [\![\mathtt{skip} \;\&\; pp^*]\!](c_0, A). \qquad \square
\end{aligned}$$

Note that unlike KAT and NetKAT, $[\![p^*]\!]$ is *not* the same as the infinite sum of powers $[\![\&_n\, p^n]\!]$. The latter fails to capture the sequential nature of iteration in the presence of probabilistic choice.

5.5 Guards

ProbNetKAT's *guards* generalize tests, which are predicates defined by their behavior on the first packet in a history, to predicates over the entire history. Recall that a guard is simply an element $g \in 2^H$ used as a deterministic program with semantics

$$[\![g]\!](a) \triangleq \delta_{a \cap g}.$$

A test $x = n$ is a special case in which $g = \{\pi{:}\tau \mid \pi(x) = n\}$. Note that unlike other ProbNetKAT atomic programs, guards are not necessarily determined by their action on the head packet.

By Lemma 1, guards extend to measures as follows:

$$[\![g]\!](\mu) = \lambda A.\mu(\{a \mid a \cap g \in A\}).$$

Using this construct, we can define encodings of conditionals and while loops as follows:

$$\mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q = bp \;\&\; \overline{b}q \qquad \mathsf{while}\ b\ \mathsf{do}\ p = (bp)^*\overline{b}.$$

Importantly, unlike treatments involving subprobability measures found in previous work [29, 46], the output here is always a probability measure, even if the program does not halt. For example, the output of the program $\mathsf{while\ true\ do\ skip}$ is the Dirac measure $\delta_\varnothing$.

6. Properties

Having defined the formal semantics of ProbNetKAT in terms of Markov kernels, we now develop some essential properties of ProbNetKAT that provide further evidence in support of our semantics.

- We prove that ProbNetKAT is a conservative extension of NetKAT—i.e., every deterministic ProbNetKAT program behaves like the corresponding NetKAT program.
- We present some additional properties enjoyed by ProbNetKAT programs.
- We show that ProbNetKAT programs can generate continuous measures from discrete inputs, which shows that our use of Markov kernels is truly necessary and that no semantics based on discrete measures would suffice.
- Finally, we present a tempting alternative "uncorrelated" semantics and show that it is inadequate for defining the semantics of ProbNetKAT.

6.1 Conservativity of the Extension

Although ProbNetKAT extends NetKAT with new probabilistic operators, the addition of these operators does not affect the behavior of purely deterministic programs. We will prove that this property is indeed true of our semantics—i.e., ProbNetKAT is a conservative extension of NetKAT.

First, we show that programs that do not use random choice are deterministic:

Lemma 2. *All syntactically deterministic ProbNetKAT programs p (those without an occurrence of $\oplus_r$) are (semantically) deterministic; that is, for any $a \in 2^H$, $[\![p]\!](a)$ is a point mass.*

Next we show that on deterministic programs, the ProbNetKAT and NetKAT semantics agree. Let $[\![\cdot]\!]_{\mathrm{N}}$ denote the NetKAT semantic map and $[\![\cdot]\!]_{\mathrm{P}}$ the ProbNetKAT semantic map.

Theorem 2. *For deterministic programs, ProbNetKAT semantics and NetKAT semantics agree in the following sense. For $a \in 2^H$, define $[\![p]\!]_{\mathrm{N}}(a) = \bigcup_{\tau \in a} [\![p]\!]_{\mathrm{N}}(\tau)$. Then for any $a, b \in 2^H$,*

$$[\![p]\!]_{\mathrm{N}}(a) = b \;\Leftrightarrow\; [\![p]\!]_{\mathrm{P}}(a) = \delta_b.$$

The proofs of this lemma and theorem are given in Appendix B.

Using the fact that the NetKAT axioms are sound and complete with respect to the denotational semantics [2, Theorems 1 and 2], we immediately obtain the following corollary:

Corollary 1. *The NetKAT axioms are sound and complete for deterministic ProbNetKAT programs.*

Together, these results provide additional justification that our probabilistic semantics captures the desired behavior.

6.2 Further Properties

Next, we identify several natural equations that are satisfied by ProbNetKAT programs. The first two equations show that `drop` is a left and right unit for the parallel composition operator &:

$$[\![p \;\&\; \mathtt{drop}]\!] = [\![p]\!] = [\![\mathtt{drop} \;\&\; p]\!]$$

This equation makes intuitive sense as deterministically dropping all inputs should have no affect when composed in parallel with any other program. The next equation states that $\oplus$ is idempotent:

$$[\![p \oplus_s p]\!] = [\![p]\!]$$

Again, this equation makes sense intuitively as randomly choosing between p and itself is the same as simply executing p. The next few equations show that parallel composition is associative,

$$[\![(p \;\&\; q) \;\&\; r]\!] = [\![p \;\&\; (q \;\&\; r)]\!]$$

and commutative:

$$[\![p \;\&\; q]\!] = [\![q \;\&\; p]\!]$$

The next equation shows that the arguments to random choice can be exchanged, provided the bias is complemented:

$$[\![p \oplus_s q]\!] = [\![q \oplus_{1-s} p]\!]$$

The final equation describes how to reassociate expressions involving random choice with explicit biases:

$$[\![\left(p \oplus_{\frac{a}{a+b}} q\right) \oplus_{\frac{a+b}{a+b+c}} r]\!] = [\![p \oplus_{\frac{a}{a+b+c}} \left(q \oplus_{\frac{b}{b+c}} r\right)]\!]$$

Next we develop some additional properties involving deterministic programs.

Lemma 3. *Let p be deterministic with $[\![p]\!](a) = \delta_{f(a)}$. The function $f : 2^H \to 2^H$ is measurable, and for any measure μ,*

$$[\![p]\!](\mu) = \mu \circ f^{-1}.$$

As we have seen in Lemma 1(vi), & is not idempotent except in the deterministic case. Neither does sequential composition distribute over & in general. However,

Lemma 4. *If p is deterministic, then*

$$[\![p(q \;\&\; r)]\!] = [\![pq \;\&\; pr]\!] \qquad [\![(q \;\&\; r)p]\!] = [\![qp \;\&\; rp]\!].$$

Neither equation holds unconditionally.

The proofs are given in Appendix B.

Finally, we give examples involving iteration. Consider the program $\mathtt{skip} \oplus_r \mathtt{dup}$. This program does nothing with probability r and duplicates the head packet with probability $1 - r$, where $r \in [0, 1)$. We claim that independent of r, the value of the starred program on any single packet π is the point mass

$$[\![(\mathtt{skip} \oplus_r \mathtt{dup})^*]\!](\pi) = \delta_{\{\pi^n \mid n \geq 1\}}. \tag{6.1}$$

The argument is given in Appendix B.

Note that the equation in the statement of Theorem 1 does not determine $[\![p^*]\!]$ uniquely. For example, it can be shown that a probability measure μ is a solution of

$$[\![\mathtt{skip}^*]\!](\pi) = [\![\mathtt{skip} \mathbin{\&} \mathtt{skip} \mathbin{;} \mathtt{skip}^*]\!](\pi)$$

if and only if $\mu(B_\pi) = 1$. That is, π appears in the output set of $[\![\mathtt{skip}^*]\!](\pi)$ with probability 1.

6.3 A Continuous Measure

Without the Kleene star operator or dup, a ProbNetKAT program can generate only a discrete measure. This raises the question of whether it is possible to generate a continuous measure at all, even in the presence of * and dup. This question is important, because with only discrete measures, we would have no need for measure theory or integrals and the semantics would be significantly simpler. It turns out that the answer to this question is yes, it is possible to generate a continuous measure, therefore discrete measures do not suffice.

To see why, let 0 and 1 be distinct packets and let p be the program that changes the current packet to either 0 or 1 with equal probability. Now consider the following program:

$$p \mathbin{;} (\mathtt{dup} \mathbin{;} p)^*.$$

Operationally, this program first sets the input packet to either 0 or 1 with equal probability, then repeats the following three steps forever:

1. output the current packet,
2. duplicate the current packet, and
3. set the new current packet to 0 or 1 with equal probability.

This procedure produces outcomes a with exactly one packet history of every length and linearly ordered by the suffix relation. Thus each possible outcome a corresponds to a complete path in an infinite binary tree. Moreover, the probability that a history τ is generated is $2^{-|\tau|}$, thus any particular set is generated with probability 0, because the probability that a set is generated cannot be greater than the probability that any one of its elements is generated.

Theorem 3. *Let μ be the measure $[\![p \mathbin{;} (\mathtt{dup} \mathbin{;} p)^*]\!](0)$.*

(i) *For $\tau \in H$, the probability that τ is a member of the output set is $2^{-|\tau|}$.*

(ii) *Two packet histories of the same length are generated with probability 0.*

(iii) *$\mu(\{a\}) = 0$ for all $a \in 2^H$, thus μ is a continuous measure.*

A complete proof is given in Appendix B.

In fact, the measure μ is the uniform measure on the subspace of 2^H consisting of all sets that contain exactly one packet history of each length and are linearly ordered by the suffix relation. This subspace is homeomorphic to the Cantor space.

6.4 Uncorrelated Semantics

It is tempting to consider a weaker *uncorrelated semantics*

$$[p] : 2^H \to [0, 1]^H$$

in which $[p](a)(\tau)$ gives the probability that τ is contained in the output set on input a. Indeed, this semantics can be obtained from the standard ProbNetKAT semantics as follows:

$$[p](a)(\tau) \triangleq [\![p]\!](a)(B_\tau).$$

However, although it is simpler in that it does not require continuous measures, one loses correlation between packets.

Worse, it is not compositional, as the following example shows. Let π_0, π_1 be two packets and consider the programs $\pi_0! \oplus \pi_1!$ and $(\pi_0! \mathbin{\&} \pi_1!) \oplus \mathtt{drop}$, where $\pi!$ is the program that sets the current packet to π. Both programs have the same uncorrelated meaning:

$$[\pi_0! \oplus \pi_1!](a)(\pi) = [(\pi_0! \mathbin{\&} \pi_1!) \oplus \mathtt{drop}](a)(\pi) = \tfrac{1}{2}$$

for $\pi \in \{\pi_0, \pi_1\}$ and $a \neq \varnothing$ and 0 otherwise. However, their standard meanings differ:

$$\begin{aligned} [\![\pi_0! \oplus \pi_1!]\!](a) &= \tfrac{1}{2}\delta_{\{\pi_0\}} + \tfrac{1}{2}\delta_{\{\pi_1\}} \\ [\![(\pi_0! \mathbin{\&} \pi_1!) \oplus \mathtt{drop}]\!](a) &= \tfrac{1}{2}\delta_{\{\pi_0, \pi_1\}} + \tfrac{1}{2}\delta_{\varnothing}, \end{aligned}$$

Moreover, composing on the right with $\pi_0!$ yields $\delta_{\{\pi_0\}}$ and $\frac{1}{2}\delta_{\{\pi_0\}} + \frac{1}{2}\delta_{\varnothing}$, respectively, which have different uncorrelated meanings as well. Thus we have no choice but to reject the uncorrelated semantics as a viable alternative.

7. Applications

In this section, we demonstrate the expressiveness of ProbNetKAT's probabilistic operators and power of its semantics by presenting three case studies drawn from scenarios that commonly arise in real-world networks. Specifically, we show how ProbNetKAT can be used to model and analyze expected delivery in the presence of failures, expected congestion with randomized routing schemes, and expected convergence with gossip protocols. To the best of our knowledge, ProbNetKAT is the first high-level SDN language and reasoning framework that adequately handles these and other examples involving probabilistic behavior.

7.1 Fault Tolerance

Failures are a fact of life in real-world networks. Devices and links fail due to factors ranging from software and hardware bugs to interference from the environment such as loss of power or cables being severed. A recent empirical study of data center networks by Gill et al. [16] found that failures occur frequently and can cause issues ranging from degraded performance to service disruptions. Hence, it it important for network operators to be able to understand the impact of failures—e.g., they may elect to use routing schemes that divide traffic over many diverse paths in order to minimize the impact of any given failure.

We can encode failures in ProbNetKAT using random choice and drop: the idiom $p \oplus_d \mathtt{drop}$ encodes a program that succeeds and executes p with probability d, or fails and executes drop with probability $1 - d$. Note that since drop produces no packets, it accurately models a device or link that has crashed. We can then compute the probability that traffic will be delivered under an arbitrary forwarding scheme.

As a concrete example, consider the topology depicted in Figure 3 (a), with four switches connected in a diamond. Suppose that we wish to forward traffic from S_1 to S_2 and we know that the link between S_1 and S_4 fails with 10% probability (for simplicity, in this example, we will assume that the switches and all other links are reliable). What is the probability that a packet that originates at S_1 will be successfully delivered to S_4, as desired?

Obviously the answer to this question depends on the configuration of the network—using different forwarding paths will lead to different outcomes! To investigate this question, we will encode the overall behavior of the network using several terms: a term p

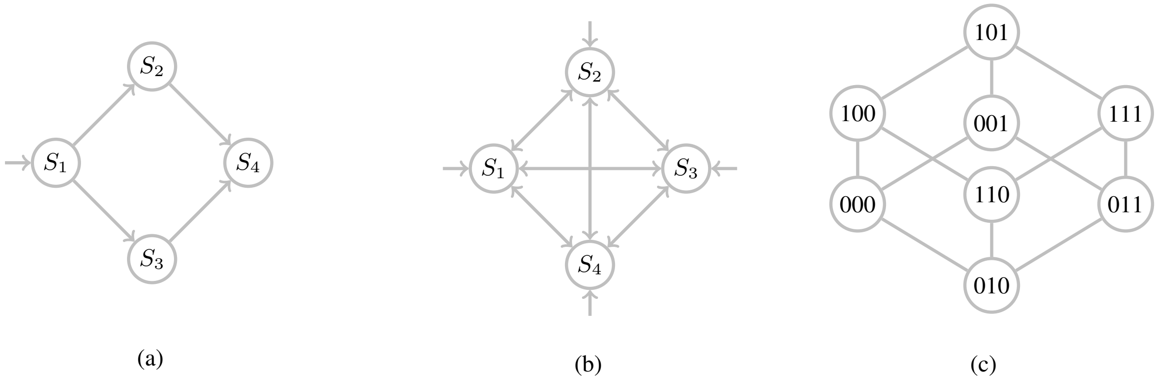

Figure 3. Topologies used in case studies: (a) fault tolerance, (b) load balancing, and (c) gossip protocols.

that encodes the local forwarding behavior of the switches; a term t that encodes the forwarding behavior of the network topology; and a term e that encodes the network egresses.

The standard way to model a link ℓ is as the sequential composition of terms that (i) test the location (i.e., switch and port) at one end of the link; (ii) duplicate the head packet, and (iii) update the location to the other end of the link. However, because we are only concerned with end-to-end packet delivery in this example, we can safely elide the dup term. Hence, using the idiom discussed above, we would model a link ℓ that fails with probability $1 - d$ as $\ell \oplus_d \texttt{drop}$. Hence, since there is a 10% probability of failure of the link $S_1 \to S_2$, we encode the topology t as follows:

$$\begin{aligned}
t \triangleq &(sw = S_1; pt = 2; ((sw \leftarrow S_2; pt \leftarrow 1) \oplus_{.9} \texttt{drop}))\\
&\&\ (sw = S_1; pt = 3; sw \leftarrow S_3; pt \leftarrow 1)\\
&\&\ (sw = S_2; pt = 4; sw \leftarrow S_4; pt \leftarrow 2)\\
&\&\ (sw = S_3; pt = 4; sw \leftarrow S_4; pt \leftarrow 3).
\end{aligned}$$

We adopt the convention that each port is named according to the identifier of the switch it connects to—e.g., port 1 on switch S_2 connects to switch S_1.

Next, we define the local forwarding policy p that encodes the behavior on switches. Suppose that we forward traffic from S_1 to S_4 via S_2. Then p would be defined as follows:

$$p \triangleq (sw = S_1; pt \leftarrow 2)\ \&\ (sw = S_2; pt \leftarrow 4).$$

Finally, the egress predicate e is simply:

$$e \triangleq sw = S_4.$$

The complete network program is then $(p; t)^*; e$. That is, the network alternates between forwarding on switches and topology, iterating these steps until the packet is either dropped or exits the network.

Using our semantics for ProbNetKAT, we can evaluate this program on a packet starting at S_1: we obtain a distribution in which there is a 90% chance that the packet is delivered to S_4 and a 10% chance that it is dropped.

As an extension to the example, we can model a more fault-tolerant forwarding scheme that divides traffic across multiple paths to reduce the impact of any single failure. The following program p' divides traffic from S_1 evenly between S_2 and S_3:

$$\begin{aligned}
p' \triangleq &(sw = S_1; (pt \leftarrow 2 \oplus pt \leftarrow 3))\\
&\&\ (sw = S_2; pt \leftarrow 4)\\
&\&\ (sw = S_3; pt \leftarrow 4).
\end{aligned}$$

As expected, evaluating this policy on a packet starting at S_1 gives us a 95% chance that the packet is delivered to S_4 and only a 5% chance that it is dropped. The positive effect with respect to failures has also been observed in previous work on randomized routing [62].

7.2 Load Balancing.

In many networks, operators must balance demands for traffic while optimizing for various criteria such as minimizing the maximum amount of congestion on any given link. An attractive approach to these traffic engineering problems is to use routing schemes based on randomization: the operator computes a collection of paths that utilize the full capacity of the network and then maps incoming traffic flows onto those paths randomly. By spreading traffic over a diverse set of paths, such schemes ensure that (in expectation) the traffic will closely approximate the optimal solution, even though they only require a static set of paths in the core of the network.

Valiant load balancing (VLB) [58] is a classic randomized routing scheme that provides low expected congestion for any traffic demands in a mesh topology. VLB forwards packets using a simple two-phase strategy: in the first phase, the ingress switch forwards the packet to a randomly selected neighbor, without considering the the packet's ultimate destination; in the second phase, the neighbor forwards the packet to the egress switch that is connected to the destination.

As an example, consider the four-node mesh topology shown in Figure 3 (b). When a packet destined for a host connected to S_3 arrives at S_1, the switch will first pick one of S_2, S_3, or S_4 as the intermediate hop. Suppose it picks S_4. When S_4 receives the packet, it forwards the packet directly to S_3, which will in turn forwards it along to the destination host.

We assume that each switch has ports named $1, 2, 3, 4$, that port i on switch i connects to the outside world, and that all other ports j connect to switch j. We can write a ProbNetKAT program for this load balancing scheme by splitting it into two parts, one for each phase of routing.

VLB often requires that traffic be tagged in each phase so that switches know when to forward it randomly or deterministically, but in this example, we can use topological information to distinguish the phases. Packets coming in from the outside (port i on switch i) are forwarded randomly, and packets on internal ports are forwarded deterministically.

We model the initial (random) phase with a term p_1:

$$p_1 \triangleq \mathop{\Large\&}_{k=1}^{4} (sw = k; pt = k; \bigoplus_{j \neq k} pt \leftarrow j).$$

Here we tacitly use an n-ary version of $\oplus$ that chooses each each summand with equal probability.

Similarly, we can model the second (deterministic) phase with a term p_2:

$$p_2 \triangleq \left(\mathop{\&}_{k=1}^{4}(sw = k; pt \neq k)\right); \left(\mathop{\&}_{k=1}^{4}(dst = k; pt \leftarrow k)\right)$$

Note that the guards $sw = k; pt \neq k$ restrict to second-phase packets. The overall switch term p is simply $p_1 \ \& \ p_2$.

The topology term t is encoded with $\mathtt{dup}$ terms to record the paths, as described in §7.1.

The power of VLB is its ability to route $nr/2$ load in a network with n switches and internal links with capacity r. In our example, $n = 4$ and r is 1 packet, so we can route 2 packets of random traffic with no expected congestion. We can model this demand with a term d that generates two packets with random origins and random destinations (writing $\pi_{i,j,k}$ for a sequence of assignments setting the switch to i, the port to j, and the identifier to k):

$$d \triangleq (\bigoplus_{k=1}^{4}(\pi_{k,k,0}!) \ \& \ \bigoplus_{k=1}^{4}(\pi_{k,k,1}!)); (\bigoplus_{k=1}^{4} dst \leftarrow k)$$

The full network program to analyze is then $d; (p; t)^*; p$. Even on this small example, the output of the program is too large to reasonably write down here. Instead, as in the congestion example from §2, we can define a random variable to extract the information we care about. Let $X_{\max}$ be a random variable equal to the maximum number of packets traversing a single internal link. Then the expected value of $X_{\max}$ is 1 packet. That is, there is no congestion.

7.3 Gossip Protocols

Gossip (or epidemic) protocols are randomized algorithms that are often used to efficiently disseminate information in large-scale distributed systems [11]. An attractive feature of gossip protocols and other epidemic algorithms is that they are able to rapidly converge to a consistent global state while only requiring bounded worst-case communication. Operationally, a gossip protocol proceeds in loosely synchronized rounds: in each round, every node communicates with a randomly selected peer and the nodes update their state using information shared during the exchange. For example, in a basic anti-entropy protocol, a "rumor" is injected into the system at a single node and spreads from node to node through pair-wise communication. In practice, such protocols can rapidly disseminate information in well-connected graphs with high probability.

We can use ProbNetKAT to model the convergence of gossip protocols. We introduce a single packet to model the "rumor" being gossiped by the system: when a node receives the packet, it randomly selects one of its neighbors to infect (by sending it the packet), and also sends a copy back to itself to maintain the infection. In gossip terminology, this would be characterized as a "push" protocol since information propagates from the node that initiates the communication to the recipient rather than the other way around.

We can make sure that nodes do not send out more than one infection packet per round by using a single incoming port (port 0) on each switch and exploiting ProbNetKAT's set semantics: because the infection packets are identical modulo topology information, multiple infection packets arriving at the same port are identified.

To simplify the ProbNetKAT program, we assume that the network topology is a hypercube, as shown in Figure 3 (c). The program for gossiping on a hypercube is highly uniform—assuming that switches are numbered in binary, we can randomly select a neighbor by flipping a single bit.

The fragment of the switch program p for switch 000 is as follows:

$$sw = 000; ((pt \leftarrow 001 \oplus pt \leftarrow 010 \oplus pt \leftarrow 100) \ \& \ pt \leftarrow 0).$$

This overall forwarding policy can be obtained by combining analogous fragments for the other switches using parallel composition.

Encoding the topology of the hypercube as t, we can then analyze $(p; t)^*$ and calculate the expected number of infected nodes after a given number of rounds X_{infected} using the ProbNetKAT semantics. The results for the first few rounds are as follows:

Rounds	$E[X_{\text{infected}}]$
0	1.00
1	2.00
2	3.33
3	4.86
4	6.25
5	7.17
6	7.66

The above examples, despite their simplicity, illustrate the expressiveness and versatility of ProbNetKAT. The ability to reason probabilistically is an important step to fully realize the vision of SDN and the framework presented in the paper provides strong semantic foundations to this end.

8. Related Work

Work related to ProbNetKAT can be divided into two categories: (i) models and semantics for proabilistic programs and (i) domain-specific frameworks for specifying and reasoning about network programs. This section summarizes the most relevant pieces of prior work in each of these categories.

8.1 Probabilistic Programming

Computational models and logics for probabilistic programming have been extensively studied for many years. Denotational and operational semantics for probabilistic while programs were first studied by Kozen [28]. Early logical systems for reasoning about probabilistic programs were proposed in a sequence of separate papers by Saheb-Djahromi, Ramshaw, and Kozen [29, 49, 53]. There are also numerous recent efforts [18, 19, 31, 34, 44, 50]. Our semantics for ProbNetKAT builds on the foundation developed in these papers and extends it to the new domain of network programming.

Probabilistic programming in the context of artificial intelligence has also been extensively studied in recent years [5, 17, 52]. However, the goals of this line of work are different than ours in that it focuses on codification of Bayesian inference.

Probabilistic automata in several forms have been a popular model going back to the early work of Paz [48], as well as many other recent efforts [3, 10, 38, 54, 55]. Probabilistic automata are a suitable operational model for probabilistic programs and play a crucial role in the development of decision procedures for bisimulation equivalence, logics to reason about behavior, in the synthesis of probabilistic programs, and in model checking procedures [1, 3, 7, 12, 24, 32, 33, 35, 36]. In the present paper, we do not touch upon any of these issues so the connections to probabilistic automata theory are thin. However, we expect they will play an important role in our future work—see below.

Denotational models combining probability and nondeterminism have been proposed in papers by several authors [22, 23, 39, 40, 57, 59], and general models for labeled Markov processes, primarily based on Markov kernels, have been studied extensively [4, 13, 46, 47]. Because ProbNetKAT does not have nondeterminism, we have not encountered the extra challenges arising in the combination of nondeterministic and probabilistic behavior.

All the above mentioned systems provide semantics and logical formalisms for specifying and reasoning about state-transition systems involving probabilistic choice. A crucial difference between our work and these efforts is in that our model is not really a state-transition model in the usual sense, but rather a packet-filtering model that filters, modifies, and forwards packets. Expressions denote functions that consume sets of packet histories as input and produce probability distributions of sets of packet histories as output. As demonstrated by our example applications, this view is appropriate for modeling the functionality of packet-switching networks. It has its own peculiarities and is different enough from standard state-based computation that previous semantic models in the literature do not immediately apply. Nevertheless, we have drawn much inspiration from the literature and exploited many similarities to provide a powerful formalism for modeling probabilistic behavior in packet-switching networks.

8.2 Network Programming

Recent years have seen an incredible growth of languages and systems for programming and reasoning about networks. Network programming languages such as Frenetic [14], Pyretic [43], Maple [60], NetKAT [2], and FlowLog [45] have introduced high-level abstractions and semantics that enable programmers to reason precisely about the behavior of networks. However, as mentioned previously, all of these language are based on deterministic packet-processing functions, and do not handle probabilistic traffic models or forwarding policies. Of all these frameworks, NetKAT is the most closely related as ProbNetKAT builds directly on its features.

In addition to programming languages, a number of network verification tools have been developed, including Header Space Analysis [25], VeriFlow [26], the NetKAT verifier [15], and Libra [61]. Similar to the network programming languages described above, these tools only model deterministic networks and verify deterministic properties.

Network calculus is a general framework for modeling and analyzing the network behavior using tools from queuing theory [9]. It models the low-level behavior of network devices in significant detail, including features such as traffic arrival rates, switch propagation delays, and the behaviors of components like buffers and queues. This enables reasoning about quantitative properties such as latency, bandwidth, congestion, etc. Past work on network calculus can be divided into two branches: deterministic [37] and stochastic [21]. Like ProbNetKAT, the stochastic branch of network calculus provides tools for reasoning about the probabilistic behavior, especially in the presence of statistical multiplexing. However, network calculus is generally known to be difficult to use, since it can require the use of external facts from queuing theory to establish many desired results. In contrast, ProbNetKAT is a self-contained, language-based framework that offers general programming constructs and a complete denotational semantics.

9. Conclusion

Previous work [2, 15] has described NetKAT, a language and logic for specifying and reasoning about the behavior of packet-switching networks. In this paper we have introduced ProbNetKAT, a conservative extension of NetKAT with constructs for reasoning about the probabilistic behavior of such networks. To our knowledge, this is the first language-based framework for specifying and verifying probabilistic network behavior. We have developed a formal semantics for ProbNetKAT based on Markov kernels and shown that the extension is conservative over NetKAT. Finally, we have presented several case studies that illustrate the use of ProbNetKAT on real-world examples.

Our examples have used the semantic definitions directly in the calculation of distributions, fault tolerance, load balancing, and a probabilistic gossip protocol. Although we have exploited several general properties of our system in these arguments, we have made no attempt to assemble them into a formal deductive system or decision procedure as was done previously for NetKAT [2, 15]. These questions remain topics for future investigation. We are hopeful that the coalgebraic perspective developed in [15] will be instrumental in obtaining a sound and complete axiomatization and a practical decision procedure for equivalence of ProbNetKAT expressions.

We are also fascinated by the question of the appropriate notion of convergence by which an iterate p^* is approximated by its partial computations p^n. The semantics of iteration is defined operationally by an infinite stochastic process and denotationally in terms of a product measure on the ω-power of the space of sets of packet histories. The definition is canonical and satisfies the desired fixpoint equation (Theorem 1), but the notion of approximation involves both the inclusion order on sets of packet histories and convergent probabilities and seems not to satisfy any natural order. Thus, standard order-theoretic domains for probabilistic programming that have been proposed in the literature seem inadequate for describing the present situation.

As a more practical next step, we would like to augment the existing NetKAT compiler [56] with tools for handling the probabilistic constructs of ProbNetKAT along with a formal proof of correctness. Features such as OpenFlow [41] "group tables" support for simple forms of randomization and emerging platforms such as P4 [6] offer additional flexibility. Hence, there already exist machine platforms that could serve as a compilation target for (restricted fragments of) ProbNetKAT.

Another interesting topic is whether we can learn ProbNetKAT programs from partial traces of a system, enabling active learning of running network policies. This is interesting for many applications. We are particularly interested in applications involving security and multiple administrative domains. For example, learning algorithms might be useful for detecting compromised nodes in a network. Alternatively, a network operator might use information from `traceroute` to learn a model that provides partial information about the paths from their own network to another autonomous system on the Internet.

References

[1] A. Abate, M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking of labelled Markov processes via finite approximate bisimulations. In *Horizons of the Mind – P. Panangaden Festschrift*, pages 40–58. Springer, 2014.

[2] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: Semantic foundations for networks. In *Proc. 41st ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'14)*, pages 113–126, San Diego, California, USA, January 2014. ACM.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[4] Richard Blute, Jose Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, pages 95–106, 1997.

[5] Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, and Jurgen Van Gael. Measure transformer semantics for Bayesian machine learning. In *European Symposium on Programming*. Springer Verlag, July 2011.

[6] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[7] S. Cattani and R. Segala. Decision algorithms for probabilistic bisimulation. In *CONCUR*, volume 2421 of *LNCS*, pages 371–385. Springer, 2002.

[8] K. L. Chung. *A Course in Probability Theory*. Academic Press, 2nd edition, 1974.

[9] R. Cruz. A calculus for network delay, parts I and II. *IEEE Transactions on Information Theory*, 37(1):114–141, January 1991.

[10] Benoit Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wasowski. Abstract probabilistic automata. In *VMCAI*, volume 6538 of *LNCS*, pages 324–339. Springer, 2011.

[11] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, New York, NY, USA, 1987. ACM.

[12] Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 179(2):163–193, December 2002.

[13] Ernst-Erich Doberkat. *Stochastic Relations: Foundations for Markov Transition Systems*. Studies in Informatics. Chapman Hall, 2007.

[14] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A network programming language. In *ICFP*, pages 279–291, September 2011.

[15] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In *Proc. 42nd ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'15)*, pages 343–355, Mumbai, India, January 2015. ACM.

[16] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *SIGCOMM*, pages 350–361, August 2011.

[17] Andrew D. Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver. Tabular: A schema-driven probabilistic programming language. Technical Report MSR-TR-2013-118, Microsoft Research, December 2013.

[18] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *International Conference on Software Engineering (ICSE Future of Software Engineering)*. IEEE, May 2014.

[19] Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Annabelle McIver, and Federico Olmedo. Conditioning in probabilistic programming. *CoRR*, abs/1504.00198, 2015.

[20] P. R. Halmos. *Measure Theory*. Van Nostrand, 1950.

[21] Y. Jiang. A basic stochastic network calculus. In *SIGCOMM*, pages 123–134, September 2006.

[22] C. Jones. *Probabilistic Nondeterminism*. PhD thesis, Edinburgh University, 1990.

[23] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Proc. 4th IEEE Symp. Logic in Computer Science (LICS'89)*, pages 186–195. IEEE, 1989.

[24] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *LICS*, pages 266–277. IEEE, 1991.

[25] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *NSDI*, 2012.

[26] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: Verifying network-wide invariants in real time. In *NSDI*, 2013.

[27] A. N. Kolmogorov and S. V. Fomin. *Introductory Real Analysis*. Prentice Hall, 1970.

[28] Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22:328–350, 1981.

[29] Dexter Kozen. A probabilistic *PDL. J. Comput. Syst. Sci.*, 30(2):162–178, April 1985.

[30] Dexter Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.

[31] Dexter Kozen, Radu Mardare, and Prakash Panangaden. Strong completeness for Markovian logics. In Krishnendu Chatterjee and Jirí Sgall, editors, *Proc. 38th Symp. Mathematical Foundations of Computer Science (MFCS 2013)*, volume 8087 of *Lect. Notes in Computer Science*, pages 655–666, Klosterneuburg, Austria, August 2013. Springer.

[32] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, June 2002.

[33] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, 2007.

[34] Kim G. Larsen, Radu Mardare, and Prakash Panangaden. Taking it to the limit: Approximate reasoning for Markov processes. In *Mathematical Foundations of Computer Science*, 2012.

[35] Kim G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:456–471, 1991.

[36] Kim Guldstrand Larsen and Arne Skou. Compositional verification of probabilistic processes. In *CONCUR*, pages 456–471, 1992.

[37] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[38] A. K. McIver, E. Cohen, C. Morgan, and C. Gonzalia. Using probabilistic Kleene algebra pKA for protocol verification. *J. Logic and Algebraic Programming*, 76(1):90–111, 2008.

[39] A. K. McIver and C. C. Morgan. Probabilistic predicate transformers: Part 2. Technical Report PRG-TR-5-96, Programming Research Group, Oxford University, 1996.

[40] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.

[41] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, 2008.

[42] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. In *SIGCOMM*, pages 161–174, August 2002.

[43] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software defined networks. In *NSDI*, April 2013.

[44] Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.

[45] Tim Nelson, Andrew D. Ferguson, Michael J. G. Scheer, and Shriram Krishnamurthi. Tierless programming and reasoning for software-defined networks. In *NSDI*, 2014.

[46] Prakash Panangaden. Probabilistic relations. In *School of Computer Science, McGill University, Montreal*, pages 59–74, 1998.

[47] Prakash Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009.

[48] A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.

[49] L. H. Ramshaw. *Formalizing the Analysis of Algorithms*. PhD thesis, Stanford University, 1979.

[50] Robert Rand and Steve Zdancewic. VPHL: A verified partial-correctness logic for probabilistic programs. In *MFPS 2015*, 2015.

[51] M. M. Rao. *Measure Theory and Integration*. Wiley-Interscience, 1987.

[52] Daniel M. Roy. *Computability, inference and modeling in probabilistic programming*. PhD thesis, Massachusetts Institute of Technology, 2011.

[53] N. Saheb-Djahromi. Probabilistic LCF. In *Mathematical Foundations of Computer Science*, volume 64 of *LNCS*, pages 442–451. Springer, May 1978.

[54] R. Segala. Probability and nondeterminism in operational models of concurrency. In *CONCUR*, volume 4137 of *LNCS*, pages 64–78. Springer, 2006.

[55] R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. In *NJC*, volume 2, pages 250–273, 1995.

[56] Steffen Smolka, Spiridon Eliopoulos, Nate Foster, and Arjun Guha. A fast compiler for NetKAT. In *ACM SIGPLAN International Conference on Functional Programming (ICFP)*, Vancouver, BC, September 2015.

[57] R. Tix, K. Keimel, and G. Plotkin. Semantic domains for combining probability and nondeterminism. *Electronic Notes in Theoretical Computer Science*, 222:3–99, 2009.

[58] L. Valiant. A Scheme for Fast Parallel Communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.

[59] D. Varacca and G. Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006.

[60] Andreas Voellmy, Junchang Wang, Y. Richard Yang, Bryan Ford, and Paul Hudak. Maple: Simplifying SDN programming using algorithmic policies. In *SIGCOMM*, 2013.

[61] Hongyi Zeng, Shidong Zhang, Fei Ye, Vimalkumar Jeyakumar, Mickey Ju, Junda Liu, Nick McKeown, and Amin Vahdat. Libra: Divide and conquer to verify forwarding tables in huge networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, April 2014.

[62] Rui Zhang-Shen and Nick McKeown. Designing a predictable Internet backbone with Valiant load-balancing. In *International Workshop on Quality of Service (IWQoS)*, pages 178–192, 2005.

Appendix

A. Semantics of Iteration via the Kolmogorov Extension Theorem

The Kolmogorov extension theorem identifies conditions under which a family of measures on finite subproducts of an infinite product space extend to a measure on the whole space. This theorem can be used to create a sample space for an iterative process when the behavior of each individual step of the process is known.

For our application, it is convenient to have a slightly more general version that applies to countable chains of spaces connected by Markov kernels. In this section we formulate this version and use it to justify our treatment of $[\![p^*]\!]$ in §A.1.

First, a few definitions. A topological space is *separable* if it contains a countable dense subset—i.e., there is a sequence $(x_i)_{i<\omega}$ of elements such that every nonempty open set contains at least one element of the sequence. A space X is *metrizable* if there is a metric $d : X \times X \to \mathbb{R}$ that induces the topology. A metric space (X, d) is called *complete* if every Cauchy sequence in X has a limit that is also in X. Finally, a topological space is said to be *completely metrizable* if there is a metric d such that (X, d) is complete and d induces the topology. A *Polish space* is a separable completely metrizable topological space.

Suppose we have a sequence of Polish spaces $(S_n, \mathcal{B}_n)$, $n \in \omega$ along with measurable functions $f_{nm} : S_n \to S_m$ for $m \leq n$ such that for all $k \leq m \leq n$,

$$f_{nk} = f_{mk} \circ f_{nm} \qquad\qquad f_{nn} = 1_{S_n}. \tag{A.1}$$

This sequence has a limit $(S_\omega, \mathcal{B}_\omega)$ in the category of measurable spaces and measurable functions, where

$$S_\omega = \{(s_n \mid n \in \omega) \in \prod_{n\in\omega} S_n \mid \forall m \leq n \; f_{nm}(s_n) = s_m\}$$

and $\mathcal{B}_\omega$ is the weakest σ-algebra on S_ω such that all projections $\pi_m : S_\omega \to S_m$ are measurable. The space S_ω, being a closed subspace of a countable product of Polish spaces, is itself a Polish space.

Now suppose that we have Markov kernels $P_{kn} : S_k \to S_n$ for each $k, n < \omega$ such that for all $k, m, n < \omega$,

$$P_{kn}(s, A) = \int_{t\in S_m} P_{mn}(t, A) \cdot P_{km}(s, dt) \tag{A.2}$$

$$P_{kn}(s) = \delta_{f_{kn}(s)}, \; n \leq k. \tag{A.3}$$

In particular, $P_{nn}(s) = \delta_s$.

The following local consistency condition corresponds to the premise needed to apply the Kolmogorov extension theorem (see [8, Theorem 3.3.6]).

Lemma 5. *For all k, m, n with $m \leq n$,*

$$P_{km}(s, A) = P_{kn}(s, f_{nm}^{-1}(A)).$$

Proof. Starting from the left-hand side and using the change-of-variable rule (3.1) at the crucial step,

$$\begin{aligned} P_{km}(s, A) &= \int_{t\in S_n} P_{nm}(t, A) \cdot P_{kn}(s, dt) \\ &= \int_{t\in S_n} \chi_A(f_{nm}(t)) \cdot P_{kn}(s, dt) \\ &= \int_{u\in S_m} \chi_A(u) \cdot P_{kn}(s, f_{nm}^{-1}(du)) \\ &= \int_{u\in A} P_{kn}(s, f_{nm}^{-1}(du)) \\ &= P_{kn}(s, f_{nm}^{-1}(A)). \end{aligned}$$

□

There is another condition needed for the application of the Kolmogorov extension theorem, namely *inner regularity*. This is automatically satisfied because S_ω is a Polish space; see [51, Theorem 2.3.10].

Let $\mathcal{R}$ be the set of finite Boolean combinations of measurable sets $\pi_m^{-1}(A_m)$ for $A_m \in \mathcal{B}_m$. By the monotone class theorem (see [8, Theorem 2.1.2] or [20, Theorem 6.A]), the σ-algebra $\mathcal{B}_\omega$ is the smallest set containing $\mathcal{R}$ and closed under unions of countable ascending chains and intersections of countable descending chains.

Lemma 6. *Every element of $\mathcal{R}$ is of the form $\pi_n^{-1}(A_n)$ for sufficiently large $n \in \omega$ and some $A_n \in \mathcal{B}_n$. Moreover, for all sufficiently large m, n with $m \leq n$, we can take $A_n = f_{nm}^{-1}(A_m)$.*

Proof. Every finite Boolean combination $B(\pi_m^{-1}(A_m) \mid m \in F)$ depends on only finitely many generators $\pi_m^{-1}(A_m)$ for $m \in F$, where F is a finite set of indices. But for any $n \geq \max F$,

$$\begin{aligned} B(\pi_m^{-1}(A_m) \mid m \in F) &= B(\pi_n^{-1}(f_{nm}^{-1}(A_m)) \mid m \in F) \\ &= \pi_n^{-1}(B(f_{nm}^{-1}(A_m) \mid m \in F)), \end{aligned}$$

and $B(f_{nm}^{-1}(A_m) \mid m \in F) \in \mathcal{B}_n$. For the last statement, if $A_n = f_{nm}^{-1}(A_m)$, then

$$\pi_n^{-1}(A_n) = \pi_n^{-1}(f_{nm}^{-1}(A_m)) = \pi_m^{-1}(A_m).$$

□

Now for each $\pi_m^{-1}(A_m) \in \mathcal{R}$, define

$$P_{n\omega}(s, \pi_m^{-1}(A_m)) \triangleq P_{nm}(s, A_m). \tag{A.4}$$

We must argue that $P_{n\omega}$ is well defined. If $\pi_m^{-1}(A_m) = \pi_k^{-1}(A_k)$ with $m \leq k$, then for any $s \in S_\omega$,

$$\begin{aligned} \pi_k(s) \in A_k &\Leftrightarrow s \in \pi_k^{-1}(A_k) \Leftrightarrow s \in \pi_m^{-1}(A_m) \Leftrightarrow \pi_m(s) \in A_m \\ &\Leftrightarrow f_{km}(\pi_k(s)) \in A_m \Leftrightarrow \pi_k(s) \in f_{km}^{-1}(A_m). \end{aligned}$$

As the π_k are surjective (we can discard any element of S_k not appearing as a component of any element of S_ω), we have that $A_k = f_{km}^{-1}(A_m)$. Then

$$\begin{aligned} P_{nk}(s, A_k) &= P_{nk}(s, f_{km}^{-1}(A_m)) \\ &= (P_{nk}\,; P_{km})(s, A_m) = P_{nm}(s, A_m). \end{aligned}$$

Theorem 4. *The map $P_{n\omega} : S_n \times \mathcal{R} \to \mathbb{R}$ extends to a Markov kernel $P_{n\omega} : S_n \to S_\omega$.*

Proof. We must show:

(i) For fixed $s \in S_n$, the map $\lambda A.P_{n\omega}(s, A) : \mathcal{R} \to \mathbb{R}$ extends to a measure $\lambda A.P_{n\omega}(s, A) : \mathcal{B}_\omega \to \mathbb{R}$.

(ii) For fixed $A \in \mathcal{B}_\omega$, the map $\lambda s.P_{n\omega}(s, A) : S_n \to \mathbb{R}$ is a measurable function.

For (i), using inner regularity one can show that for fixed $s \in S_n$, the map $\lambda A.P_{n\omega}(s, A) : \mathcal{R} \to \mathbb{R}$ is countably additive on $\mathcal{R}$, therefore by the Carathéodory extension theorem (see [20, Theorem 13.A] or [27, Theorem 7.27.7]) extends to a measure $\lambda A.P_{n\omega}(s, A) : S_\omega \to \mathbb{R}$. This is essentially the Kolmogorov extension theorem in this setting.

For (ii), the proof is by induction. The basis is (A.4). For the induction step, we use the monotone class theorem and the fact that the pointwise supremum of a countable ascending chain of uniformly bounded measurable functions is measurable. For a chain $A_0 \subseteq A_1 \subseteq \cdots$, we know that the functions $\lambda s.P_{n\omega}(s, A_i)$ are measurable by the inductive hypothesis, and $\lambda s.P_{n\omega}(s, \bigcup_i A_i)$

is the pointwise supremum of the $\lambda s.P_{n\omega}(s,A_i)$, therefore measurable. The argument for intersections of countable descending chains is similar. □

A.1 Definition of $[\![p^*]\!]$

In this section we apply Theorem 4 to obtain the semantics of p^* for a ProbNetKAT program p. Suppose we have determined the semantics of p as a Markov kernel $[\![p]\!] : 2^H \to 2^H$ and we wish to define $[\![p^*]\!] : 2^H \to 2^H$. Let $(S_n, \mathcal{B}_n)$ be the product space $(2^H)^n$. For $n \geq m$, let $f_{nm} : S_n \to S_m$ be the projection onto the first m components: $f_{nm}(a_0,\ldots,a_{n-1}) = (a_0,\ldots,a_{m-1})$. For $n \geq m$, the Markov kernels $P_{mn} : S_m \to S_n$ are the maps that extend $(a_0,\ldots,a_{m-1})$ to $(a_0,\ldots,a_{n-1})$ by choosing $a_m,\ldots,a_{n-1}$ successively according to $[\![p]\!]$; that is,

$$P_{m,m+1}(a_0,\ldots,a_{m-1},(2^H)^{m-1}\times A) = [\![p]\!](a_{m-1},A).$$

By Theorem 4, the P_{nm} give rise to Markov kernels $P_{n\omega} : (2^H)^n \to (2^H)^\omega$.

Now consider the following infinite process:

1. Start with a given initial set $a_0 \in 2^H$.
2. At stage $n+1$, having constructed $a_0,\ldots,a_n$, sample $[\![p]\!](a_n)$ to obtain a_{n+1}.

The outcome after the first n steps of the process is a sequence $a_0,\ldots,a_n$. For $A \in \mathcal{B}_{n+1}$, the probability that this sequence lies in A is $P_{0,n+1}(a_0,A)$. After ω steps, the outcome is an infinite sequence $a_0,a_1,\ldots \in (2^H)^\omega$, and the probability of event $A \in \mathcal{B}_\omega$ is $P_{0\omega}(a_0,A)$.

We can now compose this process with the deterministic process $\bigcup : (2^H)^\omega \to 2^H$ that takes the union of a countable sequence of sets. This is the result of the process $[\![p^*]\!](a_0)$. Formally, for $A \in \mathcal{B}$,

$$\begin{aligned}[\![p^*]\!](a_0,A) &\triangleq \int_{s\in(2^H)^\omega} \chi_A(\textstyle\bigcup s)\cdot P_{0\omega}(a_0,ds)\\ &= \int_{a\in 2^H} \chi_A(a)\cdot P_{0\omega}(a_0,\textstyle\bigcup^{-1}(da))\\ &= \int_{a\in A} P_{0\omega}(a_0,\textstyle\bigcup^{-1}(da)) = P_{0\omega}(a_0,\textstyle\bigcup^{-1}(A)).\end{aligned}$$

A.2 Colimit Construction

In fact, more can be said. Recall that a Markov kernel $P : S \to T$ is *deterministic* iff for every $s \in S$, there is a Borel measurable function $f : S \to T$ such that

$$P(s,A) = \delta_{f(s)}(A) = \delta_s(f^{-1}(A)) = \chi_A(f(s)).$$

Let us call a Markov kernel $P : S \to T$ *reversible* if it has a deterministic right[2] inverse $f : T \to S$; thus

$$\delta_s(A) = (P\,;f)(s,A) = P(s,f^{-1}(A)).$$

The measurable spaces and reversible Markov kernels form a subcategory of the Kleisli category of the Giry monad.

Theorem 5. *The space $(S_\omega,\mathcal{B}_\omega)$ is the colimit of the $(S_n,\mathcal{B}_n)$ with coprojections $P_{n\omega} : S_n \to S_\omega$ in the category of measurable spaces and reversible Markov kernels.*

Proof. First, we show that the $P_{n\omega}$ commute with the P_{nm} in the sense that $P_{n\omega} = P_{nm};P_{m\omega}$. It suffices to consider their behavior on generators $\pi_k^{-1}(A)$, that is,

$$P_{n\omega}(s,\pi_k^{-1}(A)) = \int_{t\in S_m} P_{nm}(s,dt)\cdot P_{m\omega}(t,\pi_k^{-1}(A)).$$

By (A.4), we wish to show

$$P_{nk}(s,A) = \int_{t\in S_m} P_{nm}(s,dt)\cdot P_{mk}(t,A).$$

But this is immediate from (A.3). Moreover, $P_{m\omega}$ is reversible with right inverse π_m:

$$(P_{m\omega}\,;\pi_m)(s,A) = P_{m\omega}(s,\pi_m^{-1}(A)) = P_{mm}(s,A) = \delta_s(A).$$

To show the universality of the construction, let $(T,\mathcal{B}_T)$ be any measurable space with reversible Markov kernels $Q_n : S_n \to T$, each with a deterministic right inverse $g_n : T \to S_n$ such that $Q_m = P_{mn}\,;Q_n$ for all m,n. In particular, $g_m = g_n\,;f_{nm}$ for all $m < n$. Since $(S_\omega,\mathcal{B}_\omega)$ is the limit of the $(S_n,\mathcal{B}_n)$ in the category of measurable spaces and measurable functions, there is a measurable function $g : T \to S_\omega$ such that $g_n = g\,;\pi_n$ for all n. Now define the Markov kernel $Q_\omega : S_\omega \to T$ by

$$Q_\omega = \pi_n\,;Q_n.$$

The choice of n does not matter: for $m \leq n$,

$$\pi_m\,;Q_m = \pi_n\,;f_{nm}\,;Q_m = \pi_n\,;P_{nm}\,;Q_m = \pi_n\,;Q_n.$$

The kernel Q_ω is reversible with right inverse g:

$$Q_\omega\,;g\,;\pi_n = \pi_n\,;Q_n\,;g\,;\pi_n = \pi_n\,;Q_n\,;g_n = \pi_n\,;1_{S_n} = \pi_n,$$

and by the universality of the product,

$$Q_\omega\,;g = 1_{S_\omega}.$$

Finally, the Q_n factor through S_ω via the universal arrow Q_ω:

$$Q_n = 1_{S_n}\,;Q_n = P_{n\omega}\,;\pi_n\,;Q_n = P_{n\omega}\,;Q_\omega.$$

□

B. Omitted Proofs

Proof of Lemma 1. Associativity and commutativity are clear from (5.1). For (ii), let $A \times B$ be a measurable rectangle. We have

$$\begin{aligned}((a\mu+b\nu)\times\xi)(A\times B) &= (a\mu+b\nu)(A)\cdot\xi(B)\\ &= a\mu(A)\cdot\xi(B)+b\nu(A)\cdot\xi(B)\\ &= a(\mu\times\xi)(A\times B)+b(\nu\times\xi)(A\times B)\\ &= (a(\mu\times\xi)+b(\nu\times\xi))(A\times B),\end{aligned}$$

thus

$$(a\mu+b\nu)\times\xi = a(\mu\times\xi)+b(\nu\times\xi). \qquad \text{(B.1)}$$

Then for any C,

$$\begin{aligned}&((a\mu+b\nu)\ \&\ \xi)(C)\\ &= ((a\mu+b\nu)\times\xi)(\{(a,b)\mid a\cup b\in C\})\\ &= (a(\mu\times\xi)+b(\nu\times\xi))(\{(a,b)\mid a\cup b\in C\}) \quad \text{by B.1}\\ &= a(\mu\times\xi)(\{(a,b)\mid a\cup b\in C\})+b(\nu\times\xi)(\{(a,b)\mid a\cup b\in C\})\\ &= a(\mu\ \&\ \xi)(C)+b(\nu\ \&\ \xi)(C)\\ &= (a(\mu\ \&\ \xi)+b(\nu\ \&\ \xi))(C).\end{aligned}$$

For (iii), since

$$\begin{aligned}&(\delta_a\times\mu)(\{(b,c)\mid b\cup c\in A\}\cap(\sim\{a\}\times 2^H))\\ &\leq (\delta_a\times\mu)(\sim\{a\}\times 2^H) = \delta_a(\sim\{a\})\mu(2^H) = 0,\end{aligned}$$

[2] in diagrammatic order

we have

$$\begin{aligned}(\delta_a \mathbin{\&} \mu)(A) &= (\delta_a \times \mu)(\{(b,c) \mid b \cup c \in A\})\\ &= (\delta_a \times \mu)(\{(b,c) \mid b \cup c \in A\} \cap (\{a\} \times 2^H))\\ &\quad + (\delta_a \times \mu)(\{(b,c) \mid b \cup c \in A\} \cap (\sim\{a\} \times 2^H))\\ &= (\delta_a \times \mu)(\{(b,c) \mid b \cup c \in A\} \cap (\{a\} \times 2^H))\\ &= (\delta_a \times \mu)(\{a\} \times \{c \mid a \cup c \in A\})\\ &= \delta_a(\{a\})\mu(\{c \mid a \cup c \in A\})\\ &= \mu(\{c \mid a \cup c \in A\}).\end{aligned}$$

Properties (iv) and (v) follow directly from (iii).

Finally, for (vi), $\delta_a \mathbin{\&} \delta_a = \delta_a$ is immediate from (iv). Now suppose $\mu \mathbin{\&} \mu = \mu$. For any μ and ν, we have

$$\begin{aligned}(\mu \mathbin{\&} \nu)(\sim B_\tau) &= (\mu \times \nu)(\{(a,b) \mid a \cup b \in \sim B_\tau\})\\ &= (\mu \times \nu)(\{(a,b) \mid \tau \notin a \cup b\})\\ &= (\mu \times \nu)(\{(a,b) \mid a \in \sim B_\tau, b \in \sim B_\tau\})\\ &= (\mu \times \nu)(\sim B_\tau \times \sim B_\tau)\\ &= \mu(\sim B_\tau) \cdot \nu(\sim B_\tau),\end{aligned}$$

therefore $(\mu \mathbin{\&} \mu)(\sim B_\tau) = \mu(\sim B_\tau)^2$, and this equals $\mu(\sim B_\tau)$ iff $\mu(\sim B_\tau) \in \{0, 1\}$. Since $\mu(B_\tau) = 1 - \mu(\sim B_\tau)$, it must be that exactly one of $\mu(B_\tau)$ and $\mu(\sim B_\tau)$ is 1 and the other is 0. Let $a = \{\tau \mid \mu(B_\tau) = 1\}$. Then

$$\begin{aligned}a \in B_\tau &\Leftrightarrow \tau \in a \Leftrightarrow \mu(B_\tau) = 1\\ a \in \sim B_\tau &\Leftrightarrow \tau \notin a \Leftrightarrow \mu(B_\tau) \neq 1 \Leftrightarrow \mu(\sim B_\tau) = 1,\end{aligned}$$

so

$$\begin{aligned}\{a\} &= \bigcap\{B_\tau \mid a \in B_\tau\} \cap \bigcap\{\sim B_\tau \mid a \in \sim B_\tau\}\\ &= \bigcap\{B_\tau \mid \mu(B_\tau) = 1\} \cap \bigcap\{\sim B_\tau \mid \mu(\sim B_\tau) = 1\}\\ \mu(\{a\}) &= \mu(\bigcap\{B_\tau \mid \mu(B_\tau) = 1\} \cap \bigcap\{\sim B_\tau \mid \mu(\sim B_\tau) = 1\})\\ &= 1,\end{aligned}$$

therefore $\mu = \delta_a$. □

Proof of Lemma 2. All primitive ProbNetKAT programs p (assignments, tests, dup) are by definition semantically deterministic. That & preserves semantic determinacy is immediate from Lemma 1(iv).

The sequential composition *pq* of two semantically deterministic programs is semantically deterministic, since if $\llbracket p \rrbracket(a) = \delta_b$, then $\llbracket pq \rrbracket(a) = \llbracket q \rrbracket(\llbracket p \rrbracket(a))$ and $\llbracket q \rrbracket(\delta_b) = \llbracket q \rrbracket(b)$.

To argue that p^* is semantically deterministic, we must show that the construction of §5.4 yields a point mass whenever $\llbracket p \rrbracket$ is semantically deterministic. This is true because the sets c_n generated by the process are uniquely determined by the start set c_0, since $\llbracket p \rrbracket(c_n) = \delta_{c_{n+1}}$. The result is the point mass on $\bigcup_n c_n$. □

Proof of Theorem 2. The proof is by induction on the structure of the expression. This is clear for assignments, tests, and dup by inspection. The parallel composition operator & in ProbNetKAT corresponds to the sum operator + in NetKAT. Here we have, for $\llbracket p \rrbracket_{\mathrm{N}}(a) = b$ and $\llbracket q \rrbracket_{\mathrm{N}}(a) = c$,

$$\begin{aligned}\llbracket p + q \rrbracket_{\mathrm{N}}(a) &= \llbracket p \rrbracket_{\mathrm{N}}(a) \cup \llbracket q \rrbracket_{\mathrm{N}}(a) = b \cup c\\ \llbracket p \mathbin{\&} q \rrbracket_{\mathrm{P}}(a) &= \llbracket p \rrbracket_{\mathrm{P}}(a) \mathbin{\&} \llbracket q \rrbracket_{\mathrm{P}}(a) = \delta_b \mathbin{\&} \delta_c = \delta_{b \cup c}\end{aligned}$$

by Lemma 1(iv).

For sequential composition, suppose $\llbracket p \rrbracket_{\mathrm{N}}(a) = b$ and $\llbracket q \rrbracket_{\mathrm{N}}(b) = c$. Then

$$\begin{aligned}\llbracket pq \rrbracket_{\mathrm{N}}(a) &= \llbracket q \rrbracket_{\mathrm{N}}(\llbracket p \rrbracket_{\mathrm{N}}(a)) = \llbracket q \rrbracket_{\mathrm{N}}(b) = c\\ \llbracket pq \rrbracket_{\mathrm{P}}(a) &= \llbracket q \rrbracket_{\mathrm{P}}(\llbracket p \rrbracket_{\mathrm{P}}(a)) = \llbracket q \rrbracket_{\mathrm{P}}(\delta_b) = \llbracket q \rrbracket_{\mathrm{P}}(b) = \delta_c.\end{aligned}$$

Finally, for iteration, given c_0, let $c_{n+1} = \llbracket p \rrbracket_{\mathrm{N}}(c_n)$ for $n \geq 0$. Then

$$\llbracket p^* \rrbracket_{\mathrm{N}}(c_0) = \bigcup_n \llbracket p^n \rrbracket_{\mathrm{N}}(c_0) = \bigcup_n c_n,$$

and as argued in the proof of Lemma 2, the deterministic process $\llbracket p^* \rrbracket_{\mathrm{P}}$ produces the point mass on the same set $\bigcup_n c_n$. □

Proof of Lemma 3. For any $A \in \mathcal{B}$,

$$\begin{aligned}f^{-1}(A) &= \{a \mid f(a) \in A\}\\ &= \{a \mid \delta_{f(a)}(A) = 1\} = \{a \mid \llbracket p \rrbracket(a)(A) = 1\},\end{aligned}$$

which is a measurable set since $\llbracket p \rrbracket$ is a Markov kernel. By the change-of-variable rule (3.1),

$$\begin{aligned}\llbracket p \rrbracket(\mu)(A) &= \int_a \llbracket p \rrbracket(a)(A) \cdot \mu(da) = \int_a \delta_{f(a)}(A) \cdot \mu(da)\\ &= \int_a \chi_A(f(a)) \cdot \mu(da) = \int_c \chi_A(c) \cdot \mu(f^{-1}(dc))\\ &= \int_{c \in A} \mu(f^{-1}(dc)) = \mu(f^{-1}(A)).\end{aligned}$$

□

Proof of Lemma 4. Suppose p is deterministic with $\llbracket p \rrbracket(a) = \delta_{f(a)}$. For the left-hand equation,

$$\begin{aligned}\llbracket p(q \mathbin{\&} r) \rrbracket(a) &= \llbracket q \mathbin{\&} r \rrbracket(\llbracket p \rrbracket(a)) = \llbracket q \mathbin{\&} r \rrbracket(b) = \llbracket q \rrbracket(b) \mathbin{\&} \llbracket r \rrbracket(b)\\ &= \llbracket q \rrbracket(\llbracket p \rrbracket(a)) \mathbin{\&} \llbracket r \rrbracket(\llbracket p \rrbracket(a)) = \llbracket pq \rrbracket(a) \mathbin{\&} \llbracket pr \rrbracket(a)\\ &= \llbracket pq \mathbin{\&} pr \rrbracket(a).\end{aligned}$$

For the right-hand equality, we have

$$\begin{aligned}\llbracket (q \mathbin{\&} r)p \rrbracket(a) &= \llbracket p \rrbracket(\llbracket q \rrbracket(a) \mathbin{\&} \llbracket r \rrbracket(a))\\ \llbracket qp \mathbin{\&} rp \rrbracket(a) &= \llbracket p \rrbracket(\llbracket q \rrbracket(a)) \mathbin{\&} \llbracket p \rrbracket(\llbracket r \rrbracket(a)),\end{aligned}$$

so it suffices to show for any μ, ν that

$$\llbracket p \rrbracket(\mu \mathbin{\&} \nu) = \llbracket p \rrbracket(\mu) \mathbin{\&} \llbracket p \rrbracket(\nu).$$

By Lemma 3, it suffices to show that

$$(\mu \mathbin{\&} \nu) \circ f^{-1} = \mu \circ f^{-1} \mathbin{\&} \nu \circ f^{-1}.$$

By Lemma 1(iv) and Theorem 2 we have $f(a \cup b) = f(a) \cup f(b)$. Let $B = \{(a,b) \mid a \cup b \in A)\}$. Then

$$\begin{aligned}(\mu \circ f^{-1} \mathbin{\&} \nu \circ f^{-1})(A) &= (\mu \circ f^{-1} \times \nu \circ f^{-1})(B),\\ ((\mu \mathbin{\&} \nu) \circ f^{-1})(A) &= (\mu \mathbin{\&} \nu)(f^{-1}(A))\\ &= (\mu \times \nu)(\{(a,b) \mid a \cup b \in f^{-1}(A))\}\\ &= (\mu \times \nu)(\{(a,b) \mid f(a) \cup f(b) \in A)\}\\ &= (\mu \times \nu)(\{(a,b) \mid (f(a), f(b)) \in B\}\\ &= (\mu \times \nu)(F^{-1}(B)),\end{aligned}$$

where $F(a,b) = (f(a), f(b))$. It therefore remains to show that the measures $\mu \circ f^{-1} \times \nu \circ f^{-1}$ and $(\mu \times \nu) \circ F^{-1}$ are equal. But on measurable rectangles $C \times D$, both are easily seen to give the same value $\mu(f^{-1}(C)) \cdot \nu(f^{-1}(D))$.

Neither equation holds unconditionally. For both equations, take p to be any program that is not deterministic and $q = r = \mathtt{skip}$. As $\llbracket \mathtt{skip} \mathbin{\&} \mathtt{skip} \rrbracket = \llbracket \mathtt{skip} \rrbracket$ and $\llbracket p\,; \mathtt{skip} \rrbracket = \llbracket \mathtt{skip}\,; p \rrbracket = \llbracket p \rrbracket$, both equations reduce to $\llbracket p \rrbracket = \llbracket p \mathbin{\&} p \rrbracket$, which is false by Lemma 1(vi). □

Proof of Equation (6.1). Let $\mu = \llbracket (\mathtt{skip} \oplus_r \mathtt{dup})^* \rrbracket(\pi)$. Then

$$\begin{aligned}\llbracket \mathtt{skip} \oplus_r \mathtt{dup} \rrbracket(\pi) &= r\llbracket \mathtt{skip} \rrbracket(\pi) + (1 - r)\llbracket \mathtt{dup} \rrbracket(\pi)\\ &= r\delta_{\{\pi\}} + (1 - r)\delta_{\{\pi^2\}},\end{aligned}$$

$$\begin{aligned}
&\llbracket(\texttt{skip} \oplus_r \texttt{dup})^*\rrbracket(\llbracket\texttt{skip} \oplus_r \texttt{dup}\rrbracket(\pi))\\
&\quad = r\llbracket(\texttt{skip} \oplus_r \texttt{dup})^*\rrbracket(\pi) + (1-r)\llbracket(\texttt{skip} \oplus_r \texttt{dup})^*\rrbracket(\pi^2)\\
&\quad = r\mu + (1-r)\mu\!:\!\pi,
\end{aligned}$$

where for $A \in \mathcal{B}$, $a \in 2^H$, $\sigma, \tau \in H$, $\sigma\!:\!\tau$ denotes the concatenation of σ and τ and

$$a\!:\!\tau \triangleq \{\sigma\!:\!\tau \mid \sigma \in a\} \quad A/\tau \triangleq \{a \mid a\!:\!\tau \in A\} \quad (\mu\!:\!\tau)(A) \triangleq \mu(A/\tau).$$

The set $A/\tau \in \mathcal{B}$, because the function $\lambda a.a\!:\!\tau$ is measurable:

$$\{a \mid a\!:\!\tau \in B_\sigma\} = \{a \mid \sigma \in a\!:\!\tau\} = \begin{cases} B_\upsilon, & \text{if } \sigma = \upsilon\!:\!\tau,\\ \varnothing, & \text{otherwise.}\end{cases}$$

By Lemma 1(iii),

$$\begin{aligned}
\mu(A) &= (r\mu + (1-r)\mu\!:\!\pi)(\{c \mid \{\pi\} \cup c \in A\})\\
&= r\mu(\{c \mid \{\pi\} \cup c \in A\}) + (1-r)\mu(\{c \mid \{\pi\} \cup c \in A\}/\pi).
\end{aligned}$$

In particular, for $A = B_\tau$,

$$\begin{aligned}
\mu(B_\tau) &= r\mu(\{c \mid \{\pi\} \cup c \in B_\tau\}) + (1-r)\mu(\{c \mid \{\pi\} \cup c \in B_\tau\}/\pi)\\
&= r\mu(\{c \mid \tau \in \{\pi\} \cup c\}) + (1-r)\mu(\{c \mid \tau \in \{\pi\} \cup c\}/\pi).
\end{aligned}$$

For the three mutually exclusive and exhaustive cases $\tau = \pi$, $\tau = \sigma\!:\!\pi$ with $|\sigma| \geq 1$, and $\tau = \sigma\!:\!\rho$ with $\rho \neq \pi$, we have

$$\begin{aligned}
\{c \mid \pi \in \{\pi\} \cup c\} &= 2^H\\
\{c \mid \sigma\!:\!\pi \in \{\pi\} \cup c\} &= B_{\sigma:\pi}\\
\{c \mid \sigma\!:\!\rho \in \{\pi\} \cup c\} &= B_{\sigma:\rho}
\end{aligned}$$

$$\begin{aligned}
\{c \mid \pi \in \{\pi\} \cup c\}/\pi &= 2^H/\pi = 2^H\\
\{c \mid \sigma\!:\!\pi \in \{\pi\} \cup c\}/\pi &= B_{\sigma:\pi}/\pi = B_\sigma\\
\{c \mid \sigma\!:\!\rho \in \{\pi\} \cup c\}/\pi &= B_{\sigma:\rho}/\pi = \varnothing.
\end{aligned}$$

In these three cases, we have

$$\begin{aligned}
\mu(B_\pi) &= r\mu(2^H) + (1-r)\mu(2^H) = 1\\
\mu(B_{\sigma:\pi}) &= r\mu(B_{\sigma:\pi}) + (1-r)\mu(B_\sigma)\\
\mu(B_{\sigma:\rho}) &= r\mu(B_{\sigma:\rho}) + (1-r)\cdot 0 = r\mu(B_{\sigma:\rho}),
\end{aligned}$$

thus

$$\mu(B_\pi) = 1 \quad \mu(B_{\sigma:\pi}) = \mu(B_\sigma) \quad \mu(B_{\sigma:\rho}) = 0,\ \rho \neq \pi.$$

We thus have $\mu(B_{\pi^n}) = 1$ for $n \geq 1$ and $\mu(B_\tau) = 0$ for all other τ, therefore $\mu = \delta_{\{\pi^n \mid n \geq 1\}}$. □

Proof of Theorem 3. For $x \in \{0,1\}^*$, let $\mathsf{suf}\,x$ be the set of all nonnull suffixes of x; for example,

$$\mathsf{suf}\,01001 = \{01001, 1001, 001, 01, 1\}.$$

Note that $\mathsf{suf}\,\varepsilon = \varnothing$.

Let $\mu = \llbracket p\,;(\texttt{dup}\,;p)^*\rrbracket(0)$, let $\mu_i = \llbracket(\texttt{dup}\,;p)^*\rrbracket(i)$ for $i \in \{0,1\}$, and let $f_x(b) = \mathsf{suf}\,x \cup b\!:\!x$ for $x \in \{0,1\}^*$ and $b \in 2^H$. We start with a few claims.

(A) $f_\varepsilon(b) = b$ and $f_{xy} = f_y \circ f_x$

(B) $\mu = \frac{1}{2}(\mu_0 + \mu_1)$

(C) $\mu_i = \mu \circ f_i^{-1}$, $i \in \{0,1\}$

(D) For all n, $\mu = 2^{-n}\sum_{|x|=n} \mu \circ f_x^{-1}$.

For (A),

$$\begin{aligned}
f_\varepsilon(b) &= \mathsf{suf}\,\varepsilon \cup b\!:\!\varepsilon = \varnothing \cup b = b,\\
f_y(f_x(b)) &= \mathsf{suf}\,y \cup f_x(b)\!:\!y = \mathsf{suf}\,y \cup (\mathsf{suf}\,x \cup b\!:\!x)\!:\!y\\
&= \mathsf{suf}\,y \cup (\mathsf{suf}\,x)\!:\!y \cup (b\!:\!x)\!:\!y = \mathsf{suf}\,xy \cup b\!:\!xy\\
&= f_{xy}(b).
\end{aligned}$$

For (B),

$$\begin{aligned}
\mu &= \llbracket p\,;(\texttt{dup}\,;p)^*\rrbracket(0) = \llbracket(\texttt{dup}\,;p)^*\rrbracket(\llbracket p\rrbracket(0))\\
&= \llbracket(\texttt{dup}\,;p)^*\rrbracket(\tfrac{1}{2}0 + \tfrac{1}{2}1)\\
&= \tfrac{1}{2}\llbracket(\texttt{dup}\,;p)^*\rrbracket(0) + \tfrac{1}{2}\llbracket(\texttt{dup}\,;p)^*\rrbracket(1) = \tfrac{1}{2}(\mu_0 + \mu_1).
\end{aligned}$$

For (C), for $A \in \mathcal{B}$ and $i, j \in \{0,1\}$,

$$\begin{aligned}
(\delta_{\{i\}} \mathbin{\&} \mu_j\!:\!i)(A) &= (\mu_j\!:\!i)(\{a \mid \{i\} \cup a \in A\})\\
&= \mu_j(\{a \mid \{i\} \cup a \in A\}/i)\\
&= \mu_j(\{b \mid b\!:\!i \in \{a \mid \{i\} \cup a \in A\}\})\\
&= \mu_j(\{b \mid \{i\} \cup b\!:\!i \in A\})\\
&= \mu_j(f_i^{-1}(A)),
\end{aligned}$$

thus $\delta_{\{i\}} \mathbin{\&} \mu_j\!:\!i = \mu_j \circ f_i^{-1}$. Then

$$\begin{aligned}
\mu_i &= \llbracket(\texttt{dup}\,;p)^*\rrbracket(i)\\
&= \llbracket\texttt{skip}\rrbracket(i) \mathbin{\&} \llbracket(\texttt{dup}\,;p)^*\rrbracket(\llbracket\texttt{dup}\,;p\rrbracket(i))\\
&= \delta_{\{i\}} \mathbin{\&} \llbracket(\texttt{dup}\,;p)^*\rrbracket(\llbracket p\rrbracket(ii))\\
&= \delta_{\{i\}} \mathbin{\&} \llbracket(\texttt{dup}\,;p)^*\rrbracket(\tfrac{1}{2}0i + \tfrac{1}{2}1i)\\
&= \delta_{\{i\}} \mathbin{\&} (\tfrac{1}{2}\llbracket(\texttt{dup}\,;p)^*\rrbracket(0i) + \tfrac{1}{2}\llbracket(\texttt{dup}\,;p)^*\rrbracket(1i))\\
&= \tfrac{1}{2}(\delta_{\{i\}} \mathbin{\&} \llbracket(\texttt{dup}\,;p)^*\rrbracket(0i)) + \tfrac{1}{2}(\delta_{\{i\}} \mathbin{\&} \llbracket(\texttt{dup}\,;p)^*\rrbracket(1i))\\
&= \tfrac{1}{2}(\delta_{\{i\}} \mathbin{\&} \llbracket(\texttt{dup}\,;p)^*\rrbracket(0)\!:\!i) + \tfrac{1}{2}(\delta_{\{i\}} \mathbin{\&} \llbracket(\texttt{dup}\,;p)^*\rrbracket(1)\!:\!i)\\
&= \tfrac{1}{2}(\delta_{\{i\}} \mathbin{\&} \mu_0\!:\!i) + \tfrac{1}{2}(\delta_{\{i\}} \mathbin{\&} \mu_1\!:\!i)\\
&= \tfrac{1}{2}\mu_0 \circ f_i^{-1} + \tfrac{1}{2}\mu_1 \circ f_i^{-1}\\
&= \tfrac{1}{2}(\mu_0 + \mu_1) \circ f_i^{-1}\\
&= \mu \circ f_i^{-1}.
\end{aligned}$$

For (D), we proceed by induction on n. The basis $n = 0$ is trivial, as f_ε^{-1} is the identity on $\mathcal{B}$. For the induction step,

$$\begin{aligned}
\mu &= 2^{-n}\sum_{|x|=n} \mu \circ f_x^{-1} = 2^{-n}\sum_{|x|=n} \tfrac{1}{2}(\mu_0 + \mu_1) \circ f_x^{-1}\\
&= 2^{-(n+1)}\sum_{|x|=n} (\mu_0 + \mu_1) \circ f_x^{-1}\\
&= 2^{-(n+1)}\sum_{|x|=n} (\mu \circ f_0^{-1} + \mu \circ f_1^{-1}) \circ f_x^{-1}\\
&= 2^{-(n+1)}\sum_{|x|=n} (\mu \circ f_0^{-1} \circ f_x^{-1} + \mu \circ f_1^{-1} \circ f_x^{-1})\\
&= 2^{-(n+1)}\sum_{|x|=n} (\mu \circ f_{0x}^{-1} + \mu \circ f_{1x}^{-1}) = 2^{-(n+1)}\sum_{|x|=n+1} \mu \circ f_x^{-1}.
\end{aligned}$$

Now on to (i)–(iii) of the theorem. Recall that $\mathcal{B}$ is generated by the sets $B_\tau = \{a \mid \tau \in a\}$. We have

$$\begin{aligned}
f_x^{-1}(B_\tau) &= \{a \mid f_x(a) \in B_\tau\} = \{a \mid \tau \in \mathsf{suf}\,x \cup a\!:\!x\}\\
&= \begin{cases} 2^H, & \tau \in \mathsf{suf}\,x,\\ B_\sigma, & \tau = \sigma\!:\!x,\\ \varnothing, & \text{otherwise.}\end{cases}
\end{aligned}$$

Thus for any n, if $|x| = |\tau| = n$, then

$$f_x^{-1}(B_\tau) = \begin{cases} 2^H, & \tau = x,\\ \varnothing, & \tau \neq x\end{cases}$$

and if $|x| = |\tau| = |\sigma| = n$ and $\tau \neq \sigma$, then

$$\begin{aligned} f_x^{-1}(B_\tau \cap B_\sigma) &= f_x^{-1}(B_\tau) \cap f_x^{-1}(B_\sigma) \\ &= \begin{cases} 2^H, & \tau = x, \\ \varnothing, & \tau \neq x \end{cases} \cap \begin{cases} 2^H, & \sigma = x, \\ \varnothing, & \sigma \neq x \end{cases} \\ &= \varnothing, \end{aligned}$$

thus

$$\begin{aligned} \mu(B_\tau) &= 2^{-n} \sum_{|x|=n} \mu(f_x^{-1}(B_\tau)) \\ &= 2^{-n}(\sum_{x=\tau} \mu(2^H) + \sum_{x \neq \tau} \mu(\varnothing)) = 2^{-n} \\ \mu(B_\tau \cap B_\sigma) &= 2^{-n} \sum_{|x|=n} \mu(f_x^{-1}(B_\tau \cap B_\sigma)) \\ &= 2^{-n} \sum_{|x|=n} \mu(\varnothing) = 0. \end{aligned}$$

These two equations verify (i) and (ii), respectively. For (iii), we have

$$\{a\} \subseteq \bigcap_{\substack{|\tau|=n \\ \tau \in a}} B_\tau,$$

and it follows from (i) and (ii) that for any n,

$$\mu(\{a\}) = \mu(\{a\} \cap \bigcap_{\substack{|\tau|=n \\ \tau \in a}} B_\tau) \leq 2^{-n}.$$

As n was arbitrary, $\mu(\{a\}) = 0$. □