Probabilistic NetKAT

Nate Foster[1], Dexter Kozen[1], Konstantinos Mamouras[2*],
Mark Reitblatt[3*], and Alexandra Silva[4]

[1] Cornell University
[2] University of Pennsylvania
[3] Facebook
[4] University College London

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

1 Introduction

Formal specification and verification of networks has become a reality in recent years with the emergence of network-specific programming languages and property-checking tools. Programming languages like Frenetic [11], Pyretic [37], Maple [53], FlowLog [39], and others are enabling programmers to specify the intended behavior of a network in terms of high-level constructs such as boolean predicates and functions on packets. Verification tools like Header Space Analysis [22], VeriFlow [23], and NetKAT [12] are making it possible to check properties such as connectivity, loop freedom, and traffic isolation automatically.

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

- **Congestion:** the network operator wishes to calculate the expected degree of congestion on each link given a model of the demands for traffic.
- **Failure:** the network operator wishes to calculate the probability that packets will be delivered to their destination, given that devices and links fail with a certain probability.

[*] Work performed at Cornell University.

– **Randomization:** the network operator wishes to use randomized routing schemes such as equal cost multi-path routing (ECMP) or Valiant load balancing (VLB) to balance load across multiple paths.

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

Background. As its name suggests, ProbNetKAT is based on NetKAT, a network programming language developed in prior work [1, 12, 49]. NetKAT is an extension of Kleene algebra with tests (KAT), an algebraic system for propositional verification of imperative programs that has been extensively studied for nearly two decades [27]. At the level of syntax, NetKAT offers a rich collection of intuitive constructs including: conditional tests; primitives for modifying packet headers and encoding topologies; and sequential, parallel, and iteration operators. The semantics of the language can be understood in terms of a denotational model based on functions from packet histories to sets of packet histories (where a history records the path through the network taken by a packet) or equivalently, using an equational deductive system that is sound and complete with respect to the denotational semantics. NetKAT has a PSPACE decision procedure that exploits the coalgebraic structure of the language and can solve many verification problems automatically [12]. Several practical applications of NetKAT have been developed, including algorithms for testing reachability and non-interference, a syntactic correctness proof for a compiler that translates programs to hardware instructions for SDN switches, and an implementation that handles programs written against virtual topologies [49].

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

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

distributions—the latter models multicast—as is the Kleene star operator since it requires showing that fixpoints exist.

Evaluation. To evaluate our design, we prove that the probabilistic semantics of ProbNetKAT is a conservative extension of the standard NetKAT semantics. This is a crucial point of our work: the language developed in this paper is based on NetKAT, which in turn is an extension of KAT, a well-established framework for program verification. Hence, this work can be seen as the next step in the modular development of an expressive network programming language, with increasingly sophisticated set of features, based on a sound and long-standing mathematical foundation. We also develop a number of case studies that illustrate the use of the semantics on examples inspired by real-world scenarios. Our case studies model congestion, failure, and randomization, as discussed above, as well as a gossip protocol that disseminates information through a network.

Contributions. Overall, the contributions of this paper are as follows:

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

Outline. The rest of this paper is organized as follows: §2 introduces the basic ideas behind ProbNetKAT through an example; §3 reviews concepts from measure theory needed to define the semantics; §4 and §5 present the syntax and semantics of ProbNetKAT; §6 further illustrates the semantics by proving conservativity and some natural equations; §8 discusses applications of the semantics to real-world examples. We discuss related work in §9 and conclude in §10. Proofs and further details on the semantics of iteration can be found in the extended version of this paper [13].

2 Overview

This section introduces ProbNetKAT using a simple example and discusses some of the key challenges in designing the language.

Preliminaries. A *packet* π is a record with fields x_1 to x_k ranging over standard header fields (Ethernet and IP addresses, TCP ports, etc.) as well as special switch and port fields indicating its location in the network:

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

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

Example. Consider the network shown in Fig. 1 with six switches arranged into a "barbell" topology. Suppose the network operator wants to configure the switches to forward traffic on the two left-to-right paths from I_1 to E_1 and I_2 to E_2. We can implement this in ProbNetKAT as follows:

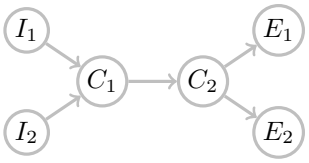

Fig. 1. Barbell topology.

$$p \triangleq (sw = I_1; \texttt{dup}; sw \leftarrow C_1; \texttt{dup}; sw \leftarrow C_2; \texttt{dup}; sw \leftarrow E_1) \,\&\, (sw = I_2; \texttt{dup}; sw \leftarrow C_1; \texttt{dup}; sw \leftarrow C_2; \texttt{dup}; sw \leftarrow E_2)$$

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

Now suppose the network operator wants to calculate not just *where* traffic is routed but also *how much* traffic is sent across each link. The deterministic semantics we have seen so far calculates the trajectories that packets take through the network. Hence, for a given set of inputs, we can use the semantics to calculate the set of output histories and then count how many packets traversed each link, yielding an upper bound on congestion. But now suppose we want to *predict* the amount of congestion that could be induced from a model that encodes expectations about the set of possible inputs. Such models, which are often represented as traffic matrices, can be built from historical monitoring data using a variety of statistical techniques [36]. Unfortunately, even simple calculations of how much congestion is likely to occur on a given link cannot be performed using the deterministic semantics.

Returning to the example, suppose that we wish to represent the following traffic model in ProbNetKAT: in each time period, the number of packets originating at I_1 is either 0, 1 or 2, with equal probability, and likewise for I_2. Let π_1 to π_4 be distinct packets, and write $\pi_{I_j,i}!$ for the sequence of assignments that produces the packet π_i located at switch I_j. We can encode the distributions at I_1 and I_2 using the following ProbNetKAT terms:[5]

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

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

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

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

$\{\,\}$,
$\{E_{1,1}{:}C_{2,1}{:}C_{1,1}{:}I_{1,1}\}$,
$\{E_{1,1}{:}C_{2,1}{:}C_{1,1}{:}I_{1,1},\ E_{1,2}{:}C_{2,2}{:}C_{1,2}{:}I_{1,2}\}$,
$\{E_{2,3}{:}C_{2,3}{:}C_{1,3}{:}I_{2,3}\}$,
$\{E_{2,3}{:}C_{2,3}{:}C_{1,3}{:}I_{2,3},\ E_{2,4}{:}C_{2,4}{:}C_{1,4}{:}I_{2,4}\}$,
$\{E_{1,1}{:}C_{2,1}{:}C_{1,1}{:}I_{1,1},\ E_{2,3}{:}C_{2,3}{:}C_{1,3}{:}I_{2,3}\}$
$\{E_{1,1}{:}C_{2,1}{:}C_{1,1}{:}I_{1,1},\ E_{1,2}{:}C_{2,2}{:}C_{1,2}{:}I_{1,2},\ E_{2,3}{:}C_{2,3}{:}C_{1,3}{:}I_{2,3}\}$
$\{E_{1,1}{:}C_{2,1}{:}C_{1,1}{:}I_{1,1},\ E_{2,3}{:}C_{2,3}{:}C_{1,3}{:}I_{2,3},\ E_{2,4}{:}C_{2,4}{:}C_{1,4}{:}I_{2,4}\}$
$\{E_{1,1}{:}C_{2,1}{:}C_{1,1}{:}I_{1,1},\ E_{1,2}{:}C_{2,2}{:}C_{1,2}{:}I_{1,2},\ E_{2,3}{:}C_{2,3}{:}C_{1,3}{:}I_{2,3},\ E_{2,4}{:}C_{2,4}{:}C_{1,4}{:}I_{2,4}\}$

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

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

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

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

Hence, even in a simple example where forwarding is deterministic, our semantics for ProbNetKAT is quite useful: it enables making predictions about quantitative properties such as congestion, which can be used to provision capacity, inform traffic engineering algorithms, or calculate the risk that service-level agreements may be violated. More generally, ProbNetKAT can be used to express much richer behaviors such as randomized routing, faulty links, gossip, etc., as shown by the examples presented in Section 8.

Challenges. We faced several challenges in formulating the semantics of ProbNetKAT in a satisfactory way. The deterministic semantics of NetKAT [1, 12] interprets programs as packet-processing functions on sets of packet histories. This is different enough from other probabilistic models in the literature that it was not obvious how to apply standard approaches. On the one hand, we wanted to extend the deterministic semantics conservatively—i.e., a ProbNetKAT pro-

gram that makes no probabilistic choices should behave the same as under the deterministic NetKAT semantics. This goal was achieved (Theorem 2) using the notion of a *Markov kernel*, well known from previous work in probabilistic semantics [26, 10, 41]. Among other things, conservativity enables using NetKAT axioms to reason about deterministic sub-terms of ProbNetKAT programs. On the other hand, when moving to the probabilistic domain, several properties enjoyed by the deterministic version are lost, and great care was needed to formulate the new semantics correctly. Most notably, it is no longer the case that the meaning of a program on an input set of packet histories is uniquely determined by its action on individual histories (§6.4). The parallel composition operator (&), which supplants the union operator (+) of NetKAT, is no longer idempotent except when applied to deterministic programs (Lemma 1(vi)), and distributivity no longer holds in general (Lemma 4). Nevertheless, the semantics provides a powerful means of reasoning that is sufficient to derive many interesting and useful properties of networks (§8).

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

Another challenge was the observation that in the presence of both duplication (`dup`) and iteration (*), models based on discrete distributions do not suffice, and it is necessary to base the semantics on an uncountable state space with continuous measures and sequential composition defined by integration. Most models in the literature only deal with discrete distributions, with a few noteable exceptions (e.g. [10, 25, 26, 41, 40]). To see why a discrete semantics suffices in the absence of either duplication or iteration note that H is a countable set. Without iteration, we could limit our attention to distributions on finite subsets of H, which is also countable. Similarly, with iteration but without duplication, the set of histories that could be generated by a program is actually finite. Hence a discrete semantics would suffice in that case as well, even though iterative processes would not necessarily converge after finitely many steps as with deterministic processes. However, in the presence of both duplication and iteration, infinite sets and continuous measures are unavoidable (§6.3), although in specific applications, discrete distributions sometimes suffice.

3 Measure Theory Primer

This section introduces the background mathematics necessary to understand the semantics of ProbNetKAT. Because ProbNetKAT requires continuous prob-

ability distributions, we review some basic measure theory. See Halmos [18], Chung [5], or Rao [44] for a more thorough treatment.

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

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

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

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

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

Let $(S, \mathcal{B}_S)$ and $(T, \mathcal{B}_T)$ be measurable spaces. A function $f : S \to T$ is *measurable* if the inverse image $f^{-1}(B) = \{x \in S \mid f(x) \in B\}$ of every measurable subset $B \subseteq T$ is a measurable subset of S. For the particular case where T is generated by the collection $\mathcal{F}$, we have the following criterion for measurability: f is measurable if and only if $f^{-1}(B)$ is measurable for every $B \in \mathcal{F}$.

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

For every $a \in S$, the Dirac measure on a is the probability measure:

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

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

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

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

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

These properties allow integration on the left and right respectively.

The measurable spaces and Markov kernels form a category, the *Kleisli category of the Giry monad*; see [40, 41, 10]. In this context, we occasionally write $P : (S, \mathcal{B}_S) \to (T, \mathcal{B}_T)$ or just $P : S \to T$. Composition is given by integration: for $P : S \to T$ and $Q : T \to U$,

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

Associativity of composition is essentially Fubini's theorem (see Chung [5] or Halmos [18]). Markov kernels were first proposed as a model of probabilistic while programs by Kozen [26].

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

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

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

4 Syntax

ProbNetKAT extends NetKAT [1, 12], which is itself based on Kleene algebra with tests (KAT) [27], a generic equational system for reasoning about partial correctness of programs.

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

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

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

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

The axioms of KA and KAT (both elided here) capture natural conditions such as associativity of $\cdot$; see the original paper by Kozen for a complete listing [27]. Note that the KAT axioms do not hold for arbitrary ProbNetKAT programs—e.g., parallel composition is not idempotent—although they do hold for the deterministic fragment of the language.

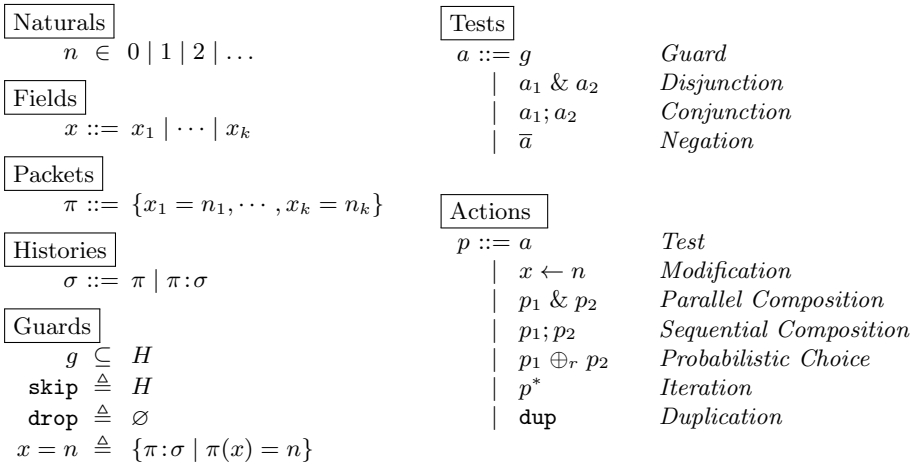

Fig. 2. ProbNetKAT Syntax.

4.2 NetKAT Syntax

NetKAT [1, 12] extends KAT with network-specific primitives for filtering, modifying, and forwarding packets, along with additional axioms for reasoning about programs built using those primitives. Formally, NetKAT is KAT with atomic tests $x = n$ and actions $x \leftarrow n$ and `dup`. The test $x = n$ checks whether field x of the current packet contains the value n; the assignment $x \leftarrow n$ assigns the value n to the field x in the current packet; the action `dup` duplicates the packet in the packet history, which keeps track of the path the packet takes through the network. In NetKAT, we write ; instead of $\cdot$, `skip` instead of 1, and `drop` instead of 0, as these names capture their intuitive use as programming constructs. We often use juxtaposition to indicate sequential composition in examples. As an example, the NetKAT expression

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

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

4.3 ProbNetKAT Syntax

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

- A *random choice* operation $p \oplus_r q$, where p and q are expressions and r is a real number in the interval $[0, 1]$. The expression $p \oplus_r q$ intuitively behaves according to p with probability r and q with probability $1 - r$. We frequently omit the subscript r, in which case r is understood to implicitly be $1/2$.
- A *parallel composition* operation $p \,\&\, q$, where p and q are expressions. The expression $p \,\&\, q$ intuitively says to perform both p and q, making

$$[\![x \leftarrow n]\!](\pi : \sigma) = \{\pi[n/x] : \sigma\}$$

$$[\![x = n]\!](\pi : \sigma) = \begin{cases} \{\pi : \sigma\}, & \pi(x) = n \\ \varnothing, & \pi(x) \neq n \end{cases}$$

$$[\![\texttt{dup}]\!](\pi : \sigma) = \{\pi : \pi : \sigma\}$$

$$[\![\texttt{skip}]\!](\sigma) = \{\sigma\}$$

$$[\![\texttt{drop}]\!](\sigma) = \varnothing$$

$$[\![p + q]\!](\sigma) = [\![p]\!](\sigma) \cup [\![q]\!](\sigma)$$

$$[\![p\,;q]\!](\sigma) = \bigcup_{\tau \in [\![p]\!](\sigma)} [\![q]\!](\tau)$$

$$[\![p^*]\!](\sigma) = \bigcup_n [\![p^n]\!](\sigma)$$

$$[\![\bar{a}]\!](\sigma) = \begin{cases} \{\sigma\}, & \text{if } [\![a]\!](\sigma) = \varnothing \\ \varnothing, & \text{if } [\![a]\!](\sigma) = \{\sigma\} \end{cases}$$

Fig. 3. Semantics of NetKAT: on the left, semantics of the primitive actions and tests; on the right, semantics of KAT operations.

any probabilistic choices in p and q independently, and combine the results. The operation & serves the same purpose as + in NetKAT and replaces it syntactically. We use the notation & to distinguish it from +, which is used in the semantics to add measures and measurable functions as in [25, 26].

– *Guards* g which generalize NetKAT's tests by allowing them to operate on the entire packet history rather than simply the head packet. Formally a guard g is just an element of 2^H. The guard `skip` is defined as the set of all packet histories and `drop` is the empty set. An atomic test $x = n$ is defined as the set of all histories σ where the x field of the head packet of σ is n. As we saw in §2, guards are often useful for reasoning probabilistically about properties such as congestion.

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

5 Semantics

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

5.1 NetKAT Semantics

Formally, a (nonprobabilistic) NetKAT term p denotes a function

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

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

The semantics of NetKAT is shown in Figure 3. Intuitively, a test $x = n$ drops the packet if the test is not satisfied and passes it through unaltered if it is satisfied—i.e., tests behave like filters. The `dup` construct duplicates the head packet π, yielding a fresh copy that can be modified by other constructs. Hence, the `dup` construct can be used to encode paths through the network, with each occurrence of `dup` marking an intermediate hop. Note that + behaves like a disjunction operation when applied to tests and like a union operation when applied to actions. Similarly, ; behaves like a conjunction operation when applied to tests and like a sequential composition when applied to actions. Negation is only ever applied to tests, as is enforced by the syntax of the language.

5.2 Sets of Packet Histories as a Measurable Space

To give a denotational semantics to ProbNetKAT, we must first identify a suitable space of mathematical objects. Because we want to reason about probability distributions over sets of network paths, we construct a *measurable space* (as defined in §3) from sets of packet histories, and then define the semantics using Markov kernels on this space. The powerset 2^H of packet histories H forms a topological space with topology generated by basic clopen sets,

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

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

5.3 The Operation &

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

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

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

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

Intuitively, this is the probability that the union $a \cup b$ of two independent samples taken with respect to μ and ν lies in A. The & operation enjoys a number of useful properties, as captured by the following lemma:

Lemma 1.

(i) & *is associative and commutative.*
(ii) & *is linear in both arguments.*
(iii) $(\delta_a \mathbin{\&} \mu)(A) = \mu(\{b \mid a \cup b \in A\})$.
(iv) $\delta_a \mathbin{\&} \delta_b = \delta_{a \cup b}$.
(v) $\delta_\varnothing$ *is a two-sided identity for* &.
(vi) $\mu \mathbin{\&} \mu = \mu$ *iff* $\mu = \delta_a$ *for some* $a \in 2^H$.

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

5.4 ProbNetKAT Semantics

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

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

which can be curried variously as

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

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

The semantics of the atomic operations are defined as follows for $a \in 2^H$:

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

Note that if no elements of a satisfy the test $x = n$, the result is $\delta_\varnothing$, which is the Dirac measure on the emptyset, not the constant 0 measure.

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

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

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

Note that the semantics of composition requires us to extend $[\![q]\!]$ to allow measures as inputs. This is done by integration as described in §3:

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

It is not surprising that this extension is needed: in NetKAT, the semantics is similarly extended to sets of histories to define the semantics of sequential composition. Both phenomena are consequences of sequential composition taking place in the Kleisli category of the powerset and Giry monads respectively.

5.5 Semantics of Iteration

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

Consider the following infinite stochastic process. Starting with $c_0 \in 2^H$, create a sequence $c_0, c_1, c_2, \ldots$ inductively. After n steps, say we have constructed $c_0, \ldots, c_n$. Let c_{n+1} be the outcome obtained by sampling 2^H according to the distribution $[\![p]\!](c_n)$. Continue this process forever to get an infinite sequence $c_0, c_1, c_2, \ldots \in (2^H)^\omega$. Take the union of the resulting sequence $\bigcup_n c_n$ and ask whether it is in A. The probability of this event is taken to be $[\![p^*]\!](c_0, A)$. This intuitive operational definition can be justified denotationally. However, the formal development is quite technical and depends on an application of the Kolmogorov extension theorem—see the full version of this paper [13].

The next theorem shows that the iteration operator satisfies a natural fixpoint equation. In fact, this property was the original motivation behind the operational definition we just gave. It can be used to describe the iterated processing performed by a network (§8), and to define the semantics of loops (§5.6).

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

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

$c_0 \cup c \in A$. Thus

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

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

5.6 Guards

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

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

A test $x = n$ is a special case in which $g = \{\pi : \tau \mid \pi(x) = n\}$. Note that unlike other ProbNetKAT atomic programs, guards are not necessarily determined by their action on the head packet. By Lemma 1, guards extend to measures:

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

With this construct, we can define encodings of conditionals and while loops:

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

Importantly, unlike treatments involving subprobability measures found in previous work [26, 40], the output here is always a probability measure, even if the program does not halt. For example, the output of the program $\mathsf{while\ true\ do\ skip}$ is the Dirac measure $\delta_{\varnothing}$.

6 Properties

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

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

6.1 Conservativity of the Extension

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

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

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

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

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

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

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

Besides providing further evidence that our probabilistic semantics captures the intended behavior, these theorems also have a pragmatic benefit: they allow us to use NetKAT to reason about deterministic terms in ProbNetKAT programs.

6.2 Further Properties

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

$$[\![p \mathbin{\&} \texttt{drop}]\!] = [\![p]\!] = [\![\texttt{drop} \mathbin{\&} p]\!]$$

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

$$[\![p \oplus_r p]\!] = [\![p]\!]$$

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

$$[\![(p \mathbin{\&} q) \mathbin{\&} s]\!] = [\![p \mathbin{\&} (q \mathbin{\&} s)]\!]$$

$$[\![p \mathbin{\&} q]\!] = [\![q \mathbin{\&} p]\!]$$

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

$$[\![p \oplus_r q]\!] = [\![q \oplus_{1-r} p]\!]$$

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

$$[\![\Big(p \oplus_{\frac{a}{a+b}} q\Big) \oplus_{\frac{a+b}{a+b+c}} s]\!] = [\![p \oplus_{\frac{a}{a+b+c}} \Big(q \oplus_{\frac{b}{b+c}} s\Big)]\!]$$

Next we develop some additional properties involving deterministic programs.

Lemma 3. *Let p be deterministic with $[\![p]\!](a) = \delta_{f(a)}$. The function $f : 2^H \to 2^H$ is measurable, and for any measure μ, we have $[\![p]\!](\mu) = \mu \circ f^{-1}$.*

As we have seen in Lemma 1(vi), & is not idempotent except in the deterministic case. Neither does sequential composition distribute over & in general. However, if the term being distributed is deterministic, then the property holds:

Lemma 4. *If p is deterministic, then*

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

Neither equation holds unconditionally.

Finally, consider the program $\texttt{skip} \oplus_r \texttt{dup}$. This program does nothing with probability r and duplicates the head packet with probability $1 - r$, where $r \in [0, 1)$. We can show that independent of r, the value of the iterated program on any single packet π is the point mass

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

Note that the equation in the statement of Theorem 1 does not determine $[\![p^*]\!]$ uniquely. For example, it can be shown that a probability measure μ is a solution of $[\![\texttt{skip}^*]\!](\pi) = [\![\texttt{skip} \,\&\, \texttt{skip}\,;\texttt{skip}^*]\!](\pi)$ if and only if $\mu(B_\pi) = 1$. That is, π appears in the output set of $[\![\texttt{skip}^*]\!](\pi)$ with probability 1.

6.3 A Continuous Measure

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

To see why, let π_0 and π_1 be distinct packets and let p be the program that changes the current packet to either π_0 or π_1 with equal probability. Then consider the program, $p\,;(\texttt{dup}\,;p)^*$. Operationally, it first sets the input packet to either 0 or 1 with equal probability, then repeats the following steps forever:

(i) output the current packet,
(ii) duplicate the current packet, and
(iii) set the new current packet to π_0 or π_1 with equal probability.

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

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

(i) *For $\tau \in H$, the probability that τ is a member of the output set is $2^{-|\tau|}$.*
(ii) *Two packet histories of the same length are generated with probability 0.*
(iii) *$\mu(\{a\}) = 0$ for all $a \in 2^H$, thus μ is a continuous measure.*

For the proof, see the full version of this paper [13].

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

6.4 Uncorrelated Semantics

It is tempting to consider a weaker *uncorrelated semantics*

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

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

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

However, although it is simpler in that it does not require continuous measures, one loses correlation between packets. Worse, it is not compositional, as the following example shows. Let π_0, π_1 be two packets and consider the programs $\pi_0! \oplus \pi_1!$ and $(\pi_0! \,\&\, \pi_1!) \oplus \mathtt{drop}$, where $\pi!$ is the program that sets the current packet to π. Both programs have the same uncorrelated meaning:

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

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

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

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

7 Approximation

Approximation in the context of bisimulation of Markov processes has been studied by many authors [8–10, 30, 40, 41]. In this section we identify a suitable notion of approximation for the iterates of a loop and show that every program is arbitrarily closely approximated by a loop-free program.

7.1 Weak Convergence of $p^{(m)}$ to p^*

In §5.5, we defined $[\![p^*]\!]$ operationally in terms of an infinite process. To get $[\![p^*]\!](c_0, A)$, we compute an infinite sequence $c_0, c_1, \ldots$ where in the n^{th} step we sample c_n to get c_{n+1}. Then we take the union of the c_n and ask whether it is in A. We proved that the resulting kernel exists and satisfies $[\![p^*]\!] = [\![\texttt{skip} \mathbin{\&} p\,; p^*]\!]$.

Now let $c_0, c_1, \ldots, c_{m-1}$ be the outcome of the first m steps of this process, and let $[\![p^{(m)}]\!](c_0, A)$ be the probability that $\bigcup_{n=0}^{m-1} c_n \in A$. This gives an approximation to $[\![p^*]\!](c_0, A)$. Formally, define

$$p^{(0)} = \texttt{skip} \qquad\qquad p^{(n+1)} = \texttt{skip} \mathbin{\&} p\,; p^{(n)}.$$

Note that $p^{(n)}$ is not p^n, nor is it $p^0 \mathbin{\&} \cdots \mathbin{\&} p^n$.

The appropriate notion of convergence is *weak convergence*. A sequence of measures μ_n converge weakly to a measure μ if for all bounded continuous real-valued functions f, the expected values of f with respect to the measures μ_n converge to the expected value of f with respect to μ.

Theorem 4. *The measures $[\![p^{(m)}]\!](c)$ converge weakly to $[\![p^*]\!](c)$.*

7.2 Approximation by *-Free Programs

We have observed that *-free programs only generate finite discrete distributions on finite inputs. In this section we show that every program is weakly approximated to arbitrary precision by *-free programs. The approximating programs are obtained by replacing each p^* with $p^{(m)}$ for sufficiently large m.

This explains why we see only finite discrete distributions in most applications. In most cases, we start with finite sets and iterate only finitely many times. For instance, this will happen whenever there is a bound on the number of occurrences of `dup` in any string generated by the program as a regular expression. So although the formal semantics requires continuous distributions and integration, in many real-world scenarios we can get away with only finite discrete distributions.

Theorem 5. *For every ProbNetKAT program p, there is a sequence of *-free programs that converge weakly to p.*

The proof uses Theorem 4 and the fact that all program constructors are continuous with respect to weak convergence.

8 Applications

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

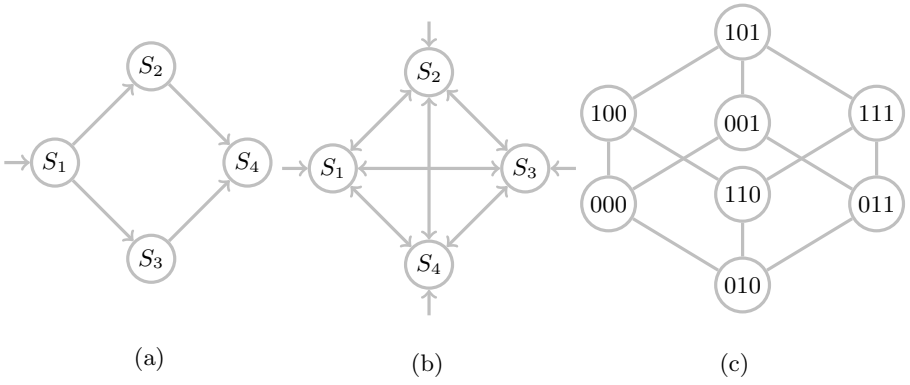

Fig. 4. Topologies used in case studies: (a) fault tolerance, (b) load balancing, and (c) gossip protocols.

8.1 Fault Tolerance

Failures are a fact of life in real-world networks. Devices and links fail due to factors ranging from software and hardware bugs to interference from the environment such as loss of power or cables being severed. A recent empirical study of data center networks by Gill et al. [14] found that failures occur frequently and can cause issues ranging from degraded performance to service disruptions. Hence, it it important for network operators to be able to understand the impact of failures—e.g., they may elect to use routing schemes that divide traffic over many diverse paths in order to minimize the impact of any given failure.

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

As a concrete example, consider the topology depicted in Figure 4 (a), with four switches connected in a diamond. Suppose that we wish to forward traffic from S_1 to S_4 and we know that the link between S_1 and S_2 fails with 10% probability (for simplicity, in this example, we will assume that the switches and all other links are reliable). What is the probability that a packet that originates at S_1 will be successfully delivered to S_4, as desired?

Obviously the answer to this question depends on the configuration of the network—using different forwarding paths will lead to different outcomes! To investigate this question, we will encode the overall behavior of the network using several terms: a term p that encodes the local forwarding behavior of the switches; a term t that encodes the forwarding behavior of the network topology; and a term e that encodes the network egresses.

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

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

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

Next, we define the local forwarding policy p that encodes the behavior on switches. Suppose that we forward traffic from S_1 to S_4 via S_2. Then p would be defined as follows: $p \triangleq (sw = S_1; pt \leftarrow 2)$ & $(sw = S_2; pt \leftarrow 4)$ Finally, the egress predicate e is simply: $e \triangleq sw = S_4$

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

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

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

$$p' \triangleq (sw = S_1; (pt \leftarrow 2 \oplus pt \leftarrow 3)) \ \& \ (sw = S_2; pt \leftarrow 4) \ \& \ (sw = S_3; pt \leftarrow 4)$$

As expected, evaluating this policy on a packet starting at S_1 gives us a 95% chance that the packet is delivered to S_4 and only a 5% chance that it is dropped. The positive effect with respect to failures has also been observed in previous work on randomized routing [55].

8.2 Load Balancing

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

Valiant load balancing (VLB) [51] is a classic randomized routing scheme that provides low expected congestion for any feasible demands in a full mesh. VLB forwards packets using a simple two-phase strategy: in the first phase, the ingress switch forwards the packet to a randomly selected neighbor, without considering the the packet's ultimate destination; in the second phase, the neighbor forwards the packet to the egress switch that is connected to the destination.

As an example, consider the four-node mesh topology shown in Figure 4(b). When a packet destined for a host connected to S_3 arrives at S_1, the switch will first pick one of S_2, S_3, or S_4 as the intermediate hop. Suppose it picks S_4. When S_4 receives the packet, it forwards the packet directly to S_3, which will in turn forward it along to the destination host.

We assume that each switch has ports named $1, 2, 3, 4$, that port i on switch i connects to the outside world, and that all other ports j connect to switch j. We can write a ProbNetKAT program for this load balancing scheme by splitting it into two parts, one for each phase of routing. VLB often requires that traffic be tagged in each phase so that switches know when to forward it randomly or deterministically, but in this example, we can use topological information to distinguish the phases. Packets coming in from the outside (port i on switch i) are forwarded randomly, and packets on internal ports are forwarded deterministically.

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

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

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

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

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

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

The topology term t is encoded with `dup` terms to record the paths, as described in §8.1.

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

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

The full network program to analyze is then $d; (p; t)^*; p$. We can use similar techniques as in the congestion example from §2 to reason about congestion.

We first define a random variable to extract the information we care about. Let $X_{\max}$ be a random variable equal to the maximum number of packets traversing a single internal link. Then, using the semantics, we calculate that the expected value of $X_{\max}$ is 1 packet—i.e., there is no congestion.

8.3 Gossip Protocols

Gossip (or epidemic) protocols are randomized algorithms that are often used to efficiently disseminate information in large-scale distributed systems [7]. An attractive feature of gossip protocols and other epidemic algorithms is that they are able to rapidly converge to a consistent global state while only requiring bounded worst-case communication. Operationally, a gossip protocol proceeds in loosely synchronized rounds: in each round, every node communicates with a randomly selected peer and the nodes update their state using information shared during the exchange. For example, in a basic anti-entropy protocol, a "rumor" is injected into the system at a single node and spreads from node to node through pair-wise communication. In practice, such protocols can rapidly disseminate information in well-connected graphs with high probability.

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

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

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

Fig. 5. Gossip results.

To simplify the ProbNetKAT program, we assume that the network topology is a hypercube, as shown in Figure 4 (c). The program for gossiping on a hypercube is highly uniform—assuming that switches are numbered in binary, we can randomly select a neighbor by flipping a single bit.

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

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

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

Encoding the topology of the hypercube as t, we can then analyze $(p;t)^*$ and calculate the expected number of infected nodes after a given number of rounds X_{infected} using the ProbNetKAT semantics. The results for the first few rounds are shown in Fig. 5.

9 Related Work

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

9.1 Probabilistic Programming

Computational models and logics for probabilistic programming have been extensively studied for many years. Denotational and operational semantics for probabilistic while programs were first studied by Kozen [25]. Early logical systems for reasoning about probabilistic programs were proposed in a sequence of separate papers by Saheb-Djahromi, Ramshaw, and Kozen [46, 43, 26]. There are also numerous recent efforts [38, 28, 30, 16, 17]. Our semantics for ProbNetKAT builds on the foundation developed in these papers and extends it to the new domain of network programming.

Probabilistic programming in the context of artificial intelligence has also been extensively studied in recent years [45, 2, 15]. However, the goals of this line of work are different than ours in that it focuses on Bayesian inference.

Probabilistic automata in several forms have been a popular model going back to the early work of Paz [42], as well as many other recent efforts [47, 48, 33]. Probabilistic automata are a suitable operational model for probabilistic programs and play a crucial role in the development of decision procedures for bisimulation equivalence, logics to reason about behavior, in the synthesis of probabilistic programs, and in model checking procedures [31, 8, 4, 21, 29]. In the present paper, we do not touch upon any of these issues so the connections to probabilistic automata theory are thin. However, we expect they will play an important role in our future work—see below.

Denotational models combining probability and nondeterminism have been proposed in papers by several authors [20, 34, 52, 50], and general models for labeled Markov processes, primarily based on Markov kernels, have been studied extensively [40, 41, 10]. Because ProbNetKAT does not have nondeterminism, we have not encountered the extra challenges arising in the combination of nondeterministic and probabilistic behavior.

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

literature do not immediately apply. Nevertheless, we have drawn much inspiration from the literature and exploited many similarities to provide a powerful formalism for modeling probabilistic behavior in packet-switching networks.

9.2 Network Programming

Recent years have seen an incredible growth of languages and systems for programming and reasoning about networks. Network programming languages such as Frenetic [11], Pyretic [37], Maple [53], NetKAT [1], and FlowLog [39] have introduced high-level abstractions and semantics that enable programmers to reason precisely about the behavior of networks. However, as mentioned previously, all of these language are based on deterministic packet-processing functions, and do not handle probabilistic traffic models or forwarding policies. Of all these frameworks, NetKAT is the most closely related as ProbNetKAT builds directly on its features.

In addition to programming languages, a number of network verification tools have been developed, including Header Space Analysis [22], VeriFlow [23], the NetKAT verifier [12], and Libra [54]. Similar to the network programming languages described above, these tools only model deterministic networks and verify deterministic properties.

Network calculus is a general framework for analyzing network behavior using tools from queuing theory [6]. It models the low-level behavior of network devices in significant detail, including features such as traffic arrival rates, switch propagation delays, and the behaviors of components like buffers and queues. This enables reasoning about quantitative properties such as latency, bandwidth, congestion, etc. Past work on network calculus can be divided into two branches: deterministic [32] and stochastic [19]. Like ProbNetKAT, the stochastic branch of network calculus provides tools for reasoning about the probabilistic behavior, especially in the presence of statistical multiplexing. However, network calculus is generally known to be difficult to use, since it can require the use of external facts from queuing theory to establish many desired results. In contrast, ProbNetKAT is a self-contained, language-based framework that offers general programming constructs and a complete denotational semantics.

10 Conclusion

Previous work [1, 12] has described NetKAT, a language and logic for specifying and reasoning about the behavior of packet-switching networks. In this paper we have introduced ProbNetKAT, a conservative extension of NetKAT with constructs for reasoning about the probabilistic behavior of such networks. To our knowledge, this is the first language-based framework for specifying and verifying probabilistic network behavior. We have developed a formal semantics for ProbNetKAT based on Markov kernels and shown that the extension is conservative over NetKAT. We have also determined the appropriate notion of approximation and have shown that every ProbNetKAT program is arbitrarily

closely approximated by loop-free programs. Finally, we have presented several case studies that illustrate the use of ProbNetKAT on real-world examples.

Our examples have used the semantic definitions directly in the calculation of distributions, fault tolerance, load balancing, and a probabilistic gossip protocol. Although we have exploited several general properties of our system in these arguments, we have made no attempt to assemble them into a formal deductive system or decision procedure as was done previously for NetKAT [1, 12]. These questions remain topics for future investigation. We are hopeful that the coalgebraic perspective developed in [12] will be instrumental in obtaining a sound and complete axiomatization and a practical decision procedure for equivalence of ProbNetKAT expressions.

As a more practical next step, we would like to augment the existing NetKAT compiler [49] with tools for handling the probabilistic constructs of ProbNetKAT along with a formal proof of correctness. Features such as OpenFlow [35] "group tables" support for simple forms of randomization and emerging platforms such as P4 [3] offer additional flexibility. Hence, there already exist machine platforms that could serve as a compilation target for (restricted fragments of) ProbNetKAT.

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

Acknowledgements

The authors wish to thank the members of the Cornell PLDG and DIKU COPLAS group 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, and CCF-1253165; 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

1. C. Anderson, N. Foster, A. Guha, J. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. In *Proc. POPL'14*, ACM.
2. J. Borgström, A. Gordon, M. Greenberg, J. Margetson, and J. Gael. Measure transformer semantics for Bayesian machine learning. In *ESOP'11*. Springer Verlag.
3. P. Bosshart, D. Daly, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.
4. S. Cattani and R. Segala. Decision algorithms for probabilistic bisimulation. In *CONCUR*, volume 2421 of *LNCS*, pages 371–385. Springer, 2002.
5. K. L. Chung. *A Course in Probability Theory*. Academic Press, 2nd edition, 1974.
6. R. Cruz. A calculus for network delay, parts I and II. *IEEE Transactions on Information Theory*, 37(1):114–141, January 1991.
7. A. Demers, D. Greene, et al. Epidemic algorithms for replicated database maintenance. In *Proceedings PODC '87*, ACM.
8. J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 179(2):163–193, 2002.
9. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. A metric for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.
10. E. Doberkat. *Stochastic Relations: Foundations for Markov Transition Systems*. Studies in Informatics. Chapman Hall, 2007.
11. N. Foster, R. Harrison, M. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. In *ICFP'11*, ACM.
12. N. Foster, D. Kozen, M. Milano, A. Silva, and L. Thompson. A coalgebraic decision procedure for NetKAT. In *Proc. POPL'15*, ACM.
13. N. Foster, D. Kozen, K. Mamouras, M. Reitblatt, and A. Silva. Probabilistic NetKAT (full version). Available at `http://alexandrasilva.org/pbNK_full.pdf`
14. P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *SIGCOMM'11*, ACM.
15. A. Gordon, T. Graepel, et al. Tabular: A schema-driven probabilistic programming language. Technical Report MSR-TR-2013-118, Microsoft Research, 2013.
16. A. Gordon, T. Henzinger, A. Nori, and S. Rajamani. Probabilistic programming. In *ICSE'14*. IEEE.
17. F. Gretz, N. Jansen, B. Kaminski, J. Katoen, A. McIver, and F. Olmedo. Conditioning in probabilistic programming. *MFPS'15*, Elsevier.
18. P. R. Halmos. *Measure Theory*. Van Nostrand, 1950.
19. Y. Jiang. A basic stochastic network calculus. In *SIGCOMM'06*, ACM.
20. C. Jones. *Probabilistic Nondeterminism*. PhD thesis, Edinburgh University, 1990.
21. B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *LICS'91*, IEEE.
22. P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *NSDI*, 2012.
23. A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *NSDI*, 2013.
24. A. Kolmogorov and S. Fomin. *Introductory Real Analysis*. Prentice Hall, 1970.
25. D. Kozen. Semantics of probabilistic programs. *JCSS*, 22:328–350, 1981.
26. D. Kozen. A probabilistic *PDL*. *JCSS*, 30(2):162–178, 1985.
27. D. Kozen. Kleene algebra with tests. *TOPLAS*, 19(3):427–443, 1997.
28. D. Kozen, R. Mardare, and P. Panangaden. Strong completeness for Markovian logics. In *Proc. MFCS 2013*, Springer.

29. M. Kwiatkowska, G. Norman, et al. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, 2007.
30. Kim G. Larsen, R. Mardare, and P. Panangaden. Taking it to the limit: Approximate reasoning for Markov processes. In *Proc. MFCS'12*, Springer.
31. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:456–471, 1991.
32. J. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.
33. A. McIver, E. Cohen, C. Morgan, and C. Gonzalia. Using probabilistic Kleene algebra pKA for protocol verification. *JLAP*, 76(1):90–111, 2008.
34. A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.
35. N. McKeown, T. Anderson, et al. Openflow: Enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, 2008.
36. A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. In *SIGCOMM'02*, ACM.
37. C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing software defined networks. In *NSDI*, 2013.
38. C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *TOPLAS*, 18(3):325–353, 1996.
39. T. Nelson, A. Ferguson, M. Scheer, and S. Krishnamurthi. Tierless programming and reasoning for software-defined networks. In *NSDI*, 2014.
40. P. Panangaden. Probabilistic relations. In *School of Computer Science, McGill University, Montreal*, pages 59–74, 1998.
41. P. Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009.
42. A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.
43. L. H. Ramshaw. *Formalizing the Analysis of Algorithms*. PhD thesis, Stanford University, 1979.
44. M. M. Rao. *Measure Theory and Integration*. Wiley-Interscience, 1987.
45. D. Roy. *Computability, inference and modeling in probabilistic programming*. PhD thesis, Massachusetts Institute of Technology, 2011.
46. N. Saheb-Djahromi. Probabilistic LCF. In *Proc. MFCS'78*, Springer.
47. R. Segala. Probability and nondeterminism in operational models of concurrency. In *Proc. CONCUR'06*, Springer.
48. R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. In *NJC*, volume 2, pages 250–273, 1995.
49. S. Smolka, S. Eliopoulos, N. Foster, and A. Guha. A fast compiler for NetKAT. In *ICFP'15*, ACM.
50. R. Tix, K. Keimel, and G. Plotkin. Semantic domains for combining probability and nondeterminism. *ENTCS*, 222:3–99, 2009.
51. L. Valiant. A Scheme for Fast Parallel Communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.
52. D. Varacca and G. Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006.
53. A. Voellmy, J. Wang, Y. Yang, B. Ford, and P. Hudak. Maple: Simplifying SDN programming using algorithmic policies. In *SIGCOMM*, 2013.
54. H. Zeng, S. Zhang, et al. Libra: Divide and conquer to verify forwarding tables in huge networks. In *NSDI*, 2014.
55. R. Zhang-Shen and N. McKeown. Designing a predictable Internet backbone with Valiant load-balancing. In *IWQoS*, 2005.