

Weighted NetKAT

A Programming Language for Quantitative Network Verification

EMMANUEL SUÁREZ ACEVEDO, Cornell University, USA

TIAGO FERREIRA, University College London, United Kingdom

KEVIN BATZ, Cornell University, USA

OLIVER BØVING, Technical University of Denmark, Denmark

NATE FOSTER, EPFL, Switzerland and Jane Street, USA

ALEXANDRA SILVA, Cornell University, USA

We introduce weighted NetKAT, a domain-specific language for modeling and verifying *quantitative* network properties. The language is parametric on a *semiring*, enabling the treatment of a wide range of quantities in a uniform way. We provide a denotational semantics and an equivalent operational semantics, the latter based on a novel model of *weighted NetKAT automata* (WNKA) capturing the stateful behavior of our language. With WNKA, we obtain a class of generic decision procedures for reasoning about *quantitative safety and reachability* in a fully automatic way, even in the presence of possibly unbounded iteration. We demonstrate the applicability of our framework in a case study using Internet2's Abilene network as the underlying topology.

CCS Concepts: • **Theory of computation** → **Verification by model checking**; **Formal languages and automata theory**; **Models of computation**.

Additional Key Words and Phrases: network verification, quantitative verification, weighted automata, NetKAT

ACM Reference Format:

Emmanuel Suárez Acevedo, Tiago Ferreira, Kevin Batz, Oliver Bøving, Nate Foster, and Alexandra Silva. 2026. Weighted NetKAT: A Programming Language for Quantitative Network Verification. *Proc. ACM Program. Lang.* 10, PLDI, Article 240 (June 2026), 24 pages. <https://doi.org/10.1145/3808318>

1 Introduction

The field of network verification has emerged as a significant success story for the programming languages community in recent years. The idea is to see the *network as a program*, and model the topology of a network and the configurations of its devices as programs in a domain-specific language, which can then be analyzed to verify properties of interest. This basic approach has been applied successfully at scale in industry, where it has shown to improve the correctness and reliability of networks by catching errors at design time [1, 16].

Among the numerous network verification frameworks that have been proposed, NetKAT [3] stands out for its strong theoretical foundations based on Kleene Algebra with Tests (KAT) [18]. Indeed, the deep connection between NetKAT and finite automata has been instrumental in facilitating production-grade [4, 23] scalable verification based on automata-theoretic methods [11, 24, 25].

Authors' Contact Information: [Emmanuel Suárez Acevedo](mailto:es2278@cornell.edu), es2278@cornell.edu, Cornell University, Ithaca, USA; [Tiago Ferreira](mailto:t.ferreira@ucl.ac.uk), University College London, London, United Kingdom, t.ferreira@ucl.ac.uk; [Kevin Batz](mailto:ksb239@cornell.edu), ksb239@cornell.edu, Ithaca, Cornell University, USA; [Oliver Bøving](mailto:oembo@dtu.dk), oembo@dtu.dk, Technical University of Denmark, Kongens Lyngby, Denmark; [Nate Foster](mailto:nate.foster@epfl.ch), EPFL, Lausanne, Switzerland and Jane Street, New York, USA, nate.foster@epfl.ch; [Alexandra Silva](mailto:alexandra.silva@cornell.edu), Cornell University, Ithaca, USA, alexandra.silva@cornell.edu.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2475-1421/2026/6-ART240

<https://doi.org/10.1145/3808318>

However, NetKAT has a critical limitation: its semantics only captures the packet-forwarding behavior of the network. Hence, it can be used to capture basic properties involving the paths that packets take (e.g., reachability, isolation, forwarding loops, etc.). But in many situations, network operators need to reason about richer *quantitative* properties, such as bandwidth, latency, reliability, or security, that cannot be gleaned from topologies and device configurations, but are important for applications such as traffic engineering, fault tolerance, and security.

This paper presents weighted NetKAT (wNetKAT), a new framework for modeling and reasoning about such quantitative network properties. wNetKAT enriches NetKAT with new syntactic constructs for assigning and manipulating weights and a semantics that assigns a weight to each execution. At a technical level, we model weights as elements of a semiring—intuitively, semirings arise in networking as their operations model both alternative (e.g., sum) and joint (e.g., product) use of information, across all possible paths in the topology.

Although extending NetKAT with weights may appear straightforward initially, there are numerous challenges that arise in the design of the language and in formulating the semantics correctly. For the latter, one has to carefully restrict the semiring to ensure that iteration can be computed and, more interestingly, the operational semantics of the language requires a new automaton model that captures both the presence of weights (very much in line with classical weighted automata) but also correctly accounts for the idiosyncrasies of NetKAT semantics. At the level of the expressiveness of the language one has to take into account that the weights needed to compute the relevant quantities might be associated with different parts of the network (e.g. a switch, a link, a port) and therefore the new syntactic constructs need to offer that flexibility.

We provide a thorough formalization of the language and its metatheory including a denotational semantics, language model, operational model using *wNetKAT automata*, and theorems that equate these different models. The soundness of our automata construction then enables the verification of wNetKAT policies at the level of automata. At time of verification the semiring parametricity shines: in different contexts, the nature of the relevant weights varies. For latency we might want to use integers whereas for security we might want to use an ordered set of permission levels. Moreover, the way these quantities need to be combined to propagate through the network to yield the answer to a verification question also varies (e.g., worst-case latency or best-case reliability).

We focus on two classes of quantitative properties in our verification quest: *r*-safety (“Do all paths in the network have weight *at most r*?”) and *r*-reachability (“Does there exist a path in the network with weight *at least r*?”). We provide algorithms to decide these verification questions and then illustrate their applicability in a case study. In particular, although wNetKAT cannot precisely model the kind of quality of service (QoS) properties that depend on flow-level interactions such as congestion, these verification questions encompass a broad range of network performance characteristics. For example, many quantitative aspects of networks—such as reliability (from historical packet loss) or security (whether a link is trusted)—do not depend on modeling dynamic packet-processing behavior at all. Increased bandwidths and packet-processing rates in networks have also recently enabled network performance to be modeled at a coarser granularity that does not need to directly consider queuing or packet-level congestion [15].

In summary, this paper makes the following contributions:

- We develop wNetKAT, a semiring-based framework to facilitate reasoning about quantitative network behaviors such as bandwidth constraints, latency measurements, and reliability metrics.
- We provide a comprehensive formal treatment of the semantics of wNetKAT, including denotational (Section 3), language-theoretic (Section 4), and operational (Section 5) models. The latter is based on a new automaton model—wNetKAT automata. We develop a sound translation, akin to the classical Thompson construction, from wNetKAT expressions to wNetKAT automata.

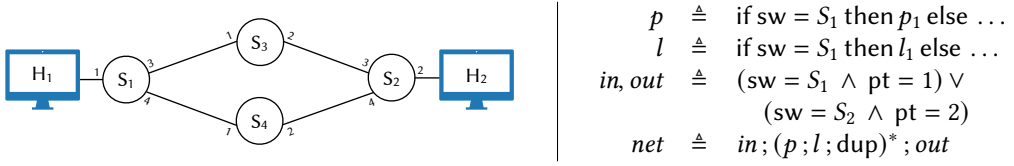


Fig. 1. Sample network and its encoding in wNetKAT.

- We establish the exact computation of wNetKAT expressions—in particular, providing the first computable semantics for probabilistic extensions of NetKAT. This enables algorithms for verifying quantitative network properties: r -safety and r -reachability (Section 6).
- We showcase the applicability of wNetKAT in the setting of Internet2’s Abilene backbone network, providing worst-/best-case network guarantees over a range of practical network phenomena with automatically generated concrete witnesses and/or counterexamples (Section 7).

We also show that wNetKAT subsumes the original semantics of NetKAT as well as the guarded fragment of ProbNetKAT [10, 32]. We include proofs of all formal claims in the appendix [34].

2 Quantitative Network Verification with wNetKAT

In this section, we give an overview of the *quantitative network verification* enabled by wNetKAT. First, we briefly recap modeling networks with NetKAT, after which we discuss the challenges of quantitative network behavior. Finally, we describe the verification of quantitative properties, namely r -safety and r -reachability, through a computable semantics based on wNetKAT automata.

2.1 Background: Encoding Networks in wNetKAT

wNetKAT is a conservative extension of NetKAT [3], a domain-specific language for modeling a networks’ forwarding policies. When disregarding quantitative aspects, modeling in wNetKAT is thus analogous to modeling networks in NetKAT. Let us illustrate this by means of an example.

Consider the network in Figure 1 (left), consisting of two *hosts* and four *switches*. These hosts and switches are connected via links at designated *ports*, giving rise to the network’s topology. Every switch operates according to a forwarding table. For instance, if S_1 receives a packet destined for H_2 , then S_1 will send the packet either via port 3 or port 4.

In wNetKAT, we model networks as intuitive programs, which are called *policies*. More specifically, a policy models how the *fields* of a packet that is being sent through the network are modified over time. In our example, a packet consists of the fields *sw* (holding the switch the packet is currently at), *pt* (holding the port the packet is currently at), and *dst* (holding the packet’s destination). The high-level structure of the policy modeling our example network is depicted in Figure 1 (right). Consider the top-level policy *net* and let us go over each of its components separately:

$$in; (p; l; dup)^*; out.$$

in and *out* are predicates specifying the network’s ingress/egress points: a packet can enter/leave the network at port 1 of S_1 or at port 2 of S_2 . The expression $(p; l; dup)^*$ then models the iterative behavior of the network and is intuitively to be read as follows: “*p*” look up in the current switch’s forwarding table at which port the packet is to be placed next, then “*l*” send the packet via the corresponding link, then “*dup*” log the current packet’s state in a history, and “ $(\dots)^*$ ” repeat.

Both *p* and *l* are basically case distinctions on the current packet’s switch. For instance, if the packet is currently at switch S_1 , then the corresponding sub-policies are given by

$$\begin{aligned} p_1 &\triangleq \text{if } dst = H_2 \text{ then } (pt \leftarrow 3 \oplus pt \leftarrow 4) \text{ else if } dst = H_1 \text{ then } pt \leftarrow 1 \text{ else drop} \\ l_1 &\triangleq \text{if } pt = 3 \text{ then } (sw \leftarrow S_3; pt \leftarrow 1) \text{ else if } pt = 4 \text{ then } (sw \leftarrow S_4; pt \leftarrow 1) \text{ else } pt = 1 \end{aligned} \quad (1)$$

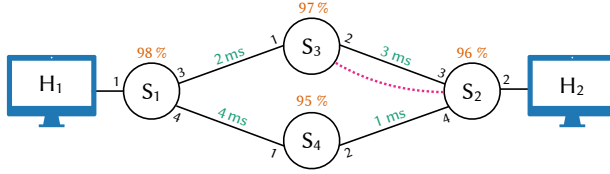


Fig. 2. Sample network with links now weighted by **latency** and switches weighted by **reliability**, and an optional retry **link** (dotted line) in the network.

p_1 branches on the packet’s destination: If the destination is H_2 , then the packet is forwarded via port 3 or 4, which is modeled via $wNetKAT$ ’s *choice operator* \oplus . Similarly, l_1 branches on the current port, and modifies the fields `sw` and `pt` according to the network’s topology.

2.2 Modeling Quantitative Network Behavior in $wNetKAT$

We have just exemplified how to model a network’s packet forwarding behavior in $wNetKAT$ when disregarding quantitative aspects. Let us now consider $wNetKAT$ ’s novel and generic capability of *modeling various quantitative aspects of networks*. Again, we proceed example-driven.

Consider the network in Figure 2. The topology and packet-forwarding behavior of this network coincides with the one from Figure 1. Both the switches and the links are now annotated with *quantitative information*: the switches are annotated by **success/failure rates**, i.e., the probability of succeeding in forwarding a packet. Links are annotated by **latencies**, i.e., the time it takes for packet to be sent via a particular link. $wNetKAT$ enables modeling these aspects in a natural manner.

The key idea in modeling this behavior is to introduce a *weighting operation* $r \odot p$, where p is a policy and r is an element from a fixed *semiring*. Intuitively, this operation says “execute policy p with weight r .” $wNetKAT$ is parametric in that fixed semiring, rendering it a generic language for modeling all kinds of quantitative aspects. Figure 3 provides an overview of different semirings and what aspects they are capable of modeling. Details on semirings and their operations are provided in Section 3. Let us, for now, gain some intuition for our example network.

With the appropriate semiring (Tropical or Arctic, depending on whether we are interested in best- or worst-case behavior), we can weight policies by **latencies**. To model the latencies attached to the *links* in Figure 2, we extend, e.g., the policy l_1 as follows:

$$l_1 \triangleq \text{if pt} = 3 \text{ then } 2\text{ms} \odot (\text{sw} \leftarrow S_3 ; \text{pt} \leftarrow 1) \text{ else} \\ \text{if pt} = 4 \text{ then } 4\text{ms} \odot (\text{sw} \leftarrow S_4 ; \text{pt} \leftarrow 1) \text{ else pt} = 1$$

Alternatively, by choosing the Viterbi semiring, we model a *switch*’s forwarding success rate (i.e., its reliability) by weighting the policy encoding its forwarding table. For, e.g., S_1 and S_2 , we have:

$$p_1 \triangleq 98\% \odot \text{if dst} = H_2 \text{ then } (\text{pt} \leftarrow 3 \oplus \text{pt} \leftarrow 4) \text{ else } \dots \\ p_2 \triangleq 96\% \odot \text{if dst} = H_2 \text{ then } \text{pt} \leftarrow 2 \text{ else } \dots$$

Note how the construct $r \odot p$ can be placed in different parts of the network policy to model quantities associated with different components (in the above links and switches). We also emphasize that the choice of the semiring determines the interplay of weighting \odot and choice \oplus , which influences whether we are modeling best- or worst-case behavior. Recall that switch S_1 may *choose* between forwarding the packet via port 3 or 4 if the destination is H_2 (modeled by \oplus in p_1 from (1) on page 3). Regarding, e.g., latencies it is thus natural to distinguish between the best- and the worst-case latency of a packet. Which of these cases we actually model depends on the semiring: The Tropical semiring resolves choices in a latency-minimizing manner (since \oplus is interpreted as a minimum) and the Arctic semiring resolves them in a maximizing manner (since \oplus maximizes).

Arctic semiring to model Latency or Information Leakage		$(\mathbb{N} \cup \{\infty, -\infty\}, \max, +, -\infty, 0)$
Weighting $r \odot p$: Policy p has latency r ms (or reveals r bits of information)	Choice $p \oplus q$: Choose policy with worse latency (or leakage)	Interpretation of $\llbracket p \rrbracket(h)$: Worst-case latency (or information leakage) of network paths
Probabilistic-union semiring to model Failure Rates		$([0, 1] \cup \{-\infty\}, \max, \uplus, -\infty, 0)$ where $r_1 \uplus r_2$ is the <i>probabilistic union</i> $r_1 + r_2 - r_1 \cdot r_2$
Weighting $r \odot p$: Policy p has a failure rate of r	Choice $p \oplus q$: Choose policy with higher failure rate	Interpretation of $\llbracket p \rrbracket(h)$: Worst-case failure rate of network paths
Tropical semiring to model Confidentiality or Cost		$(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$
Weighting $r \odot p$: Policy p reveals r bits of information (or has cost r)	Choice $p \oplus q$: Choose whichever policy reveals less bits of information (or has cheaper cost)	Interpretation of $\llbracket p \rrbracket(h)$: Best-case confidentiality (or cost) of network paths
Bottleneck semiring to model Network Bandwidth		$(\mathbb{N} \cup \{\infty, -\infty\}, \max, \min, -\infty, \infty)$
Weighting $r \odot p$: Restrict bandwidth of policy p to r Mbps	Choice $p \oplus q$: Choose policy with higher bandwidth	Interpretation of $\llbracket p \rrbracket(h)$: Best-case bandwidth of network paths
Viterbi semiring to model Reliability		$([0, 1], \max, \cdot, 0, 1)$
Weighting $r \odot p$: Policy p has a success rate of r	Choice $p \oplus q$: Choose policy with higher success rate	Interpretation of $\llbracket p \rrbracket(h)$: Best-case reliability of network paths
Security semiring to model Security Levels		$(0 < L < M < H, \max, \min, 0, H)$
Weighting $r \odot p$: Policy p has security level r	Choice $p \oplus q$: Choose policy with higher security level	Interpretation of $\llbracket p \rrbracket(h)$: Best-case security level of network paths
Why semiring to model Resource Tracking		(propositional <i>positive</i> DNF, $\vee, \wedge, 0, 1$)
Weighting $r \odot p$: Policy p uses resource r	Choice $p \oplus q$: Use resources from p or resources from q	Interpretation of $\llbracket p \rrbracket(h)$: Resources used by network paths
Boolean semiring to model NetKAT [3]		$(\{0, 1\}, \vee, \wedge, 0, 1)$
Weighting $r \odot p$: $1 \odot p = p$ and $0 \odot p = \text{drop}$	Choice $p \oplus q$: Nondeterministic choice between p and q	Interpretation of $\llbracket p \rrbracket(h)$: All possible network paths
Real Numbers semiring to model ProbNetKAT [10]		$(\mathbb{R}^{\geq 0} \cup \{\infty\}, +, \cdot, 0, 1)$
Weighting $r \odot p$: p has probability r	Choice $p \oplus q$: Probabilistic choice between p and q	Interpretation of $\llbracket p \rrbracket(h)$: Probability of each network path

Fig. 3. Instances of wNetKAT: examples of semirings $(S, +, \cdot, 0, \mathbb{1})$ and their use in networking.

2.3 From Modeling to Verification

Our first goal was to design a language that inherits the modeling aspects from classic NetKAT while enabling to model quantitative aspects in a generic and natural way. Our ultimate goal, however, is to obtain *effective procedures to fully automate quantitative reasoning about networks*.

Developing these effective procedures is challenging. In Section 3, we will present a semiring-valued denotational semantics $\llbracket p \rrbracket : \text{Pk} \rightarrow (\text{H} \rightarrow \mathcal{S})$ where Pk denotes the finite set of packets, H denotes the countably infinite set of histories (think: “traces” of packets) and \mathcal{S} denotes the chosen semiring. Intuitively, $\llbracket p \rrbracket(\pi)(h)$ is the weight (e.g., worst-case latency) p associates with the history h on input packet π . While the denotational semantics provides us with a ground truth for assigning meanings to (weighted) policies, it is not immediately amenable to automation—wNetKAT features unbounded iteration, thus denotationally it does not provide a finitary executable description.

Therefore, in the second part of our paper, we develop the novel notion of (finite-state) wNetKAT Automata (WNKA, for short). The idea is to *compile*—in an algorithmic manner—every wNetKAT policy p to a WNKA \mathcal{A}_p , which accepts a *weighted language over guarded strings*, i.e.,

$$\llbracket \mathcal{A}_p \rrbracket : \text{GS} \rightarrow \mathcal{S}, \quad \text{where the set of guarded strings is } \text{GS} \cong \text{Pk} \cdot (\text{Pk} \cdot \text{dup})^* \cdot \text{Pk}.$$

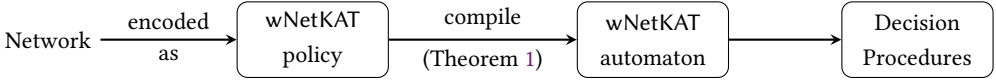


Fig. 4. Quantitative verification pipeline of wNetKAT instantiated with a semiring \mathcal{S} .

Guarded strings represent complete packet traces; intuitively, we can think of a guarded string x as the concatenation of an input packet π and an output history h , i.e., morally $x = \pi \cdot h$, so that

$$\underbrace{\llbracket \mathcal{A}_p \rrbracket (\pi \cdot h)}_{\text{operational semantics enabling effective reasoning}} = \underbrace{\llbracket p \rrbracket (\pi)(h)}_{\text{denotational semantics}} .$$

The WNKA \mathcal{A}_p does provide us with the finitary description required for the algorithmic verification of quantitative network aspects. To produce these automata models we have devised a Thompson-like construction specific to wNetKAT (Table 1). Like the classic Thompson construction for regular languages and NFAs, it operates recursively on the structure of a policy. However, as we will see in Section 5, this construction is far from being a trivial generalization as we cannot rely in ε -transitions when composing sub-automata (obtained recursively) and we must therefore employ a rather complex on-the-fly epsilon-elimination procedure in a weighted setting. The construction is further complicated by wNetKAT's so-called *carry-on packet semantics*: unlike traditional regular expressions, NetKAT is *stateful*, and the output packet of a transition is carried on to the next state. In wNetKAT this carry-on packet *additionally* incurs a weight that needs to be accounted for in the Thompson construction. This carry-on packet is also the reason why we cannot directly use classical weighted automata for the operational semantics and need to introduce a new automaton model. Let us now consider the algorithmic problems WNKAs enable us to tackle effectively.

2.4 Automatic Reasoning in wNetKAT

Our verification pipeline is depicted in Figure 4. With WNKAs, we tackle the following problems:

- (1) **r -Safety**: Do *all*—out of possibly infinitely many—traces through the network satisfy a given *upper bound* r on, e.g., latency or the overall probability of failure?
- (2) **r -Reachability**: Does *there exist* a trace through the network satisfying a given *lower bound* r on, e.g., a confidentiality measure or the probability of successfully transmitting a packet?
- (3) **Computing Weights**: Given an input packet π and a history h , what is the weight $\llbracket p \rrbracket (\pi)(h)$ policy p assigns to h on π ?

Our corresponding decision procedures are as generic as wNetKAT's modeling capabilities: In Section 6, we provide sufficient conditions on the semiring in order for the above problems to be decidable, and we provide corresponding generic decision procedures based on our WNKAs. The rest of this section is devoted to examples for the above problems.

2.4.1 Deciding Safety. The verification question of r -safety naturally arises when upper-bounding worst-case quantities associated with a network:

Example 1. *We can decide that all traffic in the network from Figure 2 has a latency of at most 5ms. For that, we encode the network in wNetKAT (Arctic semiring) and check whether the resulting policy p satisfies 5-safety: For all input packets π and all histories h , we have $\llbracket p \rrbracket (\pi)(h) \leq 5$.*

Moreover, if the safety property is violated, our decision procedure provides a *witness*: a trace with a weight greater than the safety threshold r . For example, suppose a network provider was considering adding the retry [link](#) (dotted line) to Figure 2. In this case, the network would no longer be 5-safe as a packet might be repeatedly forwarded back and forth between switches S_3 and S_2

A *semiring* is a structure $\mathcal{S} = (S, +, \cdot, \mathbb{0}, \mathbb{1})$, where S is a set equipped with two binary operations $+$, $\cdot : S \times S \rightarrow S$, and constants $\mathbb{0}, \mathbb{1} \in S$ satisfying:

- (1) $(S, +, \mathbb{0})$ is a commutative monoid,
- (2) $(S, \cdot, \mathbb{1})$ is a monoid,
- (3) *multiplication* distributes over *addition*.
- (4) multiplying with $\mathbb{0}$ is *annihilating*.

An ω -*continuous semiring* is a structure (\mathcal{S}, \preceq) :

- (1) \mathcal{S} is a semiring,
- (2) (\mathcal{S}, \preceq) is an ω -complete partial order,
- (3) \preceq is positive, i.e., $\mathbb{0}$ is least element of \preceq ,
- (4) both $+$ and \cdot are ω -continuous,
- (5) \mathcal{S} admits countable sums, defined as:

$$\sum_{i \in \mathbb{N}} s_i = \bigsqcup_{n \in \mathbb{N}} \underbrace{\sum_{i=0}^n s_i}_{\text{supremum of partial sums}}$$

Fig. 5. Definition of (ω -continuous) semirings. Semirings offer an algebraic basis to formalize weights and relevant operations, whereas ω -continuity captures the existence of adequate countable sums, essential for the semantics of iteration.

(accumulating an unbounded latency). Our decision procedure identifies this updated network as violating the safety property and provides a witness including the new link.

2.4.2 Deciding Reachability. The question of r -reachability is more natural to ask in settings that model *best-case* analyses, such as modeling reliability with the Viterbi semiring. In this setting, r -reachability corresponds to finding reliable paths between nodes.

Example 2. We verify, for the network from Figure 2, that host H_2 is reachable from H_1 with a reliability of at least 90% by deciding if the policy

$$\text{sw} = H_1 ; \text{net} ; \text{sw} = H_2$$

is 0.9-reachable (in the setting of $w\text{NetKAT}$ instantiated with the Viterbi semiring). If the property is satisfied, we provide a sample trace, i.e., an input packet and output history such that the associated weight is ≥ 0.9 . In this particular instance, the property is satisfied and the sample trace corresponds to the network path $S_1 \rightarrow S_3 \rightarrow S_2$ with a 91.26% reliability.

While the network in Figure 2 is simple, we can reason about much more complicated networks in $w\text{NetKAT}$. In Section 7, we model Internet2’s Abilene backbone network in $w\text{NetKAT}$. Abilene features several *nodes* across major cities in the United States; we use the network to showcase the verification of quantitative network behavior in a real-world setting.

In the rest of the paper, we make precise the notions covered throughout this section and then tie everything back to these examples of quantitative network verification with a case study over the Abilene network. We first provide a formal definition of the syntax and semantics of $w\text{NetKAT}$ (Section 3), followed by a language model (Section 4) and an operational semantics based on $w\text{NetKAT}$ automata (Section 5). Finally, we describe the verification of r -safety and r -reachability through decision procedures at the level of $w\text{NetKAT}$ automata (Section 6), and apply them in the real-world setting of Internet2’s Abilene backbone network (Section 7).

3 $w\text{NetKAT}$: Syntax and Semantics

$w\text{NetKAT}$ is parameterized by an ω -continuous semiring, a class of semirings admitting *countably infinite* sums (see Figure 5). We restrict to ω -continuous semirings as unbounded iteration $(-)^*$ is naturally captured by an infinite choice—the semantics of which then needs to make use of these countable sums to be well-defined. We discuss this further in Section 3.2.

We fix an ω -continuous semiring \mathcal{S} throughout this section and the rest of the paper. To emphasize the instantiation with a given semiring \mathcal{S} , we sometimes refer to \mathcal{S} -wNetKAT (and write just wNetKAT when the semiring is clear from context).

3.1 Syntax

The syntax of wNetKAT is shown in Figure 6 (left). We fix a *finite* set of (*packet header*) values Val that each of the finitely many *fields* F can take. A *packet* π in the set Pk is thus a finite function of type $F \rightarrow \text{Val}$, assigning a value to each field. We usually write $\pi.f$ instead of $\pi(f)$ and denote packets in record notation $\{f_1 = n_1, \dots, f_k = n_k\}$. A (*packet*) *history* $h = \pi::\tilde{h}$ is a *non-empty* list of packets, where π is the *head packet* and \tilde{h} is the (possibly empty) *tail*. We sometimes abuse notation, and write $\pi::h$ instead of $\pi::\tilde{h}$. Finally, elements r of the semiring \mathcal{S} are called *weights*.

Predicates t are Boolean combinations of false, true, and *tests* of the form $f = n$. They act as filters: If the current packet does not satisfy t , the packet is dropped. The *modification* $f \leftarrow n$ assigns the value n to the field f in the current packet. *Sequential composition* $p; q$ first executes p and then executes q . *Weighting* $r \circ p$ weights the execution of p by the semiring element r . The *choice* $p \oplus q$ executes either p or q . The primitive dup can be understood as a logging command for keeping track of a packet's trajectory through a network. *Iteration* p^* is, intuitively, a countably infinite choice between terminating or keeping iterating p , i.e., schematically, p^* is equivalent to

$$\text{skip} \oplus p \oplus p;p \oplus p;p;p \oplus \dots,$$

where we write skip (drop) instead of true (false) to emphasize the behavior of the predicate when used as a policy. Finally, we can also encode usual control-flow structures in wNetKAT:

$$\begin{aligned} \text{skip} &\triangleq \text{true} & \text{if } t \text{ then } p \text{ else } q &\triangleq t;p \oplus \neg t;q \\ \text{drop} &\triangleq \text{false} & \text{while } t \text{ do } p &\triangleq (t;p)^*; \neg t \end{aligned}$$

These encodings are justified by the semantics of wNetKAT, which we go over next.

3.2 Semantics

The semantics of wNetKAT is shown in Figure 6 (right). In what follows, we first introduce so-called *weightings*—the central semantic objects. We then detail the semantics of the individual constructs and state various desirable properties such as a fixed point characterization of iteration.

3.2.1 Weightings. Intuitively, a policy p takes as input a history h and produces a set of output histories h' , where *each output history is weighted by some element from \mathcal{S}* . To formalize an appropriate semantic domain, we introduce the following central objects:

Definition 1 (Weightings). *We define weightings over a set X as the tuple $(\mathcal{W}_{\mathcal{S}}(X), \eta, \gg=)$, where:*

- (1) $\mathcal{W}_{\mathcal{S}}(X) \triangleq \{m: X \rightarrow \mathcal{S} \mid \text{supp}(m) \text{ is countable}\}$ with $\text{supp}(m) \triangleq \{x \in X \mid m(x) \neq 0\}$,
- (2) $\eta: X \rightarrow \mathcal{W}_{\mathcal{S}}(X)$ is the unit, defined as $\eta(x) \triangleq \lambda y. [x = y]$.¹
- (3) $\gg=: \mathcal{W}_{\mathcal{S}}(X) \rightarrow (X \rightarrow \mathcal{W}_{\mathcal{S}}(X)) \rightarrow \mathcal{W}_{\mathcal{S}}(X)$ is the bind, defined as (using infix notation)

$$m \gg= f \triangleq \lambda y. \sum_{x \in \text{supp}(m)} m(x) \cdot f(x)(y)$$

This sum is well-defined as $\text{supp}(m)$ is countable and ω -continuous semirings admit countable sums. Δ

We often call weightings the elements of $\mathcal{W}_{\mathcal{S}}(X)$ and denote them by m, m' , and variations thereof. $\text{supp}(m)$ is called the *support* of m . It is easy to check that the monad axioms hold for the operations as defined above, and therefore $(\mathcal{W}_{\mathcal{S}}(X), \eta, \gg=)$ forms a monad.

¹Here, and throughout the paper, we use Iverson bracket notation: $[P] = 1$ if the proposition P holds and $[P] = 0$ otherwise.

Syntax

Values	$\text{Val} \ni n ::= v_1 \mid \dots \mid v_n$
Fields	$F \ni f ::= f_1 \mid \dots \mid f_k$
Packets	$\text{Pk} \ni \pi, \alpha, \beta, \gamma ::= \{f_1 = n_1, \dots, f_k = n_k\}$
Histories	$\text{H} \ni h ::= \pi :: \tilde{h}$ $\tilde{h} ::= \langle \rangle \mid \pi :: \tilde{h}$
Weights	$\mathcal{S} \ni r, s$
Predicates	$\text{Pred} \ni t, u ::= \text{false} \quad \text{False/Drop}$ $\quad \quad \quad \mid \text{true} \quad \text{True/Skip}$ $\quad \quad \quad \mid f = n \quad \text{Test}$ $\quad \quad \quad \mid t \vee u \quad \text{Disjunction}$ $\quad \quad \quad \mid t \wedge u \quad \text{Conjunction}$ $\quad \quad \quad \mid \neg t \quad \text{Negation}$
Policies	$\text{Pol} \ni p, q ::= t \quad \text{Filter}$ $\quad \quad \quad \mid f \leftarrow n \quad \text{Modification}$ $\quad \quad \quad \mid \text{dup} \quad \text{Duplication}$ $\quad \quad \quad \mid p ; q \quad \text{Seq. Comp.}$ $\quad \quad \quad \mid r \odot p \quad \text{Weighting}$ $\quad \quad \quad \mid p \oplus q \quad \text{Choice}$ $\quad \quad \quad \mid p^* \quad \text{Iteration}$

Semantics

$\llbracket t \rrbracket_{\text{Pred}} : \text{Pk} \rightarrow 2$
$\llbracket \text{false} \rrbracket_{\text{Pred}}(\pi) = 0$
$\llbracket \text{true} \rrbracket_{\text{Pred}}(\pi) = 1$
$\llbracket f = n \rrbracket_{\text{Pred}}(\pi) = [\pi.f = n]$
$\llbracket t \vee u \rrbracket_{\text{Pred}}(\pi) = \llbracket t \rrbracket_{\text{Pred}}(\pi) \vee \llbracket u \rrbracket_{\text{Pred}}(\pi)$
$\llbracket t \wedge u \rrbracket_{\text{Pred}}(\pi) = \llbracket t \rrbracket_{\text{Pred}}(\pi) \wedge \llbracket u \rrbracket_{\text{Pred}}(\pi)$
$\llbracket \neg t \rrbracket_{\text{Pred}}(\pi) = [\llbracket t \rrbracket_{\text{Pred}}(\pi) = 0]$
$\llbracket p \rrbracket : \text{H} \rightarrow \mathcal{W}_{\mathcal{S}}(\text{H})$
$\llbracket t \rrbracket(\pi :: \tilde{h}) = [\llbracket t \rrbracket_{\text{Pred}}(\pi) = 1] \cdot \eta(\pi :: \tilde{h})$
$\llbracket f \leftarrow n \rrbracket(\pi :: \tilde{h}) = \eta(\pi[f := n] :: \tilde{h})$
$\llbracket \text{dup} \rrbracket(\pi :: \tilde{h}) = \eta(\pi :: \pi :: \tilde{h})$
$\llbracket p ; q \rrbracket(h) = \llbracket p \rrbracket(h) \gg \llbracket q \rrbracket$
$\llbracket r \odot p \rrbracket(h) = r \cdot \llbracket p \rrbracket(h)$
$\llbracket p \oplus q \rrbracket(h) = \llbracket p \rrbracket(h) + \llbracket q \rrbracket(h)$
$\llbracket p^* \rrbracket(h) = \sum_{n \in \mathbb{N}} \llbracket p^{(n)} \rrbracket(h)$
where $p^{(0)} = \text{skip}$ and $p^{(n+1)} = p ; p^{(n)}$

Fig. 6. Syntax and Semantics of \mathcal{S} -wNetKAT, where $\mathcal{S} = (\mathcal{S}, +, \cdot, \mathbb{0}, \mathbb{1})$ is an ω -continuous semiring. We assume that the operators bind stronger in the order: $\neg, \wedge, \vee, ;, \odot, \oplus$.

We lift the operations and the order of the semiring \mathcal{S} pointwise to $\mathcal{W}_{\mathcal{S}}(X)$, i.e., for $r \in \mathcal{S}$, $m_1, m_2 \in \mathcal{W}_{\mathcal{S}}(X)$, and $\{m_i\}_{i \in I} : I \rightarrow \mathcal{W}_{\mathcal{S}}(X)$ a family in $\mathcal{W}_{\mathcal{S}}(X)$ indexed by I , we define:

$$\begin{aligned}
 r \cdot m &\triangleq \lambda x. r \cdot m(x) & \mathbb{0} &\triangleq \lambda x. \mathbb{0} \\
 m \cdot r &\triangleq \lambda x. m(x) \cdot r & m_1 + m_2 &\triangleq \lambda x. m_1(x) + m_2(x) \\
 m_1 \preceq m_2 &\text{ iff } \forall x: m_1(x) \preceq m_2(x) & \sum_{i \in I} m_i &\triangleq \lambda x. \sum_{i \in I} m_i(x)
 \end{aligned}$$

Finally, as weightings and their associated operations are lifted from ω -continuous semirings, they satisfy many expected commutativity, associativity, and distributivity properties .

3.2.2 The Denotational Semantics of Policies. Intuitively, a policy p takes as input a history h and produces a set of output histories h' , each output being weighted by some element from \mathcal{S} . The notion of weightings formalizes this: the semantics $\llbracket p \rrbracket$ of a policy p is of type $\text{H} \rightarrow \mathcal{W}_{\mathcal{S}}(\text{H})$, i.e., each input history h is mapped to a weighting $\llbracket p \rrbracket(h)$ of (output) histories, and $\llbracket p \rrbracket(h)(h')$ is the weight p assigns to the output history h' when executed on the input history h . The set of all histories produced by p on input h is $\text{supp}(\llbracket p \rrbracket(h))$, i.e., all histories to which $\llbracket p \rrbracket(h)$ assigns a non- $\mathbb{0}$ weight. It is useful to note that, operationally, only the *head* of the input h is relevant for the execution of p in the sense that for all packets π and all histories h, h' , we have

$$\llbracket p \rrbracket(\pi :: \langle \rangle)(h') = \llbracket p \rrbracket(\pi :: h)(h' :: h) .$$

Predicates, modification, and duplication produce at most one output history (behaving analogously to NetKAT [3]). We embed their semantics into wNetKAT via the unit η of $\mathcal{W}_{\mathcal{S}}(\text{H})$. Sequential composition, weighting, choice, and iteration yield wNetKAT's generic capabilities for modeling *quantitative* aspects of networks and require a more involved treatment.

Predicates. Recall that predicates t are Boolean combinations of false, true, and tests $f = n$. Intuitively, if the head packet π satisfies t , then executing t does not alter the input history—it is effectless in the sense that it simply outputs the input history. Otherwise, i.e., if π does not satisfy t , then π is dropped. Semantically, this is captured as follows: $\llbracket t \rrbracket(\pi::\hat{h})(h') = [\pi \text{ satisfies } t \text{ and } \pi::\hat{h} = h']$.

Modification. The policy $f \leftarrow n$ sets the field f of the input history's head packet to n . To capture this, we use $\pi[f := n]$ to denote the packet obtained from π by updating the value of the field f to the value n and define $\llbracket f \leftarrow n \rrbracket(\pi::\hat{h}) = \eta(\pi[f := n]::\hat{h})$, which is $\mathbb{1}$ only when $h' = \pi[f := n]::\hat{h}$.

Duplication. `dup` is intended to be a logging statement for keeping track of a packet's trajectory through a network. Its semantics makes this explicit: $\llbracket \text{dup} \rrbracket(\pi::\hat{h}) = \eta(\pi::\pi::\hat{h})$.

Weighting and Choice. All constructs considered so far produce only $\{0, \mathbb{1}\}$ -valued weightings. Weighting and choice bring the capability of modeling *quantitative* aspects of networks to wNetKAT. Let us consider an example to illustrate how these constructs act in concert.

Example 3. Let $\mathcal{S} = (\mathbb{N} \cup \{\infty, -\infty\}, \max_{\mathbb{N}}, +_{\mathbb{N}}, -\infty, 0)$ be the Arctic semiring and consider the simple policies

$$p_1 = 3 \odot f \leftarrow 1 \quad \text{and} \quad p_2 = 5 \odot f \leftarrow 2.$$

We have $\llbracket p_1 \rrbracket(\pi::\hat{h})(h') = 3 +_{\mathbb{N}} \llbracket f \leftarrow 1 \rrbracket(\pi::\hat{h})(h')$, where $\llbracket f \leftarrow 1 \rrbracket(\pi::\hat{h})(h') = 0$ if $h' = \pi[f := 1]::\hat{h}$ and $-\infty$ otherwise. Hence, $\llbracket p_1 \rrbracket(\pi::\hat{h})(h') = 3$ when $h' = \pi[f := 1]::\hat{h}$ enabling us to model that

“Modifying the input packet by setting f to 1 incurs a cost (latency) of 3”

and similarly for p_2 . Let us now combine p_1 and p_2 via a choice, i.e., let $p = p_1 \oplus p_2$. We have

$$\llbracket p \rrbracket(\pi::\hat{h}) = \max_{\mathbb{N}}(\llbracket p_1 \rrbracket(\pi::\hat{h}), \llbracket p_2 \rrbracket(\pi::\hat{h})) = \lambda \pi'::\hat{h}'. \begin{cases} 3 & \text{if } \pi' = \pi[f := 1] \text{ and } \hat{h} = \hat{h}' \\ 5 & \text{if } \pi' = \pi[f := 2] \text{ and } \hat{h} = \hat{h}' \\ -\infty & \text{otherwise.} \end{cases}$$

Here, because the assignments in each summand of p are different the result of the semiring addition $\max_{\mathbb{N}}(\llbracket p_1 \rrbracket(\pi::\hat{h}), \llbracket p_2 \rrbracket(\pi::\hat{h}))$ will be of the shape $\max_{\mathbb{N}}(n, -\infty)$ for $n \in \{3, 5\}$, producing two different output histories in the support. Now consider a small change in p_2 using the same output history:

$$p_1 = 3 \odot f \leftarrow 1 \quad \text{and} \quad p_2 = 5 \odot \boxed{f \leftarrow 1}.$$

How does $p = p_1 \oplus p_2$ behave now? We have $\llbracket p \rrbracket(\pi::\hat{h})(h) = \max_{\mathbb{N}}(\llbracket p_1 \rrbracket(\pi::\hat{h})(h), \llbracket p_2 \rrbracket(\pi::\hat{h})(h)) = \max_{\mathbb{N}}(3, 5)$ (or vice-versa), which is 5 if $h = \pi[f := 1]::\hat{h}$. i.e., the choice is resolved in a cost maximizing manner. This emphasizes how the semiring operations determine the interplay of weighting and choice. If, e.g., instead of the Arctic semiring, we were to choose the Tropical semiring $\mathcal{S} = (\mathbb{N} \cup \{\infty\}, \min_{\mathbb{N}}, +_{\mathbb{N}}, \infty, 0)$, the choice is resolved in a cost minimizing manner and $\llbracket p \rrbracket(\pi::\hat{h})(h) = 3$.

Sequential Composition. The weightings produced by sequentially composing policies is naturally captured by the bind $\gg=$ operation of the monad $\mathcal{W}_{\mathcal{S}}(H)$ of weightings. We have

$$\llbracket p ; q \rrbracket(h)(h') = (\llbracket p \rrbracket(h) \gg= \llbracket q \rrbracket)(h') = \sum_{h'' \in \text{supp}(\llbracket p \rrbracket(h))} \llbracket p \rrbracket(h)(h'') \cdot \llbracket q \rrbracket(h'')(h').$$

This is intuitive: The weight $p ; q$ assigns to h' on input h is obtained by summing over all *intermediate outputs* h'' that p produces on h . For each such h'' , we multiply the weight p assigns to h'' on input h by the weight q assigns to h' on input h'' , which captures the sequential behavior of $p ; q$. Note that the semantics of conjunction of two predicates coincide with their sequencing, i.e., $\llbracket t \wedge u \rrbracket_{\text{Pred}} = \llbracket t ; u \rrbracket$; we sometimes use the two combinators interchangeably for predicates.

Complete Tests

$$\text{Pk}^? \ni \pi? \triangleq f_1 = \pi.f_1 \wedge \dots \wedge f_k = \pi.f_k$$

Complete Assignments

$$\text{Pk}^! \ni \pi! \triangleq f_1 \leftarrow \pi.f_1; \dots; f_k \leftarrow \pi.f_k$$

Reduced Policies

$$\begin{aligned} \text{Pol}^\downarrow \ni p, q &::= \pi? \mid \pi! \mid \text{dup} \mid p; q \\ &\mid r \odot p \mid p \oplus q \mid p^* \end{aligned}$$

Fig. 7. Reduced wNetKAT Syntax.

Iteration. Recall that p^* is, intuitively, a countably infinite choice between terminating or keeping iterating p . Semantically, this behavior is captured by the countable sum

$$\llbracket p^* \rrbracket(h) = \sum_{n \in \mathbb{N}} \llbracket p^{(n)} \rrbracket(h) = \llbracket \text{skip} \rrbracket(h) \oplus \llbracket p \rrbracket(h) \oplus \llbracket p; p \rrbracket(h) \oplus \dots$$

As a sanity check for this definition, we establish the usual least fixed characterization from KAT and NetKAT [3], i.e., $\llbracket p^* \rrbracket$ is the least solution of $\llbracket p^* \rrbracket = \llbracket \text{skip} \oplus p; p^* \rrbracket$.

4 Language Model

In this section, we define the language model of wNetKAT, in which each policy p is assigned a *weighting of guarded strings*, generalizing NetKAT's languages of guarded strings.

4.1 Reduced Syntax

First, we restrict all wNetKAT policies to a reduced syntax, see Figure 7, without loss of expressivity. At the core of our reduced syntax are *complete tests* and *complete assignments*. A *complete test* is a conjunction of tests $f_1 = n_1 \wedge \dots \wedge f_k = n_k$, covering all $f_i \in \mathbb{F}$. In particular, note that this conjunction over all fields means that complete tests precisely match one and only one packet: $\pi \triangleq \{f_1 = n_1, \dots, f_k = n_k\}$. As such the complete test matching packet π is labelled $\pi?$, and the set of all complete tests denoted $\text{Pk}^?$. A complete test is often called an *atom* as complete tests are precisely the minimal nonzero elements of the Boolean algebra generated by basic tests $f_i = n_i$. Dually, a *complete assignment* is an expression $\pi! \triangleq f_1 \leftarrow n_1; \dots; f_k \leftarrow n_k$. We call $\text{Pk}^!$ the set of all complete assignments. It is easy to see that there are isomorphisms between $\text{Pk}^!$, Pk , and $\text{Pk}^?$. Hence, we often use simply $\pi \in \text{Pk}$ to represent the respective complete test or assignment.

Note that every reduced policy $p \in \text{Pol}^\downarrow$ is itself a standard wNetKAT policy (i.e., $\text{Pol}^\downarrow \subset \text{Pol}$). Most importantly, every policy can be converted to a semantically equivalent *reduced* policy. Therefore, from now on, we *assume w.l.o.g. that all policies are reduced*.

4.2 Guarded Strings: Basic Notation and Operations

Guarded strings appeared originally in the work of Kaplan [17] and later played a prominent role in the work of Kozen [18] to reason about program (trace) equivalence. Formally, guarded strings are elements of the set $\text{GS} \subset \text{Pol}^\downarrow$, defined inductively as:

$$\text{GS} \triangleq \bigcup_{i \in \mathbb{N}} \text{GS}^i \quad \text{GS}^0 \triangleq \{\alpha?; \beta! \mid \alpha, \beta \in \text{Pk}\} \quad \text{GS}^{i+1} \triangleq \{x; \text{dup}; \gamma! \mid x \in \text{GS}^i, \gamma \in \text{Pk}\}$$

Guarded strings encompass the minimal nonzero elements of the standard model of NetKAT and represent *complete packet traces*. This is analogous to the language models of KA(T) in which expressions are interpreted as regular sets of minimal nonzero (join-irreducible) terms. For convenience, we will exploit the above mentioned isomorphisms of complete tests and assignments, and represent guarded strings as regular strings over packets ($\text{GS} \cong \text{Pk} \cdot (\text{Pk} \cdot \text{dup})^* \cdot \text{Pk}$). A crucial difference, however, between guarded and regular strings is that the concatenation operation captures the consistency of state between two sequentially composed programs. Given two guarded strings

$p \in \text{Pol}$	$G(p)(x) \in \mathcal{S}$	Guarded Strings
$\pi?$	$[x = \pi \pi]$	$\text{GS} \cong \text{Pk} \cdot (\text{Pk} \cdot \text{dup})^* \cdot \text{Pk}$
$\pi!$	$[\exists \alpha \in \text{Pk}. x = \alpha \pi]$	Guarded Concatenation: $\diamond: \text{GS}^2 \rightarrow \text{GS}$
dup	$[\exists \alpha \in \text{Pk}. x = \alpha \alpha \text{ dup } \alpha]$	$\alpha x \beta \diamond \gamma y \xi = \begin{cases} \alpha x y \xi & \beta = \gamma \\ \text{undefined} & \beta \neq \gamma \end{cases}$
$p_1 \oplus p_2$	$G(p_1)(x) + G(p_2)(x)$	Lifted Guarded Concatenation
$p_1 ; p_2$	$(G(p_1) \diamond G(p_2))(x)$	$\diamond: \mathcal{W}_{\mathcal{S}}(\text{GS})^2 \rightarrow \mathcal{W}_{\mathcal{S}}(\text{GS})$
$r \odot p_1$	$r \cdot G(p_1)(x)$	$(m_1 \diamond m_2)(x) \triangleq \sum_{\substack{x_i \in \text{supp}(m_i) \\ x = x_1 \diamond x_2}} m_1(x_1) \cdot m_2(x_2)$
p_1^*	$\sum_{n \in \mathbb{N}} G(p_1^{(n)})(x)$	

Fig. 8. Language Model.

$\alpha x \beta, \gamma y \xi \in \text{GS}$ their *guarded concatenation* (see Figure 8 (right)) $\alpha x \beta \diamond \gamma y \xi$ is a *partial* operation. The final state β of the first string has to be compatible with the initial state γ of the second string, that is $\beta = \gamma$ for the concatenation to be defined: $\alpha x \beta \diamond \gamma y \xi = \alpha x y \xi$. Note that guarded concatenation can be lifted to a *total* operation over weightings of guarded strings (see Figure 8).

4.3 Language Model for wNetKAT

We now have all the ingredients to define the language model of wNetKAT as a class of functions $G: \text{Pol} \rightarrow \mathcal{W}_{\mathcal{S}}(\text{GS})$. We remark that NetKAT models were given by regular sets of guarded strings, or equivalently, functions $G: \text{Pol} \rightarrow 2^{\text{GS}}$ so we are generalizing the Boolean semiring underlying sets to an arbitrary ω -continuous semiring (note that $2^{\text{GS}} \cong \mathcal{W}_2(\text{GS})$).

Definition 2 (Language Model). *Let p be a wNetKAT policy. We define the weighted language $G(p)$ of p as the weighting $G(p): \mathcal{W}_{\mathcal{S}}(\text{GS})$ given by the table in Figure 8 (left), where we write $G(p)(x)$ for the weight attributed to $x \in \text{GS}$ by p .* \triangle

Lemma 1 (Denotational–Language Correspondence). *For all $h \in \text{H}$ and $p \in \text{Pol}$:*

$$\llbracket p \rrbracket(h) = \sum_{x \in \text{supp}(G(p))} G(p)(x) \cdot \llbracket x \rrbracket(h)$$

The reader familiar with formal power series [6] might notice the similarity between the inductive definition $G(p)$ and *rational functions*. There is a crucial difference with the presence of \diamond , but we will show that, for ω -continuous semirings, such functions can be recognized by a special finite automaton (Section 5). Furthermore, the language model enables reasoning about the behavior of wNetKAT policies on *complete traces* of the network, rather than separate input/output histories.

Example 4. *Consider the wNetKAT policy $(3 \odot \text{dup})^*$ over the semiring of the extended naturals $\mathbb{N}^\infty = (\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$. Then over a single $\alpha \in \text{Pk}$, $G(p)(\alpha (\alpha \text{ dup})^{(n)} \alpha) = 3^n$.*

Note how in Example 4 the star allows for the unbounded sequencing of zero or more dups, each accruing a weight multiplier of 3. This is because, if we intuitively unfold our example policy, we would see that the non-zero output weight of the guarded string $x = \alpha (\alpha \text{ dup})^{(n)} \alpha$ is produced entirely by the subexpression $(3 \odot \text{dup})^{(n)}$:

$$\underbrace{(3 \odot \text{dup})^*}_{x \mapsto 3^n} \equiv \bigoplus_{n \in \mathbb{N}} (3 \odot \text{dup})^{(n)} \equiv \underbrace{\text{skip}}_{x \mapsto 0} \oplus \underbrace{(3 \odot \text{dup})}_{x \mapsto 0} \oplus \underbrace{(3 \odot \text{dup})^2}_{x \mapsto 0} \oplus \cdots \oplus \underbrace{(3 \odot \text{dup})^{(n)}}_{x \mapsto 3^n} \oplus \cdots$$

This is by no accident: unlike common string concatenation, one cannot construct longer guarded strings through the concatenation of dup-free strings. For instance: $\alpha \alpha \diamond \alpha \alpha = \alpha \alpha$. It is the dup construct that entirely determines the length of guarded strings, preventing their concatenation from collapsing the intermediate state. After all, the purpose of dup from a denotational perspective is precisely to *duplicate* the current packet, freezing it and producing a longer history as its output. By attributing a weight to dup, we are associating the weight (be it cost, latency, reliability, etc.) of taking another *hop* in our network, thus allowing us to reason about its paths. We can, however, also reason about the overall input-output behavior of a network, without restricting to specific network paths. In that case, we consider only dup-free policies.

Example 5. Consider the policy $((\{a\} \odot \alpha!) \oplus (\{b\} \odot \beta!))^*$ over complete assignments $\alpha!, \beta!$, and the formal language semiring $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$, for $\Sigma = \{a, b\}$. We compute $G(p)(x)$ for subexpressions p of our policy. We consider only dup-free strings, as dup-free policies always assign \emptyset to longer strings.

$x \in \mathbf{GS}$	$\{a\} \odot \alpha!$	$\{b\} \odot \beta!$	$(\{a\} \odot \alpha!) \oplus (\{b\} \odot \beta!)$	$((\{a\} \odot \alpha!) \oplus (\{b\} \odot \beta!))^*$
$\alpha \alpha$	$\{a\}$	\emptyset	$\{a\}$	$\{\varepsilon, wa \mid w \in \Sigma^*\}$
$\alpha \beta$	\emptyset	$\{b\}$	$\{b\}$	$\{wb \mid w \in \Sigma^*\}$
$\beta \alpha$	$\{a\}$	\emptyset	$\{a\}$	$\{wa \mid w \in \Sigma^*\}$
$\beta \beta$	\emptyset	$\{b\}$	$\{b\}$	$\{\varepsilon, wb \mid w \in \Sigma^*\}$

Note that unlike our dup example above, a dup-free expression does not constrain at all the summands $p^{(n)}$ that produce non-zero weights as part of the star computation. For $p \triangleq ((\{a\} \odot \alpha!) \oplus (\{b\} \odot \beta!))^*$, and an example input dup-free guarded string $x \triangleq \alpha \alpha$ we have that:

$$\underbrace{p^*}_{x \mapsto \{\varepsilon, wa \mid w \in \Sigma^*\}} \equiv \bigoplus_{n \in \mathbb{N}} (p)^{(n)} \equiv \underbrace{\text{skip}}_{x \mapsto \{\varepsilon\}} \oplus \underbrace{p}_{x \mapsto \{a\}} \oplus \underbrace{p^{(2)}}_{x \mapsto \{aa, ba\}} \oplus \dots \oplus \underbrace{p^{(n)}}_{x \mapsto \{wa \mid w \in \Sigma^n\}} \oplus \dots$$

One may find it unintuitive that the same dup-less string $x \triangleq \alpha \alpha$ is assigned non-zero weight not only by skip, but also by every positive n -th exponentiation of p . After all, the semantics of n -th iteration requires that the input guarded string be a representative trace of the “ n -times sequencing” of p . However, note that the nuance of guarded concatenation (Figure 8) allows for precisely this:

$$\begin{aligned} G(p^{(2)})(\alpha \alpha) &= G(p; p)(\alpha \alpha) = (G(p) \diamond G(p))(\alpha \alpha) = G(p)(\alpha \alpha) \cdot G(p)(\alpha \alpha) \\ &\quad + G(p)(\alpha \beta) \cdot G(p)(\beta \alpha) \end{aligned}$$

Intuitively, this is because guarded concatenation of dup-free strings will always collapse any intermediate state back into a minimal, dup-free string: $\alpha \alpha \diamond \alpha \alpha = \alpha \alpha = \alpha \beta \diamond \beta \alpha$. We can extend this behavior to as many concatenations of dup-free strings as needed to “match” a given $p^{(n)}$. This is crucial to model the input-output behavior of networks with cycles, where in the absence of trace information, one has no control over how many times a cycle is traversed from a given input-output packet pair. As such, computing the exact semantics of star requires that we compute the semantics of an infinite amount of policies: namely, the n -th iterates of the policy, and then sum them. Any finite approximation will yield an incorrect result. As seen above, $G(p^*)(\alpha \alpha)$ is an infinite language, but all its n -th iterates produce only finite languages. We achieve the exact computation of wNetKAT policies through the use of an automata-based operational semantics.

5 wNetKAT Automata

In this section, we will present an operational semantics for wNetKAT. We will define a special weighted automaton model—wNetKAT automata—and show how to construct a finite automaton from any wNetKAT expression. wNetKAT Automata (WNKA) resemble classic weighted automata, albeit adjusted to the specifics of wNetKAT and its guarded string-based language model. While classic weighted automata would only consume a symbol at a time, WNKA consume packets in *linked pairs*, as processing guarded strings requires keeping a state of the “previous packet”.

Definition 3 (wNetKAT Automaton). *A wNetKAT automaton (WNKA) is a 4-tuple $\mathcal{A} = (Q, \iota, \delta, \lambda)$ where Q is a finite set of states, $\iota: \mathcal{W}_S(Q)$ is the initial weighting, δ is a family of transition functions $\delta_{\alpha\beta}: Q \rightarrow \mathcal{W}_S(Q)$ indexed by packet pairs, and λ is a family of output weightings $\lambda_{\alpha\beta}: \mathcal{W}_S(Q)$. \triangle*

The above definition is similar to that of weighted automata: weightings $\mathcal{W}_S(Q)$ are simply S -valued “vectors” over Q (i.e., elements of the Q -semimodule over S), and the transition functions $\delta_{\alpha\beta}: Q \rightarrow \mathcal{W}_S(Q)$ are matrices $Q \times Q \rightarrow S$. We push this analogy further and note that functions $\delta_{\alpha\beta}: Q \rightarrow \mathcal{W}_S(Q)$ are isomorphic to $\mathcal{W}_S(Q \times Q)$, which we shall call *weighting matrices*. Given weighting matrices $m: \mathcal{W}_S(X \times Y)$ and $m': \mathcal{W}_S(Y \times Z)$ we define their product $m \times m': \mathcal{W}_S(X \times Z)$:

$$m \times m' \triangleq \lambda(x, z). \sum_{(x, y) \in \text{supp}(m)} m(x, y) \cdot m'(y, z)$$

The initial and output weightings can be equivalently represented as matrices $\iota: \mathcal{W}_S(1 \times Q)$ and $\lambda_{\alpha\beta}: \mathcal{W}_S(Q \times 1)$. With these notational conventions in hand, we can now more easily define the weighted language of guarded strings $\llbracket \mathcal{A} \rrbracket: \mathcal{W}_S(\text{GS})$ recognized by an WNKA \mathcal{A} :

$$\llbracket \mathcal{A} \rrbracket(\pi_0 \pi_1 \text{ dup } \pi_2 \text{ dup } \dots \text{ dup } \pi_n) \triangleq \iota \times \delta_{\pi_0 \pi_1} \times \delta_{\pi_1 \pi_2} \times \dots \times \delta_{\pi_{n-2} \pi_{n-1}} \times \lambda_{\pi_{n-1} \pi_n}$$

Note that when $n = 1$ we obtain $\llbracket \mathcal{A} \rrbracket(\pi_0 \pi_1) = \iota \times \lambda_{\pi_0 \pi_1}$.

Example 6. *Consider the policy $p \triangleq (3 \odot \text{dup})^*$ from Example 4. We define a minimal automaton \mathcal{A}_p such that $\llbracket \mathcal{A}_p \rrbracket = G(p)$, as below. We use single-line arrows for transitions between states and for initial weights, and double-line arrows for the output weight function on each state.*

$$\begin{array}{ccc} \begin{array}{c} \xrightarrow{1} s_0 \\ \bullet \\ \downarrow \\ [\alpha = \beta] \end{array} & \begin{array}{c} \xrightarrow{3 \cdot [\alpha = \beta]} \\ \bullet \\ \xrightarrow{3 \cdot [\alpha = \beta]} \end{array} & \begin{array}{l} \iota(s_0) \triangleq 1 \quad \delta_{\alpha\beta}(s_0) \triangleq 3 \cdot [\alpha = \beta] \\ \lambda_{\alpha\beta}(s_0) \triangleq [\alpha = \beta] \end{array} \end{array}$$

Note how the automaton’s transitions are labeled by conditions on both the current packet and the previous one—the packet-pair semantics of our WNKA. We can then perform the computation of $G(p)(\alpha (\alpha \text{ dup})^{(n)} \alpha)$ as done through the automaton:

$$\llbracket \mathcal{A}_p \rrbracket(\alpha (\alpha \text{ dup})^{(n)} \alpha) = \iota \times \delta_{\alpha\alpha}^n \times \lambda_{\alpha\alpha} = (1) \times (3 \cdot [\alpha = \alpha])^n \times ([\alpha = \alpha]) = 3^n$$

Although simple, the automaton of Example 6 captures a good intuition of the behavior of `dup` when iterated by star. As captured by the semantics of WNKA, a transition can only be taken by consuming a `dup` from the input guarded string. If the `dup` is furthermore part of an expression being iterated with star, then we must allow for the *unbounded* consumption of such `dups`. As our automata are finite, this is achieved by *looping* the transition. Equally, a `dup`-free policy must not consume any `dups` from input guarded strings, and as such the corresponding automaton will not have any effectively traversable transitions. In practice, this means that although our automaton may still be comprised of many states with no transitions, these can be collapsed into a single state automaton, where the state output entirely captures the language for `dup`-free guarded strings.

Table 1. wNetKAT Thompson construction. For expression p , we inductively build $\mathcal{A}_p \triangleq (S_p, \iota_p, \delta^p, \lambda^p)$. We use $+$ to denote coproducts of sets; given a coproduct $X + Y$, $f: X \rightarrow Z$, and $g: Y \rightarrow Z$ the function $[f, g]: X + Y \rightarrow Z$ is the copairing of f and g . We denote by $\iota_{p_1} \boxtimes \lambda^{p_1}$ the square matrix $\alpha\beta \mapsto \iota_{p_1} \times \lambda_{\alpha\beta}^{p_1}$.

p	S_p	$\iota_p: \mathcal{W}_S(S_p)$	$\delta_{\alpha\beta}^p: S_p \rightarrow \mathcal{W}_S(S_p)$	$\lambda_{\alpha\beta}^p: \mathcal{W}_S(S_p)$
$\pi?$	$\{\heartsuit\}$	$\eta(\heartsuit)$	\emptyset	$\heartsuit \mapsto [\alpha = \beta = \pi]$
$\pi!$	$\{\heartsuit\}$	$\eta(\heartsuit)$	\emptyset	$\heartsuit \mapsto [\beta = \pi]$
dup	$\{\heartsuit, \clubsuit\}$	$\eta(\heartsuit)$	$s \mapsto \begin{cases} \eta(\clubsuit) & s = \heartsuit \wedge \alpha = \beta \\ \emptyset & \text{otherwise} \end{cases}$	$s \mapsto \begin{cases} [\alpha = \beta] & s = \clubsuit \\ \emptyset & s = \heartsuit \end{cases}$
$r \odot p_1$	S_{p_1}	$r \cdot \iota_{p_1}$	$\delta_{\alpha\beta}^{p_1}$	$\lambda_{\alpha\beta}^{p_1}$
$p_1 \oplus p_2$	$S_{p_1} + S_{p_2}$	$[\iota_{p_1}, \iota_{p_2}]$	$\left[\left[\delta_{\alpha\beta}^{p_1}, \emptyset \right], \left[\emptyset, \delta_{\alpha\beta}^{p_2} \right] \right]$	$\left[\lambda_{\alpha\beta}^{p_1}, \lambda_{\alpha\beta}^{p_2} \right]$
$p_1 ; p_2$	$S_{p_1} + S_{p_2}$	$[\iota_{p_1}, \emptyset]$	$\left[\left[\delta_{\alpha\beta}^{p_1}, \sum_Y \lambda_{\alpha Y}^{p_1} \times \iota_{p_2} \times \delta_{Y\beta}^{p_2} \right], \left[\emptyset, \delta_{\alpha\beta}^{p_2} \right] \right]$	$\left[\sum_Y \lambda_{\alpha Y}^{p_1} \times \iota_{p_2} \times \lambda_{Y\beta}^{p_2}, \lambda_{\alpha\beta}^{p_2} \right]$
p_1^*	$S_{p_1} + \{\heartsuit\}$	$[\emptyset, \mathbb{1}]$	$\left[\left[\delta'_{\alpha\beta}, \emptyset \right], \left[(\lambda^\heartsuit \times \iota_{p_1} \times \lambda^{p_1})_{\alpha\beta}, \emptyset \right] \right]$	$\left[(\lambda^{p_1} \times \lambda^\heartsuit)_{\alpha\beta}, \lambda_{\alpha\beta}^\heartsuit \right]$
where $\delta' = \delta^{p_1} + \lambda^{p_1} \times \lambda^\heartsuit \times \iota_{p_1} \times \delta^{p_1}$ and $\lambda^\heartsuit = (\iota_{p_1} \boxtimes \lambda^{p_1})^*$				

5.1 From Expressions To Automata

Classically, NetKAT automata are constructed on-the-fly using *Brzowski derivatives*. However, in the presence of weights, Brzowski derivatives are known to generally not yield finite automata [8]. To avoid this, we instead describe a generalized Thompson's construction (Table 1), that is guaranteed to terminate by traversing the syntax of the given expression.

Crucially, this construction is guaranteed to produce an automaton whose language matches the language of its expression. By transitivity, our construction computes precisely the denotational semantics of the expression, by converting the input histories into a guarded string.

Lemma 2 (Soundness of Thompson). *Given a policy $p \in \text{Pol}$: $\llbracket \mathcal{A}_p \rrbracket = G(p)$.*

Corollary 1 (Equivalence of wNetKAT policies and wNetKAT automata). *Given a policy $p \in \text{Pol}$:*

$$\llbracket p \rrbracket (\pi_0 :: \langle \rangle) (\pi_n :: \dots :: \pi_1 :: \langle \rangle) = \llbracket \mathcal{A}_p \rrbracket (\pi_0 \pi_1 \text{ dup } \dots \text{ dup } \pi_n).$$

Like the classic Thompson construction, our construction works by combining simpler base automata to achieve more complex ones. This is achieved by composing the vector and matrix components in adequate ways. For example, for $p_1 \oplus p_2$, the two automata are simply juxtaposed into one, with no interaction between the two, as depicted below.

$$\iota^\oplus = \left(\begin{array}{|c|} \hline \iota^{p_1} \\ \hline \end{array} \quad \begin{array}{|c|} \hline \iota^{p_2} \\ \hline \end{array} \right) \quad \delta_{\alpha\beta}^\oplus = \left(\begin{array}{|c|} \hline \delta_{\alpha\beta}^{p_1} \\ \hline \end{array} \quad \begin{array}{|c|} \hline \emptyset \\ \hline \end{array} \right) \quad \lambda_{\alpha\beta}^\oplus = \left(\begin{array}{|c|} \hline \lambda_{\alpha\beta}^{p_1} \\ \hline \end{array} \quad \begin{array}{|c|} \hline \lambda_{\alpha\beta}^{p_2} \\ \hline \end{array} \right)$$

The ‘‘quadrants’’ of the transition matrices created by copairing allow us to specify the behavior of two classes of transitions: the top left, and bottom right quadrants capture transitioning *inside* the two component automata, while the top right and bottom left quadrants capture transitioning

²The soundness of our syntax and semantics (i.e., Lemmas 1 and 2 and Corollary 1) is additionally mechanized in Lean; the mechanization can be found at <https://github.com/cornell-pl/wnetkat-lean/blob/pldi2026/WeightedNetKAT/Papers/PLDI2026.lean>.

between the two component automata. As expected, the construction for choice does not allow transitioning between automata, and the internal automata transitions remain the same.

If we look at sequencing, however, we see a more interesting case. When sequencing two automata, we want to preserve their independent transitions, while additionally being able to “jump” from the first automaton to the second. Traditionally, this is done by introducing ε -transitions, non-deterministically bridging each state of the first automaton into the start state of the second. WNKAs do not have ε -transitions and we produce these “jumps” by using the packets consumed on exiting a state, to instead transition into the second automaton. Formally we do this by: exiting the first automaton ($\lambda_{\alpha\gamma}^{p_1}$), entering the second automaton (ι^{p_2}), and finally taking a first transition ($\delta_{\gamma\beta}^{p_2}$). This is equivalent to transitioning directly from the first automaton to the second, while accumulating the three weights, i.e. $\alpha\beta \mapsto \sum_Y \lambda_{\alpha\gamma}^{p_1} \times \iota^{p_2} \times \delta_{\gamma\beta}^{p_2}$ (note that $\alpha\beta = \alpha\gamma \diamond \gamma\beta$).

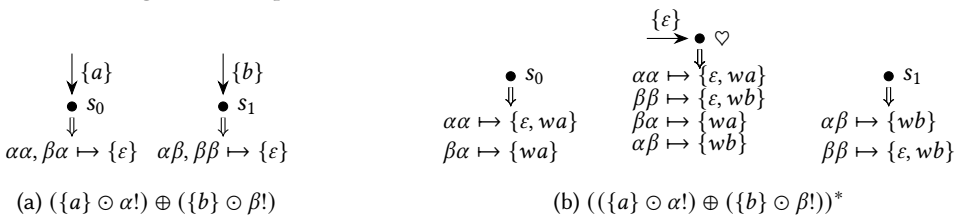
The most interesting case for the Thompson construction, however, is the one of iteration. In the classic Thompson construction, iteration is achieved by introducing a new (accepting) start state (\heartsuit), and whenever a state of the automaton would be accepting, instead ε -transitioning to the new start state, where the string is either accepted or free to begin another iteration. Generalizing this principle requires care, due to the specific semantics of WNKAs. This is best observed in two classes of cases: when iterating over expressions with dup vs. dup-free expressions.

Firstly, as demonstrated in Example 5, a dup-free policy, even if iterated, will only attribute non-zero weight to dup-free guarded string inputs. However, these atomic guarded strings may still be iterated unboundedly, and unobservedly, due to the lack of a dup. For example, $\alpha\xi = \alpha\beta \diamond \beta\gamma \diamond \gamma\xi$, so the weight of $\alpha\xi$ must take into account its “longer”, unobserved equivalents. Processing such a string is akin to entering the start state (ι^{p_1}) and exiting it directly (λ^{p_1})—in this case three times—for each atomic component being sequenced. In general, this must be done for an unbounded number of intermediate atomic guarded strings, meaning the output of the new start state (λ^{\heartsuit}) must be the star of the combined $\iota^{p_1} \times \lambda^{p_1}$ matrix, which naively would require computing an infinite sum.

Definition 4 (Matrix Star). *For a weighting square matrix $M: \mathcal{W}_S(X \times X)$, we define its star as $M^* \triangleq \sum_{n \in \mathbb{N}} M^n$, where $M_{xy}^0 \triangleq [x = y]$ and $M^{n+1} = M^n \times M$. \triangle*

The star of a matrix is, however, a common operation in automata theory, also known as the matrix closure. In the past, Bloom and Ésik [7] developed an algorithm for computing the matrix star in terms of only the underlying semiring operations, together with the *semiring star*, a shorthand defined as $s^* \triangleq \sum_{n \in \mathbb{N}} s^n$. Although computing the star of a semiring element requires computing a countable sum, for all our documented semirings and their combinations, this operation is easily computed, usually in constant time [26]. The Thompson construction depicted in Example 7 below provides a good view of the matrix star in action, as used to compute the output weights of state \heartsuit . For instance, as expected the policy maps the guarded string $\alpha\alpha$ to the infinite regular language $\{w\alpha \mid w \in \Sigma^*\}$, which is computed entirely by λ^{\heartsuit} , the starred output matrix.

Example 7. *For the policy $p \triangleq ((\{a\} \circ \alpha!) \oplus (\{b\} \circ \beta!))^*$ from Example 5, we obtain the following automaton through the Thompson construction:*



Our second key consideration for the case of iteration is best seen when iterating expressions with dup . This is the “classical” setting, where it is possible to non-deterministically return to the start state whenever an end state is reached, ready to process another iteration (as is the case in Example 4). However, wNetKAT automata do not have a single start and end state. In fact, *every* state can be both initial and final, as determined by the initial and output weights. This makes looping the automaton more delicate, and requires an entirely matrix-based treatment, as demonstrated by component δ^p of the star construction: To transition inside the automaton is to either take an internal transition as normal (δ^{p_1}), or non-deterministically (+) exit the state (λ^{p_1}) and, via the new start state (λ^\heartsuit), loop back into the automaton (i^{p_1}), and transition through (δ^{p_1}).

5.2 Computable Semantics of wNetKAT

Finally, the effective computation of the Thompson construction allows us to compute the automaton for any given wNetKAT policy. Using the correctness of the semantics of the automaton (Corollary 1), and the fact that the automaton semantics is computed through matrix multiplication, which is itself computed in terms of semiring addition and multiplication, we obtain the following:

THEOREM 1 (COMPUTABLE SEMANTICS). *Given a computable semiring (\mathcal{S}, \preceq) , the semantics of every \mathcal{S} - wNetKAT policy is computable by compiling it to its corresponding wNetKAT automaton.*

Furthermore, given that the semantics of every wNetKAT policy is computable by compiling to a wNetKAT automaton, we can verify the questions of r -safety and r -reachability through decision procedures that we develop at the level of wNetKAT automata in the following section.

6 Decidability Results for wNetKAT

In this section, we tie the technical results of the previous sections back to the verification questions in Section 2. Consider again verifying whether a network encoded in wNetKAT as p is r -safe or r -reachable. Semantically, these questions correspond to the following two properties:

$$\begin{aligned} \forall \pi \in \text{Pk}, h \in \text{H}: \quad \llbracket p \rrbracket(\pi::\langle \rangle)(h) &\preceq r, && (p \text{ is } r\text{-safe}) \\ \exists \pi \in \text{Pk}, h \in \text{H}: \quad \llbracket p \rrbracket(\pi::\langle \rangle)(h) &\succeq r. && (p \text{ is } r\text{-reachable}) \end{aligned}$$

In words, r -safety says that *all* (out of possibly *infinitely many*) traces produced by the policy p have weight at most r . Dually, r -reachability says that there exists *some* trace with weight at least r .

We generalize techniques by Almagor et al. [2] to obtain *generic* decision procedures for r -safety and r -reachability for a broad class of semirings. These procedures can produce *witnesses*, providing operators with diagnostic information (when r -safety is violated), and synthesize traces satisfying some desired lower bound on a quantity of interest. We illustrate this in Section 7 via a case study.

6.1 Decidability of r -Safety in wNetKAT

We begin by establishing the decidability of r -safety for semirings that model a *worst-case* analysis.

THEOREM 2 (DECIDABILITY OF r -SAFETY). *Let (\mathcal{S}, \preceq) be a computable semiring such that*

$$s_1 + s_2 \preceq s_3 \quad \text{iff} \quad s_1 \preceq s_3 \text{ and } s_2 \preceq s_3,$$

and let p be an \mathcal{S} - wNetKAT policy. Then “ p is r -safe” is decidable. Moreover, if \preceq is total and p is not r -safe, then we can compute a witness, i.e., $\pi \in \text{Pk}$ and $h \in \text{H}$ such that $\llbracket p \rrbracket(\pi::\langle \rangle)(h) \not\preceq r$.

Before we describe the decision procedure, let us gain some intuition on the conditions imposed on \mathcal{S} . Computability is necessary to effectively compute the semantics of the wNetKAT policy. The condition on + expresses that \mathcal{S} models worst-case behavior: the semiring addition “chooses” a

worst-case scenario so that upper-bounding $s_1 + s_2$ is equivalent to upper-bounding both s_1 and s_2 . This condition is satisfied by the Arctic, Probabilistic-union, and Why semirings (cf. Figure 3).

The decision procedure works as follows. First, we invoke Corollary 1, which gives us

$$p \text{ is } r\text{-safe} \quad \text{iff} \quad \forall x \in \text{GS}: \llbracket \mathcal{A}_p \rrbracket(x) \preceq r .$$

It follows from ω -continuity of \mathcal{S} and the side condition on $+$ that the latter is equivalent to

$$\sum_{x \in \text{GS}} \llbracket \mathcal{A}_p \rrbracket(x) \preceq r .$$

We then proceed by showing that this infinite sum over all guarded strings is *computable*, which implies the claim. The key idea is to reduce the computation of this infinite sum to the computation of a matrix star (cf. Definition 4), which can be done via well-established algorithms [7]. In case r -safety is violated, we are—by the totality of \preceq —guaranteed to find a witness by enumerating guarded strings $x \in \text{GS}$ in a breadth-first search manner until we find one with $\llbracket \mathcal{A}_p \rrbracket(x) \not\preceq r$, where we use Theorem 1 to compute the weight \mathcal{A}_p assigns to x . By Corollary 1, x can then be turned into an appropriate witness.

6.2 Decidability of r -Reachability in wNetKAT

Next, we establish the decidability of r -reachability for semirings that model a *best-case* analysis.

THEOREM 3 (DECIDABILITY OF r -REACHABILITY FOR wNETKAT POLICIES). *Let (\mathcal{S}, \preceq) be a computable semiring such that $s_1 \succeq s_1 \cdot s_2$ and*

$$s_1 + s_2 \succeq s_3 \quad \text{iff} \quad s_1 \succeq s_3 \text{ or } s_2 \succeq s_3 ,$$

and let p be a \mathcal{S} -wNetKAT policy. Then “ p is r -reachable” is decidable. Moreover, if p is r -reachable, we can compute a witness, i.e., $\pi \in \text{Pk}$ and $h \in \text{H}$ such that $\llbracket p \rrbracket(\pi::\langle \rangle)(h) \succeq r$.

Let us again gain some intuition on the imposed conditions. Dually to r -safety, the condition on the semiring addition expresses that \mathcal{S} models best-case behavior: lower-bounding $s_1 + s_2$ is equivalent to lower-bounding one of s_1, s_2 . The condition on the semiring multiplication expresses that making traces longer can only make things worse since $s_1 \cdot s_2$ will always be smaller than s_1 . These conditions are satisfied by the Tropical, Viterbi, and Bottleneck semirings (cf. Figure 3).

Our decision procedure works as follows. First, we invoke Corollary 1 to get

$$p \text{ is } r\text{-reachable} \quad \text{iff} \quad \exists x \in \text{GS}: \llbracket \mathcal{A}_p \rrbracket(x) \succeq r .$$

We then exploit the conditions on $+$ and \cdot to conclude that the above is equivalent to the existence of a guarded string $x \in \text{GS}$ corresponding to a *cycle-free run* of \mathcal{A}_p —for a notion of runs we define over the underlying graph structure of \mathcal{A}_p . There are only finitely many cycle-free runs, so it suffices to check $\llbracket \mathcal{A}_p \rrbracket(x) \succeq r$ for only finitely many x to decide r -reachability. If we find such a guarded string x , we use Corollary 1 to turn it into a witness.

7 Case Studies

In this section, we demonstrate how our marriage of classic NetKAT’s modeling capabilities and weighted reasoning enables the automatic quantitative analysis of intricate network configurations. For that, we use a topology based on Internet2’s Abilene backbone network (see Figure 10)³, which features *nodes* across several cities in the United States. Traffic can enter or exit the network from any node (e.g., a network packet entering at BAY destined for NYC), and every node is able to forward packets to nodes it is linked to (e.g., KAN can forward packets to DEN, HOU, and IND). We have additionally annotated the network topology in Figure 10 with several quantities, such as

³The TikZ code used in Figure 10 was produced with the help of a generative AI software tool (Claude, Sonnet 4.6).

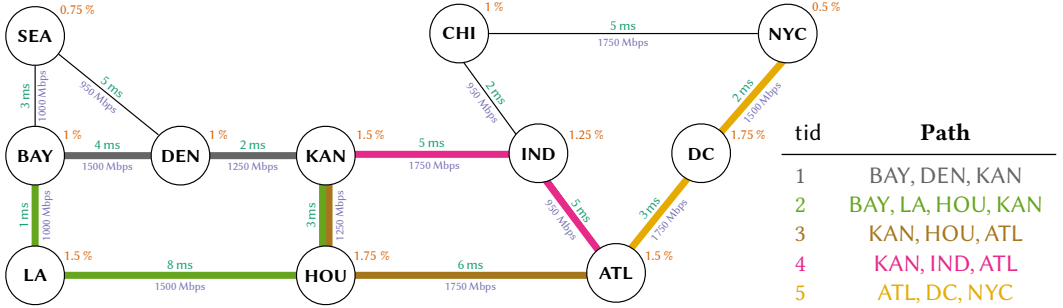


Fig. 10. Topology from Abilene network with links between nodes weighted by *latency/bandwidth* and nodes by forwarding *failure rate*. Network tunnels are mapped to their corresponding tunnel ID (tid) in table.

the associated *failure rates* of each node or the *bandwidth* of each link between nodes (e.g., based on forwarding failure metrics or historical average bandwidths).

Suppose that for certain source-destination pairs it is preferable to forward traffic through *tunnels* instead of with the usual forwarding behavior (e.g., based on *shortest-paths* or some other routing scheme). Figure 10 highlights example tunnels available in the network to be used by specific nodes for NYC-bound traffic. For example, KAN is configured to use either of the tunnels KAN → HOU → ATL (tid 3) or KAN → IND → ATL (tid 4). Both tunnels exit into ATL, which in turn is configured to forward NYC-bound traffic through tunnel 5. In what follows, we first model this tunneling behavior in classic NetKAT and then demonstrate how wNetKAT enables quantitative reasoning. We model the network’s forwarding behavior using choice and *nested iteration*:

$$\begin{aligned}
 & \text{if tid} = 0 \text{ then} \\
 & \quad \text{if dst} = \text{NYC} \text{ then } (\text{tid} \leftarrow 3 \oplus \text{tid} \leftarrow 4) \\
 & \quad \text{else } \dots \quad (* \text{Default behavior} *) \\
 p_{\text{KAN}} & \triangleq \text{else if tid} = 1 \vee \text{tid} = 2 \text{ then tid} \leftarrow 0 \\
 & \quad \text{else if tid} = 3 \text{ then node} \leftarrow \text{HOU} \\
 & \quad \text{else if tid} = 4 \text{ then node} \leftarrow \text{IND} \\
 & \quad \text{else drop} \\
 & \quad p \triangleq \text{if node} = \text{ATL} \text{ then} \\
 & \quad \quad (p_{\text{ATL}})^*; \text{node} \neq \text{ATL} \\
 & \quad \text{else if node} = \text{BAY} \text{ then} \\
 & \quad \quad (p_{\text{BAY}})^*; \text{node} \neq \text{BAY} \\
 & \quad \text{else } \dots \\
 \text{abilene} & \triangleq (p; \text{dup})^*
 \end{aligned}$$

Similarly to Section 2.1, *abilene* models the process of (i) forwarding a packet according to its current node (modeled by p), (ii) recording the packet’s state in the history (using dup), and (iii) repeating this process (using iteration). Policy p branches on the packet’s current node and invokes the corresponding routing policy. Notice that these routing policies are *also iterated*—an idea dating back to Gupta et al. [14]. The reason for that becomes apparent when considering p_{KAN} : we use the field tid to keep track of the tunnel the packet is currently in ($\text{tid} = 0$ meaning no tunnel). KAN thus marks the end of tunnels 1 and 2, and the start of tunnels 3 and 4. In particular, if KAN receives a packet with $\text{tid} = 0$ and destined for NYC, this packet may be forwarded *either* via tunnel 3 *or* 4 (using choice). Now, if KAN receives a packet destined for NYC on tunnel 1, say, then we have to execute p_{KAN} *twice* to ensure that, subsequently, it is correctly forwarded via tunnel 3 or 4. Iterating p_{KAN} naturally captures the necessity of possibly having to execute the policy multiple times.

Consider BAY, which is configured analogously to KAN (i.e., NYC-bound traffic is forwarded via tunnels 1 or 2). The full encoding of BAY (and for all other examples throughout this section) is included in the appendix [34]. By instantiating wNetKAT with the Boolean semiring we can, as with NetKAT, already verify that the tunneled paths between BAY and NYC are configured so that the two nodes are connected. This amounts to checking that the following policy is 1-reachable:

$$(\text{node} = \text{BAY} \wedge \text{dst} = \text{NYC}); \text{abilene}; (\text{node} = \text{NYC} \wedge \text{tid} \neq 0)$$

7.1 Verifying Reliability of Tunneled Paths with Theorem 2

Let us now illustrate how wNetKAT can serve as a *verification tool* for bounds on the *reliability* of a network's configuration. Consider again network traffic entering at BAY destined for NYC. Beyond checking that BAY and NYC are indeed connected, a network provider may additionally wish to ensure that all tunneled paths between BAY and NYC are sufficiently reliable. This corresponds to upper-bounding the *worst-case failures across all combinations of tunnels a packet might take*. We take the encoding from the previous section and *weight* each node's (iterated) routing policy by its forwarding **failure rate**, instantiating wNetKAT with the Probabilistic-union semiring (cf. Figure 3):

$$p_{\text{rel}} \triangleq \begin{array}{l} \text{if node = ATL then} \\ \quad 1.5\% \odot (p_{\text{ATL}})^* ; \text{node} \neq \text{ATL} \\ \text{else ...} \end{array} \quad \text{abilene}_{\text{rel}} \triangleq (p_{\text{rel}} ; \text{dup})^*$$

Now assume we wish to check that all tunneled paths between BAY and NYC have a failure rate of at most 10%. We use Theorem 2 to decide whether the following is 0.1-safe:

$$(\text{node} = \text{BAY} \wedge \text{dst} = \text{NYC}) ; \text{abilene}_{\text{rel}} ; (\text{node} = \text{NYC} \wedge \text{tid} \neq 0)$$

The decision procedure from Theorem 2 answers negatively and provides a witness: KAN is configured to always forward NYC-bound traffic through tunnel 3 (KAN \rightarrow HOU \rightarrow ATL). However, when a packet at KAN has just exited tunnel 2 (BAY \rightarrow LA \rightarrow HOU \rightarrow KAN), KAN will unnecessarily reroute traffic through HOU and incur more probability of failure. As a result, the tunneled path 2 \rightarrow 3 \rightarrow 5 between BAY and NYC will have a failure rate of 10.3%. We remark that this is no longer a straightforward cycle detection as discussed in Section 2. wNetKAT enables the automatic identification of a specific combination of network tunnels that cause reliability issues.

A network provider can now use the generated witness to reconfigure KAN to only forward packets destined for NYC through tunnel 3 if they have not just exited tunnel 2:

$$p_{\text{KAN, safe}} \triangleq \begin{array}{l} \text{if tid} = 0 \text{ then ...} \\ \text{else if tid} = 2 \wedge \text{dst} = \text{NYC} \text{ then tid} \leftarrow 4 \quad (* \text{ Forward directly } *) \\ \text{else if tid} = 1 \vee \text{tid} = 2 \text{ then tid} \leftarrow 0 \\ \text{else ...} \end{array}$$

In particular, packets exiting tunnel 2 which are destined for NYC are now directly forwarded through tunnel 4. We can once again use the decision procedure from Theorem 2, which this time answers positively: all tunneled paths guarantee a failure rate of at most 10%.

7.2 Finding High-Bandwidth Tunneled Paths with Theorem 3

We now demonstrate how wNetKAT can aid network providers as a *design tool* when configuring the routing behavior: With wNetKAT, we can synthesize paths within a network satisfying specific quantitative properties; after which the network can be reconfigured appropriately.

Consider the following scenario: although there are several tunneled paths from BAY to NYC, we wish to refine the network to use high-bandwidth tunnels specifically for *video traffic*, which should be delivered with at least 1000Mbps of **bandwidth**. We modify the encoding of the network from the previous section (for which it is already guaranteed that all tunneled paths from BAY to NYC have a failure rate of at most 10%) by weighting the *forwarding actions* in each tunnel by the corresponding link's **bandwidth**, instantiating wNetKAT with the Bottleneck semiring (cf. Figure 3):

$$p_{\text{KAN, band}} \triangleq \begin{array}{l} \text{if ...} \\ \text{else if tid} = 3 \text{ then } 1250 \text{ Mbps} \odot \text{node} \leftarrow \text{HOU} \\ \text{else if tid} = 4 \text{ then } 1750 \text{ Mbps} \odot \text{node} \leftarrow \text{IND} \\ \text{else drop} \end{array} \quad \begin{array}{l} p_{\text{band}} \triangleq \dots \\ \text{abilene}_{\text{band}} \triangleq (p_{\text{band}} ; \text{dup})^* \end{array}$$

If it exists, we can now synthesize a tunneled path between BAY and NYC with a bandwidth of at least 1000Mbps. By Theorem 3, this can be decided by checking the 1000-reachability of

$$(\text{node} = \text{BAY} \wedge \text{dst} = \text{NYC}) ; \text{abilene}_{\text{band}} ; (\text{node} = \text{NYC} \wedge \text{tid} \neq 0) .$$

The decision procedure from Theorem 3 answers positively and provides the sought-after tunneled path: $1 \rightarrow 3 \rightarrow 5$ has a bandwidth of 1250Mbps. We now reconfigure the network to always choose this tunneled path for video traffic (identified via the field `vid`) destined for NYC, e.g.,

$$p_{\text{KAN,vid}} \triangleq \begin{array}{l} \text{if tid} = 0 \text{ then} \\ \quad \text{if dst} = \text{NYC} \text{ then (if vid} = \text{TRUE then tid} \leftarrow 3 \text{ else (tid} \leftarrow 3 \oplus \text{tid} \leftarrow 4)) \\ \quad \text{else ...} \\ \text{else ...} \end{array}$$

In particular, video traffic (`vid = TRUE`) destined for NYC is now always tunneled through tunnel 3. BAY would similarly be reconfigured to always tunnel NYC-bound video traffic through tunnel 1. The forwarding behavior for regular traffic remains unchanged from our previous example.

In summary, we first used wNetKAT to verify that the tunneled paths configured in the Abilene network provide a reliability of at least 90%; then we used wNetKAT to refine our network configuration to use high-bandwidth tunneled paths for video traffic. We focus on reliability and bandwidth in these examples, but our framework (and decision procedures) remains parametric on a semiring and can verify several network phenomena as shown in Figure 3. For example, we can similarly verify that for the final network configuration above, all *video traffic* in these tunneled paths is *additionally* delivered within 20ms. Instantiating wNetKAT with the Arctic semiring and weighting by *latency*, this would correspond to checking that the following policy is 20-safe:

$$(\text{node} = \text{BAY} \wedge \text{dst} = \text{NYC} \wedge \text{vid} = \text{TRUE}) ; \text{abilene}_{\text{vid}} ; (\text{node} = \text{NYC} \wedge \text{tid} \neq 0)$$

As before, this property is decidable by Theorem 2; the decision procedure answers in the positive.

8 Related Work

wNetKAT is an extension of NetKAT [3], which is itself an extension of KAT [18] to reason about network behavior. wNetKAT is inspired in large part by ProbNetKAT [10, 31], which in turn extends NetKAT to model probabilistic network behavior. wNetKAT generalizes this idea to support modeling several different quantitative behaviors based on the choice of semiring. Nevertheless, wNetKAT remains a conservative extension by subsuming both NetKAT and the guarded fragment of ProbNetKAT. Although wNetKAT subsumes only a fragment of ProbNetKAT, an even smaller fragment (guarded and dup-free) has previously been studied in a practical setting [32]. Singh [29] and Larsen et al. [21] both introduce extensions to NetKAT parametric on semirings to model latency and other quantitative properties. However, neither extension provides a sound translation to weighted automata nor decision procedures for verifying quantitative network properties. Both extensions are more expressive at the syntax-level than wNetKAT (e.g. including quantitative tests), our extension instead is intentionally chosen so that the syntax is expressive enough to model interesting network behavior while still being able to use techniques based on weighted automata.

More generally, several frameworks have been proposed for reasoning about programming systems that are parametric on semirings (e.g., see [5, 9, 12]). In particular, Batz et al. [5] propose *weighted programming* as a paradigm for specifying mathematical models beyond probability distributions. wNetKAT follows a similar approach (and is likewise parametric on ω -continuous semirings), though our focus is specifically on extending the power of NetKAT to reasoning about the quantitative behavior of networks. Many of our example semirings, however, are based on their applications in weighted programming. wGKAT [35] and KAWT [28] are both extensions to

KAT that likewise follow a similar approach to the work by Batz et al. [5]. Van Koevering et al. [35] extend Guarded Kleene Algebra with Tests (GKAT) [30] to the weighted setting and show the decidability of equivalence for weighted automata up to bisimilarity. Unfortunately, we cannot apply their results as wNetKAT features unguarded iteration; we refer the reader to the work of Wasserstein [36] for the incompatibility of NetKAT and GKAT. Sedlár [27] shows a completeness result for a more general version of KAWT (Kleene Algebra with Weights and Tests). wNetKAT is most similar to KAWT (albeit in the setting of NetKAT which introduces further subtleties as we discuss throughout the paper). However, KAWT does not provide a general language model and computable operational semantics, limiting this only to finite semirings.

Decision procedures for weighted automata have been studied extensively in the literature [19, 20]. Our decision procedures for r -safety and r -reachability for wNetKAT automata adapt results by Almagor et al. [2] for the Tropical semiring over the natural numbers. Recent work by Moeller et al. [25] contributed techniques for efficient implementations of decision procedures over *NetKAT automata* (which wNetKAT automata subsume). In particular, they develop *symbolic* versions of NetKAT automata that do not explicitly enumerate their packet space and provide an ecosystem of supporting algorithms. Their approach is, however, highly tailored to deciding *equivalence*, a property that is (i) undecidable in general for weighted automata (ii) often too strong when it comes to verifying quantitative network properties. Applying these techniques in our setting is fundamentally different as it would require not only the development of a novel, symbolic representation of our wNetKAT automata but also direct symbolic decision procedures for r -safety and r -reachability rather than equivalence.

Finally, semirings have been used as the foundation of other frameworks in the networking domain, including network calculus [22] and routing algebras [13, 33]. Network calculus is a mathematical framework designed for modeling and reasoning about quantitative properties. It provides primitives for modeling the arrival, buffering, and departure of traffic in a deterministic queueing system and also models interactions between multiple flows. Unlike wNetKAT, the focus is more on pencil-and-paper proofs of performance bounds rather than automated verification of safety and reachability properties. Routing algebras model the behavior of distributed control-plane protocols like OSPF and BGP, whereas wNetKAT focuses on behavior at the data-plane level. Deepening the connections between these frameworks is an excellent direction for future work.

9 Conclusion

We introduced wNetKAT, a framework for quantitative network verification. We developed a denotational semantics of wNetKAT and an equivalent language model. We then presented an operational semantics based on wNetKAT automata to compute the exact semantics of wNetKAT. This enabled the design of decision procedures for reasoning about r -safety and r -reachability in networks. We then used the framework to reason about worst-/best-case network guarantees over a range of practical network phenomena in the setting of Abilene backbone network.

As future work, we would like to implement practical versions of these decision procedures over efficient representations of wNetKAT automata (e.g., as in [24, 25]). Separately, network verification with NetKAT requires having accurate models of such systems; which can be tedious and error prone. This is only more true in the weighted setting, and so we would like to explore learning for wNetKAT automata [24]. Finally, we would be interested in extensions to wNetKAT that make the language more expressive. In particular, we would like to consider variants with quantitative tests (e.g., as in [21, 29]), which would allow expressing network behavior dependent on quantities (e.g., load balancing). These extensions would however complicate our denotational and operational model, and importantly, would not allow us to reduce our properties to wNetKAT automata. We leave them as a possible direction for future work.

Acknowledgments

We are grateful to our PLDI reviewers and shepherd who helped us improve our paper significantly. We also thank Thomas Lu for helpful discussions on our case study, as well as the Cornell PLDG and UCL PPLV group for their feedback on early drafts. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001125CE018 (Approved for public release; distribution is unlimited.). Additionally, this work was supported by ERC grant Autoprobe (no. 101002697), NSF grant DGE-2139899, DFF project AuRoRA, and a Royal Society Wolfson fellowship, as well as a gift from the VMware University Research Fund. Finally, a generative AI software tool (Claude, Sonnet 4.6) was used for finding typographical errors in our paper and for help producing the TikZ code used in one figure.

References

- [1] Kinan Dak Albab, Jonathan DiLorenzo, Stefan Heule, Ali Kheradmand, Steffen Smolka, Konstantin Weitz, Muhammad Timarzi, Jiaqi Gao, and Minlan Yu. 2022. SwitchV: Automated SDN switch validation with P4 models. In *Proceedings of the ACM SIGCOMM Conference*. 365–379. <https://doi.org/10.1145/3544216.3544220>
- [2] Shaull Almagor, Udi Boker, and Orna Kupferman. 2022. What’s decidable about weighted automata? *Information and Computation* 282 (2022), 104651. <https://doi.org/10.1016/j.ic.2020.104651> Special issue on 9th International Workshop Weighted Automata: Theory and Applications (WATA 2018).
- [3] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. 2014. NetKAT: semantic foundations for networks. *SIGPLAN Not.* 49, 1 (Jan. 2014), 113–126. <https://doi.org/10.1145/2578855.2535862>
- [4] The NetKAT authors. 2025. NetKAT. <https://github.com/google/netkat>.
- [5] Kevin Batz, Adrian Gallus, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Tobias Winkler. 2022. Weighted programming: a programming paradigm for specifying mathematical models. *Proc. ACM Program. Lang.* 6, OOPSLA1, Article 66 (April 2022), 30 pages. <https://doi.org/10.1145/3527310>
- [6] Jean Berstel and Christophe Reutenauer. 2010. *Noncommutative Rational Series with Applications*. Encyclopedia of Mathematics and its Applications, Vol. 137. Cambridge University Press, Cambridge, UK.
- [7] Stephen L. Bloom and Zoltán Ésik. 1993. *Matrix Iteration Theories*. Springer Berlin Heidelberg, Berlin, Heidelberg, 289–351. https://doi.org/10.1007/978-3-642-78034-9_10
- [8] Filippo Bonchi, Marcello M. Bonsangue, Helle H. Hansen, Prakash Panangaden, Jan J. M. M. Rutten, and Alexandra Silva. 2014. Algebra-coalgebra duality in brzozowski’s minimization algorithm. *ACM Trans. Comput. Logic* 15, 1, Article 3 (March 2014), 29 pages. <https://doi.org/10.1145/2490818>
- [9] Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. 2014. A Core Quantitative Coeffect Calculus. In *Programming Languages and Systems*, Zhong Shao (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 351–370.
- [10] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. 2016. Probabilistic NetKAT. In *Proceedings of the 25th European Symposium on Programming Languages and Systems - Volume 9632*. Springer-Verlag, Berlin, Heidelberg, 282–309. https://doi.org/10.1007/978-3-662-49498-1_12
- [11] Nate Foster, Dexter Kozen, Mae Milano, Alexandra Silva, and Laure Thompson. 2015. A Coalgebraic Decision Procedure for NetKAT. *SIGPLAN Not.* 50, 1 (Jan. 2015), 343–355. <https://doi.org/10.1145/2775051.2677011>
- [12] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (Beijing, China) (PODS '07)*. Association for Computing Machinery, New York, NY, USA, 31–40. <https://doi.org/10.1145/1265530.1265535>
- [13] Timothy G. Griffin and João Luís Sobrinho. 2005. Metarouting. In *Proceedings of the ACM SIGCOMM Conference*. 1–12. <https://doi.org/10.1145/1080091.1080094>
- [14] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P. Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. 2014. SDX: a software defined internet exchange (SIGCOMM ’14). Association for Computing Machinery, New York, NY, USA, 551–562. <https://doi.org/10.1145/2619239.2626300>
- [15] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 3–14. <https://doi.org/10.1145/2534169.2486019>
- [16] Karthick Jayaraman, Nikolaj Bjørner, Jitu Padhye, Amar Agrawal, Ashish Bhargava, Paul-Andre C Bissonnette, Shane Foster, Andrew Helwer, Mark Kasten, Ivan Lee, Anup Namdhari, Haseeb Niaz, Aniruddha Parkhi, Hanukumar Pinnamraju, Adrian Power, Neha Milind Raje, and Parag Sharma. 2019. Validating datacenters at scale. In *Proceedings*

- of the ACM SIGCOMM Conference. 200–213. <https://doi.org/10.1145/3341302.3342094>
- [17] Donald M. Kaplan. 1969. Regular Expressions and the Equivalence of Programs. *J. Comput. Syst. Sci.* 3, 4 (1969), 361–386. [https://doi.org/10.1016/S0022-0000\(69\)80027-9](https://doi.org/10.1016/S0022-0000(69)80027-9)
- [18] Dexter Kozen. 1997. Kleene Algebra with Tests. *ACM Trans. Program. Lang. Syst.* 19, 3 (1997), 427–443. <https://doi.org/10.1145/256167.256195>
- [19] Daniel Kroh. 1992. The Equality Problem for Rational Series with Multiplicities in the Tropical Semiring is Undecidable. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*. Springer-Verlag, Berlin, Heidelberg, 101–112.
- [20] Daniel Kroh. 1994. Some consequences of a Fatou property of the tropical semiring. *Journal of Pure and Applied Algebra* 93, 3 (1994), 231–249. [https://doi.org/10.1016/0022-4049\(94\)90090-6](https://doi.org/10.1016/0022-4049(94)90090-6)
- [21] Kim G. Larsen, Stefan Schmid, and Bingtian Xue. 2016. WNetKAT: A Weighted SDN Programming and Verification Language. arXiv:1608.08483 [cs.NI] <https://arxiv.org/abs/1608.08483>
- [22] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network calculus: a theory of deterministic queuing systems for the Internet*. Springer-Verlag, Berlin, Heidelberg.
- [23] Mark Moeller. [n. d.]. Galois Internship Round 2: A Second Summer Intern Experience. <https://web.archive.org/web/20251010101506/https://www.galois.com/articles/galois-internship-round-2-a-second-summer-intern-experience>. Accessed: 2025-11-13.
- [24] Mark Moeller, Tiago Ferreira, Thomas Lu, Nate Foster, and Alexandra Silva. 2025. Active Learning of Symbolic NetKAT Automata. *Proc. ACM Program. Lang.* 9, PLDI, Article 192 (June 2025), 24 pages. <https://doi.org/10.1145/3729295>
- [25] Mark Moeller, Jules Jacobs, Olivier Savary Bélanger, David Darais, Cole Schlesinger, Steffen Smolka, Nate Foster, and Alexandra Silva. 2024. KATch: A Fast Symbolic Verifier for NetKAT. *Proc. ACM Program. Lang.* 8, PLDI (2024), 1905–1928. <https://doi.org/10.1145/3656454>
- [26] Mehryar Mohri. 2009. *Weighted Automata Algorithms*. Springer Berlin Heidelberg, Berlin, Heidelberg, 213–254.
- [27] Igor Sedlár. 2024. Completeness of Finitely Weighted Kleene Algebra with Tests. In *Logic, Language, Information, and Computation*, George Metcalfe, Thomas Studer, and Ruy de Queiroz (Eds.). Springer Nature Switzerland, Cham, 210–224.
- [28] Igor Sedlár. 2023. Kleene Algebra With Tests for Weighted Programs. In *2023 IEEE 53rd International Symposium on Multiple-Valued Logic (ISMVL)*. 111–116. <https://doi.org/10.1109/ISMVL57333.2023.00031>
- [29] Avaljot Singh. 2021. *Cost InterNetKAT: Basics of Algebraic Network Routing*. Ph. D. Dissertation. INDIAN INSTITUTE OF TECHNOLOGY DELHI.
- [30] Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. 2019. Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time. *Proc. ACM Program. Lang.* 4, POPL, Article 61 (Dec. 2019), 28 pages. <https://doi.org/10.1145/3371129>
- [31] Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. 2017. Cantor meets Scott: semantic foundations for probabilistic networks. *SIGPLAN Not.* 52, 1 (Jan. 2017), 557–571. <https://doi.org/10.1145/3093333.3009843>
- [32] Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. 2019. Scalable verification of probabilistic networks. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) (PLDI 2019). Association for Computing Machinery, New York, NY, USA, 190–203. <https://doi.org/10.1145/3314221.3314639>
- [33] João Luís Sobrinho. 2005. An algebraic theory of dynamic network routing. *IEEE/ACM Transactions on Networking (ToN)* 13, 5 (Oct. 2005), 1160–1173. <https://doi.org/10.1109/TNET.2005.857111>
- [34] Emmanuel Suárez Acevedo, Tiago Ferreira, Kevin Batz, Oliver Bøving, Nate Foster, and Alexandra Silva. 2026. Weighted NetKAT: A Programming Language For Quantitative Network Verification. arXiv:2604.13987 [cs.PL] <https://arxiv.org/abs/2604.13987>
- [35] Spencer Van Koeveering, Wojciech Różowski, and Alexandra Silva. 2025. Weighted GKAT: Completeness and Complexity. In *52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 334)*, Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 172:1–172:18. <https://doi.org/10.4230/LIPIcs.ICALP.2025.172>
- [36] Jacob Wasserstein. 2023. Guarded NetKAT: Soundness, Partial-Completeness, Decidability. (May 2023). <https://doi.org/10.7298/Y5X5-JR17> Publisher: Cornell University Library.

Received 2025-11-14; accepted 2026-04-03