Undecidable Problems for Probabilistic Network Programming

David M. Kahn

Cornell University, Department of Computer Science, Ithaca NY, USA
`dmk254@cornell.edu`

Abstract

The software-defined networking language NetKAT is able to verify many useful properties of networks automatically via a PSPACE decision procedure for program equality. However, for its probabilistic extension ProbNetKAT, no such decision procedure is known. We show that several potentially useful properties of ProbNetKAT are in fact undecidable, including emptiness of support intersection and certain kinds of distribution bounds and program comparisons. We do so by embedding the Post Correspondence Problem in ProbNetKAT via direct product expressions, and by directly embedding probabilistic finite automata.

1998 ACM Subject Classification D.2.4 Software/Program Verification, D.3.1 Formal Definitions and Theory, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.3 Formal Languages

Keywords and phrases Software-defined networking, NetKAT, ProbNetKAT, Undecidability, Probabilistic finite automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.68

1 Introduction

The NetKAT family of programming languages aims to simplify network programming and its verification [1, 10]. The NetKAT languages do this is by centralizing network control in a scheme known as software-defined networking, and by exploiting the algebraic structure of Kleene algebra with tests. This leaves the language NetKAT sufficiently well-behaved so as to admit a relatively fast (PSPACE-complete) decision procedure for program equivalence [5]. Several important networking problems, such as waypointing and reachability, are reducible to program equivalence in NetKAT, so this decision procedure is quite useful for automated verification of NetKAT programs.

NetKAT, however, has limitations. NetKAT is deterministic and cannot express probabilistic concepts that often arise in networking, such as randomized routing algorithms and connection failure chance [4]. The language ProbNetKAT answers this by conservatively extending NetKAT with an operator for probabilistic choice. However, the addition of this new operator further distances the language from the well-behavedness of a pure Kleene algebra with tests. This opens up many new questions concerning decidability and deductive completeness. If the benefits of the NetKAT framework are to be applied to probabilistic networking, it is important to now determine what can or cannot be decided about ProbNetKAT. As of the time of this paper, few ProbNetKAT decidability results have been put forward.

This paper introduces several properties of ProbNetKAT programs that are not decidable, alongside some contexts in which they might arise. These undecideable properties include emptiness of support intersection, support size, and certain kinds of program distribution bounds and comparisons. We prove these problems undecidable through two different embeddings. First, the Post Correspondence Problem is embedded in ProbNetKAT via direct product expressions, which are expressions in the direct product of regular expressions. And second, probabilistic finite automata are then directly embedded in ProbNetKAT, and their associated undecidable problems translated over.

2 Definition of ProbNetKAT

To understand ProbNetKAT, it is important to understand how NetKAT treats its *packets*, the information that is passed around a network. A given packet $\pi \in Pk$ is a collection of fields of information, such that $\pi(x)$ can be considered to be the value of field x. Each of these fields may be assigned value using the $\leftarrow$ operator, or *tested* using the $=$ operator. These tests act as filters, dropping all packets that do not satisfy them. Any such test can also be complemented using the $\neg$ operator.

For bookkeeping purposes, these packets are understood to be arranged in *histories*, $\pi_1 :: \pi_2 :: ... :: \pi_n$, which are sequences of snapshots of a given packet over time, from youngest to oldest. (Such a history is often shortened notationally to $\pi_1 :: \sigma$.) The operation which takes this snapshot is called *dup*, because it *duplicates* the latest packet, the head packet π_1, and appends it to the front of the sequence. Only the head packet ever actually exists in the network in execution, but tracking the history that a given program would generate for a packet allows for answering important questions about that packet, like where it has been.

In addition to these basic operations of test and assignment, NetKAT also employs the operators and constants of Kleene algebra. Kleene algebra's addition is given with & (parallel composition), its multiplication is given with ; (sequential composition), its asterate is given with * (iteration), its annihilator element 0 is given with *drop* (drops all packets), and its identity element 1 is given with *skip* (does nothing). Together, these form a Kleene algebra on the generating set of assignments and tests. This means, for instance, that & is commutative. Interestingly, in most contexts, we interpret Kleene algebras as treating & disjunctively, but in NetKAT, & is conjunctive. And, paired with the negation operator ($\neg$), these also can form a Boolean algebra on only tests ($=$). For a Boolean algebra, *drop* act as 0, *skip* acts as 1, & acts as disjunction, and ; acts as conjunction. This means, for instance, that both ; and & are commutative on the tests.

This Kleene algebra structure paired with Boolean algebra comprises the Kleene algebra with tests, or KAT, that NetKAT is named after. This algebra is sufficient to characterize a lot of programmatic structure, allowing for manipulation of such structure into different semantically equivalent forms. For instance, letting the term b below be such a Boolean term, and letting p and q be general expressions, we can represent the programming idioms of the while loop and if-then-else clause as below. The symbols used will be defined formally shortly.

$$\text{if } b \text{ then } p \text{ else } q = b; p \& \neg b; q \qquad \text{while } b \text{ do } p = (b; p)^*; \neg b$$

The relevant syntax of NetKAT can therefore be given just by KAT expressions of the above operators on an alphabet of tests, assignments, and *dup*s, where tests are the Boolean elements.

Before we provide the formal definition of the above pieces of NetKAT, it will be useful to mention the Haskell language's monad operators, *return* (η) and *bind* ($>\!\!>\!\!=$). Letting $T(A)$ be a type predicated on the type A (like a *list* of elements from A), the monad operators are of the types $>\!\!>\!\!=: T(A) \to (A \to T(B)) \to T(B)$ and $\eta : A \to T(A)$ for some types A and B.

These operators allow the functional language of Haskell to translate functions on A into $T(A)$ in a useful way, called *lifting*. Bind and return satisfy the following axioms, corresponding to something similar to left identity, right identity, and associativity, respectively.

$$\eta(x) \gg= f = f(x) \quad t \gg= \eta = t \quad t \gg= (\lambda x. f(x) \gg= g) = (t \gg= f) \gg= g$$

Formally, a NetKAT program p or q is usually interpreted as a function that maps a history in $\pi :: \sigma \in H$ to a set of histories $a \in 2^H$, with the semantics defined as follows [12]. These definitions will be presented alongside some monadic counterparts to better clarify their structure and emphasize the ability to lift the operators to act on sets of histories. The relevant monad for these operations is the powerset monad.

$$\eta(\pi :: \sigma) = \{\pi :: \sigma\} \quad a \gg= [\![p]\!] = \bigcup_{h \in a} [\![p]\!](h)$$

$$\begin{array}{lclcl}
[\![skip]\!](\pi :: \sigma) & = & \{\pi :: \sigma\} & = & \eta(\pi :: \sigma) \\
[\![drop]\!](\pi :: \sigma) & = & \emptyset & & \\
[\![x \leftarrow n]\!](\pi :: \sigma) & = & \{\pi[n/x] :: \sigma\} & = & \eta(\pi[n/x] :: \sigma) \\
[\![x = n]\!](\pi :: \sigma) & = & \begin{cases} \{\pi :: \sigma\} & \pi(x) = n \\ \emptyset & otherwise \end{cases} & = & \begin{cases} \eta(\pi :: \sigma) & \pi(x) = n \\ \emptyset & otherwise \end{cases} \\
[\![dup]\!](\pi :: \sigma) & = & \{\pi :: \pi :: \sigma\} & = & \eta(\pi :: \pi :: \sigma) \\
[\![\neg b]\!](\pi :: \sigma) & = & \{\pi :: \sigma\} - [\![b]\!](\pi :: \sigma) & = & \begin{cases} \emptyset & [\![b]\!](\pi :: \sigma) = \eta(\pi :: \sigma) \\ \eta(\pi :: \sigma) & otherwise \end{cases} \\
[\![p \& q]\!](\pi :: \sigma) & = & [\![p]\!](\pi :: \sigma) \cup [\![q]\!](\pi :: \sigma) & = & [\![p]\!](\pi :: \sigma) \\
 & & & & \gg= \lambda s.([\![q]\!](\pi :: \sigma) \gg= \lambda t.\eta(s) \cup \eta(t))) \\
[\![p; q]\!](\pi :: \sigma) & = & \bigcup_{h \in [\![p]\!](\pi :: \sigma)} [\![q]\!](h) & = & [\![p]\!](\pi :: \sigma) \gg= [\![q]\!] \\
[\![p^*]\!](\pi :: \sigma) & = & [\![skip \& p^*; p]\!](\pi :: \sigma) & = & [\![skip \& p; p^*]\!](\pi :: \sigma) \\
 & = & \bigcup_{n \in \mathbb{N}} [\![p^n]\!](\pi :: \sigma) & &
\end{array}$$

To get ProbNetKAT, we start by lifting these operators with bind as $\lambda a.a \gg= [\![p]\!]$ to get a new set of operators that allow us to interpret NetKAT programs as functions from *sets* of histories to sets of histories. Then we add in the probabilistic choice operator $\oplus$, which, with a given probability r, chooses to continue to the code on its left, and otherwise the code on its right. Syntactically it may be used with the same generality as & or ;, such that all take two ProbNetKAT programs and return one. The new operator's randomness introduces distributions on the output, so a ProbNetKAT program can be interpreted as a function that maps a set of histories $a \in 2^H$ to a distribution on sets of histories μ. The semantics for the this interpretation is defined below, where many of the definitions show the same structure as in NetKAT. These semantics use the Dirac delta function δ for probability measure (which yields a distribution that puts all weight on a single point) and the product measure $\times$. The relevant monad here is the Giry monad.

$$\delta_a(A) = \begin{cases} 1 & a \in A \\ 0 & else \end{cases} \quad \eta(a) = \delta_a \quad \mu \gg= [\![p]\!] = \lambda A. \int_{a \in 2^H} [\![p]\!](a)(A) \cdot \mu(da)$$

$$\begin{array}{lclcl}
[\![skip]\!](a) & = & \delta_a & = & \eta(a) \\
[\![drop]\!](a) & = & \delta_\emptyset & = & \eta(\emptyset) \\
[\![x \leftarrow n]\!](a) & = & \delta_{\{\pi[n/x]::\sigma | \pi::\sigma \in a\}} & = & \eta(\{\pi[n/x] :: \sigma | \pi :: \sigma \in a\}) \\
[\![x = n]\!](a) & = & \delta_{\{\pi::\sigma \in a | \pi(x)=n\}} & = & \eta(\{\pi :: \sigma | \pi :: \sigma \in a \wedge \pi(x) = n\}) \\
[\![dup]\!](a) & = & \delta_{\{\pi::\pi::\sigma \ |\pi::\sigma \in a\}} & = & \eta(\{\pi :: \pi :: \sigma \ |\pi :: \sigma \in a\}) \\
[\![\neg b]\!](a) & = & \delta_{\{\pi::\sigma \in a | \pi \not\models b\}} & = & [\![b]\!](a) >\!\!>\!= \lambda s.\eta(a - s) \\
[\![p\&q]\!](a) & = & ([\![p]\!](a) \times [\![q]\!](a))(\{(s,t) | s \cup t \in a\}) & = & [\![p]\!](\pi :: \sigma) \\
 & & & & >\!\!>\!= \lambda s.([\![q]\!](\pi :: \sigma) \\
 & & & & >\!\!>\!= \lambda t.\eta(s \cup t))) \\
[\![p;q]\!](a) & = & [\![q]\!]([\![p]\!](a)) & = & [\![p]\!](\pi :: \sigma) >\!\!>\!= [\![q]\!] \\
[\![p \oplus_r q]\!](a) & = & r \cdot [\![p]\!](a) + (1 - r) \cdot [\![q]\!](a) & & \\
[\![p^{(0)}]\!](a) & = & [\![skip]\!](a) & & \\
[\![p^{(n+1)}]\!](a) & = & [\![skip\&p; p^{(n)}]\!](a) & & \\
[\![p^*]\!](a) & = & \bigsqcup_{n \in \mathbb{N}} [\![p^{(n)}]\!](a) & = & [\![skip\&p;p^*]\!](a)
\end{array}$$

This maintains many, but not all, of the typical properties we expect a Kleene algebra with tests to have. For instance, & is still commutative (thanks to the Frobenius theorem) and so are the test operations, but $[\![skip\&p^*;p]\!](\pi :: \sigma)$ is no longer an identity of p^*.

It will next be useful to define an extended form of some operators to act as short-hand for writing them down multiple times. Let p_i be a program predicated on the variable i, and let I be a finite set of i values indexed 1 through n, and let r_i be a probability predicated on i.

$$\mathop{;}_{i \in I} p_i = p_{i_1}; p_{i_2}; ...; p_{i_n} \qquad \mathop{\&}_{i \in I} p_i = p_{i_1} \& p_{i_2} \& ... \& p_{i_n}$$

$$\mathop{\oplus_{r_i}}_{i \in I} p_i = p_{i_1} \oplus_{r_1} (p_{i_2} \oplus_{r_2/(1-r_1)} (...(p_{i_n} \oplus_{r_n/(1-\sum\limits_{1 \le i < n} r_i)} drop)...))$$

This definition of sequential composition across multiple terms is only well-defined where the p_i are commutative. We will only be using it in commutative contexts like with test terms, so it is sufficient for our purposes. The sequential composition of multiple choice is designed such that each term p_i is picked with the given probability r_i. This requires that the sum of the probabilities does not exceed 1.

It is also useful to replace the assignment operator $\leftarrow$ and test operator $=$ with a complete assignment operator ! and complete test operator ? defined below, where X is the set of all fields in a packet. The complete assignment operator assigns all fields of a packet to match those of another, turning the first packet into the second. The complete test operator tests all fields of a packet against those of another, dropping the first if it does not perfectly match the second.

$$[\![\pi?]\!](a) = [\![\mathop{;}_{x \in X} x = \pi(x)]\!](a) \qquad [\![\pi!]\!](a) = [\![\mathop{;}_{x \in X} x \leftarrow \pi(x)]\!](a)$$

3 A Random Loop

Some of our results rely on a program structure that loops a random number of times before exiting the loop. The following ProbNetKAT code achieves that purpose. Π will be a set of packets where every packet has a boolean field x assigned to *true*. C will stand in for the body of the loop, written to only use packets of Π. Assigning *false* to the Π packets' field x yields a new set of packets Ψ, in one-to-one correspondance. Because the following code results in the execution of C a number of times that is a geometric random variable with parameter r, this code will be denoted $[\![C^{G_r}]\!]$.

$$[\![C^{G_r}]\!] = [\![(\text{while } x = \mathit{true} \text{ do } \{C \oplus_r x \leftarrow \mathit{false}\}); x \leftarrow \mathit{true}]\!]$$

Suppose this program is given a set of histories with head packets in Π, and consider dynamically each step the code takes. First it will enter a while loop conditioned on x being *true*, so that while the packets are in Π, the loop continues. Thus, being in Ψ at the end of the while loop acts as the marker to break the loop, and the loop will never execute its body on packets from Ψ. In each iteration of the while loop, the code makes a probabilistic choice. With probability r the code runs C, which, since C is designed to run only using packets from Π, will go through C's manipulations as expected, only extending the histories with packets from Π, and thus exiting its code block with all head packets in Π. Otherwise, with probability $1 - r$, the code maps the packets into Ψ, signalling the the end of the looping. After exiting the while loop, the code ends by re-mapping all head packets back into Π.

Thus, the while loop either exits with probability $1 - r$, or it runs C on the current head packets from Π and loops again. This is the structure of running Bernoulli trials until success, so the number of times C is run is geometrically distributed with respect to r. Specifically, the probability of running C n times is given by $(1 - r) * r^n$. This accomplishes the desired goal, as a geometric distribution assigns every natural number a positive probability, so C could be run for any number of iterations.

4 Main Results

4.1 A Post-Correspondance Embedding

Direct Product Expressions (DPEs) For the first undecidable problem that will be presented, it is best to lay out an intermediate undecidable problem involving a variation on regular expressions, which will be called DPEs as shorthand for "direct product expressions". DPEs, rather than describing sets of the usual words from Σ^* (the free monoid on a generating alphabet Σ), instead describe words from direct product of n copies of Σ^* in the category of monoids, denoted $(\Sigma^*)^n$. In this direct product, multiplication is defined component-wise, such that $(a_1, a_2, ..., a_n) \cdot (b_1, b_2..., b_n) = (a_1 \cdot b_1, a_2 \cdot b_2, ..., a_n \cdot b_n)$.

Consider the minimal generating set for $(\Sigma^*)^n$, elements of which are tuples made entirely of the identity element ϵ, except for one position in which sits an element of Σ. Call this set Γ. For brevity, refer to the element of Γ containing $a \in \Sigma$ in position i as $a_{(i)}$, and in general, the tuple containing only ϵ except for $w \in \Sigma^*$ at index i as $w_{(i)}$.

$$\Gamma = \{a_{(i)} | a \in \Sigma \wedge 1 \leq i \leq n\}$$

Now, Γ^*, being the free monoid generated by Γ, can be treated in the usual manner as the alphabet for a regular expression. And because $(\Sigma^*)^n$ is also a monoid generated by Γ, there exists a unique canonical epimorphism $h : \Gamma^* \rightarrow (\Sigma^*)^n$ that acts as an identity on elements of Γ. This h is the map that performs component-wise multiplication, introducing a sort of commutativity in its image such that, while $a_{(1)} \cdot a_{(2)} \neq a_{(2)} \cdot a_{(1)}$, it is the case that $h(a_{(1)} \cdot a_{(2)}) = (a, a) = h(a_{(2)} \cdot a_{(1)})$. It is in the image of this map h that we would like to consider DPEs, though the regular expressions yielding the DPEs will be easier to write out and work with here.

Addition on DPEs continues to remain a nondeterministic choice between terms, so that the set of words matching $(a, a) + (b, b)$ is just the set containing only (a, a) and (b, b). Note that addition is therefore not componentwise and introduces a sort of choice dependence between the choices made at different indices; (a, b) matches $(a+b, a+b)$, but not $(a, a)+(b, b)$. Finally, the asterate is defined as per usual, with w^* being the supremum of across all n of w^n, satisfying $w^* = 1 + w \cdot w^*$, for w a word in the direct product and 1 the multiplicative identity. Informally, we can then see that a DPE looks like a set of regular expressions, each on their own separate track corresponding to index, which can behave dependently on one another. Expoitation of this dependency will yield the undecidability that we seek.

Certain properties of DPEs are decidable just as easily as regular expressions. Emptiness can be decided simply by checking if the expression is ϵ. Membership of some word w can be decided by considering each of the finite permutations of elements from Γ that map to w under h, and checking membership of any of those in any regular expression that maps to the DPE under h. The union of two DPEs x and y is also decidable, simply as $x + y$.

Unlike regular expressions, however, the intersection of DPEs is not decidable, as can be seen with an embedding of the Post-Correspondence Problem (PCP) below. It should be noted that similar undecidable results concerning Kleene algebras with commutativity conditions do already exist in the literature [8, 9]. However, the formulation of the proof below in terms of DPEs is more direct and intuitive for application to ProbNetKAT, as an indexed set of words looks quite similar to a set of histories marked by head packet. I conjecture that the related undecidable results for such commutative Kleene algebras can be embedded into ProbNetKAT similarly to the method shown.

▶ **Theorem 1.** *For arbitrary DPEs A and B, $A \cap B = \emptyset$ is undecidable.*

Proof. To show the undecideability of emptiness of intersection, we will start by defining notation for summing in an expression across multiple terms. Let I be a finite, indexed set of n with element m denoted i_m, and let e_{i_m} be an expression predicated on element i_m. Because addition of expressions is commutative and there are only finitely many terms, this is well-defined.

$$\sum_{i \in I} e_i = e_{i_1} + e_{i_2} + ... + e_{i_n}$$

Now take an arbitrary instance of the PCP. The PCP is the following decision problem: Given an indexed set of word pairs $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ on an alphabet (given here by Σ), does there exist a non-empty sequence of indices $i_1, i_2, ..., i_m$ such that $x_{i_1} \cdot x_{i_2} \cdot ... \cdot x_{i_m} = y_{i_1} \cdot y_{i_2} \cdot ... \cdot y_{i_m}$?

We then provide the two regular X and Y below with $n = 2$, involving the left and right sides of the PCP pairs. This uses the summation definition and $1 \leq k \leq n$ as shorthand for $k \in \{m \in \mathbb{N} | 1 \leq m \leq n\}$.

$$X = [\sum_{1 \leq k \leq n} (x_k)_{(1)} \cdot (y_k)_{(2)}] \cdot [\sum_{1 \leq k \leq n} (x_k)_{(1)} \cdot (y_k)_{(2)}]^*$$

$$Y = [\sum_{a \in \Sigma} a_{(1)} \cdot a_{(2)}] \cdot [\sum_{a \in \Sigma} a_{(1)} \cdot a_{(2)}]^*$$

X generates every possible non-empty ordering of the word pairs from the PCP with each side of the pairs at its own index, and Y generates every possible non-empty word in Σ^* copied at each index.

Mapping these sets under h now actually separates the words and indices into their designated positions, such that we are left with a pair of two words. For example, if (wo, w) and (rd, ord) were PCP pairs, then we would see $h((wo)_{(1)} \cdot w_{(2)} \cdot (rd)_{(1)} \cdot (ord)_{(2)}) = (word, word)$.

In that way, the mapping of X under h to get the DPE A gives us every pair of words generated by different sides of the PCP pairs with the same index sequence, and the mapping of Y to get the DPE B gives us every pair of matching words. Thus, if we could decide if there exists a word (s, t) in both A and B, we would know that the s and t are matching words made from the same index sequence, comprising an affirmative solution to the PCP problem. And if such a word did not exist, then there would be no solution, as every possible word from every possible sequence of indices is represented. This therefore decides the PCP. But, the PCP is undecidable, so the existence of any such word (s, t) in the intersection is also undecidable. ◀

ProbNetKAT DPEs. Suppose one has two ProbNetKAT programs p and q and wishes to decide if their support sets (where support is used in the discrete sense as the set given by $\mathsf{supp}(\mu)$ of points for which the distribution μ assigns positive mass). This could be used to determine if q successfully avoids every set of paths through waypoints that p routes through, regardless of the probability with which any permutation of waypoints is used. Perhaps some of p's route sets have been compromised, but exactly which ones are unknown. Or it could serve as a preliminary test for equality of the programs' distributions, as the decision procedure for equality might be difficult, and is in fact unknown at the time of this writing; certainly, if the supports are completely different, the distributions must also differ.

Or suppose that one wishes to check the size of the output set of a program on a given input. Perhaps this would be to ensure that packets never get too congested by using too few routes.

Unfortunately, the first of these problems is undecidable for fixed inputs with more than one distinct head packet. This can be shown in a discrete distribution case by showing that support intersection emptiness decides the emptiness of intersection of DPEs. The same example can further be used to show that the second problem, determining output size, is undecidable as well. These results do not reflect on the decidability of intersection or output size of non-random NetKAT programs, however. NetKAT programs can in fact be represented with regular sets of guarded strings [1] , so these two questions are easily decidable in NetKAT.

To show this, we will construct an embedding for DPEs in ProbNetKAT. This embedding will satisfy that $w = (w_1, w_2, ...w_n)$ is in a given DPE with alphabet Σ if and only if, given the set $\{\pi_1, \pi_2...\pi_n\}$ to start, the corresponding ProbNetKAT program puts out the history set $\{\pi_1 :: w_1, \pi_2 :: w_2, ...\pi_n :: w_n\}$ with positive probability, where the set of packets Pk is given by $\Sigma \cup \Pi \cup \Psi$, $\Pi = \{\pi_i | 1 \leq i \leq n\}$, and Ψ the appropriate packet set for the random loop from earlier. The embedding g is defined inductively below on regular expressions over $a_{(i)} \in \Gamma$, which yield the appropriate DPEs under h.

$$g(\epsilon) = [\![skip]\!]$$

$$g(a_{(i)}) = [\![\text{if } \pi_i? \text{ then } a!; dup; \pi_i! \text{ else } skip]\!]$$

$$g(x \cdot y) = g(y); g(x)$$

$$g(x + y) = g(x) \oplus g(y)$$

$$g(x^*) = [\![g(x)^{G_{0.5}}]\!]$$

▶ **Theorem 2.** *The function g yields a valid embedding of DPEs into ProbNetKAT, i.e., for a regular expression s on Γ, g satisfies that*

$$(w_1, ...w_n) \in h(s) \leftrightarrow g(s)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0.$$

Proof. This follows routinely by induction on the structure of regular expressions. See the appendix for the explicit proof. The trick is that each head packet acts as a label for its history, denoting the index or track of the regular expression to which that history matches in the DPE. ◀

▶ **Theorem 3.** *For arbitrary $a \in 2^H$ and ProbNetKAT programs p and q, whether or not* $\mathsf{supp}([\![p]\!](a)) \cap \mathsf{supp}([\![q]\!](a)) = \emptyset$ *is undecidable.*

Proof. Take the regular expressions X and Y from the section on DPEs. Because $h(X) \cap h(Y)$ is undecidable (as was proved in the above section on DPEs), we find that $\mathsf{supp}(g(X)(\{\pi_1, \pi_2\})) \cap \mathsf{supp}(g(Y)(\{\pi_1, \pi_2\})) = \emptyset$ is also undecidable. This is an example of the type of problem we wish to prove undecidable, so we are done. ◀

▶ **Theorem 4.** *For arbitrary $a \in 2^H$, $n \in \mathbb{N}$, and ProbNetKAT program p, whether or not* $\exists b \in \mathsf{supp}([\![p]\!](a))$ *such that $|b| = n$ is undecidable.*

Proof. Take the regular expressions X and Y from the section on DPEs. Consider the program given by $[\![g(X)\&g(Y)]\!]$.

Recognize that the program made by g always outputs a set of size 2 given $\{\pi_1, \pi_2\}$ as input. No base operation of g changes the number of elements in the set it is given, so this follows easily by induction. Then recall that the & operation essentially performs each of its two arguments' code independently, before taking the union of the results, respecting probability. The union of the results of $g(X)$ and $g(Y)$ on the input $\{\pi_1, \pi_2\}$ is thus of size 2 with positive probability iff there is some output made by both programs with positive probability, because the only way the union of two sets of the same finite size does not go up in size is if the two sets are the same. This means if we could decide if no output in the support was of size two, we could also decide the emptiness of support intersection. Emptiness of support intersection is undecidable, so we also cannot decide if the program $[\![g(X)\&g(Y)]\!]$ given $\{\pi_1, \pi_2\}$ has an output of size two in its support. ◀

▶ **Theorem 5.** *For arbitrary $a \in 2^H$, $n \in \mathbb{N}$, and ProbNetKAT program p, whether or not* $\exists b \in \mathsf{supp}([\![p]\!](a))$ *such that $|b| \leq n$ is undecidable.*

Proof. Consider the same set up as in theorem 4. Note again that the program made by g always outputs a set of size 2 given $\{\pi_1, \pi_2\}$ as input, and that the union of two sets is never less than the size of the either set. Thus, all sets in the support of $[\![g(X)\&g(Y)]\!]\{\pi_1, \pi_2\}$ are of size 2 or more. If there was a set b in that support of size at most 2, it would in fact be of size 2, and such a b of size 2 is of course size at most 2. Therefore, deciding if there is such b of size at most 2 is precisely the same as deciding if there is such a b of size 2 exactly. It is undecidable if there is such a set b of size 2, as proven in theorem 4, so it also undecideable if there is a set of size at most 2. ◀

4.2 A PFA embedding

PFAs Probabilistic finite automata (PFAs), are the probabilistic extension of DFAs. Rather than each transition deterministically moving from the current state to another state on a given symbol, each transition moves to one of several states with specific probability. The language, rather than being a set of strings, is a distribution on strings. This can be formally defined with the tuple $(Q, \Sigma, \Delta, q_0, F)$, where Q is a finite set of states, Σ is a finite set of symbols, Δ is a set of matrices, q_0 is the starting state, and F represents the set of final states. More specifically, Δ is a set of square stochastic matrices Δ_a indexed by states, such that for each $a \in \Sigma$, $\Delta_a(x, y)$ is the probability that state x transitions to state y on input symbol a. It will be useful as well to let q_0 also stand in for the horizontal vector of zeroes save for a one at index q_0, and F stand in for the vertical vector of zeroes save for those indices that are in F.

Using the notation that $\Delta_{w \cdot w'} = \Delta_w \cdot \Delta_{w'}$ for $w, w' \in \Sigma^*$, we can then define the word distribution $D(M)$ for a PFA M as the function from $w \in \Sigma^*$ to $q_0 \cdot \Delta_w \cdot F$, which returns the probability that w will be accepted. This can then be used to define the language of M for a given threshold λ as follows.

$$L_\lambda(M) = \{w \in \Sigma^* | D(M)(w) \geq \lambda\}$$

There are a few undecidable problems associated with PFAs from previous work on the subject [6, 2], some of which are closely related. These include:

- The emptiness problem: $\exists w \in \Sigma^*.D(M)(w) \geq \lambda$
  (Is the language empty for a given cutpoint?)
- The strict emptiness problem: $\exists w \in \Sigma^*.D(M)(w) > \lambda$
  (Is the strict variant of the language empty for a given cutpoint?)
- The equality problem: $\exists w \in \Sigma^*.D(M)(w) = 0.5$
  (Is there any word accepted exactly half the time?)
- The isolation problem: $\exists \epsilon > 0.\forall w \in \Sigma^*.|D(M)(w) - \lambda| \geq \epsilon$
  (Are there words that are accepted with probability arbitrarily close to a given value?)
- The value 1 problem: $\exists \epsilon > 0.\forall w \in \Sigma^*.D(M)(w) \leq 1 - \epsilon$
  (Are there words that are accepted with arbitrarily high probability?)

Equivalence and certain kinds of approximations between PFAs, however, are decidable [3, 7, 13]. This means equivalence and those approximations of PFAs will not be as useful here for finding undecidable problems. Nor do they directly yield analagous decision procedures for general ProbNetKAT programs, as the provided embedding does not fully encompass the forms that ProbNetKAT programs can take.

ProbNetKAT PFAs. It is easy to create ProbNetKAT code that only does meaningful work on certain input sets, like code that begins by dropping everything with a certain head packet. This might arise in networking if some subset of the packets Pk are the only ones properly formatted as request packets for the network, so any other packet is discarded. It is also easy to create code that produces some output with some form of geometric distribution, as the random loop does. For this reason, it would be useful to be able to compare a ProbNetKAT program's distribution on specific sets to a geometric distribution, to see whether or not they are equal within a certain error, or whether one's probabilities dominate the other's.

One might also like to do the same sorts of comparisons between programs. Because these are probabilistic algorithms, approximation within a certain error is often just as good in application as actual equality. Determining whether one program's distribution on certain sets dominates another's could be useful in a context where one is trying to improve successful service probability. For such a case, suppose that one's current neworking program drops all packets (fails) with a positive probability, and that one wishes to improve it so that it fails less often, but doesn't lower the probability of any of its successful output. This would be solved by determining if a candidate replacement program dominated the original on non-empty outputs.

Unfortunately, because PFAs can be embedded in ProbNetKAT, these sorts of problems are often undecidable. These problems also do not have any clear analogues in non-random NetKAT, as they are intimately concerned with the properties of probability distributions.

The function g performs this embedding for a PFA $M = (Q, \Sigma, \Delta, q_0, F)$. Define $Pk = \Pi \cup \Psi \cup \Sigma$, where $\Pi = \{\pi_q | q \in Q\}$, Ψ as the image of Π under $x \leftarrow false$, $0 < r < 1$, and $0 < s \leq \frac{1}{|\Sigma|}$. Both Π and Ψ maintain their roles as given in section 3 for random looping. We can then give g with the following, which works as will be described in theorem 6.

$$g(M) = [\underset{q\in Q}{\&} \text{ if } \pi_q? \text{ then } \underset{a\in\Sigma}{\oplus_s} a!; dup; (\underset{t\in Q}{\oplus_{\Delta_a(q,t)}} \pi_t!) \text{ else } drop]^{G_r}; (\underset{f\in F}{\&} \pi_f?); \pi_{q_0}!$$

▶ **Theorem 6.** *The function g yields a valid embedding of PFAs into ProbNetKAT, i.e., for a PFA M, g satisfies that*

$$g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: rev(w)\}) = (rs)^{|w|} \cdot (1-r) \cdot D(M)(w)$$

where $rev(x)$ is the function that reverses x.

Proof. This validity of this statement can be seen by considering traversal through the code dynamically. The head packet is used to record the state of the PFA, so it starts at the packet corresponding to the starting state of the PFA, q_0. The code then enters a random loop. Each iteration of this loop starts by determining which state the configuration is in by checking head packets, and picking a random letter/packet to add to the front of the history. By putting letters on the front each time, the sequence of letters chosen is recorded as the reverse of the packet history. If it randomly drops here instead of picking a letter, then every branch of parallel execution will have dropped, so the code will output the empty set. If a letter is picked and duped into the history, then, knowing which state the configuration is in and which letter it chose, the code chooses a new head packet with probability in accordance with Δ.

Thus each iteration of the loop corresponds to a transition in the PFA, and either maintains a singleton set or drops to the empty set. For each iteration of the random loop that doesn't drop to the empty set, not only is the probability multiplied by r to continue, but also by s to pick a letter, and further by the transition probability. If the sequence of letters chosen after $|w|$ iterations is the word w, then these transition probabilities multiply to $D(w)$ as in the PFA, for a total probability of $(rs)^{|w|}D(w)$ for having reached that iteration with those choices. When the random loop finally exits, it does so with a probability $1 - r$, for a total probability of $(rs)^{|w|}D(w) \cdot (1 - r)$.

After going through this loop some arbitrary number of times and finally exiting, the code checks to see if the state is now a final state. If not, it drops, but if so, it standardizes the head packet to that of the initial state. (Nothing here changes the probability.) Thus, any non-empty outputs from the input $\{\pi_{q_0}\}$ correspond to having reached a final state through PFA transitions, and the history of the output is the reverse of the word that led there. ◀

▶ **Theorem 7.** *For an aribtrary ProbNetKAT program p, a set of non-empty history sets B, an arbitrary $a \in 2^H$, arbitrary values u and v, and a function f taking a non-empty set of histories to a linear combination of the contained histories' lengths, whether or not $\exists b \in B.[\![p]\!](a)(b) \geq u \cdot v^{f(b)}$, i.e., whether the distribution on non-empty outputs of p on input a can be bounded by a geometric function of output history length, is undecidable.*

Proof. Take the emptiness problem for PFAs on alphabet Σ. Translate it into ProbNetKAT using g as follows, noting that the reverse of a word in $w \in \Sigma^*$ always exists and always is the same length as w.

$$\begin{aligned}\exists w \in \Sigma^*.D(M)(w) \geq \lambda &\iff \exists w \in \Sigma^*.(rs)^{|w|} \cdot (1-r) \cdot D(M)(w) \geq (rs)^{|w|} \cdot (1-r) \cdot \lambda \\ &\iff \exists w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: rev(w)\}) \geq (rs)^{|w|} \cdot (1-r) \cdot \lambda \\ &\iff \exists w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) \geq (rs)^{|w|} \cdot (1-r) \cdot \lambda\end{aligned}$$

Because the emptiness problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, $u = (1-r)\lambda$, $v = rs$, and $f(b) = \sum_{h \in b} |h|$, we find that said statement is an example of the type of problem we are trying to prove undecidable, so we are done. ◀

▶ **Theorem 8.** *For an arbitrary ProbNetKAT programs p and q, an arbitrary $a \in 2^H$, arbitrary values u and v, and a function f taking a non-empty set of histories to a linear combination of the contained histories' lengths, whether or not $\exists b \in B. |[\![p]\!](a)(b) - [\![q]\!](a)(b)| \geq u \cdot v^{f(b)}$, i.e., whether p's distribution approximates q's on non-empty outputs to within an error exponentially decaying with history length, is undecidable.*

Proof. Let q be the program $[\![drop]\!]$. This program's output distribution always assigns probability 0 to non-empty sets.

$$\begin{aligned}\exists b \in B. |[\![p]\!](a)(b) - [\![q]\!](a)(b)| \geq u \cdot v^{f(b)} &\iff \exists b \in B. |[\![p]\!](a)(b) - [\![drop]\!](a)(b)| \geq u \cdot v^{f(b)} \\ &\iff \exists b \in B. |[\![p]\!](a)(b) - 0| \geq u \cdot v^{f(b)} \\ &\iff \exists b \in B. [\![p]\!](a)(b) \geq u \cdot v^{f(b)}\end{aligned}$$

The statement in the final line was shown undecidable in thoerem 7, so our desired theorem is undecidable in the instance when $q = [\![drop]\!]$. It is therefore undecidable. ◀

▶ **Theorem 9.** *For an aribtrary ProbNetKAT program p, a set of non-empty history sets B, an arbitrary $a \in 2^H$, arbitrary values u and v, and a function f taking a non-empty set of histories to a linear combination of the contained histories' lengths, the following are undecidable.*

- $\exists b \in B. [\![p]\!](a)(b) > u \cdot v^{f(b)}$
  (Is p's distribution on non-empty outputs bounded from above by a given geometric function that changes with output history length?)
- $\exists b \in B. [\![p]\!](a)(b) = u \cdot v^{f(b)}$
  (Does p's distribution on non-empty outputs ever coincide with a given geometric function that changes with output history length?)
- $\exists \epsilon > 0. \forall b \in B. |[\![p]\!](a)(b) - u \cdot v^{f(b)}| \geq u \cdot v^{f(b)} \cdot \epsilon$
  (Does p's distribution on non-empty outputs get within an arbitrarily small scalar of a given geometric function that changes with output history length?)
- $\exists \epsilon > 0. \forall b \in B. [\![p]\!](a)(b) \leq u \cdot v^{f(b)} \cdot (1 - \epsilon)$
  (Can p's distribution on non-empty outputs be bounded from above by a scalar<1 of a given geometric function that changes with output history length?)

Proof. Follow the same translation procedure as theorem 7. Starting from the strict emptiness problem, equality problem, isolation problem, and value 1 problem for PFAs, respectively. See the appendix for an explicit proof. ◀

▶ **Theorem 10.** *For arbitrary ProbNetKAT programs p and q, a set of non-empty history sets B, and arbitrary $a \in 2^H$, the following are undecidable.*

- $\exists b \in B. [\![p]\!](a)(b) \geq [\![q]\!](a)(b)$
  (Is p's distribution on non-empty outputs bounded strictly from above (strongly dominated) by q's?)
- $\exists b \in B. [\![p]\!](a)(b) > [\![q]\!](a)(b)$
  (Is p's distribution on non-empty outputs bounded from above (weakly dominated) by q's?)

- $\exists b \in B.[\![p]\!](a)(b) = [\![q]\!](a)(b)$
  (Does p's distribution on non-empty outputs ever coincide with q's?)
- $\exists \epsilon > 0.\forall b \in B.|[\![p]\!](a)(b) - [\![q]\!](a)(b)| \geq [\![q]\!](a)(b) \cdot \epsilon$
  (Does p's distribution on non-empty outputs get within an arbitrarily small scalar of q's?)
- $\exists \epsilon > 0.\forall b \in B.[\![p]\!](a)(b) \leq [\![q]\!](a)(b) \cdot (1 - \epsilon)$
  (Can p's distribution on non-empty outputs be bounded from above by a scalar<1 of q's?)

Proof. Consider the following program p

$$[\![p]\!] = [\![(\underset{a\in\Sigma}{\oplus_s} a!; dup)^{G_r}; \pi_{q_0}! \oplus_\lambda drop]\!]$$

This program's output distribution on input $\{\pi_{q_0}\}$ is given for non-empty output sets by

$$[\![p]\!](\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) = (rs)^{|w|} \cdot (1-r) \cdot \lambda$$

This distribution is precisely the term that we showed was undecideably comparable to arbitrary programs in theorems 7 and 9. Substituting that term, not with $u \cdot v^{f(b)}$ as was done in theorems 7 and 9, but rather with $[\![q]\!](a)(b)$, yields instances of each of the above problems. As values were merely substituted for identical values, the statements are still undecidable. Thus, each of the problems named are not generally decidable. ◀

5 Conclusion

By encoding the Post Correspondence Problem and various undecidable problems of probabilistic finite automata, we have been able to show that various problems for ProbNetKAT are undecidable. These include emptiness of support intersection, size of the output set, dominance of distribution probabilities over other programs', the satisfaction of geometric distribution bounds, and more.

However, it is still open whether or not ProbNetKAT program equality is decidable. NetKAT's equality is decidable, and many useful networking problems like waypointing are reducible to program equality. There is hope that the same power could be achieved in ProbNetKAT. It is known that equality of ProbNetKAT programs is decidable if one removes random choice (since that is just NetKAT), if one removes *dup* (which can be shown with some linear algebra and Markov chains [11]), and if one removes the asterate (since distributions become finite). It still remains to be seen if ProbNetKAT program equality can be decided if all three are present. Unfortunately it may be the case that with all three features, program equality becomes undecidable. In such an event, embeddings like those shown here may be instrumental in proving undecidability.

Future work could also be done to determine if it is decidable whether one program approximates another to within a constant error bound. Being probabilistic, this is often as good as equality in application. We have shown here that it is undecidable on non-empty outputs where the error decays exponentially with history length (theorem 8).

Acknowledgements. Thanks goes to Dexter Kozen for mentorship and support throughout this project, Steffen Smolka for insightful discussions about ProbNetKAT and beyond, and Eric Perdew for helpful revision. Additional thanks goes to the anonymous reviewers for their valuable comments.

References

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

2 Vincent D Blondel, Vincent Canterini, et al. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computing systems*, 36(3):231–245, 2003.

3 Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. *On the Computation of Some Standard Distances Between Probabilistic Automata*, pages 137–149. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/11812128_14.

4 Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic NetKAT. In Peter Thiemann, editor, *25th European Symposium on Programming (ESOP 2016)*, volume 9632 of *Lecture Notes in Computer Science*, pages 282–309, Eindhoven, The Netherlands, April 2016. Springer.

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

6 Hugo Gimbert and Youssouf Oualhadj. *Probabilistic Automata on Finite Words: Decidable and Undecidable Problems*, pages 527–538. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. URL: http://dx.doi.org/10.1007/978-3-642-14162-1_44, doi:10.1007/978-3-642-14162-1_44.

7 Stefan Kiefer, Andrzej S. Murawski, Joël Ouaknine, Björn Wachter, and James Worrell. *On the Complexity of the Equivalence Problem for Probabilistic Automata*, pages 467–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-28729-9_31.

8 Dexter Kozen. Kleene algebra with tests and commutativity conditions. In T. Margaria and B. Steffen, editors, *Proc. Second Int. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 14–33, Passau, Germany, March 1996. Springer-Verlag.

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

10 Dexter Kozen. NetKAT: A formal system for the verification of networks. In Jacques Garrigue, editor, *Proc. 12th Asian Symposium on Programming Languages and Systems (APLAS 2014)*, volume 8858 of *Lecture Notes in Computer Science*, Singapore, November 17–19 2014. Asian Association for Foundation of Software (AAFS), Springer.

11 Steffen Smolka, David Kahn, Praveen Kumar, and Nate Foster. Deciding probabilistic program equivalence in NetKAT. http://www.cs.cornell.edu/~smolka/papers/mcnetkat.pdf, 2017.

12 Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. Cantor meets Scott: Domain-theoretic foundations for probabilistic network programming. In *Proc. 44th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'17)*, pages 557–571, Paris, France, January 2017. ACM.

13 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21(2):216–227, 1992. doi:10.1137/0221017.

A Appendix

A.1 Theorem 2 in Detail

Theorem 2 The function g yields a valid embedding of DPEs into ProbNetKAT, i.e., for a regular expression s on Γ, g satisfies that

$$w \in h(s) \leftrightarrow g(s)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0.$$

Proof. Define the predicate $P(s)$ for regular expression s to denote that g validly embeds s. We can then use induction on regular expressions to show that P holds for all expressions.

The base case expressions of our induction are the empty expression ϵ and each symbol in the alphabet Σ.

To get the case of ϵ, note that the only element in the support of $[\![skip]\!]\{\pi_1, ..., \pi_n\}$ is $\{\pi_1, ..., \pi_n\}$ itself, which has no histories beyond the head packets. Any words made from the histories must therefore be empty.

$$\begin{aligned} &P(\epsilon): \\ w \in h(\epsilon) &\leftrightarrow \forall i, w_i = \epsilon \\ &\leftrightarrow \{\pi_1, ..., \pi_n\} = \{\pi_1 :: w_1, ..., \pi_n :: w_n\} \\ &\leftrightarrow [\![skip]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0 \\ &\leftrightarrow g(\epsilon)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0 \end{aligned}$$

To get the case of the single symbol, note that the code again only leaves a single element in the support:$\{\pi_1, ..., \pi_n\}$ with a single new packet added to the history of a single head packet. The only word made from such histories is thus that single symbol with the index of the head packet it is behind.

$$\begin{aligned} &P(a_{(i)}): \\ w \in h(a_{(i)}) &\leftrightarrow w_i = a \wedge \forall j \neq i. w_j = \epsilon \\ &\leftrightarrow \pi_i :: w_i = \pi_i :: a \wedge \forall j \neq i. \pi_j :: w_j = \pi \\ &\leftrightarrow [\![\text{ if } \pi_i? \text{ then } a!; dup; \pi_i! \text{ else } skip]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0 \\ &\leftrightarrow g(a_{(i)})(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0 \end{aligned}$$

Now, assuming that the embedding is valid for smaller expressions x and y, we consider the inductive steps for each of the operators: multiplication, addition, and asterate.

Note that adding new elements to the history stores them in the reverse order of their addition, so we end up reversing the order of x and y terms. Further recall that the computation done on a packet at a given point is determined solely by the head packet, not the history, as no operation can read from the history. This allows us to prove the case of multiplication as follows.

$$\begin{aligned} &P(x \cdot y): \\ w \in h(x \cdot y) &\leftrightarrow w \in h(x) \cdot h(y) \\ &\leftrightarrow \exists u, v \in (\Sigma^*)^n. u \in h(x) \wedge v \in h(y) \wedge u \cdot v = w \\ &\leftrightarrow \exists u, v \in (\Sigma^*)^n. g(x)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: u_1, ..., \pi_n :: u_n\}) > 0 \\ &\qquad \wedge g(y)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: v_1, ..., \pi_n :: v_n\}) > 0 \wedge u \cdot v = w \\ &\leftrightarrow \exists u, v \in (\Sigma^*)^n. [\![g(y); g(x)]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: u_1 :: v_1, ..., \pi_n :: u_n :: v_n\}) > 0 \\ &\qquad \wedge \forall i, u_i \cdot v_i = w_i \\ &\leftrightarrow [\![g(y); g(x)]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0 \\ &\leftrightarrow g(x \cdot y)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0 \end{aligned}$$

The case of addition follows because scaling a probability by 0.5 does not change its positivity.

$$\begin{aligned}
&P(x+y):\\
w \in h(x+y) &\leftrightarrow w \in h(x) \cup h(y)\\
&\leftrightarrow w \in h(x) \vee w \in h(y)\\
&\leftrightarrow g(x)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0\\
&\qquad \vee g(y)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0\\
&\leftrightarrow [\![g(x) \oplus_{0.5} g(y)]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0\\
&\leftrightarrow [\![g(x+y)]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0
\end{aligned}$$

Finally, the asterate case follows from recalling that the random loop is capable stopping after any number of iterations, and always does so with a positive probability.

$$\begin{aligned}
&P(x^*):\\
w \in h(x^*) &\leftrightarrow w \in \bigcup_{n \in \mathbb{N}} h(x)^n\\
&\leftrightarrow \exists n \in \mathbb{N}.w \in h(x)^n\\
&\leftrightarrow \exists n \in \mathbb{N}.[\![g(x)^n]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0\\
&\leftrightarrow [\![g(x)^{G_{0.5}}]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0\\
&\leftrightarrow [\![g(x^*)]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0
\end{aligned}$$

The validity of the embedding holds for all base elements and through all operations of regular expressions on Γ. Thus, by induction, the validity of the embedding holds for all regular expressions.

◀

A.2 Theorem 9 in Detail

Theorem 9 For an aribtrary ProbNetKAT program p, a set of non-empty history sets B, an arbitrary $a \in 2^H$, arbitrary values u and v, and a function f taking a non-empty set of histories to a linear combination of the contained histories' lengths, the following are undecidable.

- $\exists b \in B.[\![p]\!](a)(b) > u \cdot v^{f(b)}$
  (Is p's distribution on non-empty outputs bounded from above by a given geometric function that changes with output history length?)
- $\exists b \in B.[\![p]\!](a)(b) = u \cdot v^{f(b)}$
  (Does p's distribution on non-empty outputs ever coincide with a given geometric function that changes with output history length?)
- $\exists \epsilon > 0.\forall b \in B.|[\![p]\!](a)(b) - u \cdot v^{f(b)}| \geq u \cdot v^{f(b)} \cdot \epsilon$
  (Does p's distribution on non-empty outputs get within an arbitrarily small scalar of a given geometric function that changes with output history length?)
- $\exists \epsilon > 0.\forall b \in B.[\![p]\!](a)(b) \leq u \cdot v^{f(b)} \cdot (1-\epsilon)$
  (Can p's distribution on non-empty outputs be bounded from above by a scalar<1 of a given geometric function that changes with output history length?)

Proof. Take the strict emptiness problem for PFAs on alphabet Σ. Translate it into ProbNetKAT using g as follows, noting that the reverse of a word in $w \in \Sigma^*$ always exists and always is the same length as w.

$$\begin{aligned}\exists w \in \Sigma^*. D(M)(w) > \lambda &\iff \exists w \in \Sigma^*. (rs)^{|w|} \cdot (1-r) \cdot D(M)(w) > (rs)^{|w|} \cdot (1-r) \cdot \lambda \\ &\iff \exists w \in \Sigma^*. g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: \mathit{rev}(w)\}) > (rs)^{|w|} \cdot (1-r) \cdot \lambda \\ &\iff \exists w \in \Sigma^*. g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) > (rs)^{|w|} \cdot (1-r) \cdot \lambda\end{aligned}$$

Because the strict emptiness problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, $u = (1-r)\lambda$, $v = rs$, and $f(b) = \sum_{h \in b} |h|$, we find that said statement is an example of the first type of problem we are trying to prove undecidable, so we have proved that the first problem in the list is undecidable.

Take the equality problem for PFAs on alphabet Σ. Translate it into ProbNetKAT using g as follows.

$$\begin{aligned}\exists w \in \Sigma^*. D(M)(w) = 0.5 &\iff \exists w \in \Sigma^*. (rs)^{|w|} \cdot (1-r) \cdot D(M)(w) = (rs)^{|w|} \cdot (1-r) \cdot 0.5 \\ &\iff \exists w \in \Sigma^*. g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: \mathit{rev}(w)\}) = (rs)^{|w|} \cdot (1-r) \cdot 0.5 \\ &\iff \exists w \in \Sigma^*. g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) = (rs)^{|w|} \cdot (1-r) \cdot 0.5\end{aligned}$$

Because the equality problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, $u = (1-r)/2$, $v = rs$, and $f(b) = \sum_{h \in b} |h|$, we find that said statement is an example of the second type of problem we are trying to prove undecidable, so we have proved that the second problem in the list is undecidable.

Take the isolation problem for PFAs on alphabet Σ. Pick an instance with a positive λ. Translate it into ProbNetKAT using g as follows.

$$\begin{aligned}&\exists \epsilon > 0. \forall w \in \Sigma^*. |D(M)(w) - \lambda| \geq \epsilon \\ &\iff \exists \epsilon > 0. \forall w \in \Sigma^*. (rs)^{|w|} \cdot (1-r) \cdot |D(M)(w) - \lambda| \geq (rs)^{|w|} \cdot (1-r) \cdot \epsilon \\ &\iff \exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: \mathit{rev}(w)\}) - (rs)^{|w|} \cdot (1-r) \cdot \lambda| \geq (rs)^{|w|} \cdot (1-r) \cdot \epsilon \\ &\iff \exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) - (rs)^{|w|} \cdot (1-r) \cdot \lambda| \geq (rs)^{|w|} \cdot (1-r) \cdot \epsilon\end{aligned}$$

At this point, let $u = (1-r)\lambda$ and $v = rs$. Continue by substituting these variables into the final line, and note that for every $\epsilon/\lambda > 0$ there exists $\epsilon' = \epsilon/\lambda > 0$.

$$\begin{aligned}&\exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) - (rs)^{|w|} \cdot (1-r) \cdot \lambda| \geq (rs)^{|w|} \cdot (1-r) \cdot \epsilon \\ &\iff \exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) - u \cdot v^{|w|}| \geq u \cdot v^{|w|} \cdot \epsilon/\lambda \\ &\iff \exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) - u \cdot v^{|w|}| \geq u \cdot v^{|w|} \cdot \epsilon\end{aligned}$$

Because the isolation problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, and $f(b) = \sum_{h \in b} |h|$, we find that said statement is an example of the third type of problem we are trying to prove undecidable, so we have proved that the third problem in the list is undecidable.

Take the value 1 problem for PFAs on alphabet Σ. Pick an instance with a positive λ. Translate it into ProbNetKAT using g as follows.

$$\begin{aligned}
&\exists\epsilon > 0.\forall w \in \Sigma^*.D(M)(w) \leq 1-\epsilon \\
\iff\ &\exists\epsilon > 0.\forall w \in \Sigma^*.(rs)^{|w|}\cdot(1-r)\cdot D(M)(w) \leq (rs)^{|w|}\cdot(1-r)\cdot(1-\epsilon) \\
\iff\ &\exists\epsilon > 0.\forall w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: rev(w)\}) \leq (rs)^{|w|}\cdot(1-r)\cdot(1-\epsilon) \\
\iff\ &\exists\epsilon > 0.\forall w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) \leq (rs)^{|w|}\cdot(1-r)\cdot(1-\epsilon)
\end{aligned}$$

Because the value 1 problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, $u = (1-r)\lambda$, $v = rs$, and $f(b) = \sum_{h\in b}|h|$, we find that said statement is an example of the final type of problem we are trying to prove undecidable, so we are done. ◀