Cantor Meets Scott: Semantic Foundations for Probabilistic Networks

Steffen Smolka
Cornell University, USA

Praveen Kumar
Cornell University, USA

Nate Foster
Cornell University, USA

Dexter Kozen
Cornell University, USA

Alexandra Silva
University College London, UK

Abstract

ProbNetKAT is a probabilistic extension of NetKAT with a denotational semantics based on Markov kernels. The language is expressive enough to generate continuous distributions, which raises the question of how to compute effectively in the language. This paper gives an new characterization of ProbNetKAT's semantics using domain theory, which provides the foundation needed to build a practical implementation. We show how to use the semantics to approximate the behavior of arbitrary ProbNetKAT programs using distributions with finite support. We develop a prototype implementation and show how to use it to solve a variety of problems including characterizing the expected congestion induced by different routing schemes and reasoning probabilistically about reachability in a network.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory—Semantics

Keywords Software-defined networking, Probabilistic semantics, Kleene algebra with tests, Domain theory, NetKAT.

1. Introduction

The recent emergence of software-defined networking (SDN) has led to the development of a number of domain-specific programming languages (Foster et al. 2011; Monsanto et al. 2013; Voellmy et al. 2013; Nelson et al. 2014) and reasoning tools (Kazemian et al. 2012; Khurshid et al. 2013; Anderson et al. 2014; Foster et al. 2015) for networks. But there is still a large gap between the models provided by these languages and the realities of modern networks. In particular, most existing SDN languages have semantics based on deterministic packet-processing functions, which makes it impossible to encode probabilistic behaviors. This is unfortunate because in the real world, network operators often use randomized protocols and probabilistic reasoning to achieve good performance.

POPL'17, January 15–21, 2017, Paris, France
© 2017 ACM. 978-1-4503-4660-3/17/01...$15.00
http://dx.doi.org/10.1145/3009837.3009843

Previous work on ProbNetKAT (Foster et al. 2016) proposed an extension to the NetKAT language (Anderson et al. 2014; Foster et al. 2015) with a random choice operator that can be used to express a variety of probabilistic behaviors. ProbNetKAT has a compositional semantics based on Markov kernels that conservatively extends the deterministic NetKAT semantics and has been used to reason about various aspects of network performance including congestion, fault tolerance, and latency. However, although the language enjoys a number of attractive theoretical properties, there are some major impediments to building a practical implementation: (i) the semantics of iteration is formulated as an infinite process rather than a fixpoint in a suitable order, and (ii) some programs generate continuous distributions. These factors make it difficult to determine when a computation has converged to its final value, and there are also challenges related to representing and analyzing distributions with infinite support.

This paper introduces a new semantics for ProbNetKAT, following the approach pioneered by Saheb-Djahromi, Jones, and Plotkin (Saheb-Djahromi 1980, 1978; Jones 1989; Plotkin 1982; Jones and Plotkin 1989). Whereas the original semantics of ProbNetKAT was somewhat imperative in nature, being based on stochastic processes, the semantics introduced in this paper is purely functional. Nevertheless, the two semantics are closely related—we give a precise, technical characterization of the relationship between them. The new semantics provides a suitable foundation for building a practical implementation, it provides new insights into the nature of probabilistic behavior in networks, and it opens up several interesting theoretical questions for future work.

Our new semantics follows the order-theoretic tradition established in previous work on Scott-style domain theory (Scott 1972; Abramsky and Jung 1994). In particular, Scott-continuous maps on algebraic and continuous DCPOs both play a key role in our development. However, there is an interesting twist: NetKAT and ProbNetKAT are not *state-based* as with most other probabilistic systems, but are rather *throughput-based*. A ProbNetKAT program can be thought of as a filter that takes an input set of packet histories and generates an output randomly distributed on the measurable space 2^H of sets of packet histories. The closest thing to a "state" is a set of packet histories, and the structure of these sets (e.g., the lengths of the histories they contain and the standard subset relation) are important considerations. Hence, the fundamental domains are not flat domains as in traditional domain theory, but are instead the DCPO of sets of packet histories ordered by the subset relation. Another point of departure from prior work is that the structures used

in the semantics are not subprobability distributions, but genuine probability distributions: with probability 1, some set of packets is output, although it may be the empty set.

It is not obvious that such an order-theoretic semantics should exist at all. Traditional probability theory does not take order and compositionality as fundamental structuring principles, but prefers to work in monolithic sample spaces with strong topological properties such as Polish spaces. Prototypical examples of such spaces are the real line, Cantor space, and Baire space. The space of sets of packet histories 2^{H} is homeomorphic to the Cantor space, and this was the guiding principle in the development of the original ProbNetKAT semantics. Although the Cantor topology enjoys a number of attractive properties (compactness, metrizability, strong separation) that are lost when shifting to the Scott topology, the sacrifice is compensated by a more compelling least-fixpoint characterization of iteration that aligns better with the traditional domain-theoretic treatment. Intuitively, the key insight that underpins our development is the observation that ProbNetKAT programs are monotone: if a larger set of packet histories is provided as input, then the likelihood of seeing any particular set of packets as a subset of the output set can only increase. From this germ of an idea, we formulate an order-theoretic semantics for ProbNetKAT.

In addition to the strong theoretical motivation for this work, our new semantics also provides a source of practical useful reasoning techniques, notably in the treatment of iteration and approximation. The original paper on ProbNetKAT showed that the Kleene star operator satisfies the usual fixpoint equation $P^* = 1 \,\&\, P \,;\, P^*$, and that its finite approximants $P^{(n)}$ converge weakly (but not pointwise) to it. However, it was not characterized as a least fixpoint in any order or as a canonical solution in any sense. This was a bit unsettling and raised questions as to whether it was the "right" definition—questions for which there was no obvious answer. This paper characterizes P^* as the least fixpoint of the Scott-continuous map $X \mapsto 1 \,\&\, P \,;\, X$ on a continuous DCPO of Scott-continuous Markov kernels. This not only corroborates the original definition as the "right" one, but provides a powerful tool for monotone approximation. Indeed, this result implies the correctness of our prototype implementation, which we have used to build and evaluate several applications inspired by common real-world scenarios.

Contributions. This main contributions of this paper are as follows: (i) we develop a domain-theoretic foundation for probabilistic network programming, (ii) using this semantics, we build a prototype implementation of the ProbNetKAT language, and (iii) we evaluate the applicability of the language on several case studies.

Outline. The paper is structured as follows. In §2 we give a high-level overview of our technical development using a simple running example. In §3 we review basic definitions from domain theory and measure theory. In §4 we formalize the syntax and semantics of ProbNetKAT abstractly in terms of a monad. In §5 we prove a general theorem relating the Scott and Cantor topologies on 2^{H}. Although the Scott topology is much weaker, the two topologies generate the same Borel sets, so the probability measures are the same in both. We also show that the bases of the two topologies are related by a countably infinite-dimensional triangular linear system, which can be viewed as an infinite analog of the inclusion-exclusion principle. The cornerstone of this result is an extension theorem (Theorem 8) that determines when a function on the basic Scott-open sets extends to a measure. In §6 we give the new domain-theoretic semantics for ProbNetKAT in which programs are characterized as Markov kernels that are Scott-continuous in their first argument. We show that this class of kernels forms a continuous DCPO, the basis elements being those kernels that drop all but fixed finite sets of input and output packets. In §7 we show that ProbNetKAT's primitives are (Scott-)continuous and its program operators preserve continuity. Other operations such as product and Lebesgue integration are also treated in this framework. In proving these results, we attempt to reuse general results from domain theory whenever possible, relying on the specific properties of 2^{H} only when necessary. We supply complete proofs for folklore results and in cases where we could not find an appropriate original source. We also show that the two definitions of the Kleene star operator—one in terms of an infinite stochastic process and one as the least fixpoint of a Scott-continuous map—coincide. In §8 we apply the continuity results from §7 to derive monotone convergence theorems. In §9 we describe a prototype implementation based on §8 and several applications. In §10 we review related work. We conclude in §11 by discussing open problems and future directions.

2. Overview

This section provides motivation for the ProbNetKAT language and summarizes our main results using a simple example.

Example. Consider the topology shown in Figure 1 and suppose we are asked to implement a routing application that forwards all traffic to its destination while minimizing congestion, gracefully adapting to shifts in load, and also handling unexpected failures. This problem is known as traffic engineering in the networking literature and has been extensively studied (Fortz et al. 2002; He and Rexford 2008; Jain et al. 2013; Applegate and Cohen 2003; Räcke 2008). Note that standard shortest-path routing (SPF) does not solve the problem as stated—in general, it can lead to bottlenecks and also makes the network vulnerable to failures. For example, consider sending a large amount of traffic from host h_1 to host h_3: there are two paths in the topology, one via switch S_2 and one via switch S_4, but if we only use a single path we sacrifice half of the available capacity. The most widely-deployed approaches to traffic engineering today are based on using multiple paths and randomization. For example, Equal Cost Multipath Routing (ECMP), which is widely supported on commodity routers, selects a least-cost path for each traffic flow uniformly at random. The intention is to spread the offered load across a large set of paths, thereby reducing congestion without increasing latency.

ProbNetKAT Language. Using ProbNetKAT, it is straightforward to write a program that captures the essential behavior of ECMP. We first construct programs that model the routing tables and topology, and build a program that models the behavior of the entire network.

Routing: We model the routing tables for the switches using simple ProbNetKAT programs that match on destination addresses and forward packets on the next hop toward their destination. To randomly map packets to least-cost paths, we use the choice operator ($\oplus$). For example, the program for switch $\mathsf{S1}$ in Figure 1 is as follows:

$$\begin{aligned} p_1 \triangleq\ & (\mathsf{dst}{=}h_1 \,;\, \mathsf{pt}{\leftarrow}1) \\ \&\ & (\mathsf{dst}{=}h_2 \,;\, \mathsf{pt}{\leftarrow}2) \\ \&\ & (\mathsf{dst}{=}h_3 \,;\, (\mathsf{pt}{\leftarrow}2 \oplus \mathsf{pt}{\leftarrow}4)) \\ \&\ & (\mathsf{dst}{=}h_4 \,;\, \mathsf{pt}{\leftarrow}4) \end{aligned}$$

The programs for other switches are similar. To a first approximation, this program can be read as a routing table, whose entries are separated by the parallel composition operator (&). The first entry states that packets whose destination is h_1 should be forwarded out on port 1 (which is directly connected to h_1). Likewise, the second entry states that packets whose destination is host h_2 should be forwarded out on port 2, which is the next hop on the unique shortest path to h_2. The third entry, however, is different: it states that packets whose destination is h_3 should be forwarded out on ports 2 and 4 with equal probability. This divides traffic going to h_3 among the clockwise path via S_2 and the counter-clockwise path via S_4. The final entry states that packets whose destination is h_4

should be forwarded out on port 4, which is again the next hop on the unique shortest path to h_4. The routing program for the network is the parallel composition of the programs for each switch:

$$p \triangleq (\mathsf{sw}{=}S_1\,;p_1)\,\&\,(\mathsf{sw}{=}S_2\,;p_2)\,\&\,(\mathsf{sw}{=}S_3\,;p_3)\,\&\,(\mathsf{sw}{=}S_4\,;p_4)$$

Topology: We model a directed link as a program that matches on the switch and port at one end of the link and modifies the switch and port to the other end of the link. We model an undirected link l as a parallel composition of directed links in each direction. For example, the link between switches S_1 and S_2 is modeled as follows:

$$\begin{aligned} l_{1,2} \triangleq\ & (\mathsf{sw}{=}S_1\ ;\ \mathsf{pt}{=}2\ ;\ \mathsf{dup}\ ;\ \mathsf{sw}{\leftarrow}S_2\ ;\ \mathsf{pt}{\leftarrow}1\ ;\ \mathsf{dup}) \\ \&\ & (\mathsf{sw}{=}S_2\ ;\ \mathsf{pt}{=}1\ ;\ \mathsf{dup}\ ;\ \mathsf{sw}{\leftarrow}S_1\ ;\ \mathsf{pt}{\leftarrow}2\ ;\ \mathsf{dup}) \end{aligned}$$

Note that at each hop we use ProbNetKAT's dup operator to store the headers in the packet's history, which records the trajectory of the packet as it goes through the network. Histories are useful for tasks such as measuring path length and analyzing link congestion. We model the topology as a parallel composition of individual links:

$$t \triangleq l_{1,2}\ \&\ l_{2,3}\ \&\ l_{3,4}\ \&\ l_{1,4}$$

To delimit the network edge, we define ingress and egress predicates:

$$\begin{aligned} in &\triangleq (\mathsf{sw}{=}1\ ;\ \mathsf{pt}{=}1)\,\&\,(\mathsf{sw}{=}2\ ;\ \mathsf{pt}{=}2)\,\&\,\ldots \\ out &\triangleq (\mathsf{sw}{=}1\ ;\ \mathsf{pt}{=}1)\,\&\,(\mathsf{sw}{=}2\ ;\ \mathsf{pt}{=}2)\,\&\,\ldots \end{aligned}$$

Here, since every ingress is an egress, the predicates are identical.

Network: We model the end-to-end behavior of the entire network by combining p, t, in and out into a single program:

$$net \triangleq in\ ;\ (p\ ;\ t)^*\ ;\ p\ ;\ out$$

This program models processing each input from ingress to egress across a series of switches and links. Formally it denotes a Markov kernel that, when supplied with an input distribution on packet histories μ produces an output distribution ν.

Queries: Having constructed a probabilistic model of the network, we can use standard tools from measure theory to reason about performance. For example, to compute the expected congestion on a given link l, we would introduce a function Q from sets of packets to $\mathbb{R} \cup \{\infty\}$ (formally a random variable):

$$Q(a) \triangleq \sum_{h \in a} \#_l(h)$$

where $\#_l(h)$ is the function on packet histories that returns the number of times that link l occurs in h, and then compute the expected value of Q using integration:

$$\mathop{\mathbf{E}}_{\nu}[Q] = \int Q\, d\nu$$

We can compute queries that capture other aspects of network performance such as latency, reliability, etc. in similar fashion.

Limitations. Unfortunately there are several issues with the approach just described:

- One problem is that computing the results of a query can require complicated measure theory since a ProbNetKAT program may generate a continuous distribution in general (Lemma 4). Formally, instead of summing over the support of the distribution, we have to use Lebesgue integration in an appropriate measurable space. Of course, there are also challenges in representing infinite distributions in an implementation.
- Another issue is that the semantics of iteration is modeled in terms of an infinite stochastic process rather than a standard fixpoint. The original ProbNetKAT paper showed that it is possible to approximate a program using a series of star-free programs that weakly converge to the correct result, but the approximations need not converge monotonically, which makes this result difficult to apply in practice.
- Even worse, many of the queries that we would like to answer are not actually continuous in the Cantor topology, meaning that the weak convergence result does not even apply! The notion of distance on sets of packet histories is $d(a, b) = 2^{-n}$ where n is the length of the smallest history in a but not in b, or vice versa. It is easy to construct a sequence of histories h_n of length n such that $\lim_{n\to\infty} d(\{h_n\}, \{\}) = 0$ but $\lim_{n\to\infty} Q(\{h_n\}) = \infty$ which is not equal to $Q(\{\}) = 0$.

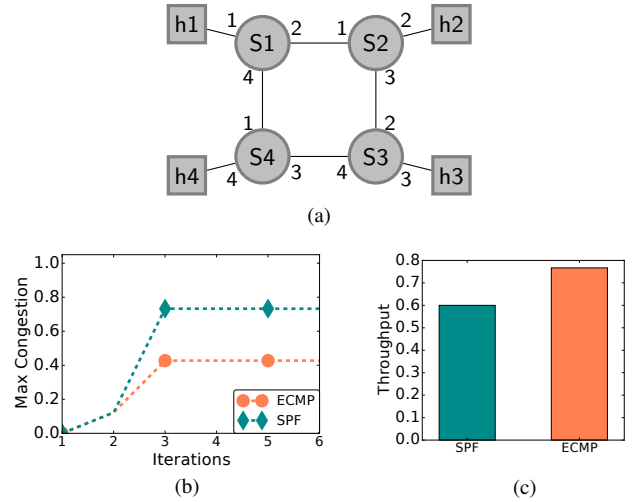


Figure 1. (a) topology, (b) congestion, (c) failure throughput.

Together, these issues are significant impediments that make it difficult to apply ProbNetKAT in many scenarios.

Domain-Theoretic Semantics. This paper develops a new semantics for ProbNetKAT that overcomes these problems and provides the key building blocks needed to engineer a practical implementation. The main insight is that we can formulate the semantics in terms of the Scott topology rather than the Cantor topology. It turns out that these two topologies generate the same Borel sets, and the relationship between them can be characterized using an extension theorem that captures when functions on the basic Scott-open sets extend to a measure. We show how to construct a DCPO equipped with a natural partial order that also lifts to a partial order on Markov kernels. We prove that standard program operators are continuous, which allows us to formulate the semantics of the language—in particular Kleene star—using standard tools from domain theory, such as least fixpoints. Finally, we formalize a notion of approximation and prove a monotone convergence theorem.

The problems with the original ProbNetKAT semantics identified above are all solved using the new semantics. Because the new semantics models iteration as a least fixpoint, we can work with finite distributions and star-free approximations that are guaranteed to monotonically converge to the analytical solution (Corollary 23). Moreover, whereas our query Q was not Cantor continuous, it is straightforward to show that it is Scott continuous. Let A be an increasing chain $a_0 \subseteq a_1 \subseteq a_2 \subseteq \ldots$ ordered by inclusion. Scott continuity requires $\bigsqcup_{a\in A} Q(a) = Q(\bigsqcup A)$ which is easy to prove. Hence, the convergence theorem applies and we can compute a monotonically increasing chain of approximations that converge to $\mathbf{E}_\nu[Q]$.

Implementation and Applications. We developed the first implementation of ProbNetKAT using the new semantics. We built an

interpreter for the language and implemented a variety of traffic engineering schemes including ECMP, k-shortest path routing, and oblivious routing (Räcke 2008). We analyzed the performance of each scheme in terms of congestion and latency on real-world demands drawn from Internet2's Abilene backbone, and in the presence of link failures. We showed how to use the language to reason probabilistically about reachability properties such as loops and black holes. Figures 1 (b-c) depict the expected throughput and maximum congestion when using shortest paths (SPF) and ECMP on the 4-node topology as computed by our ProbNetKAT implementation. We set the demand from h_1 to h_3 to be $\frac{1}{2}$ units of traffic, and the demand between all other pairs of hosts to be $\frac{1}{8}$ units. The first graph depicts the maximum congestion induced under successive approximations of the Kleene star, and shows that ECMP achieves much better congestion than SPF. With SPF, the most congested link (from S_1 to S_2) carries traffic from h_1 to h_2, from h_4 to h_2, and from h_1 to h_3, resulting in $\frac{3}{4}$ total traffic. With ECMP, the same link carries traffic from h_1 to h_2, half of the traffic from h_2 to h_4, half of the traffic from h_1 to h_3, resulting in $\frac{7}{16}$ total traffic. The second graph depicts the loss of throughput when the same link fails. The total aggregate demand is $1\frac{7}{8}$. With SPF, $\frac{3}{4}$ units of traffic are dropped leaving $1\frac{1}{8}$ units, which is 60% of the demand, whereas with ECMP only $\frac{7}{16}$ units of traffic are dropped leaving $1\frac{7}{16}$ units, which is 77% of the demand.

3. Preliminaries

This section briefly reviews basic concepts from topology, measure theory, and domain theory, and defines *Markov kernels*, the objects on which ProbNetKAT's semantics is based. For a more detailed account, the reader is invited to consult standard texts (Durrett 2010; Abramsky and Jung 1994).

Topology. A *topology* $\mathcal{O} \subseteq 2^X$ on a set X is a collection of subsets including X and $\varnothing$ that is closed under finite intersection and arbitrary union. A pair $(X, \mathcal{O})$ is called a *topological space* and the sets $U, V \in \mathcal{O}$ are called the *open sets* of $(X, \mathcal{O})$. A function $f : X \to Y$ between topological spaces $(X, \mathcal{O}_X)$ and $(Y, \mathcal{O}_Y)$ is *continuous* if the preimage of any open set in Y is open in X, *i.e.* if

$$f^{-1}(U) = \{x \in X \mid f(x) \in U\} \in \mathcal{O}_X$$

for any $U \in \mathcal{O}_Y$.

Measure Theory. A *σ-algebra* $\mathcal{F} \subseteq 2^X$ on a set X is a collection of subsets including X that is closed under complement, countable union, and countable intersection. A *measurable space* is a pair $(X, \mathcal{F})$. A *probability measure* μ over such a space is a function $\mu : \mathcal{F} \to [0, 1]$ that assigns probabilities $\mu(A) \in [0, 1]$ to the *measurable sets* $A \in \mathcal{F}$, and satisfies the following conditions:

- $\mu(X) = 1$
- $\mu(\bigcup_{i \in I} A_i) = \sum_{i \in I} \mu(A_i)$ whenever $\{A_i\}_{i \in I}$ is a countable collection of disjoint measurable sets.

Note that these conditions already imply that $\mu(\varnothing) = 0$. Elements $a, b \in X$ are called *points* or *outcomes*, and measurable sets $A, B \in \mathcal{F}$ are also called *events*. The σ-algebra $\sigma(U)$ generated by a set $U \subseteq X$ is the smallest σ-algebra containing U:

$$\sigma(U) \triangleq \bigcap \{\mathcal{F} \subseteq 2^X \mid \mathcal{F} \text{ is a } \sigma\text{-algebra and } U \subseteq \mathcal{F}\}.$$

Note that it is well-defined because the intersection is not empty (2^X is trivially a σ-algebra containing U) and intersections of σ-algebras are again σ-algebras. If $\mathcal{O} \subseteq 2^X$ are the open sets of X, then the smallest σ-algebra containing the open sets $\mathcal{B} = \sigma(\mathcal{O})$ is the *Borel algebra*, and the measurable sets $A, B \in \mathcal{B}$ are the *Borel sets* of X.

Let $P_\mu \triangleq \{a \in X \mid \mu(\{a\}) > 0\}$ denote the points (not events!) with non-zero probability. It can be shown that P_μ is countable. A probability measure is called *discrete* if $\mu(P_\mu) = 1$. Such a measure can simply be represented by a function $Pr : X \to [0, 1]$ with $Pr(a) = \mu(\{a\})$. If $|P_\mu| < \infty$, the measure is called *finite* and can be represented by a finite map $Pr : P_\mu \to [0, 1]$. In contrast, measures for which $\mu(P_\mu) = 0$ are called *continuous*, and measures for which $0 < \mu(P_\mu) < 1$ are called *mixed*. The *Dirac measure* or *point mass* puts all probability on a single point $a \in X$: $\delta_a(A) = 1$ if $a \in A$ and 0 otherwise. The uniform distribution on $[0, 1]$ is a continuous measure.

A function $f : X \to Y$ between measurable spaces $(X, \mathcal{F}_X)$ and $(Y, \mathcal{F}_Y)$ is called *measurable* if the preimage of any measurable set in Y is measurable in X, *i.e.* if

$$f^{-1}(A) \triangleq \{x \in X \mid f(x) \in A\} \in \mathcal{F}_X$$

for all $A \in \mathcal{F}_Y$. If $Y = \mathbb{R} \cup \{-\infty, +\infty\}$, then f is called a *random variable* and its *expected value* with respect to a measure μ on X is given by the Lebesgue integral

$$\mathop{\mathbf{E}}_{\mu}[f] \triangleq \int f d\mu = \int_{x \in X} f(x) \cdot \mu(dx)$$

If μ is discrete, the integral simplifies to the sum

$$\mathop{\mathbf{E}}_{\mu}[f] = \sum_{x \in X} f(x) \cdot \mu(\{x\}) = \sum_{x \in P_\mu} f(x) \cdot Pr(x)$$

Markov Kernels. Consider a probabilistic transition system with states X that makes a random transition between states at each step. If X is finite, the system can be captured by a transition matrix $T \in [0, 1]^{X \times X}$, where the matrix entry T_{xy} gives the probability that the system transitions from state x to state y. Each row T_x describes the transition function of a state x and must sum to 1. Suppose that the start state is initially distributed according to the row vector $V \in [0, 1]^X$, *i.e.* the system starts in state $x \in X$ with probability V_x. Then, the state distribution is given by the matrix product $VT \in [0, 1]^X$ after one step and by VT^n after n steps.

Markov kernels generalize this idea to infinite state systems. Given measurable spaces $(X, \mathcal{F}_X)$ and $(Y, \mathcal{F}_Y)$, a Markov kernel with source X and target Y is a function $P : X \times \mathcal{F}_Y \to [0, 1]$ (or equivalently, $X \to \mathcal{F}_Y \to [0, 1]$) that maps each source state $x \in X$ to a distribution over target states $P(x, -) : \mathcal{F}_Y \to [0, 1]$. If the initial distribution is given by a measure ν on X, then the target distribution μ after one step is given by Lebesgue integration:

$$\mu(A) \triangleq \int_{x \in X} P(x, A) \cdot \nu(dx) \qquad (A \in \mathcal{F}_Y) \tag{3.1}$$

If ν and $P(x, -)$ are discrete, the integral simplifies to the sum

$$\mu(\{y\}) = \sum_{x \in X} P(x, \{y\}) \cdot \nu(\{x\}) \qquad (y \in Y)$$

which is just the familiar vector-matrix-product VT. Similarly, two kernels P, Q from X to Y and from Y to Z, respectively, can be sequentially composed to a kernel $P \,;\, Q$ from X to Z:

$$(P \,;\, Q)(x, A) \triangleq \int_{y \in Y} P(x, dy) \cdot Q(y, A) \tag{3.2}$$

This is the continuous analog of the matrix product TT. A Markov kernel P must satisfy two conditions:

(i) For each source state $x \in X$, the map $A \mapsto P(x, A)$ must be a probability measure on the target space.

(ii) For each event $A \in \mathcal{F}_Y$ in the target space, the map $x \mapsto P(x, A)$ must be a measurable function.

Condition (ii) is required to ensure that integration is well-defined. A kernel P is called *deterministic* if $P(a, -)$ is a dirac measure for each a.

Domain Theory. A *partial order* (PO) is a pair $(D, \sqsubseteq)$ where D is a set and $\sqsubseteq$ is a reflexive, transitive, and antisymmetric relation on D. For two elements $x, y \in D$ we let $x \sqcup y$ denote their $\sqsubseteq$-least upper bound (*i.e.*, their supremum), provided it exists. Analogously, the least upper bound of a subset $C \subseteq D$ is denoted $\bigsqcup C$, provided it exists. A non-empty subset $C \subseteq D$ is *directed* if for any two $x, y \in C$ there exists *some* upper bound $x, y \sqsubseteq z$ in C. A *directed complete partial order* (DCPO) is a PO for which any directed subset $C \subseteq D$ has a supremum $\bigsqcup C$ in D. If a PO has a least element it is denoted by $\bot$, and if it has a greatest element it is denoted by $\top$. For example, the nonnegative real numbers with infinity $\mathbb{R}_+ \triangleq [0, \infty]$ form a DCPO under the natural order $\leq$ with suprema $\bigsqcup C = \sup C$, least element $\bot = 0$, and greatest element $\top = \infty$. The unit interval is a DCPO under the same order, but with $\top = 1$. Any powerset 2^X is a DCPO under the subset order, with suprema given by union.

A function f from D to E is called (*Scott-*)*continuous* if

(i) it is monotone, *i.e.* $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$, and

(ii) it preserves suprema, *i.e.* $f(\bigsqcup C) = \bigsqcup_{x \in C} f(x)$ for any directed set C in D.

Equivalently, f is continuous with respect to the *Scott topologies* on D and E (Abramsky and Jung 1994, Proposition 2.3.4), which we define next. (Note how condition (ii) looks like the classical definition of continuity of a function f, but with suprema taking the role of limits). The set of all continuous functions $f : D \to E$ is denoted $[D \to E]$.

A subset $A \subseteq D$ is called *up-closed* (or an *upper set*) if $a \in A$ and $a \sqsubseteq b$ implies $b \in A$. The smallest up-closed superset of A is called its *up-closure* and is denoted $A{\uparrow}$. A is called (*Scott-*)*open* if it is up-closed and intersects every directed subset $C \subseteq D$ that satisfies $\bigsqcup C \in A$. For example, the Scott-open sets of $\mathbb{R}_+$ are the upper semi-infinite intervals $(r, \infty]$, $r \in \mathbb{R}_+$. The Scott-open sets form a topology on D called the *Scott topology*.

DCPOs enjoy many useful closure properties:

(i) The cartesian product of any collection of DCPOs is a DCPO with componentwise order and suprema.

(ii) If E is a DCPO and D any set, the function space $D \to E$ is a DCPO with pointwise order and suprema.

(iii) The continuous functions $[D \to E]$ between DCPOs D and E form a DCPO with pointwise order and suprema.

If D is a DCPO with least element $\bot$, then any Scott-continuous self-map $f \in [D \to D]$ has a $\sqsubseteq$-least fixpoint, and it is given by the supremum of the chain $\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \sqsubseteq \ldots$:

$$\mathrm{lfp}(f) = \bigsqcup_{n \geq 0} f^n(\bot)$$

Moreover, the least fixpoint operator, $\mathrm{lfp} \in [[D \to D] \to D]$ is itself continuous, that is: $\mathrm{lfp}(\bigsqcup C) = \bigsqcup_{f \in C} \mathrm{lfp}(f)$, for any directed set of functions $C \subseteq [D \to D]$.

An element a of a DCPO is called *finite* (Abramsky and Jung (1994) use the term *compact*) if for any directed set A, if $a \sqsubseteq \bigsqcup A$, then there exists $b \in A$ such that $a \sqsubseteq b$. Equivalently, a is finite if its up-closure $\{a\}{\uparrow}$ is Scott-open. A DCPO is called *algebraic* if for every element b, the finite elements $\sqsubseteq$-below b form a directed set and b is the supremum of this set. An element a of a DCPO *approximates* another element b, written $a \ll b$, if for any directed set A, $a \sqsubseteq c$ for some $c \in A$ whenever $b \sqsubseteq \bigsqcup A$. A DCPO is called *continuous* if for every element b, the elements $\ll$-below b form a directed set and b is the supremum of this set. Every algebraic DCPO is continuous. A set in a topological space is *compact-open* if it is compact (every open cover has a finite subcover) and open.

Here we recall some basic facts about DCPOs. These are all well-known, but we state them as a lemma for future reference.

Lemma 1 (DCPO Basic Facts)**.**

(i) *Let E be a DCPO and D_1, D_2 sets. There is a homeomorphism (bicontinuous bijection)* curry *between the DCPOs $D_1 \times D_2 \to E$ and $D_1 \to D_2 \to E$, where the function spaces are ordered pointwise. The inverse of* curry *is* uncurry.

(ii) *In an algebraic DCPO, the open sets $\{a\}{\uparrow}$ for finite a form a base for the Scott topology.*

(iii) *A subset of an algebraic DCPO is compact-open iff it is a finite union of basic open sets $\{a\}{\uparrow}$.*

4. ProbNetKAT

This section defines the syntax and semantics of ProbNetKAT formally (see Figure 2) and establishes some basic properties. ProbNetKAT is a core calculus designed to capture the essential forwarding behavior of probabilistic network programs. In particular, the language includes primitives that model fundamental constructs such as parallel and sequential composition, iteration, and random choice. It does not model features such as mutable state, asynchrony, and dynamic updates, although extensions to NetKAT-like languages with several of these features have been studied in previous work (Reitblatt et al. 2012; McClurg et al. 2016).

Syntax. A packet π is a record mapping a finite set of fields $f_1, f_2, \ldots, f_k$ to bounded integers n. Fields include standard header fields such as the source (src) and destination (dst) of the packet, and two logical fields (sw for switch and pt for port) that record the current location of the packet in the network. The logical fields are not present in a physical network packet, but it is convenient to model them as proper header fields. We write $\pi.f$ to denote the value of field f of π and $\pi[f{:=}n]$ for the packet obtained from π by updating field f to n. We let Pk denote the (finite) set of all packets.

A history $h = \pi{::}\hbar$ is a non-empty list of packets with *head packet* π and (possibly empty) *tail* $\hbar$. The head packet models the packet's current state and the tail contains its prior states, which capture the trajectory of the packet through the network. Operationally, only the head packet exists, but it is useful to discriminate between identical packets with different histories. We write H to denote the (countable) set of all histories.

We differentiate between *predicates* (t, u) and *programs* (p, q). The predicates form a Boolean algebra and include the primitives *false* (0), *true* (1), and *tests* $(f{=}n)$, as well as the standard Boolean operators disjunction $(t \,\&\, u)$, conjunction $(t \,;\, u)$, and negation $(\neg t)$. Programs include *predicates* (t) and *modifications* $(f{\leftarrow}n)$ as primitives, and the operators *parallel composition* $(p \& q)$, *sequential composition* $(p \,;\, q)$, and *iteration* (p^*). The primitive dup records the current state of the packet by extending the tail with the head packet. Intuitively, we may think of a history as a log of a packet's activity, and of dup as the logging command. Finally, *choice* $p \oplus_r q$ executes p with probability r or q with probability $1 - r$. We write $p \oplus q$ when $r = 0.5$.

Predicate conjunction and sequential composition use the same syntax $(t; u)$ as their semantics coincide (as we will see shortly). The same is true for disjunction of predicates and parallel composition $(t \,\&\, u)$. The distinction between *predicates* and *programs* is merely to restrict negation to predicates and rule out programs like $\neg(p^*)$.

Syntactic Sugar. The language as presented in Figure 2 is reduced to its core primitives. It is worth noting that many useful constructs can be derived from this core. In particular, it is straightforward to

Syntax

Naturals	$n ::= 0 \mid 1 \mid 2 \mid \ldots$	
Fields	$f ::= f_1 \mid \ldots \mid f_k$	
Packets	$\mathsf{Pk} \ni \pi ::= \{f_1 = n_1, \ldots, f_k = n_k\}$	
Histories	$\mathsf{H} \ni h ::= \pi :: \hbar$	
	$\hbar ::= \langle\rangle \mid \pi :: \hbar$	
Probabilities	$[0,1] \ni r$	
Predicates	$t, u ::= 0$	*False/Drop*
	$\mid 1$	*True/Skip*
	$\mid f{=}n$	*Test*
	$\mid t \,\&\, u$	*Disjunction*
	$\mid t \,;\, u$	*Conjunction*
	$\mid \neg t$	*Negation*
Programs	$p, q ::= t$	*Filter*
	$\mid f{\leftarrow}n$	*Modification*
	$\mid \mathsf{dup}$	*Duplication*
	$\mid p \,\&\, q$	*Parallel Composition*
	$\mid p \,;\, q$	*Sequential Composition*
	$\mid p \oplus_r q$	*Choice*
	$\mid p^*$	*Iteration*

Semantics $[\![p]\!] \in 2^{\mathsf{H}} \to \mathcal{M}(2^{\mathsf{H}})$

$$
\begin{aligned}
[\![0]\!](a) &\triangleq \eta(\varnothing) \\
[\![1]\!](a) &\triangleq \eta(a) \\
[\![f{=}n]\!](a) &\triangleq \eta(\{\pi :: \hbar \in a \mid \pi.f = n\}) \\
[\![\neg t]\!](a) &\triangleq [\![t]\!](a) \gg\!= \lambda b.\eta(a - b) \\
[\![f{\leftarrow}n]\!](a) &\triangleq \eta(\{\pi[f{:=}n] :: \hbar \mid \pi :: \hbar \in a\}) \\
[\![\mathsf{dup}]\!](a) &\triangleq \eta(\{\pi :: \pi :: \hbar \mid \pi :: \hbar \in a\}) \\
[\![p \,\&\, q]\!](a) &\triangleq [\![p]\!](a) \gg\!= \lambda b_1.[\![q]\!](a) \gg\!= \lambda b_2.\eta(b_1 \cup b_2) \\
[\![p \,;\, q]\!](a) &\triangleq [\![p]\!](a) \gg\!= [\![q]\!] \\
[\![p \oplus_r q]\!](a) &\triangleq r \cdot [\![p]\!](a) + (1-r) \cdot [\![q]\!](a) \\
[\![p^*]\!](a) &\triangleq \bigsqcup_{n \in \mathbb{N}} [\![p^{(n)}]\!](a)
\end{aligned}
$$

where $p^{(0)} \triangleq 1$ and $p^{(n+1)} \triangleq 1 \,\&\, p \,;\, p^{(n)}$

Probability Monad

$$\mathcal{M}(X) \triangleq \{\mu : \mathcal{B} \to [0,1] \mid \mu \text{ is a probability measure}\}$$

$$\eta(a) \triangleq \delta_a \qquad \mu \gg\!= P \triangleq \lambda A. \int_{a \in X} P(a)(A) \cdot \mu(da)$$

Figure 2. ProbNetKAT: syntax and semantics.

encode conditionals and while loops:

$$\mathsf{if}\ t\ \mathsf{then}\ p\ \mathsf{else}\ q \triangleq t \,;\, p \,\&\, \neg t \,;\, q$$
$$\mathsf{while}\ t\ \mathsf{do}\ p \triangleq (t \,;\, p)^* \,;\, \neg t$$

These encodings are well-known from KAT (Kozen 1997). While loops are useful for implementing higher level abstractions such as network virtualization in NetKAT (Smolka et al. 2015).

Example. Consider the programs

$$p_1 \triangleq \mathsf{pt}{=}1 \,;\, (\mathsf{pt}{\leftarrow}2 \,\&\, \mathsf{pt}{\leftarrow}3)$$
$$p_2 \triangleq (\mathsf{pt}{=}2 \,\&\, \mathsf{pt}{=}3) \,;\, \mathsf{dst}{\leftarrow}10.0.0.1 \,;\, \mathsf{pt}{\leftarrow}1$$

The first program forwards packets entering at port 1 out of ports 2 and 3—a simple form of multicast—and drops all other packets. The second program matches on packets coming in on ports 2 *or* 3, modifies their destination to the IP address 10.0.0.1, and sends them out through port 1. The program $p_1 \,\&\, p_2$ acts like p_1 for packets entering at port 1, and like p_2 for packets entering at ports 2 or 3.

Monads. We define the semantics of NetKAT programs parametrically over a monad $\mathcal{M}$. This allows us to give two concrete semantics at once: the classical deterministic semantics (using the identity monad), and the new probabilistic semantics (using the probability monad). For simplicity, we refrain from giving a categorical treatment and simply model a monad in terms of three components:

- a constructor $\mathcal{M}$ that lifts X to a domain $\mathcal{M}(X)$;
- an operator $\eta : X \to \mathcal{M}(X)$ that lifts objects into the domain $\mathcal{M}(X)$; and
- an infix operator
$$\gg\!= : \mathcal{M}(X) \to (X \to \mathcal{M}(X)) \to \mathcal{M}(X)$$
that lifts a function $f : X \to \mathcal{M}(X)$ to a function
$$(- \gg\!= f) : \mathcal{M}(X) \to \mathcal{M}(X)$$

These components must satisfy three axioms:

$$\eta(a) \gg\!= f \;=\; f(a) \qquad \textbf{(M1)}$$
$$m \gg\!= \eta \;=\; m \qquad \textbf{(M2)}$$
$$(m \gg\!= f) \gg\!= g \;=\; m \gg\!= (\lambda x.f(x) \gg\!= g) \qquad \textbf{(M3)}$$

The semantics of deterministic programs (not containing probabilistic choices $p \oplus_r q$) uses as underlying objects the set of packet histories 2^{H} and the identity monad $\mathcal{M}(X) = X$: η is the identify function and $x \gg\!= f$ is simply function application $f(x)$. The identity monad trivially satisfies the three axioms.

The semantics of probabilistic programs uses the probability (or Giry) monad (Giry 1982; Jones and Plotkin 1989; Ramsey and Pfeffer 2002) that maps a measurable space to the domain of probability measures over that space. The operator η maps a to the point mass (or Dirac measure) δ_a on a. Composition $\mu \gg\!= (\lambda a.\nu_a)$ can be thought of as a two-stage probabilistic experiment where the second experiment ν_a depends on the outcome a of the first experiment μ. Operationally, we first sample from μ to obtain a random outcome a; then, we sample from ν_a to obtain the final outcome b. What is the distribution over final outcomes? It can be obtained by observing that $\lambda a.\nu_a$ is a Markov kernel (§3), and so composition with μ is given by the familiar integral

$$\mu \gg\!= (\lambda a.\nu_a) = \lambda A. \int_{a \in X} \nu_a(A) \cdot \mu(da)$$

introduced in (3.1). It is well known that these definitions satisfy the monad axioms (Kozen 1981; Giry 1982; Jones and Plotkin 1989). (**M1**) and (**M2**) are trivial properties of the Lebesgue Integral. (**M3**) is essentially Fubini's theorem, which permits changing the order of integration in a double integral.

Deterministic Semantics. In deterministic NetKAT (without $p \oplus_r q$), a program p denotes a function $[\![p]\!] \in 2^{\mathsf{H}} \to 2^{\mathsf{H}}$ mapping a set of input histories $a \in 2^{\mathsf{H}}$ to a set of output histories $[\![p]\!](a)$. Note that the input and output sets do *not* encode non-determinism but represent sets of "in-flight" packets in the network. Histories record the processing done to each packet as it traverses the network. In particular, histories enable reasoning about path properties and determining which outputs were generated from common inputs.

Formally, a predicate t maps the input set a to the subset $b \subseteq a$ of histories satisfying the predicate. In particular, the false primitive 0 denotes the function mapping any input to the empty set; the true primitive 1 is the identity function; the test $f{=}n$ retains those histories with field f of the head packet equal to n; and negation $\neg t$ returns only those histories not satisfying t. Modification $f{\leftarrow}n$ sets the f-field of all head-packets to the value n. Duplication dup

extends the tails of all input histories with their head packets, thus permanently recording the current state of the packets.

Parallel composition $p \,\&\, q$ feeds the input to both p and q and takes the union of their outputs. If p and q are predicates, a history is thus in the output iff it satisfies at least one of p or q, so that union acts like logical disjunction on predicates. Sequential composition $p\,;q$ feeds the input to p and then feeds p's output to q to produce the final result. If p and q are predicates, a history is thus in the output iff it satisfies both p and q, acting like logical conjunction. Iteration p^* behaves like the parallel composition of p sequentially composed with itself zero or more times (because $\bigsqcup$ is union in 2^{H}).

Probabilistic Semantics. The semantics of ProbNetKAT is given using the probability monad applied to the set of history sets 2^{H} (seen as a measurable space). A program p denotes a function

$$[\![p]\!] \in 2^{\mathsf{H}} \to \{\mu : \mathcal{B} \to [0,1] \mid \mu \text{ is a probability measure}\}$$

mapping a set of input histories a to a *distribution* over output sets $[\![p]\!](a)$. Here, $\mathcal{B}$ denotes the Borel sets of 2^{H} (§5). Equivalently, $[\![p]\!]$ is a Markov kernel with source and destination $(2^{\mathsf{H}}, \mathcal{B})$. The semantics of all primitive programs is identical to the deterministic case, except that they now return point masses on output sets (rather than just output sets). In fact, it follows from (**M1**) that all programs without choices and iteration are point masses.

Parallel composition $p \,\&\, q$ feeds the input a to p and q, samples b_1 and b_2 from the output distributions $[\![p]\!](a)$ and $[\![q]\!](a)$, and returns the union of the samples $b_1 \cup b_2$. Probabilistic choice $p \oplus_r q$ feeds the input to both p and q and returns a convex combination of the output distributions according to r. Sequential composition $p\,;q$ is just sequential composition of Markov kernels. Operationally, it feeds the input to p, obtains a sample b from p's output distribution, and feeds the sample to q to obtain the final distribution. Iteration p^* is defined as the least fixpoint of the map on Markov kernels $X \mapsto 1 \& [\![p]\!]\,;X$, which is continuous in a DCPO that we will develop in the following sections. We will show that this definition, which is simple and is based on standard techniques from domain theory, coincides with the semantics proposed in previous work (Foster et al. 2016).

Basic Properties. To clarify the nature of predicates and other primitives, we establish two intuitive properties:

Lemma 2. *Any predicate t satisfies $[\![t]\!](a) = \eta(a \cap b_t)$, where $b_t \triangleq [\![t]\!](\mathsf{H})$ in the identity monad.*

Proof. By induction on t, using (**M1**) in the induction step. □

Lemma 3. *All atomic programs p (i.e., predicates, dup, and modifications) satisfy*

$$[\![p]\!](a) = \eta(\{f_p(h) \mid h \in a\})$$

for some partial function $f_p : \mathsf{H} \rightharpoonup \mathsf{H}$.

Proof. Immediate from Figure 2 and Lemma 2. □

Lemma 2 captures the intuition that predicates act like packet *filters*. Lemma 3 establishes that the behavior of atomic programs is captured by their behavior on individual histories.

Note however that ProbNetKAT's semantic domain is rich enough to model interactions between packets. For example, it would be straightforward to extend the language with new primitives whose behavior depends on properties of the input set of packet histories—e.g., a rate-limiting construct $@n$ that selects at most n packets uniformly at random from the input and drops all other packets. Our results continue to hold when the language is extended with arbitrary continuous Markov kernels of appropriate type, or continuous operations on such kernels.

Another important observation is that although ProbNetKAT does not include continuous distributions as primitives, there are programs that generate continuous distributions by combining choice and iteration:

Lemma 4 (Theorem 3 in Foster et al. (2016))**.** *Let π_0, π_1 denote distinct packets. Let p denote the program that changes the head packet of all inputs to either π_0 or π_1 with equal probability. Then*

$$[\![p\,;(\mathsf{dup}\,;p)^*]\!](\{\pi\}, -)$$

is a continuous distribution.

Hence, ProbNetKAT programs cannot be modeled by functions of type $2^{\mathsf{H}} \to (2^{\mathsf{H}} \to [0,1])$ in general. We need to define a measure space over 2^{H} and consider general probability measures.

5. Cantor Meets Scott

To define continuous probability measures on an infinite set X, one first needs to endow X with a topology—some additional structure that, intuitively, captures which elements of X are close to each other or approximate each other. Although the choice of topology is arbitrary in principle, different topologies induce different notions of continuity and limits, thus profoundly impacting the concepts derived from these primitives. Which topology is the "right" one for 2^{H}? A fundamental contribution of this paper is to show that there are (at least) two answers to this question:

- The initial work on ProbNetKAT (Foster et al. 2016) uses the ***Cantor topology***. This makes 2^{H} a *standard Borel space*, which is well-studied and known to enjoy many desirable properties.
- This paper is based on the ***Scott topology***, the standard choice of domain theorists. Although this topology is weaker in the sense that it lacks much of the useful structure and properties of a standard Borel space, it leads to a simpler and more computational account of ProbNetKAT's semantics.

Despite this, one view is not better than the other. The main advantage of the Cantor topology is that it allows us to reason in terms of a metric. With the Scott topology, we sacrifice this metric, but in return we are able to interpret all program operators and programs as continuous functions. The two views yield different convergence theorem, both of which are useful. Remarkably, we can have the best of both worlds: it turns out that the two topologies generate the same Borel sets, so the probability measures are the same regardless. We will prove (Theorem 21) that the semantics in Figure 2 coincides with the original semantics (Foster et al. 2016), recovering all the results from previous work. This allows us to freely switch between the two views as convenient. The rest of this section illustrates the difference between the two topologies intuitively, defines the topologies formally and endows 2^{H} with Borel sets, and proves a general theorem relating the two.

Cantor and Scott, Intuitively. The Cantor topology is best understood in terms of a distance $d(a,b)$ of history sets a, b, formally known as a *metric*. Define this metric as $d(a,b) = 2^{-n}$, where n is the length of the shortest packet history in the symmetric difference of a and b if $a \neq b$, or $d(a,b) = 0$ if $a = b$. Intuitively, history sets are close if they differ only in very long histories. This gives the following notions of limit and continuity:

- a is the limit of a sequence $a_1, a_2, \ldots$ iff the distance $d(a, a_n)$ approaches 0 as $n \to \infty$.
- a function $f : 2^{\mathsf{H}} \to [0, \infty]$ is continuous at point a iff $f(a_n)$ approaches $f(a)$ whenever a_n approaches a.

The Scott topology cannot be described in terms of a metric. It is captured by a complete partial order $(2^{\mathsf{H}}, \sqsubseteq)$ on history sets. If we choose the subset order (with suprema given by union) we obtain the following notions:

- a is the limit of a sequence $a_1 \subseteq a_2 \subseteq \ldots$ iff $a = \bigcup_{n\in\mathbb{N}} a_n$.
- a function $f : 2^{\mathsf{H}} \to [0, \infty]$ is continuous at point a iff $f(a) = \sup_{n\in\mathbb{N}} f(a_n)$ whenever a is the limit of $a_1 \subseteq a_2 \subseteq \ldots$.

Example. To illustrate the difference between Cantor-continuity and Scott-continuity, consider the function $f(a) \triangleq |a|$ that maps a history set to its (possibly infinite) cardinality. The function is not Cantor-continuous. To see this, let h_n denote a history of length n and consider the sequence of singleton sets $a_n \triangleq \{h_n\}$. Then $d(a_n, \varnothing) = 2^{-n}$, *i.e.* the sequence approaches the empty set as n approaches infinity. But the cardinality $|a_n| = 1$ does *not* approach $|\varnothing| = 0$. In contrast, the function is easily seen to be Scott-continuous.

As a second example, consider the function $f(a) \triangleq 2^{-k}$, where k is the length of the smallest history *not* in a. This function is Cantor-continuous: if $d(a_n, a) = 2^{-n}$, then

$$|f(a_n) - f(a)| \leq 2^{-(n-1)} - 2^{-n} \leq 2^{-n}$$

Therefore $f(a_n)$ approaches $f(a)$ as the distance $d(a_n, a)$ approaches 0. However, the function is not Scott-continuous[1], as all Scott-continuous functions are monotone.

Approximation. The computational importance of limits and continuity comes from the following idea. Assume a is some complicated (say infinite) mathematical object. If $a_1, a_2, \ldots$ is a sequence of simple (say finite) objects with limit a, then it may be possible to approximate a using the sequence (a_n). This gives us a computational way of working with infinite objects, even though the available resources may be fundamentally finite. Continuity captures precisely when this is possible: we can perform a computation f on a if f is continuous in a, for then we can compute the sequence $f(a_1), f(a_2), \ldots$ which (by continuity) converges to $f(a)$.

We will show later that any measure μ can be approximated by a sequence of finite measures $\mu_1, \mu_2, \ldots$, and that the expected value $\mathbf{E}_\mu[f]$ of a Scott-continuous random variable f is continuous with respect to the measure. Our implementation exploits this to compute a monotonically improving sequence of approximations for performance metrics such as latency and congestion (§9).

Notation. We use lower case letters $a, b, c \subseteq \mathsf{H}$ to denote history sets, uppercase letters $A, B, C \subseteq 2^{\mathsf{H}}$ to denote measurable sets (*i.e.*, sets of history sets), and calligraphic letters $\mathcal{B}, \mathcal{O}, \cdots \subseteq 2^{2^{\mathsf{H}}}$ to denote sets of measurable sets. For a set X, we let $\wp_\omega(X) \triangleq \{Y \subseteq X \mid |Y| < \infty\}$ denote the finite subsets of X and $\mathbf{1}_X$ the characteristic function of $\mathbf{X}$. For a statement ϕ, such as $a \subseteq b$, we let $[\phi]$ denote 1 if ϕ is true and 0 otherwise.

Cantor and Scott, Formally. For $h \in \mathsf{H}$ and $b \in 2^{\mathsf{H}}$, define

$$B_h \triangleq \{c \mid h \in c\} \qquad B_b \triangleq \bigcap_{h\in b} B_h = \{c \mid b \subseteq c\}. \tag{5.3}$$

The Cantor space topology, denoted $\mathcal{C}$, can be generated by closing $\{B_h, {\sim}B_h \mid h \in \mathsf{H}\}$ under finite intersection and arbitrary union. The Scott topology of the DCPO $(2^{\mathsf{H}}, \subseteq)$, denoted $\mathcal{O}$, can be generated by closing $\{B_h \mid h \in \mathsf{H}\}$ under the same operations and adding the empty set. The Borel algebra $\mathcal{B}$ is the smallest σ-algebra containing the Cantor-open sets, *i.e.* $\mathcal{B} \triangleq \sigma(\mathcal{C})$. We write $\mathcal{B}_b$ for the Boolean subalgebra of $\mathcal{B}$ generated by $\{B_h \mid h \in b\}$.

Lemma 5.

(i) $b \subseteq c \Leftrightarrow B_c \subseteq B_b$
(ii) $B_b \cap B_c = B_{b\cup c}$
(iii) $B_\varnothing = 2^{\mathsf{H}}$
(iv) $\mathcal{B}_{\mathsf{H}} = \bigcup_{b\in\wp_\omega(H))} \mathcal{B}_b$.

[1] with respect to the orders $\subseteq$ on 2^{H} and $\leq$ on $\mathbb{R}$

Note that if b is finite, then so is $\mathcal{B}_b$. Moreover, the atoms of $\mathcal{B}_b$ are in one-to-one correspondence with the subsets $a \subseteq b$. The subsets a determine which of the B_h occur positively in the construction of the atom,

$$\begin{aligned} A_{ab} &\triangleq \bigcap_{h\in a} B_h \cap \bigcap_{h\in b-a} {\sim}B_h \\ &= B_a - \bigcup_{a\subset c\subseteq b} B_c = \{c \in 2^{\mathsf{H}} \mid c \cap b = a\}, \end{aligned} \tag{5.4}$$

where $\subset$ denotes proper subset. The atoms A_{ab} are the basic open sets of the Cantor space. The notation A_{ab} is reserved for such sets.

Lemma 6 (Figure 3)**.** *For b finite and $a \subseteq b$, $B_a = \bigcup_{a\subseteq c\subseteq b} A_{cb}$.*

Proof. By (5.4),

$$\begin{aligned} \bigcup_{a\subseteq c\subseteq b} A_{cb} &= \bigcup_{a\subseteq c\subseteq b} \{d \in 2^{\mathsf{H}} \mid d \cap b = c\} \\ &= \{d \in 2^{\mathsf{H}} \mid a \subseteq d\} = B_a. \end{aligned}$$

□

Scott Topology Properties. Let $\mathcal{O}$ denote the family of Scott-open sets of $(2^{\mathsf{H}}, \subseteq)$. Following are some facts about this topology.

- The DCPO $(2^{\mathsf{H}}, \subseteq)$ is algebraic. The finite elements of 2^{H} are the finite subsets $a \in \wp_\omega(\mathsf{H})$, and their up-closures are $\{a\}\uparrow = B_a$.
- By Lemma 1(ii), the up-closures $\{a\}\uparrow = B_a$ form a base for the Scott topology. The sets B_h for $h \in \mathsf{H}$ are therefore a subbase.
- Thus, a subset $B \subseteq 2^{\mathsf{H}}$ is Scott-open iff there exists $F \subseteq \wp_\omega(\mathsf{H})$ such that $B = \bigcup_{a\in F} B_a$.
- The Scott topology is weaker than the Cantor space topology, *e.g.*, ${\sim}B_h$ is Cantor-open but not Scott-open. However, the Borel sets of the topologies are the same, as ${\sim}B_h$ is a Π^0_1 Borel set.[2]
- Although any Scott-open set in 2^{H} is also Cantor-open, a Scott-continuous function $f : 2^{\mathsf{H}} \to \mathbb{R}_+$ is not necessarily Cantor-continuous. This is because for Scott-continuity we consider $\mathbb{R}_+$ (ordered by $\leq$) with the Scott topology, but for Cantor-continuity we consider $\mathbb{R}_+$ with the standard Euclidean topology.
- Any Scott-continuous function $f : 2^{\mathsf{H}} \to \mathbb{R}_+$ is measurable, because the Scott-open sets of $(\mathbb{R}_+, \leq)$ (*i.e.*, the upper semi-infinite intervals $(r, \infty] = \{r\}\uparrow$ for $r \geq 0$) generate the Borell sets on $\mathbb{R}_+$.
- The open sets $\mathcal{O}$ ordered by the subset relation forms an ω-complete lattice with bottom $\varnothing$ and top $B_\varnothing = 2^{\mathsf{H}}$.
- The finite sets $a \in \wp_\omega(\mathsf{H})$ are dense and countable, thus the space is separable.
- The Scott topology is not Hausdorff, metrizable, or compact. It is not Hausdorff, as any nonempty open set contains H, but it satisfies the weaker T_0 separation property: for any pair of points a, b with $a \not\subseteq b$, $a \in B_a$ but $b \notin B_a$.
- There is an up-closed Π^0_2 Borel set with an uncountable set of minimal elements.
- There are up-closed Borel sets with no minimal elements; for example, the family of cofinite subsets of H, a Σ^0_3 Borel set.
- The compact-open sets are those of the form $F\uparrow$, where F is a finite set of finite sets. There are plenty of open sets that are not compact-open, e.g. $B_\varnothing - \{\varnothing\} = \bigcup_{h\in\mathsf{H}} B_h$.

Lemma 7 (see Halmos (1950, Theorem III.13.A))**.** *Any probability measure is uniquely determined by its values on B_b for b finite.*

[2] References to the Borel hierarchy Σ^0_n and Π^0_n refer to the Scott topology. The Cantor and Scott topologies have different Borel hierarchies.

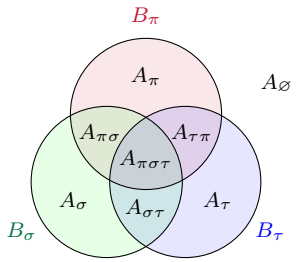


Figure 3. Relationship of the basic Scott-open sets B_a to the basic Cantor-open sets A_{ab} for $b = \{\pi, \sigma, \tau\}$ and $a \subseteq b$. The regions labeled $A_\varnothing$, A_π, $A_{\pi\sigma}$, etc. represent the basic Cantor-open sets $A_{\varnothing,b}$, $A_{\{\pi\},b}$, $A_{\{\pi,\sigma\},b}$, etc. These are the atoms of the Boolean algebra $\mathcal{B}_b$. Several basic Scott-open sets are not shown, e.g. $B_{\{\pi,\sigma\}} = B_\pi \cap B_\sigma = A_{\{\pi,\sigma\},b} \cup A_{\{\pi,\sigma,\tau\},b}$.

Proof. For b finite, the atoms of $\mathcal{B}_b$ are of the form (5.4). By the inclusion-exclusion principle (see Figure 3),

$$\mu(A_{ab}) = \mu(B_a - \bigcup_{a \subset c \subseteq b} B_c) = \sum_{a \subseteq c \subseteq b} (-1)^{|c-a|} \mu(B_c). \quad (5.5)$$

Thus μ is uniquely determined on the atoms of $\mathcal{B}_b$ and therefore on $\mathcal{B}_b$. As $\mathcal{B}_\mathsf{H}$ is the union of the $\mathcal{B}_b$ for finite b, μ is uniquely determined on $\mathcal{B}_\mathsf{H}$. By the monotone class theorem, the Borel sets $\mathcal{B}$ are the smallest monotone class containing $\mathcal{B}_\mathsf{H}$, and since $\mu(\bigcup_n A_n) = \sup_n \mu(A_n)$ and $\mu(\bigcap_n A_n) = \inf_n \mu(A_n)$, we have that μ is determined on all Borel sets. □

Extension Theorem. We now prove a useful extension theorem (Theorem 8) that identifies necessary and sufficient conditions for extending a function $\mathcal{O} \to [0,1]$ defined on the Scott-open sets of 2^H to a measure $\mathcal{B} \to [0,1]$. The theorem yields a remarkable linear correspondence between the Cantor and Scott topologies (Theorem 10). We prove it for 2^H only, but generalizations may be possible.

Theorem 8. *A function $\mu : \{B_b \mid b \text{ finite}\} \to [0,1]$ extends to a measure $\mu : \mathcal{B} \to [0,1]$ if and only if for all finite b and all $a \subseteq b$,*

$$\sum_{a \subseteq c \subseteq b} (-1)^{|c-a|} \mu(B_c) \geq 0.$$

Moreover, the extension to $\mathcal{B}$ is unique.

Proof. The condition is clearly necessary by (5.5). For sufficiency and uniqueness, we use the Carathéodory extension theorem. For each atom A_{ab} of $\mathcal{B}_b$, $\mu(A_{ab})$ is already determined uniquely by (5.5) and nonnegative by assumption. For each $B \in \mathcal{B}_b$, write B uniquely as a union of atoms and define $\mu(B)$ to be the sum of the $\mu(A_{ab})$ for all atoms A_{ab} of $\mathcal{B}_b$ contained in B. We must show that $\mu(B)$ is well-defined. Note that the definition is given in terms of b, and we must show that the definition is independent of the choice of b. It suffices to show that the calculation using atoms of $b' = b \cup \{h\}$, $h \notin b$, gives the same result. Each atom of $\mathcal{B}_b$ is the disjoint union of two atoms of $\mathcal{B}_{b'}$:

$$A_{ab} = A_{a \cup \{h\}, b \cup \{h\}} \cup A_{a, b \cup \{h\}}$$

It suffices to show the sum of their measures is the measure of A_{ab}:

$$\begin{aligned}
\mu(A_{a,b\cup\{h\}}) &= \sum_{a \subseteq c \subseteq b \cup \{h\}} (-1)^{|c-a|} \mu(B_c) \\
&= \sum_{a \subseteq c \subseteq b} (-1)^{|c-a|} \mu(B_c) + \sum_{a \cup \{h\} \subseteq c \subseteq b \cup \{h\}} (-1)^{|c-a|} \mu(B_c) \\
&= \mu(A_{ab}) - \mu(A_{a\cup\{h\}, b\cup\{h\}}).
\end{aligned}$$

To apply the Carathéodory extension theorem, we must show that μ is countably additive, *i.e.* that $\mu(\bigcup_n A_n) = \sum_n \mu(A_n)$ for any countable sequence $A_n \in \mathcal{B}_\mathsf{H}$ of pairwise disjoint sets whose union is in $\mathcal{B}_\mathsf{H}$. For finite sequences $A_n \in \mathcal{B}_\mathsf{H}$, write each A_n uniquely as a disjoint union of atoms of $\mathcal{B}_b$ for some sufficiently large b such that all $A_n \in \mathcal{B}_b$. Then $\bigcup_n A_n \in \mathcal{B}_b$, the values of the atoms are given by (5.5), and the value of $\mu(\bigcup_n A_n)$ is well-defined and equal to $\sum_n \mu(A_n)$. We cannot have an infinite set of pairwise disjoint nonempty $A_n \in \mathcal{B}_\mathsf{H}$ whose union is in $\mathcal{B}_\mathsf{H}$ by compactness. All elements of $\mathcal{B}_\mathsf{H}$ are clopen in the Cantor topology. If $\bigcup_n A_n = A \in \mathcal{B}_\mathsf{H}$, then $\{A_n \mid n \geq 0\}$ would be an open cover of A with no finite subcover. □

Cantor Meets Scott. We now establish a correspondence between the Cantor and Scott topologies on 2^H. Proofs omitted from this section can be found in the long version of this paper (Smolka et al. 2016). Consider the infinite triangular matrix E and its inverse E^{-1} with rows and columns indexed by the finite subsets of H, where

$$E_{ac} = [a \subseteq c] \qquad E^{-1}_{ac} = (-1)^{|c-a|}[a \subseteq c].$$

These matrices are indeed inverses: For $a, d \in \wp_\omega(\mathsf{H})$,

$$\begin{aligned}
(E \cdot E^{-1})_{ad} &= \sum_c E_{ac} \cdot E^{-1}_{cd} \\
&= \sum_c [a \subseteq c] \cdot [c \subseteq d] \cdot (-1)^{|d-c|} \\
&= \sum_{a \subseteq c \subseteq d} (-1)^{|d-c|} = [a = d],
\end{aligned}$$

thus $E \cdot E^{-1} = I$, and similarly $E^{-1} \cdot E = I$.

Recall that the Cantor basic open sets are the elements A_{ab} for b finite and $a \subseteq b$. Those for fixed finite b are the atoms of the Boolean algebra $\mathcal{B}_b$. They form the basis of a $2^{|b|}$-dimensional linear space. The Scott basic open sets B_a for $a \subseteq b$ are another basis for the same space. The two bases are related by the matrix $E[b]$, the $2^b \times 2^b$ submatrix of E with rows and columns indexed by subsets of b. One can show that the finite matrix $E[b]$ is invertible with inverse $E[b]^{-1} = (E^{-1})[b]$.

Lemma 9. *Let μ be a measure on 2^H and $b \in \wp_\omega(\mathsf{H})$. Let X, Y be vectors indexed by subsets of b such that $X_a = \mu(B_a)$ and $Y_a = \mu(A_{ab})$ for $a \subseteq b$. Let $E[b]$ be the $2^b \times 2^b$ submatrix of E. Then $X = E[b] \cdot Y$.*

The matrix-vector equation $X = E[b] \cdot Y$ captures the fact that for $a \subseteq b$, B_a is the disjoint union of the atoms A_{cb} of $\mathcal{B}_b$ for $a \subseteq c \subseteq b$ (see Figure 3), and consequently $\mu(B_a)$ is the sum of $\mu(A_{cb})$ for these atoms. The inverse equation $X = E[b]^{-1} \cdot Y$ captures the inclusion-exclusion principle for $\mathcal{B}_b$.

In fact, more can be said about the structure of E. For any $b \in 2^\mathsf{H}$, finite or infinite, let $E[b]$ be the submatrix of E with rows and columns indexed by the subsets of b. If $a \cap b = \varnothing$, then $E[a \cup b] = E[a] \otimes E[b]$, where $\otimes$ denotes Kronecker product. The formation of the Kronecker product requires a notion of pairing on indices, which in our case is given by disjoint set union. For example,

$$E[\{h_1\}] = \begin{array}{c} \\ \varnothing \\ \{h_1\} \end{array} \begin{array}{c} \begin{array}{cc} \varnothing & \{h_1\} \end{array} \\ \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \end{array} \qquad E[\{h_2\}] = \begin{array}{c} \\ \varnothing \\ \{h_2\} \end{array} \begin{array}{c} \begin{array}{cc} \varnothing & \{h_2\} \end{array} \\ \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \end{array}$$

$$E[\{h_1,h_2\}] = E[\{h_1\}] \otimes E[\{h_2\}]$$

$$= \begin{array}{c} \\ \varnothing \\ \{h_1\} \\ \{h_2\} \\ \{h_1,h_2\} \end{array} \begin{array}{c} \begin{array}{cccc} \varnothing & \{h_1\} & \{h_2\} & \{h_1,h_2\} \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

As $(E \otimes F)^{-1} = E^{-1} \otimes F^{-1}$ for Kronecker products of invertible matrices, we also have

$$E[\{h_1\}]^{-1} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \qquad E[\{h_2\}]^{-1} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

$$E[\{h_1,h_2\}]^{-1} = E[\{h_1\}]^{-1} \otimes E[\{h_2\}]^{-1}$$

$$= \begin{bmatrix} 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

E can be viewed as the infinite Kronecker product $\bigotimes_{h \in \mathsf{H}} E[\{h\}]$.

Theorem 10. *The probability measures on $(2^{\mathsf{H}}, \mathcal{B})$ are in one-to-one correspondence with pairs of matrices $M, N \in \mathbb{R}^{\wp_\omega(\mathsf{H}) \times \wp_\omega(\mathsf{H})}$ such that*

(i) *M is diagonal with entries in $[0,1]$,*
(ii) *N is nonnegative, and*
(iii) *$N = E^{-1}ME$.*

The correspondence associates the measure μ with the matrices

$$N_{ab} = \mu(A_{ab}) \qquad M_{ab} = [a = b] \cdot \mu(B_a). \tag{5.6}$$

6. A DCPO on Markov Kernels

In this section we define a continuous DCPO on Markov kernels. Proofs omitted from this section can be found in the long version of this paper (Smolka et al. 2016).

We will interpret all program operators defined in Figure 2 also as operators on Markov kernels: for an operator $[\![p \otimes q]\!]$ defined on programs p and q, we obtain a definition of $P \otimes Q$ on Markov kernels P and Q by replacing $[\![p]\!]$ with P and $[\![q]\!]$ with Q in the original definition. Additionally we define $\&$ on probability measures as follows:

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

The corresponding operation on programs and kernels as defined in Figure 2 can easily be shown to be equivalent to a pointwise lifting of the definition here.

For measures μ, ν on 2^{H}, define $\mu \sqsubseteq \nu$ if $\mu(B) \leq \nu(B)$ for all $B \in \mathcal{O}$. This order was first defined by Saheb-Djahromi (Saheb-Djahromi 1980).

Theorem 11 ((Saheb-Djahromi 1980))**.** *The probability measures on the Borel sets generated by the Scott topology of an algebraic DCPO ordered by $\sqsubseteq$ form a DCPO.*

Because $(2^{\mathsf{H}}, \subseteq)$ is an algebraic DCPO, Theorem 11 applies.[3] In this case, the bottom and top elements are $\delta_\varnothing$ and δ_H respectively.

Lemma 12. *$\mu \sqsubseteq \mu \,\&\, \nu$ and $\nu \sqsubseteq \mu \,\&\, \nu$.*

Surprisingly, despite Lemma 12, the probability measures do not form an upper semilattice under $\sqsubseteq$, although counterexamples are somewhat difficult to construct. See the long version of this paper (Smolka et al. 2016) for an example.

[3] A beautiful proof based on Theorem 8 can be found in the long version of this paper (Smolka et al. 2016).

Next we lift the order $\sqsubseteq$ to Markov kernels $P : 2^{\mathsf{H}} \times \mathcal{B} \to [0,1]$. The order is defined pointwise on kernels regarded as functions $2^{\mathsf{H}} \times \mathcal{O} \to [0,1]$; that is,

$$P \sqsubseteq Q \iff \forall a \in 2^{\mathsf{H}}.\, \forall B \in \mathcal{O}.\, P(a,B) \leq Q(a,B).$$

There are several ways of viewing the lifted order $\sqsubseteq$, as shown in the next lemma.

Lemma 13. *The following are equivalent:*

(i) *$P \sqsubseteq Q$, i.e., $\forall a \in 2^{\mathsf{H}}$ and $B \in \mathcal{O}$, $P(a,B) \leq Q(a,B)$;*
(ii) *$\forall a \in 2^{\mathsf{H}}$, $P(a,-) \sqsubseteq Q(a,-)$ in the DCPO $\mathcal{M}(2^{\mathsf{H}})$;*
(iii) *$\forall B \in \mathcal{O}$, $P(-,B) \sqsubseteq Q(-,B)$ in the DCPO $2^{\mathsf{H}} \to [0,1]$;*
(iv) *$\mathsf{curry}\, P \sqsubseteq \mathsf{curry}\, Q$ in the DCPO $2^{\mathsf{H}} \to \mathcal{M}(2^{\mathsf{H}})$.*

A Markov kernel $P : 2^{\mathsf{H}} \times \mathcal{B} \to [0,1]$ is *continuous* if it is Scott-continuous in its first argument; i.e., for any fixed $A \in \mathcal{O}$, $P(a,A) \leq P(b,A)$ whenever $a \subseteq b$, and for any directed set $D \subseteq 2^{\mathsf{H}}$ we have $P(\bigcup D, A) = \sup_{a \in D} P(a,A)$. This is equivalent to saying that $\mathsf{curry}\, P : 2^{\mathsf{H}} \to \mathcal{M}(2^{\mathsf{H}})$ is Scott-continuous as a function from the DCPO 2^{H} ordered by $\subseteq$ to the DCPO of probability measures ordered by $\sqsubseteq$. We will show later that all ProbNetKAT programs give rise to continuous kernels.

Theorem 14. *The continuous kernels $P : 2^{\mathsf{H}} \times \mathcal{B} \to [0,1]$ ordered by $\sqsubseteq$ form a continuous DCPO with basis consisting of kernels of the form $b\,;P\,;d$ for P an arbitrary continuous kernel and b, d filters on finite sets b and d; that is, kernels that drop all input packets except for those in b and all output packets except those in d.*

It is not true that the space of continuous kernels is algebraic with finite elements $b\,;P\,;d$. See the long version of this paper (Smolka et al. 2016) for a counterexample.

7. Continuity and Semantics of Iteration

This section develops the technology needed to establish that all ProbNetKAT programs give continuous Markov kernels and that all program operators are themselves continuous. These results are needed for the least fixpoint characterization of iteration and also pave the way for our approximation results (§8).

The key fact that underpins these results is that Lebesgue integration respects the orders on measures and on functions:

Theorem 15. *Integration is Scott-continuous in both arguments:*

(i) *For any Scott-continuous function $f : 2^{\mathsf{H}} \to [0,\infty]$, the map*

$$\mu \mapsto \int f \, d\mu \tag{7.7}$$

is Scott-continuous with respect to the order $\sqsubseteq$ on $\mathcal{M}(2^{\mathsf{H}})$.

(ii) *For any probability measure μ, the map*

$$f \mapsto \int f \, d\mu \tag{7.8}$$

is Scott-continuous with respect to the order on $[2^{\mathsf{H}} \to [0,\infty]]$.

The proofs of the remaining results in this section are somewhat long and mostly routine, but can be found in the long version of this paper (Smolka et al. 2016).

Theorem 16. *The deterministic kernels associated with any Scott-continuous function $f : D \to E$ are continuous, and the following operations on kernels preserve continuity: product, integration, sequential composition, parallel composition, choice, iteration.*

The above theorem implies that $Q \mapsto 1 \,\&\, P\,;Q$ is a continuous map on the DCPO of continuous Markov kernels. Hence $P^* = \bigsqcup_n P^{(n)}$ is well-defined as the least fixed point of that map.

Corollary 17. *Every ProbNetKAT program denotes a continuous Markov kernel.*

$$\Big(\bigsqcup_{n\geq 0} P_n\Big) \,\&\, Q \;=\; \bigsqcup_{n\geq 0}\Big(P_n \,\&\, Q\Big)$$
$$\Big(\bigsqcup_{n\geq 0} P_n\Big) \oplus_r Q \;=\; \bigsqcup_{n\geq 0}\Big(P_n \oplus_r Q\Big)$$
$$\Big(\bigsqcup_{n\geq 0} P_n\Big) ; Q \;=\; \bigsqcup_{n\geq 0}\Big(P_n ; Q\Big)$$
$$Q ; \Big(\bigsqcup_{n\geq 0} P_n\Big) \;=\; \bigsqcup_{n\geq 0}\Big(Q ; P_n\Big)$$
$$\Big(\bigsqcup_{n\geq 0} P_n\Big)^* \;=\; \bigsqcup_{n\geq 0}\Big(P_n^*\Big)$$

Figure 4. Scott-Continuity of program operators (Theorem 18).

The next theorem is the key result that enables a practical implementation:

Theorem 18. *The following semantic operations are continuous functions of the DCPO of continuous kernels: product, parallel composition,* curry*, sequential composition, choice, iteration. (Figure 4.)*

The semantics of iteration presented in (Foster et al. 2016), defined in terms of an infinite process, coincides with the least fixpoint semantics presented here. The key observation is the relationship between weak convergence in the Cantor topology and fixpoint convergence in the Scott topology:

Theorem 19. *Let A be a directed set of probability measures with respect to $\sqsubseteq$ and let $f : 2^{\mathsf{H}} \to [0,1]$ be a Cantor-continuous function. Then*

$$\lim_{\mu\in A}\int_{c\in 2^{\mathsf{H}}} f(c)\cdot d\mu = \int_{c\in 2^{\mathsf{H}}} f(c)\cdot d(\bigsqcup A).$$

This theorem implies that $P^{(n)}$ weakly converges to P^* in the Cantor topology. (Foster et al. 2016) showed that $P^{(n)}$ also weakly converges to $P^{\circledast}$ in the Cantor topology, where we let $P^{\circledast}$ denote the iterate of P as defined in (Foster et al. 2016). But since $(2^{\mathsf{H}}, \mathcal{C})$ is a Polish space, this implies that $P^* = P^{\circledast}$.

Lemma 20. *In a Polish space D, the values of*

$$\int_{a\in D} f(a)\cdot \mu(da)$$

for continuous $f : D \to [0,1]$ determine μ uniquely.

Corollary 21. $P^{\circledast} = \bigsqcup_n P^{(n)} = P^*$.

8. Approximation

We now formalize a notion of approximation for ProbNetKAT programs. Given a program p, we define the n-th approximant $[p]_n$ inductively as

$$\begin{aligned}
[p]_n &\triangleq p \quad (\text{for } p \text{ primitive})\\
[q \oplus_r r]_n &\triangleq [q]_n \oplus_r [r]_n\\
[q \,\&\, r]_n &\triangleq [q]_n \,\&\, [r]_n\\
[q ; r]_n &\triangleq [q]_n ; [r]_n\\
[q^*]_n &\triangleq ([q]_n)^{(n)}
\end{aligned}$$

Intuitively, $[p]_n$ is just p where iteration $-^*$ is replaced by bounded iteration $-^{(n)}$. Let $[\![p]\!]_n$ denote the Markov kernel obtained from the n-th approximant: $[\![[p]_n]\!]$.

Theorem 22. *The approximants of a program p form a $\sqsubseteq$-increasing chain with supremum p, that is*

$$[\![p]\!]_1 \sqsubseteq [\![p]\!]_2 \sqsubseteq \ldots \qquad \textit{and} \qquad \bigsqcup_{n\geq 0} [\![p]\!]_n = [\![p]\!]$$

Proof. By induction on p and continuity of the operators. □

This means that any program can be approximated by a sequence of star-free programs, which—in contrast to general programs (Lemma 4)—can only produce finite distributions. These finite distributions are sufficient to compute the expected values of Scott-continuous random variables:

Corollary 23. *Let $\mu \in \mathcal{M}(2^{\mathsf{H}})$ be an input distribution, p be a program, and $Q : 2^{\mathsf{H}} \to [0,\infty]$ be a Scott-continuous random variable. Let*

$$\nu \triangleq \mu \gg\!= [\![p]\!] \qquad \textit{and} \qquad \nu_n \triangleq \mu \gg\!= [\![p]\!]_n$$

denote the output distribution and its approximations. Then

$$\mathop{\mathbf{E}}_{\nu_0}[Q] \leq \mathop{\mathbf{E}}_{\nu_1}[Q] \leq \ldots \qquad \textit{and} \qquad \sup_{n\in\mathbb{N}} \mathop{\mathbf{E}}_{\nu_n}[Q] = \mathop{\mathbf{E}}_{\nu}[Q]$$

Proof. Follows directly from Theorems 22 and 15. □

Note that the approximations ν_n of the output distribution ν are always finite, provided the input distribution μ is finite. Computing an expected value with respect to ν thus simply amounts to computing a sequence of finite sums $\mathbf{E}_{\nu_0}[Q], \mathbf{E}_{\nu_1}[Q], \ldots$, which is guranteed to converge monotonically to the analytical solution $\mathbf{E}_{\nu}[Q]$. The approximate semantics $[\![-]\!]_n$ can be thought of as an executable version of the denotational semantics $[\![-]\!]$. We implement it in the next section and use it to approximate network metrics based on the above result. The rest of this section gives more general approximation results for measures and kernels on 2^{H}, and shows that we can in fact handle continuous input distributions as well.

A measure is a *finite discrete measure* if it is of the form $\sum_{a\in F} r_a \delta_a$, where $F \in \wp_\omega(\wp_\omega(H))$ is a finite set of finite subsets of packet histories H, $r_a \geq 0$ for all $a \in F$, $\sum_{a\in F} r_a = 1$. Without loss of generality, we can write any such measure in the form $\sum_{a\subseteq b} r_a \delta_a$ for any $b \in \wp_\omega(H)$ such that $\bigcup F \subseteq b$ by taking $r_a = 0$ for $a \in 2^b - F$.

Saheb-Djahromi (Saheb-Djahromi 1980, Theorem 3) shows that every measure is a supremum of a directed set of finite discrete measures. This implies that the measures form a continuous DCPO with basis consisting of the finite discrete measures. In our model, the finite discrete measures have a particularly nice characterization:

For μ a measure and $b \in \wp_\omega(H)$, define the *restriction of μ to b* to be the finite discrete measure

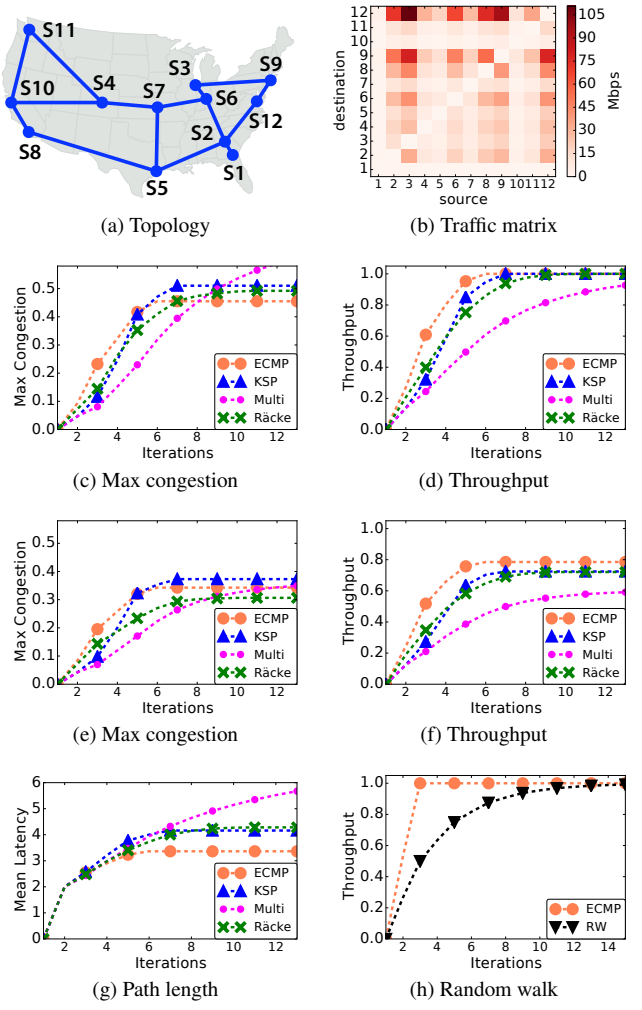
$$\mu \restriction b \triangleq \sum_{a\subseteq b} \mu(A_{ab})\delta_a.$$

Theorem 24. *The set $\{\mu \restriction b \mid b \in \wp_\omega(H)\}$ is a directed set with supremum μ. Moreover, the DCPO of measures is continuous with basis consisting of the finite discrete measures.*

We can lift the result to continuous kernels, which implies that *every program is approximated arbitrarily closely by programs whose outputs are finite discrete measures*.

Lemma 25. *Let $b \in \wp_\omega(H)$. Then $(P ; b)(a,-) = P(a,-) \restriction b$.*

Now suppose the input distribution μ in Corollary 23 is continuous. By Theorem 24, μ is the supremum of an increasing chain of finite discrete measures $\mu_1 \sqsubseteq \mu_2 \sqsubseteq \ldots$. If we redefine $\nu_n \triangleq \mu_n \gg\!= [\![p]\!]_n$ then by Theorem 15 the ν_n still approximate the output distribution ν and Corollary 23 continues to hold. Even though the input distribution is now continuous, the output distribution can still be approximated by a chain of finite distributions



(a) Topology (b) Traffic matrix

(c) Max congestion (d) Throughput

(e) Max congestion (f) Throughput

(g) Path length (h) Random walk

Figure 5. Case study with Abilene: (c, d) without loss. (e, f) with faulty links. (h) random walk in 4-cycle: all packets are eventually delivered.

and hence the expected value can still be approximated by a chain of finite sums.

9. Implementation and Case Studies

We built a simple interpreter for ProbNetKAT in OCaml that implements the denotational semantics as presented in Figure 2. Given a query, the interpreter approximates the answer through a monotonically increasing sequence of values (Theorems 22 and 23). Although preliminary in nature—more work on data structures and algorithms for manipulating distributions would be needed to obtain an efficient implementation—we were able to use our implementation to conduct several case studies involving probabilistic reasoning about properties of a real-world network: Internet2's Abilene backbone.

Routing. In the networking literature, a large number of traffic engineering (TE) approaches have been explored. We built ProbNetKAT implementations of each of the following routing schemes:

- **Equal Cost Multipath Routing (ECMP):** The network uses all least-cost paths between each source-destination pair, and maps incoming traffic flows onto those paths randomly. ECMP can reduce congestion and increase throughput, but can also perform poorly when multiple paths traverse the same bottleneck link.
- **k-Shortest Paths (KSP):** The network uses the top k-shortest paths between each pair of hosts, and again maps incoming traffic flows onto those paths randomly. This approach inherits the benefits of ECMP and provides improved fault-tolerance properties since it always spreads traffic across k distinct paths.
- **Multipath Routing (Multi):** This is similar to KSP, except that it makes an independent choice from among the k-shortest paths at each hop rather than just once at ingress. This approach dynamically routes around bottlenecks and failures but can use extremely long paths—even ones containing loops.
- **Oblivious Routing (Räcke):** The network forwards traffic using a pre-computed probability distribution on carefully constructed overlays. The distribution is constructed in such a way that guarantees worst-case congestion within a polylogarithmic factor of the optimal scheme, regardless of the demands for traffic.

Note that all of these schemes rely on some form of randomization and hence are probabilistic in nature.

Traffic Model. Network operators often use traffic models constructed from historical data to predict future performance. We built a small OCaml tool that translates traffic models into ProbNetKAT programs using a simple encoding. Assume that we are given a traffic matrix (TM) that relates pairs of hosts (u, v) to the amount of traffic that will be sent from u to v. By normalizing each TM entry using the aggregate demand $\sum_{(u,v)} TM(u, v)$, we get a probability distribution d over pairs of hosts. For a pair of source and destination (u, v), the associated probability $d(u, v)$ denotes the amount of traffic from u to v relative to the total traffic. Assuming uniform packet sizes, this is also the probability that a random packet generated in the network has source u and destination v. So, we can encode a TM as a program that generates packets according to d:

$$inp \triangleq \bigoplus_{d(u,v)} \pi_{(u,v)}!$$

$$\text{where, } \pi_{(u,v)}! \triangleq \mathsf{src}{\leftarrow}u \,;\, \mathsf{dst}{\leftarrow}v \,;\, \mathsf{sw}{\leftarrow}u$$

$\pi_{(u,v)}!$ generates a packet at u with source u and destination v. For any (non-empty) input, inp generates a distribution μ on packet histories which can be fed to the network program. For instance, consider a uniform traffic distribution for our 4-switch example (see Figure 1) where each node sends equal traffic to every other node. There are twelve (u, v) pairs with $u \neq v$. So, $d(u, v)_{u \neq v} = \frac{1}{12}$ and $d(u, u) = 0$. We also store the aggregate demand as it is needed to model queries such as expected link congestion, throughput etc.

Queries. Our implementation can be used to answer probabilistic queries about a variety of network performance properties. §2 showed an example of using a query to compute expected congestion. We can also measure expected mean latency in terms of path length:

```
let path_length (h:Hist.t) : Real.t =
  Real.of_int ((Hist.length h)/2 + 1)
let lift_query_avg
  (q:Hist.t -> Real.t) : (HSet.t -> Real.t) =
  fun hset ->
    let n = HSet.length hset in
    if n = 0 then Real.zero else
    let sum = HSet.fold hset ~init:Real.zero
      ~f:(fun acc h -> Real.(acc + q h)) in
    Real.(sum / of_int n)
```

The latency function (`path_length`) counts the number of switches in a history. We lift this function to sets and compute the expectation (`lift_query_avg`) by computing the average over all histories in the set (after discarding empty sets).

Case Study: Abilene. To demonstrate the applicability of ProbNetKAT for reasoning about a real network, we performed a case

study based on the topology and traffic demands from Internet2's Abilene backbone network as shown in Figure 5 (a). We evaluate the traffic engineering approaches discussed above by modeling traffic matrices based on NetFlow traces gathered from the production network. A sample TM is depicted in Figure 5 (b).

Figures 5 (c,d,g) show the expected maximum congestion, throughput and mean latency. Because we model a network using the Kleene star operator, we see that the values converge monotonically as the number of iterations used to approximate Kleene star increases, as guaranteed by Corollary 23.

Failures. Network failures such as a faulty router or a link going down are common in large networks (Gill et al. 2011). Hence, it is important to be able to understand the behavior and performance of a network in the presence of failures. We can model failures by assigning empirically measured probabilities to various components—e.g., we can modify our encoding of the topology so that every link in the network drops packets with probability $\frac{1}{10}$:

$$\begin{aligned}\ell_{1,2} \triangleq\ & \mathsf{sw}{=}S_1 \,;\, \mathsf{pt}{=}2 \,;\, \mathsf{dup} \,;\, ((\mathsf{sw}{\leftarrow}S_2 \,;\, \mathsf{pt}{\leftarrow}1 \,;\, \mathsf{dup}) \oplus_{0.9} 0) \\ \&\ & \mathsf{sw}{=}S_2 \,;\, \mathsf{pt}{=}1 \,;\, \mathsf{dup} \,;\, ((\mathsf{sw}{\leftarrow}S_1 \,;\, \mathsf{pt}{\leftarrow}2 \,;\, \mathsf{dup}) \oplus_{0.9} 0)\end{aligned}$$

Figures 5 (e-f) show the network performance for Abilene under this failure model. As expected, congestion and throughput decrease as more packets are dropped. As every link drops packets probabilistically, the relative fraction of packets delivered using longer links decreases—hence, there is a decrease in mean latency.

Loop detection. Forwarding loops in a network are extremely undesirable as they increase congestion and can even lead to black holes. With probabilistic routing, not all loops will necessarily result in a black hole—if there is a non-zero probability of exiting a loop, every packet entering it will eventually exit. Consider the example of random walk routing in the four-node topology from Figure 1. In a random walk, a switch either forwards traffic directly to its destination or to a random neighbor. As packets are never duplicated and only exit the network when they reach their destination, the total throughput is equivalent to the fraction of packets that exit the network. Figure 5 (h) shows that the fraction of packets exiting increases monotonically with number of iterations and converges to 1. Moreover, histories can be queried to test if it encountered a topological loop by checking for duplicate locations. Hence, given a model that computes all possible history prefixes that appear in the network, we can query it for presence of loops. We do this by removing out from our standard network model and using $in;(p;t)^*;p$ instead. This program generates the required distribution on history prefixes. Moreover, if we generalize packets with wildcard fields, similar to HSA (Kazemian et al. 2012), we can check for loops symbolically. We have extended our implementation in this way, and used it to check whether the network exhibits loops on a number of routing schemes based on probabilistic forwarding.

10. Related Work

This paper builds on previous work on NetKAT (Anderson et al. 2014; Foster et al. 2015) and ProbNetKAT (Foster et al. 2016), but develops a semantics based on ordered domains as well as new applications to traffic engineering.

Domain Theory. The domain-theoretic treatment of probability measures goes back to the seminal work of Saheb-Djahromi (Saheb-Djahromi 1980), who was the first to identify and study the CPO of probability measures. Jones and Plotkin (Jones and Plotkin 1989; Jones 1989) generalized and extended this work by giving a category-theoretical treatment and proving that the probabilistic powerdomain is a monad. It is an open problem if there exists a cartesian-closed category of continuous DCPOs that is closed under the probabilistic powerdomain; see (Jung and Tix 1998) for a discussion. This is an issue for higher-order probabilistic languages, but not for ProbNetKAT, which is strictly first-order. Edalat (Edalat 1994, 1996; Edalat and Heckmann 1998) gives a computational account of measure theory and integration for general metric spaces based on domain theory. More recent papers on probabilistic powerdomains are (Jung and Tix 1998; Heckmann 1994; Graham 1988). All this work is ultimately based on Scott's pioneering work (Scott 1972).

Probabilistic Logic and Semantics. Computational models and logics for probabilistic programming have been extensively studied. Denotational and operational semantics for probabilistic while programs were first studied by Kozen (Kozen 1981). Early logical systems for reasoning about probabilistic programs were proposed in (Kozen 1985; Ramshaw 1979; Saheb-Djahromi 1978). There are also numerous recent efforts (Gordon et al. 2014; Gretz et al. 2015; Kozen et al. 2013; Larsen et al. 2012; Morgan et al. 1996). Sankaranarayanan et al. (Sankaranarayanan et al. 2013) propose static analysis to bound the the value of probability queries. Probabilistic programming in the context of artificial intelligence has also been extensively studied in recent years (Borgström et al. 2011; Roy 2011). Probabilistic automata in several forms have been a popular model going back to the early work of Paz (Paz 1971), as well as more recent efforts (McIver et al. 2008; Segala 2006; Segala and Lynch 1995). Denotational models combining probability and nondeterminism have been proposed by several authors (McIver and Morgan 2004; Tix et al. 2009; Varacca and Winskel 2006), and general models for labeled Markov processes, primarily based on Markov kernels, have been studied extensively (Doberkat 2007; Panangaden 1998, 2009).

Our semantics is also related to the work on event structures (Nielsen et al. 1979; Varacca et al. 2006). A (Prob)NetKAT program denotes a simple (probabilistic) event structure: packet histories are events with causal dependency given by extension and with all finite subsets consistent. We have to yet explore whether the event structure perspective on our semantics could lead to further applications and connections to e.g. (concurrent) games.

Networking. Network calculus is a general framework for analyzing network behavior using tools from queuing theory (Cruz. 1991). It has been used to reason about quantitative properties such as latency, bandwidth, and congestion. The stochastic branch of network calculus provides tools for reasoning about the probabilistic behavior, especially in the presence of statistical multiplexing, but is often considered difficult to use. In contrast, ProbNetKAT is a self-contained framework based on a precise denotational semantics.

Traffic engineering has been extensively studied and a wide variety of approaches have been proposed for data-center networks (Al-Fares et al. 2010; Jeyakumar et al. 2013; Perry et al. 2014; Zhang-Shen and McKeown 2005; Shieh et al. 2010) and wide-area networks (Hong et al. 2013; Jain et al. 2013; Fortz et al. 2002; Applegate and Cohen 2003; Räcke 2008; Kandula et al. 2005; Suchara et al. 2011; He and Rexford 2008). These approaches optimize for metrics such as congestion, throughput, latency, fault tolerance, fairness etc. Optimal techniques typically have high overheads (Danna et al. 2012), but oblivious (Kodialam et al. 2009; Applegate and Cohen 2003) and hybrid approaches with near-optimal performance (Hong et al. 2013; Jain et al. 2013) have recently been adopted.

11. Conclusion

This paper presents a new order-theoretic semantics for ProbNetKAT in the style of classical domain theory. The semantics allows a standard least-fixpoint treatment of iteration, and enables new modes of reasoning about the probabilistic network behavior. We have used these theoretical tools to analyze several randomized routing protocols on real-world data.

The main technical insight is that all programs and the operators defined on them are continuous, provided we consider the right notion of continuity: that induced by the Scott topology. Continuity enables precise approximation, and we exploited this to build an implementation. But continuity is also a powerful tool for reasoning that we expect to be very helpful in the future development of ProbNetKAT's meta theory. To establish continuity we had to switch from the Cantor to the Scott topology, and give up reasoning in terms of a metric. Luckily we were able to show a strong correspondence between the two topologies and that the Cantor-perspective and the Scott-perspective lead to equivalent definitions of the semantics. This allows us to choose whichever perspective is best-suited for the task at hand.

Future Work. The results of this paper are general enough to accommodate arbitrary extensions of ProbNetKAT with continuous Markov kernels or continuous operators on such kernels. An obvious next step is therefore to investigate extension of the language that would enable richer network models. Previous work on deterministic NetKAT included a decision procedure and a sound and complete axiomatization. In the presence of probabilities we expect a decision procedure will be hard to devise, as witnessed by several undecidability results on probabilistic automata. We intend to explore decision procedures for restricted fragments of the language. Another interesting direction is to compile ProbNetKAT programs into suitable automata that can then be analyzed by a probabilistic model checker such as PRISM (Kwiatkowska et al. 2011). A sound and complete axiomatization remains subject of further investigation, we can draw inspiration from recent work (Kozen et al. 2013; Mardare et al. 2016). Another opportunity is to investigate a weighted version of NetKAT, where instead of probabilities we consider weights from an arbitrary semiring, opening up several other applications—e.g. in cost analysis. Finally, we would like to explore efficient implementation techniques including compilation, as well as approaches based on sampling, following several other probabilistic languages (Park et al. 2008; Borgström et al. 2011).

Acknowledgments

The authors wish to thank Arthur Azevedo de Amorim, David Kahn, Anirudh Sivaraman, Hongseok Yang, the Cornell PLDG, and the Barbados Crew for insightful discussions and helpful comments. Our work is supported by the National Security Agency; the National Science Foundation under grants CNS-1111698, CNS-1413972, CCF-1422046, CCF-1253165, and CCF-1535952; the Office of Naval Research under grant N00014-15-1-2177; the Dutch Research Foundation (NWO) under project numbers 639.021.334 and 612.001.113; and gifts from Cisco, Facebook, Google, and Fujitsu.

References

S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, pages 19–19, 2010.

C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. In *POPL*, pages 113–126, January 2014.

D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In *SIGCOMM*, pages 313–324, Aug. 2003.

J. Borgström, A. D. Gordon, M. Greenberg, J. Margetson, and J. V. Gael. Measure transformer semantics for Bayesian machine learning. In *ESOP*, July 2011.

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

E. Danna, S. Mandal, and A. Singh. A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering. In *INFOCOM*, pages 846–854, 2012.

E.-E. Doberkat. *Stochastic Relations: Foundations for Markov Transition Systems*. Studies in Informatics. Chapman Hall, 2007.

R. Durrett. *Probability: Theory and Examples*. Cambridge University Press, 2010.

A. Edalat. Domain theory and integration. In *LICS*, pages 115–124, 1994.

A. Edalat. The scott topology induces the weak topology. In *LICS*, pages 372–381, 1996.

A. Edalat and R. Heckmann. A computational model for metric spaces. *Theoretical Computer Science*, 193(1):53–73, 1998.

B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, 40(10):118–124, Oct. 2002.

N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. In *ICFP*, pages 279–291, Sept. 2011.

N. Foster, D. Kozen, M. Milano, A. Silva, and L. Thompson. A coalgebraic decision procedure for NetKAT. In *POPL*, pages 343–355. ACM, Jan. 2015.

N. Foster, D. Kozen, K. Mamouras, M. Reitblatt, and A. Silva. Probabilistic NetKAT. In *ESOP*, pages 282–309, Apr. 2016.

P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *SIGCOMM*, pages 350–361, Aug. 2011.

M. Giry. A categorical approach to probability theory. In *Categorical aspects of topology and analysis*, pages 68–85. Springer, 1982.

A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *FOSE*, May 2014.

S. Graham. Closure properties of a probabilistic powerdomain construction. In *MFPS*, pages 213–233, 1988.

F. Gretz, N. Jansen, B. L. Kaminski, J. Katoen, A. McIver, and F. Olmedo. Conditioning in probabilistic programming. *CoRR*, abs/1504.00198, 2015.

P. R. Halmos. *Measure Theory*. Van Nostrand, 1950.

J. He and J. Rexford. Toward internet-wide multipath routing. *IEEE Network Magazine*, 22(2):16–21, 2008.

R. Heckmann. Probabilistic power domains, information systems, and locales. In *MFPS*, volume 802, pages 410–437, 1994.

C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *SIGCOMM*, pages 15–26, Aug. 2013.

S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, pages 3–14, Aug. 2013.

V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, A. Greenberg, and C. Kim. Eyeq: Practical network performance isolation at the edge. In *NSDI*, pages 297–311, 2013.

C. Jones. *Probabilistic Non-determinism*. PhD thesis, University of Edinburgh, August 1989.

C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *LICS*, pages 186–195, 1989.

A. Jung and R. Tix. The troublesome probabilistic powerdomain. *ENTCS*, 13:70–91, 1998.

S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *SIGCOMM*, pages 253–264, Aug. 2005.

P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *NSDI*, 2012.

A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *NSDI*, 2013.

M. Kodialam, T. Lakshman, J. B. Orlin, and S. Sengupta. Oblivious routing of highly variable traffic in service overlays and ip backbones. *IEEE/ACM Transactions on Networking (TON)*, 17(2):459–472, 2009.

D. Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22: 328–350, 1981.

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

D. Kozen. Kleene algebra with tests. *ACM TOPLAS*, 19(3):427–443, May 1997.

D. Kozen, R. Mardare, and P. Panangaden. Strong completeness for Markovian logics. In *MFCS*, pages 655–666, August 2013.

M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

K. G. Larsen, R. Mardare, and P. Panangaden. Taking it to the limit: Approximate reasoning for Markov processes. In *MFCS*, 2012.

R. Mardare, P. Panangaden, and G. Plotkin. Quantitative algebraic reasoning. In *LICS*, 2016.

J. McClurg, H. Hojjat, N. Foster, and P. Cerny. Event-driven network programming. In *PLDI*, June 2016.

A. McIver and C. Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems*. Springer, 2004.

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.

C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing software defined networks. In *NSDI*, Apr. 2013.

C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM TOPLAS*, 18(3):325–353, May 1996.

T. Nelson, A. D. Ferguson, M. J. G. Scheer, and S. Krishnamurthi. Tierless programming and reasoning for software-defined networks. In *NSDI*, 2014.

M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains. In *Semantics of Concurrent Computation*, pages 266–284, 1979.

P. Panangaden. Probabilistic relations. In *PROBMIV*, pages 59–74, 1998.

P. Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009.

S. Park, F. Pfenning, and S. Thrun. A probabilistic language based on sampling functions. *ACM TOPLAS*, 31(1):1–46, Dec. 2008.

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

J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A Centralized Zero-Queue Datacenter Network. In *SIGCOMM*, August 2014.

G. D. Plotkin. Probabilistic powerdomains. In *CAAP*, pages 271–287, 1982.

H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *STOC*, pages 255–264, 2008.

N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL*, pages 154–165, Jan. 2002.

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

M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *SIGCOMM*, pages 323–334, Aug. 2012.

D. M. Roy. *Computability, inference and modeling in probabilistic programming*. PhD thesis, Massachusetts Institute of Technology, 2011.

N. Saheb-Djahromi. Probabilistic LCF. In *MFCS*, pages 442–451, May 1978.

N. Saheb-Djahromi. CPOs of measures for nondeterminism. *Theoretical Computer Science*, 12:19–37, 1980.

S. Sankaranarayanan, A. Chakarov, and S. Gulwani. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *PLDI*, pages 447–458, June 2013.

D. S. Scott. Continuous lattices. In *Toposes, Algebraic Geometry and Logic*, pages 97–136. 1972.

R. Segala. Probability and nondeterminism in operational models of concurrency. In *CONCUR*, pages 64–78, 2006.

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

A. Shieh, S. Kandula, A. G. Greenberg, and C. Kim. Seawall: Performance isolation for cloud datacenter networks. In *HotCloud*, 2010.

S. Smolka, S. Eliopoulos, N. Foster, and A. Guha. A fast compiler for NetKAT. In *ICFP*, Sept. 2015.

S. Smolka, P. Kumar, N. Foster, D. Kozen, and A. Silva. Cantor meets scott: Semantic foundations for probabilistic networks. *CoRR*, abs/1607.05830, 2016. URL `http://arxiv.org/abs/1607.05830`.

M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford. Network architecture for joint failure recovery and traffic engineering. *ACM SIGMETRICS*, pages 97–108, 2011.

R. Tix, K. Keimel, and G. Plotkin. Semantic domains for combining probability and nondeterminism. *ENTCS*, 222:3–99, 2009.

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

D. Varacca, H. Völzer, and G. Winskel. Probabilistic event structures and domains. *TCS*, 358(2-3):173–199, 2006.

A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak. Maple: Simplifying SDN programming using algorithmic policies. In *SIGCOMM*, 2013.

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