Towards an Analysis of Dynamic Gossip in NetKAT

Malvin Gattinger[1][0000−0002−2498−5073] and Jana Wagemaker[2][0000−0002−8616−3905]

[1] University of Groningen, Groningen, The Netherlands
malvin@w4eg.eu
[2] CWI, Amsterdam, The Netherlands
jana.wagemaker@cwi.nl

Abstract. In this paper we analyse the dynamic gossip problem using the algebraic network programming language NetKAT.
NetKAT is a language based on Kleene algebra with tests and describes packets travelling through networks. It has a sound and complete axiomatisation and an efficient coalgebraic decision procedure. Dynamic gossip studies how information spreads through a peer-to-peer network in which links are added dynamically.
In this paper we embed dynamic gossip into NetKAT. We show that a reinterpretation of NetKAT in which we keep track of the state of switches allows us to model Learn New Secrets, a well-studied protocol for dynamic gossip. We axiomatise this reinterpretation of NetKAT and show that it is sound and complete with respect to the packet-processing model, via a translation back to standard NetKAT.
Our main result is that many common decision problems about gossip graphs can be reduced to checks of NetKAT equivalences. We also implemented the reduction.

Keywords Dynamic gossip, Kleene algebra with tests (KAT), Network programming language, NetKAT, Peer-to-peer communication.

1 Introduction

The dynamic gossip problem is a generalisation of the classic telephone problem, in which agents exchange secrets in phone calls with the goal to spread all secrets. In the dynamic setting, also phone numbers are exchanged, hence who can call whom constantly changes. More generally, dynamic gossip provides a formal model of any peer-to-peer setting in which information has to be spread or synchronised between multiple nodes.

As of now, no formal language and logic exists that captures dynamic gossip in a satisfactory way. There is existing work on gossip using formal languages which we will discuss in section 7, but to our knowledge there is no sound and complete proof system to describe dynamic gossip axiomatically.

Anderson et al. in [1] and Foster et al. in [10], introduced NetKAT, a logic to describe packet-processing behaviour of networks. In this paper we will model dynamic gossip in NetKAT.

Our contributions are twofold. First, we show that we can simulate switch states in NetKAT without losing soundness and completeness, as long as different packets do not interact with each other. This observation is motivated by the application to dynamic gossip, but also applies to NetKAT in general. Second, we show that simulating switch states in NetKAT is a natural choice for the analysis of dynamic gossip protocols. We provide a way to compute all call sequences of the so-called Learn New Secrets (LNS) protocol by evaluating a NetKAT policy. Moreover, we reduce the decision problems whether LNS is weakly or strongly successful to a check of NetKAT equivalences. Given the complete axiomatisation and efficient decision procedure of NetKAT, any question about gossip graphs expressible as a NetKAT equivalence is decidable. More generally, this paper builds a bridge between the two areas of gossip protocols and network programming languages.

We proceed as follows. In section 2 we summarise the main definitions of NetKAT. Then we present a reinterpretation of NetKAT simulating switch states in section 3. In section 4 we recapitulate the dynamic gossip problem and in section 5 we show how to encode it in NetKAT, including proofs that these translations are sound. We describe an implementation of our methods in section 6, discuss related work in section 7 and conclude with ideas for future work in section 8. Further details can also be found in the master thesis [20] on which this paper is based on.

2 Standard NetKAT

NetKAT was first presented in [1] and is a network programming language with a strong mathematical foundation. It extends Kleene Algebra with Tests (KAT) and relies on technical results from that field.

A Kleene Algebra with Tests is a Kleene algebra — the algebra of regular expressions — with a Boolean algebra. Its soundness and completeness with respect to relational models and language-theoretic models is proven in [16]. Formally, a Kleene algebra with tests (KAT) is a two-sorted algebraic structure $(K, B, +, \cdot, ^*, 0, 1, \neg)$ where $B \subseteq K$ and $\neg$ is a unary operator defined only on B such that

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

The equational theory of KAT is PSPACE-complete as shown in [16].

The Kleene algebra operators are $+$ for non-deterministic choice, $\cdot$ for sequential composition, the Kleene star * for finite iteration, 0 for fail and 1 for skip. Elements of the Boolean algebra are called tests, and on tests the $+$ and

$\cdot$ operators behave as disjunction and conjunction respectively. The negation operator $\neg$ can only be applied to elements of the Boolean algebra.

The axioms of KAT are the axioms of Kleene algebra combined with the axioms of Boolean algebra. They can be found in [15].

NetKAT's main purpose is to describe how packets travel through a network, taking into account both the network topology and the individual switch behaviours, where a switch is a node in the network topology. To do so, NetKAT extends KAT with additional primitives for network behaviour and axioms governing those primitives. Syntactically, NetKAT expressions can be predicates and policies. Predicates are the constants true and false (1 and 0 respectively), tests ($f = n$), negation ($\neg a$), disjunction ($a + b$) and conjunction ($a \cdot b$). Policies are all predicates, modifications ($f \leftarrow n$), union ($p + q$), sequential composition ($p \cdot q$) and iteration (p^*). We assume finite NetKAT networks throughout this paper.

Definition 1. *The syntax of NetKAT is given by the following* predicates a *and* policies p*:*

$$a ::= 1 \mid 0 \mid f = n \mid a + b \mid a \cdot b \mid \neg a$$
$$p ::= a \mid f \leftarrow n \mid p + q \mid p \cdot q \mid p^*$$

where f ranges over some finite set of fields $f ::= f_1 \mid \cdots \mid f_k$ (including a switch field sw *and a port field* pt*) and n is a value from a finite domain.*

Example 1. The NetKAT expression $\mathsf{sw} = A \cdot \mathsf{pt} = 4 \cdot \mathsf{dst} \leftarrow H \cdot \mathsf{pt} \leftarrow 7$ can be read as "test whether the packet is located at port 4 of switch A and then set the destination to H and move the packet to port 7".

There exist multiple models that satisfy the NetKAT axioms, but we are mainly concerned with the packet-processing model. A packet pk is a tuple $(f_1 = v_1, \ldots, f_n = v_n)$ which for each field f_k provides a value v_n from a finite domain. Among the fields, two are used for the location of the packet in the network: switch (sw) and port (pt). We write $\mathsf{pk}.f$ for the value in field f of pk and $\mathsf{pk}[f := n]$ for the packet obtained from pk by updating field f to n.

Standard NetKAT as presented in [1] also tracks the history of packets. It contains an operator *dup* to duplicate the current packet so that a copy of it is kept in the history. We do not need histories to model gossip in NetKAT, so we leave *dup* out here and simplify the semantics to work on packets instead of histories of packets.

We show the semantics of NetKAT in Figure 1. The interpretation of a policy p is a function that maps packets to sets of packets: $[\![p]\!] \colon P \to 2^P$ where P is the set of all packets. A test or filter $f = n$ takes any input packet pk and outputs the singleton $\{\mathsf{pk}\}$ if field f of pk equals n, and $\varnothing$ otherwise. A modification $f \leftarrow n$ takes any input packet pk and yields the singleton set $\{\mathsf{pk}[f := n]\}$.

The $+$ is interpreted as a multicast operation: the outcome of the policy $(\mathsf{pt} \leftarrow 1 + \mathsf{pt} \leftarrow 2)$ are two copies of the input packet, one at port 1 and one at port 2. The policy $p \cdot q$ is the Kleisli composition of p and q which we define as $(f \bullet g)(x) = \bigcup\{g(y) \mid y \in f(x)\}$. To iterate sequential composition with * we define $F^0(h) := \{h\}$ and $F^{i+1}(h) := (F \bullet F^i)h$.

$$
\begin{aligned}
[\![1]\!](\mathsf{pk}) &:= \{\mathsf{pk}\} \\
[\![0]\!](\mathsf{pk}) &:= \varnothing \\
[\![\neg a]\!](\mathsf{pk}) &:= \{\mathsf{pk}\} \setminus [\![a]\!](\mathsf{pk}) \\
[\![f = n]\!](\mathsf{pk}) &:= \begin{cases} \{\mathsf{pk}\} & \text{if } \mathsf{pk}.f = n \\ \varnothing & \text{otherwise} \end{cases} \\
[\![f \leftarrow n]\!](\mathsf{pk}) &:= \{\mathsf{pk}[f := n]\} \\
[\![p + q]\!](\mathsf{pk}) &:= [\![p]\!]h \cup [\![q]\!]\mathsf{pk} \\
[\![p \cdot q]\!](\mathsf{pk}) &:= ([\![p]\!] \bullet [\![q]\!])(h) \\
[\![p^*]\!](\mathsf{pk}) &:= \textstyle\bigcup_{i \in \mathbb{N}} [\![p]\!]^i(\mathsf{pk})
\end{aligned}
$$

Fig. 1. Semantics of NetKAT

$$
\begin{aligned}
(f \leftarrow n \cdot f' \leftarrow n') &\equiv (f' \leftarrow n' \cdot f \leftarrow n), \text{ if } f \neq f' && \text{MOD-MOD-COMM} \\
(f \leftarrow n \cdot f' = n') &\equiv (f' = n' \cdot f \leftarrow n), \text{ if } f \neq f' && \text{MOD-FILTER-COMM} \\
(f \leftarrow n \cdot f = n) &\equiv (f \leftarrow n) && \text{MOD-FILTER} \\
(f = n \cdot f \leftarrow n) &\equiv (f = n) && \text{FILTER-MOD} \\
(f \leftarrow n \cdot f \leftarrow n') &\equiv (f \leftarrow n') && \text{MOD-MOD} \\
(f = n \cdot f = n') &\equiv 0, \text{ if } n \neq n' && \text{CONTRA} \\
\textstyle\sum_i (f = i) &\equiv 1 && \text{MATCH-ALL}
\end{aligned}
$$

Fig. 2. NetKAT axioms

The NetKAT network of switches and hosts is not used in the semantics. As shown in [1], any network topology can be incorporated into a policy. This is not important here, because we will use a totally connected NetKAT network.

The NetKAT axioms are the axioms of KAT together with additional axioms for the interactions between tests and modifications. We show these additional axioms in Figure 2. The first one for instance tells us that when modifying two different fields, the order does not matter. The last axiom implies that the values of the fields are drawn from a finite domain. These axioms are sound and complete with respect to the packet-processing model of Figure 1. For proofs, see [1].

3 Simulating Switch States in NetKAT

In standard NetKAT, a switch has to treat all incoming packets according to the same policy. For example, it cannot count how many packets of a certain type it has seen before. In this section we will demonstrate that we can reinterpret selected packet fields as switch states, which gives switches the ability to react differently, depending on previous packets. NetKAT can then express examples like dynamic gossip more naturally.

Standard NetKAT describes how packets travel through a network and it can describe the behaviour of multiple packets with the multicast interpretation of the + operator. However, each of these packets form their own network trace, and NetKAT does not allow any interaction between them.

Ideally, a global state would allow us to model interactions between packets via alterations of the global state. But this immediately yields questions about concurrency. Versions of Kleene algebra that treat concurrency can be found in [14] and [13], but to our knowledge a similar extension of KAT or NetKAT has not been found yet. For dynamic gossip as studied in [4] we do not actually need interaction between packets. It suffices to use a single packet, because in standard Dynamic Gossip no two phone calls can take place at the same time. Hence we can avoid concurrency questions by linking the global state of all switches to the current packet.

We have to change the intended meaning of the + operator in order to reinterpret selected fields as switch states. In standard NetKAT the + operator can be understood as multicasting. However, it is not realistic that a switch is in multiple states at the same time. Therefore, in NetKAT with switch states the + operator should be understood as non-deterministic choice.

For clarity, we introduce a new piece of syntax to separate the standard packet fields from the switch state fields.

Definition 2. *The syntax of NetKAT with switch states extends the one of NetKAT as follows. For f as in Definition 1, n a packet field value or a switch state from a finite domain and i a switch identifier:*

$$a ::= 1 \mid 0 \mid f = n \mid \mathsf{state}(i) = n \mid a + b \mid a \cdot b \mid \neg a$$
$$p ::= a \mid f \leftarrow n \mid \mathsf{state}(i) \leftarrow n \mid p + q \mid p \cdot q \mid p^*$$

We thus add two new operators: the state test $\mathsf{state}(i) = n$ and the state modification $\mathsf{state}(i) \leftarrow n$. They work similarly to normal tests and modifications but act specifically on the switch state fields.

The semantics of the new variant of NetKAT is shown in Figure 3. In order to also highlight the reinterpretation of selected packet fields in the semantics, we use a *state vector* $\vec{s}$ that contains all the switch state fields. For the set of all state vectors we write $\vec{S}$. The interpretation of each policy p is thus a function $[\![p]\!] \colon P \times \vec{S} \to 2^{(P \times \vec{S})}$ where $P \times \vec{S}$ is the set of all tuples $\mathsf{ps} = (\mathsf{pk}, \vec{s})$ of a packet pk and a state vector $\vec{s}$. Note the similarity to Figure 1: We only changed the underlying set from P to $P \times \vec{S}$ and added definitions for tests and assignments on the state vector.

Example 2. Consider policy $(\mathsf{sw} \leftarrow A \cdot \mathsf{state}(A) \leftarrow 1) + (\mathsf{sw} \leftarrow B \cdot \mathsf{state}(B) \leftarrow 2)$. Applied to $(\mathsf{pk}, \vec{s})$, this policy outputs $\{(\mathsf{pk}[\mathsf{sw} := A], \vec{s}[A := 1]), (\mathsf{pk}[\mathsf{sw} := B], \vec{s}[B := 2])\}$. Hence, state vectors denote the state of the network as a result of how the corresponding packet was processed by the policy.

Note that in Example 2 the packet first moves to a switch before modifying its state. The semantics of NetKAT with switch states in principle allows us to

change the state of a switch without the packet actually being there at this moment. Such policies are unrealistic. We therefore restrict ourselves to policies that only modify the state of switches when they are there.

Definition 3 (Topology Respecting). *We say that a policy is* topology respecting *iff it is equivalent to its* localised version *which is obtained by replacing every* $\mathsf{state}(x) \leftarrow n$ *with* $\mathsf{sw} = x \cdot \mathsf{state}(x) \leftarrow n$.

The axioms of NetKAT with switch states are the same as for standard NetKAT, except that we add axioms for state tests and state modifications. These are exactly the same as those shown in Figure 2 with $\mathsf{state}(i)$ replacing f.

It is easy to see that NetKAT with switch states can be translated back to standard NetKAT. This allows us to use the soundness and completeness of standard NetKAT to argue that NetKAT with switch states is sound and complete. Intuitively, the translation relies on the fact that the packet and state vector are always paired together. Hence we can simply map the state vector to extra fields w_i for each switch i.

Definition 4. *Let* $m\colon (P \times \vec{S}) \to P$ *be the translation defined by*

$$m\left(\{f_1 = v_1, \ldots, f_n = v_n\}, [s_1, \ldots, s_n]\right) := \left\{ \begin{array}{l} f_1 = v_1, \ldots, f_n = v_n, \\ w_1 = s_1, \ldots, w_n = s_n \end{array} \right\}$$

and let m^{-1} *denote its inverse. Let* t *map a Net*KAT *expression with* state *to a standard Net*KAT *expression, proceeding by recursion for compositional policies and mapping policies concerning the state vector as follows:*

$$t(\mathsf{state}(i) = n) := w_i = n \qquad t(\mathsf{state}(i) \leftarrow n) := w_i \leftarrow n$$

For all other atomic policies p, *let* $t(p) := p$.

$$\begin{aligned}
[\![1]\!](\mathsf{ps}) &:= \{\mathsf{ps}\} \\
[\![0]\!](\mathsf{ps}) &:= \varnothing \\
[\![\neg a]\!](\mathsf{ps}) &:= \{\mathsf{ps}\} \setminus ([\![a]\!](\mathsf{ps})) \\
[\![f = n]\!]((\mathsf{pk}, \vec{s})) &:= \begin{cases} \{(\mathsf{pk}, \vec{s})\} & \text{if } \mathsf{pk}.f = n \\ \varnothing & \text{otherwise} \end{cases} \\
[\![f \leftarrow n]\!]((\mathsf{pk}, \vec{s})) &:= \{(\mathsf{pk}[f := n], \vec{s})\} \\
[\![p + q]\!](\mathsf{ps}) &:= [\![p]\!](\mathsf{ps}) \cup [\![q]\!](\mathsf{ps}) \\
[\![p \cdot q]\!](\mathsf{ps}) &:= ([\![p]\!] \bullet [\![q]\!])(\mathsf{ps}) \\
[\![p^*]\!](\mathsf{ps}) &:= \textstyle\bigcup_{i \in \mathbb{N}} [\![p]\!]^i(\mathsf{ps}) \\
[\![\mathsf{state}(i) = n]\!]((\mathsf{pk}, \vec{s})) &:= \begin{cases} \{(\mathsf{pk}, \vec{s})\} & \text{if } \vec{s}(i) = n \\ \varnothing & \text{otherwise} \end{cases} \\
[\![\mathsf{state}(i) \leftarrow n]\!]((\mathsf{pk}, \vec{s})) &:= \{(\mathsf{pk}, \vec{s}[i := n])\}
\end{aligned}$$

Fig. 3. Semantics of NetKAT with switch states

Lemma 1. *For every policy p and every packet with state ps we have:*

$$[\![p]\!](\mathsf{ps}) = \{m^{-1}(\mathsf{pk}) \mid \mathsf{pk} \in [\![t(p)]\!](m(\mathsf{ps}))\}$$

Lemma 2. *For every two policies p and q we have:*

1. $[\![p]\!] = [\![q]\!]$ *if and only if* $[\![t(p)]\!] = [\![t(q)]\!]$ *and*
2. $p \equiv q$ *if and only if* $t(p) \equiv t(q)$.

Both lemmas can be proven by induction on the structure of policies, but as these proofs do not contain any new insights we omit them here.

Theorem 1 (Soundness and Completeness). *For all policies p and q of* NetKAT *with switch states we have $[\![p]\!] = [\![q]\!]$ if and only if $p \equiv q$.*

This finishes our explanation of NetKAT with switch states. In section 5 we will use it to give an intuitive formalisation of dynamic gossip.

4 Dynamic Gossip

Dynamic Gossip is an extension of the simpler gossip problem, also known as the telephone problem: A group of agents each has a secret. They can communicate via phone calls in which two agents exchange all the secrets they know. How many calls are needed, until all agents know all the secrets? This scenario was widely studied in the 1980s and a classic result is that for $n \geq 4$ agents $2n - 4$ phone calls are necessary and sufficient to distribute all secrets to everyone.

In the original setting every agent can call every other agent. Later studies removed this assumption and used a reachability graph to define which agents can communicate with each other. Different classes of graphs lead to different minimal numbers of calls. For a survey on classical static gossip, see [11].

Dynamic gossip from [7] is another variation of the problem: not only is there a reachability graph restricting who can call whom, but this graph is also manipulated when phone calls are made. Intuitively, the agents now also exchange phone numbers, in addition to the secrets. The dynamic gossip literature focuses on epistemic protocols [2, 8], which can be executed by a group of agents without a central authority. The prime example of such a protocol is "Learn New Secrets", short LNS. It allows agent a to call agent b if and only if a knows the number of b but does not know the secret of b.

In the remainder of this section we briefly state the basic definitions and a main result about the dynamic gossip problem.

Definition 5 (Gossip Graph). *Given a finite set of agents A, a* gossip graph *G is a triple (A, N, S) where N and S are binary relations over A such that $I \subseteq S \subseteq N$ where I is the identity relation on A. We write N_a as an abbreviation for $\{b \mid (a, b) \in N\}$. We abbreviate $(a, b) \in N$ with Nab. An* initial gossip graph *is a gossip graph where $S = I$. The set of all initial gossip graphs is denoted by $\mathcal{G}$. A gossip graph is called* total *if $S = A \times A$.*

The relations model the basic knowledge of the agents. We say that agent a *knows the number* of b iff Nab and that a *knows the secret* of b iff Sab. Hence a *total* gossip graph is one in which everyone knows all secrets.

Definition 6 (Possible Call; Call Execution). *A* call *is an ordered pair of agents $(a, b) \in (A \times A)$. We usually write ab instead of (a, b). Given a gossip graph $G = (A, N, S)$, a call ab is* possible *iff Nab. Given a possible call ab, G^{ab} is the graph (A', N', S') such that $A' := A$, $N'_a := N'_b := N_a \cup N_b$, $S'_a := S'_b := S_a \cup S_b$, and $N'_c := N_c$, $S'_c := S_c$ for $c \neq a, b$. For a sequence of calls $ab; cd; \ldots$ from the set $(A \times A)^*$ we write σ. The empty sequence is ϵ. We extend the notation G^{ab} to sequences of calls: $G^{\epsilon} := G$, $G^{\sigma;ab} := (G^{\sigma})^{ab}$.*

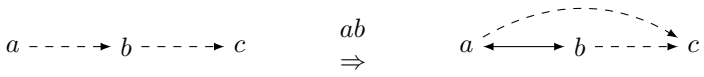
Definition 7 (LNS). *The* Learn New Secrets protocol *is defined as follows. Given a gossip graph $G = (A, N, S)$, the* set of LNS-allowed calls *is:*

$$lns(G) := \{(a, b) \in A \times A \mid Nab \text{ and not } Sab \text{ in } G\}$$

The extension of LNS *on G is the set of call sequences defined recursively by*

$$LNS(G) := \begin{cases} \{ab; \sigma \mid (a, b) \in lns(G) \text{ and } \sigma \in LNS(G^{ab})\} & \text{if } lns(G) \neq \varnothing \\ \{\epsilon\} & \text{otherwise} \end{cases}$$

Example 3. Consider an initial gossip graph G for three agents in which a knows the number of b, and b knows the number of c. Suppose that a calls b. We then obtain the gossip graph G^{ab} in which a and b know each other's secret and a now also knows the number of c. We can visualise the two graphs as follows, with dashed lines for N and solid lines for S:



There are three LNS sequences on G, namely $ab; ac; bc$, $ab; bc; ac$ and $bc; ab$. The first two sequences are successful, i.e. they lead to a total graph in which everyone knows all three secrets. The shorter sequence $bc; ab$ is stuck: no more calls are allowed according to LNS but not everyone knows all the secrets yet.

The gossip graph in Example 3 is a case in which LNS is only weakly, but not strongly successful, which we define as follows.

Definition 8 (Weak and Strong Success). *Consider a gossip graph G.*

1. *We say that LNS is* weakly successful *on G if and only if there is a call sequence $\sigma \in LNS(G)$ such that G^{σ} is total.*
2. *We say that LNS is* strongly successful *on G if and only if for all call sequences $\sigma \in LNS(G)$ we have that G^{σ} is total.*

We conclude this section with a definition and theorem from [7].

Definition 9 (Sun Graphs). *An initial gossip graph G is a* sun graph *if and only if N is strongly connected on the restriction of G to non-terminal nodes (nodes with at least one out-going edge).*

Theorem 2 (Theorem 20 in [7]). *Let G be an initial gossip graph. LNS is strongly successful on G if and only if G is a sun graph.*

5 Dynamic Gossip in NetKAT

In this section we will discuss how to translate gossip to NetKAT. We will build a NetKAT network and input packet that together represent a gossip graph. Then we will construct a policy that performs the LNS protocol.

The switches in NetKAT will represent the gossiping agents, and the state of a switch will be the list of phone numbers and secrets this agent knows. We stress that the NetKAT network does *not* describe the gossip graph. Instead, we ensure that during the execution of LNS each agent can in principle communicate with every other agent by using a totally connected NetKAT network.

The numbers and secrets an agent knows are part of the local state of the corresponding switch. Instead of encoding the N and S relation into one integer field $\mathsf{state}(i)$ for each switch i, we simplify notation and use multiple NetKAT fields Nij and Sij with values 0 and 1 to describe the state of i. For example, the field $N12$ is part of the local state of switch 1 and $Nab = 1$ means that agent a knows the phone number of agent b. We will ensure that the gossip policies will be topology respecting in the sense of Definition 3 applied to all fields Nij and Sij belonging to switch i, instead of only one field $\mathsf{state}(i)$.

In addition to the Sij and Nij fields, the input packet will have fields for the location of the packet, sw and pt, and fields call_m for denoting what call took place in round m.

Definition 10 (NetKAT network and graph packet). *Consider a group A of $n := |A|$ gossiping agents. The NetKAT network $\mathcal{N}_n$ for n agents is a fully connected network of n switches, each of which has an additional port home_i that is not connected to any other switch. Suppose we have a gossip graph $G = (A, N, S)$. The NetKAT input packet pk_G describes the S and N relations as follows. The call_m fields represent call rounds and we will explain them later.*

$$\begin{aligned}\mathsf{pk}_G := \{\mathsf{sw} = 0, \mathsf{pt} = 0\} \cup \{\mathsf{call}_m = 0 \mid m \in \{0, \ldots, n(n-1)\}\}\\ \cup \{Nij = 1 \mid (i,j) \in N\} \cup \{Nij = 0 \mid (i,j) \notin N\}\\ \cup \{Sij = 1 \mid (i,j) \in S\} \cup \{Sij = 0 \mid (i,j) \notin S\}\end{aligned}$$

We will now describe a NetKAT policy that represents the LNS protocol. On an input packet representing a gossip graph $G = (A, N, S)$, this policy should output packets corresponding to call sequences σ and the Nij and Sij fields should describe N^σ and S^σ. We want an output for each $\sigma \in LNS(G)$.

The policy describing LNS starts by distributing the initial input packet to every agent's home port — a private port not connected to any other agent. This is necessary to ensure that any agent can make the first call.

The next part of the policy will describe all LNS-allowed calls. In particular, a call from a to b is defined in policy pol_{ab} and describes a packet moving from the caller to the callee and back. We first check that the LNS conditions are satisfied. If so, the call takes place by moving the packet back and forth and updating the knowledge of the agents when the packet is located at the corresponding switch. We also keep track of the call sequences by overwriting the field call_k in round k, up to round $n(n-1)$. This is a safe upper bound on the number of calls,

because no call happens more than once in LNS. After a call has taken place, the resulting packet is again distributed to all home ports to ensure that any other agent can initiate the next call. Finally, we use the Kleene star to iterate the whole procedure.

Definition 11 (Gossip Policies). *Let A be the set of all agents and let $n := |A|$. Consider the NetKAT network $\mathcal{N}_n$ for n agents. For all $a, b \in A$, let $link_{ab}$ denote the port of agent a connected to agent b, and for all $i \in A$ let $home_i$ be the home port of agent i. For distributing the packet, we define a policy:*

$$pol_{\text{dstr}} := \sum_{i=1}^{n} (\mathsf{sw} \leftarrow i \cdot \mathsf{pt} \leftarrow home_i)$$

Suppose f is a field which takes values 0 and 1. We use the following NetKAT abbreviation for "if ... then ..." programs: $(\text{IF } f \text{ THEN } p) := (f = 1 \cdot p) + (f = 0)$. For each call (a, b) we define a policy:

$$\begin{aligned} pol_{ab} := {} & \mathsf{sw} = a \ \cdot \ \mathsf{pt} = home_a \ \cdot \ Nab = 1 \ \cdot \ Sab = 0 \\ & \cdot \ \mathsf{pt} \leftarrow link_{ab} \ \cdot \ \mathsf{sw} \leftarrow b \ \cdot \ \mathsf{pt} \leftarrow link_{ba} \\ & \cdot \ \textstyle\prod_{x \in \mathcal{A}} (\text{IF } Sax \text{ THEN } Sbx \leftarrow 1) \ \cdot \ \prod_{x \in \mathcal{A}} (\text{IF } Nax \text{ THEN } Nbx \leftarrow 1) \\ & \cdot \ \mathsf{sw} \leftarrow a \ \cdot \ \mathsf{pt} \leftarrow link_{ab} \\ & \cdot \ \textstyle\prod_{x \in \mathcal{A}} (\text{IF } Sbx \text{ THEN } Sax \leftarrow 1) \ \cdot \ \prod_{x \in \mathcal{A}} (\text{IF } Nbx \text{ THEN } Nax \leftarrow 1) \\ & \cdot \ \textstyle\sum_{k=1}^{n(n-1)} \left(\prod_{y<k} (\neg(\mathsf{call}_y = 0)) \ \cdot \ \mathsf{call}_k = 0 \ \cdot \ \mathsf{call}_k \leftarrow ab \right) \end{aligned}$$

To make any LNS call, we define the policy $pol_{\text{lns}} := \sum_{i \in A} \left(\sum_{j \in A \setminus \{i\}} pol_{ij} \right)$. For the whole LNS protocol, let $pol_{\text{LNS}} := (pol_{\text{dstr}} \cdot pol_{\text{lns}})^*$ and for any sequences of LNS calls $\sigma = c_1; \ldots; c_k$, let $pol_\sigma := pol_{\text{dstr}} \ \cdot \ pol_{c_1} \ \cdot \ \ldots \ \cdot \ pol_{\text{dstr}} \ \cdot \ pol_{c_k}$.

The pol_{LNS} policy only depends on the number of agents, and not on the gossip graph. It is topology respecting because we only update those N and S fields of agents that are emulated by the switches where the packet is at that moment of evaluation.

We now get the following correspondence between the original definition of G^σ and the result of applying pol_σ to pk_G.

Lemma 3. *Consider a gossip graph $G = (A, N, S)$, the corresponding pk_G, any LNS sequence of calls σ and the resulting graph $G^\sigma = (A, N^\sigma, S^\sigma)$. Then $[\![pol_\sigma]\!](\mathsf{pk}_G)$ describes G^σ, i.e. we have:*

$$N^\sigma = \{(a, b) \mid \forall \mathsf{pk} \in [\![pol_\sigma]\!](\mathsf{pk}_G) : \mathsf{pk}.Nab = 1\}$$

$$S^\sigma = \{(a, b) \mid \forall \mathsf{pk} \in [\![pol_\sigma]\!](\mathsf{pk}_G) : \mathsf{pk}.Sab = 1\}$$

In fact, pol_σ is such that the output is always exactly one packet.

Lemma 3 only talks about a single sequence, but we can lift it to the whole LNS protocol as follows. The LNS call sequences for a gossip graph G are the same

call sequences as those generated by applying pol_{LNS} to pk_G and the distribution of knowledge in the output packets of pol_{LNS} is the same as in the gossip graphs resulting from the same call sequences (this latter fact follows directly from Lemma 3).

Let pol_{stop} be a policy checking whether the LNS protocol has finished, i.e. testing whether for every two agents i and j we have either $Sij = 1$ or $Nij = 0$: $pol_{\text{stop}} := \prod_{i,j \in A}(Sij = 1 + Nij = 0)$.

Theorem 3. *Definition 7 and Definition 11 agree with each other. Formally, for every gossip graph $G = (A, N, S) \in \mathcal{G}$ we have:*

$$LNS(G) = \{c_0; \ldots; c_k \in (A \times A)^* \mid \exists \mathsf{pk} \in [\![pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk}_G) : \\ \forall m \leq k : \mathsf{pk}.\mathsf{call}_m = c_m \text{ and } \mathsf{pk}.\mathsf{call}_{k+1} = 0\}$$

Proof. We first show $\subseteq$. Suppose an LNS call sequence σ is a result of the LNS protocol on gossip graph G. We now prove by induction on the length of σ that this call sequence is also happening through pol_{LNS}. In case σ is ϵ, it means no calls were allowed according to LNS for gossip graph G. The initial input pk_G to pol_{LNS} will resemble this, and thus no calls happen through pol_{LNS} either.

For the induction step, we use Lemma 3 as follows. Suppose we have a sequence $xy; \sigma \in LNS(G)$ and σ consists of k calls. Then by Definition 7, we know that $\sigma \in LNS(G^{xy})$. By the induction hypothesis we then know that there exists a packet $\mathsf{pk} \in [\![pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk}_{G^{xy}})$ such that for all $m \leq k-1$ we have $\mathsf{pk}.\mathsf{call}_m = c_m$. Moreover, $\mathsf{pk} \in [\![pol_\sigma]\!](\mathsf{pk}_{G^{xy}})$. From Lemma 3 we know that $N^{xy} = \{(a,b) \mid \forall \mathsf{pk} \in [\![pol_{xy}]\!](\mathsf{pk}_G) : \mathsf{pk}.Nab = 1\}$ and $S^{xy} = \{(a,b) \mid \forall \mathsf{pk} \in [\![pol_{xy}]\!](\mathsf{pk}_G) : \mathsf{pk}.Sab = 1\}$. Thus we can conclude that $\mathsf{pk}_{G^{xy}} \in [\![pol_{xy}]\!](\mathsf{pk}_G)$, as $\mathsf{pk}_{G^{xy}}$ is the packet resembling gossip graph G^{xy} and thus such that it corresponds to N^{xy} and S^{xy}. This gives us a $\mathsf{pk}' \in [\![pol_{xy;\sigma}]\!](\mathsf{pk}_G)$ where pk' is the same as packet pk except that pk' will have one more call field. The values of the call fields are such that they match $xy; \sigma$ in the sense that $c_0 = xy$ and $c_1; \ldots; c_k = \sigma$. Moreover, $xy; \sigma$ is a finished LNS sequence. Hence we have $\mathsf{pk}' \in [\![pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk}_G)$ such that for all $m \leq k$ we have $\mathsf{pk}'.\mathsf{call}_m = c_m$.

For the converse $\supseteq$, we again proceed by induction on the length of call sequences. For the base case, if pk_G matches graph G and no calls are allowed to take place by pol_{LNS}, then also $LNS(G)$ will be empty.

For the induction step, take any outcome $\mathsf{pk} \in [\![pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk}_G)$ that is the result of $k+1$ iterations inside pol_{LNS}. Then we get a call sequence $c_0; \ldots; c_k$ such that for all $m \leq k$ we have that $\mathsf{pk}.\mathsf{call}_m = c_m$. Let us say that $c_0 = xy$ and denote $c_0; \ldots; c_k$ with $xy; \sigma$. In other words, pk is the result of policy $pol_{xy;\sigma}$ applied to pk_G where $xy; \sigma$ is a finished LNS sequence on G. Similar to what we did before, we can conclude from Lemma 3 that $\mathsf{pk}_{G^{xy}} \in [\![pol_{xy}]\!](\mathsf{pk}_G)$. Hence, $\mathsf{pk}' \in [\![pol_\sigma]\!](\mathsf{pk}_{G_{xy}})$ where pk' is the same as pk except that pk' has one less call field as pk and the call fields of pk' correspond to σ which is a finished call sequence on G^{xy}. We then get that $\mathsf{pk}' \in [\![pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk}_{G^{xy}})$ such that there is a sequence $\sigma = c_0; \ldots; c_k$ where for all $m \leq k$ we have that $\mathsf{pk}'.\mathsf{call}_m = c_m$. By our induction hypothesis we can now conclude that $\sigma \in LNS(G^{xy})$. From Definition 7

we know that to show that $xy;\sigma \in LNS(G)$ we need to have that $(x,y) \in lns(G)$. This holds because $[\![pol_{xy}]\!](\mathsf{pk}_G)$ was nonempty and pk_G corresponds to G. □

This connection between the standard gossip definitions and our definitions in NetKAT might seem obvious to the reader — of course we defined the policies exactly to get this correspondence. Theorem 3 is still useful because it means that we can use NetKAT to compute all LNS call sequences and to check whether LNS is successful, with the following translation of Definition 8 to NetKAT.

Definition 12 (Graphs and Success in NetKAT). *Given a gossip graph G, we define a policy pol_G to check whether the current packet encodes G:*

$$pol_G := \begin{pmatrix} \prod\{Nij = 1 \mid Nij \text{ in } G\} \cdot \prod\{Nij = 0 \mid \text{not } Nij \text{ in } G\} \cdot \\ \prod\{Sij = 1 \mid Sij \text{ in } G\} \cdot \prod\{Sij = 0 \mid \text{not } Sij \text{ in } G\} \end{pmatrix}$$

We also define a policy to test whether LNS has been successful, i.e. whether the S relation encoded in a given packet is total: $pol_{\text{success}} := \prod_{i,j \in A}(Sij = 1)$.

Theorem 4. *The LNS protocol is weakly successful on gossip graph G if and only if the following NetKAT equivalence holds:*

$$pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{success}} \not\equiv 0$$

Similarly, we can reduce the check whether LNS is strongly successful to NetKAT as follows. See the appendix for proofs of Theorem 4 and Theorem 5.

Theorem 5. *The LNS protocol is strongly successful on gossip graph G if and only if the following NetKAT equivalence holds:*

$$pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{stop}} \equiv pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{success}}$$

We can also translate the notion of a sun graph from Definition 9 to NetKAT. For this we need to express that an agent has an N-path to every other agent.

Definition 13 (N-paths in NetKAT). *For any set of agents A, we define the following two policies:*

$$pol_{\text{num}} := \left(\sum_{x \in A} \sum_{y \in A \setminus \{x\}} \begin{pmatrix} \mathsf{sw} = x \cdot \mathsf{pt} = home_x \cdot Nxy = 1 \cdot \mathsf{pt} \leftarrow link_{xy} \\ \cdot \mathsf{sw} \leftarrow y \cdot \mathsf{pt} \leftarrow link_{yx} \cdot \mathsf{pt} \leftarrow home_y \end{pmatrix} \right)^*$$

$$pol_{Ni} := \prod_{j \in A \setminus \{i\}} (\mathsf{sw} = i \cdot \mathsf{pt} = home_i \cdot pol_{\text{num}} \cdot \mathsf{sw} = j \cdot \mathsf{sw} \leftarrow i \cdot \mathsf{pt} \leftarrow home_i)$$

Lemma 4. *Consider a gossip graph $G = (A, N, S)$. An agent $i \in A$ has an N-path in to every other agent if and only if we have $pol_G \cdot pol_{Ni} \not\equiv 0$.*

The policy pol_{num} distributes a packet following the N-relation between any two agents. We prepend and append it with location tests to check whether we can go from agent i to another agent j. As we want an N-path to every other agent in the network, we take a product over all agents. If this does not return the empty set, we know that agent i has an N-path to every agent in the network.

Theorem 6 (Sun Graphs in NetKAT). *A gossip graph G is a sun graph if and only if the following NetKAT equivalence holds:*

$$pol_G \cdot \prod_{i \in A} (\mathsf{sw} \leftarrow i \cdot \mathsf{pt} \leftarrow home_i \cdot (pol_{Ni} + pol_{\mathrm{self}_i})) \not\equiv 0$$

where pol_{self_i} is a policy checking whether agent i only has its own number.

The proof is straightforward and can be found in the appendix.

Corollary 1. *For every gossip graph G, "LNS is strongly successful on G if and only if G is a sun" can be expressed and proven using NetKAT equivalences.*

Admittedly, this corollary is the easy direction: By Theorem 2, Theorem 6 and the completeness of NetKAT, we know that for each G there *exists* a proof of Theorem 2 in NetKAT — see appendix. We leave the more interesting and challenging task as future work: to actually find these algebraic proofs. Moreover, it would be interesting to find a proof on the meta-level by working with schemata of NetKAT expressions instead of a specific G. This could yield a completely new proof of Theorem 2 or one can try to translate the proof from [7] to NetKAT.

6 Implementation

The embedding of dynamic gossip into NetKAT allows us to reduce decision problems about gossip graphs to NetKAT equivalence checking. We implemented the methods described in this paper in *Haskell*. The code is available at `https://github.com/janawagemaker/GossipKATS` and can be used in three ways.

1. Gossip-only: We provide a direct and explicit implementation of dynamic gossip, using custom data types in Haskell. A similar implementation is described in [20]. These methods do not use NetKAT at all and we only use them as a reference to check other methods for correctness.
2. Explicit-NetKAT: Also in Haskell, we implemented the packet-processing model for NetKAT, Definition 10 and Definition 11. We thus take a gossip graph G and generate the corresponding packet pk_G. We then apply the LNS policy on this packet. The result is a set of packets from which we can obtain $LNS(G)$ by reading the call_k fields.
3. Equivalence-NetKAT: If we are not interested in the call sequences, but only in whether LNS is successful on a given graph, then we can use Theorem 4 and Theorem 5. Given a gossip graph, we generate the NetKAT equivalence that holds iff LNS is weakly or strongly successful on it. We then pass this statement to the implementation of [10], a coalgebraic decision procedure for NetKAT equivalence. The implementation we use is part of the general network programming framework *frenetic* available at `https://github.com/frenetic-lang/frenetic`.[3]

[3] To be precise, we use the `verification_and_felix` branch currently at commit `be47c929ed84904f9bdb81bf9765a0432db63069`. We would like to thank Steffen Smolka and Nate Foster for their help to get the decision method running.

All three methods are fully automated, i.e. the user only needs to input an initial gossip graph. We also provide automated tests that randomly generate gossip graphs to verify that the methods agree.

Unfortunately, the third method is currently too slow for interesting examples and the implementation is mainly a proof of work. We hope to improve it in the near future. In particular, we plan to switch to a symbolic decision method as discussed in [18] and currently being developed by the authors of [19].

7 Related Work

The idea of NetKAT with additional state is not new: another version of it is discussed in [17]. However, the system developed there is no longer a KAT and hence cannot base a soundness and completeness proof on the corresponding proofs for KAT. Such a proof is also not given via a different route. Our system is much less expressive but still sound and complete. Also related to our work is the temporal version of NetKAT presented in [6]. The authors add temporal operators to inspect histories of packets. It also seems possible to embed the gossip problem into temporal NetKAT, but we expect this to be less intuitive.

Static and dynamic gossip has mainly been studied in the logic community, with a focus on how the primitive and higher-order knowledge of gossiping agents develops [3, 12]. Some of these works also use formal languages and define logics for gossip. For example, in [4] action models of Dynamic Epistemic Logic are used to describe the effects of different gossip calls. This also yields an axiomatization via standard reduction axioms. However, the action models and axioms are of size exponential in the number of agents. This makes an axiomatization of gossip via action models impractical. Another language for the static gossip problem based on Propositional Dynamic Logic is studied in [9]. It is used to distinguish different variants of the gossip problem, but no axiomatization is given.

8 Conclusion

We have seen that switch states can be simulated in NetKAT without losing soundness and completeness. This reinterpretation of NetKAT provides a natural framework to describe dynamic gossip. We translated the LNS protocol to a NetKAT policy, and the definitions of strong/weak success and sun graphs. With these translations we can answer questions about gossip graphs by deciding NetKAT equivalences. By completeness we know that any question about gossip that is expressible in NetKAT can be decided this way. As mentioned above, we hope to find algebraic proofs and to improve the performance of our implementation in the future.

We also plan to model social influence and diffusion phenomena in NetKAT. They are often similar to dynamic gossip and have been modelled in dynamic epistemic logics, see for example [5]. A crucial difference however, is that social influence settings are, in contrast to dynamic gossip, not monotone: agents do not forget secrets or phone numbers, but they can change their behaviour and

influence back and forth. Describing the fixpoints in these settings can thus be a challenge. We think that NetKAT is a suitable language to formalise such non-monotone phenomena, given that its fixpoints are allowed to exist of sets of packets describing multiple outcomes.

Acknowledgements We received helpful feedback from Jan van Eijck, Tobias Kappé, Jurriaan Rot, Jan Rutten and the anonymous RAMiCS reviewers. The first author was affiliated with the ILLC at the University of Amsterdam during most of this work. The research of the second author is funded by the Dutch NWO project 612.001.210.

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: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '14. Extended version at `https://hdl.handle.net/1813/34445`. 2014, pages 113–126. ISBN: 978-1-4503-2544-8. DOI: `10.1145/2535838.2535862`.

[2] Krzysztof R. Apt, Davide Grossi, and Wiebe van der Hoek. "Epistemic Protocols for Distributed Gossiping". In: *Proceedings of TARK 2015*. Edited by Ramanujam. 2015. DOI: `10.4204/EPTCS.215.5`.

[3] Krzysztof R. Apt and Dominik Wojtczak. "Common Knowledge in a Logic of Gossips". In: *Proceedings of TARK 2017*. Edited by Jérôme Lang. 2017. DOI: `10.4204/EPTCS.251.2`.

[4] Maduka Attamah, Hans van Ditmarsch, Davide Grossi, and Wiebe van der Hoek. "Knowledge and Gossip". In: *Proceedings of the Twenty-first European Conference on Artificial Intelligence*. Frontiers in Artificial Intelligence and Applications. 2014, pages 21–26. ISBN: 978-1-61499-418-3. DOI: `10.3233/978-1-61499-419-0-21`.

[5] Alexandru Baltag, Zoé Christoff, Rasmus K. Rendsvig, and Sonja Smets. "Dynamic Epistemic Logics of Diffusion and Prediction in Social Networks". In: *Proceedings of the Twelfth Conference on Logic and the Foundations of Game and Decision Theory*. 2016. URL: `https://is.gd/DiffDEL`.

[6] Ryan Beckett, Michael Greenberg, and David Walker. "Temporal NetKAT". In: *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '16. 2016, pages 386–401. ISBN: 978-1-4503-4261-2. DOI: `10.1145/2908080.2908108`.

[7] Hans van Ditmarsch, Jan van Eijck, Pere Pardo, Rahim Ramezanian, and François Schwarzentruber. "Dynamic Gossip". In: *Bulletin of the Iranian Mathematical Society* (Sept. 2018). DOI: `10.1007/s41980-018-0160-4`.

[8] Hans van Ditmarsch, Jan van Eijck, Pere Pardo, Rahim Ramezanian, and François Schwarzentruber. "Epistemic protocols for dynamic gossip". In: *Journal of Applied Logic* 20 (2017), pages 1–31. DOI: `10.1016/j.jal.2016.12.001`.

[9] Hans van Ditmarsch, Davide Grossi, Andreas Herzig, Wiebe van der Hoek, and Louwe B. Kuijer. "Parameters for Epistemic Gossip Problems". In: *Proceedings of the Twelfth Conference on Logic and the Foundations of Game and Decision Theory*. 2016. URL: https://is.gd/GosPar.

[10] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. "A Coalgebraic Decision Procedure for NetKAT". In: *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '15. 2015, pages 343–355. ISBN: 978-1-4503-3300-9. DOI: 10.1145/2676726.2677011.

[11] Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. "A survey of gossiping and broadcasting in communication networks". In: *Networks* 18.4 (1988), pages 319–349. DOI: 10.1002/net.3230180406.

[12] Andreas Herzig and Faustine Maffre. "How to share knowledge by gossiping". In: *AI Communications* 30.1 (2017), pages 1–17. DOI: 10.3233/AIC-170723.

[13] Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. "Concurrent Kleene Algebra and its Foundations". In: *The Journal of Logic and Algebraic Programming* 80.6 (2011). Relations and Kleene Algebras in Computer Science, pages 266–296. DOI: 10.1016/j.jlap.2011.04.005.

[14] Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. "Concurrent Kleene Algebra: Free Model and Completeness". In: *Programming Languages and Systems (ESOP 2018)*. Edited by Amal Ahmed. 2018, pages 856–882. ISBN: 978-3-319-89884-1. DOI: 10.1007/978-3-319-89884-1_30. URL: https://arxiv.org/abs/1710.02787.

[15] Dexter Kozen. "Kleene Algebra with Tests". In: *ACM Transactions on Programming Languages and Systems* 19.3 (May 1997), pages 427–443. DOI: 10.1145/256167.256195.

[16] Dexter Kozen and Frederick Smith. "Kleene algebra with tests: Completeness and decidability". In: *Computer Science Logic*. Edited by Dirk van Dalen and Marc Bezem. 1997, pages 244–259. ISBN: 978-3-540-69201-0. DOI: 10.1007/3-540-63172-0_43.

[17] Jedidiah McClurg, Hossein Hojjat, Nate Foster, and Pavol Černý. "Event-driven Network Programming". In: *ACM SIGPLAN Notices*. PLDI 2016 51.6 (June 2016), pages 369–385. DOI: 10.1145/2908080.2908097.

[18] Damien Pous. "Symbolic Algorithms for Language Equivalence and Kleene Algebra with Tests". In: *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '15. 2015, pages 357–368. ISBN: 978-1-4503-3300-9. DOI: 10.1145/2676726.2677007.

[19] Steffen Smolka, Spiridon Eliopoulos, Nate Foster, and Arjun Guha. "A Fast Compiler for NetKAT". In: *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*. ICFP 2015. 2015, pages 328–341. ISBN: 978-1-4503-3669-7. DOI: 10.1145/2784731.2784761.

[20] Jana Wagemaker. "Gossip in NetKAT". Master's thesis. ILLC, University of Amsterdam, 2017. URL: https://eprints.illc.uva.nl/1552/.

Appendix

Proof of Lemma 3

By induction on the call sequence σ. The base case follows instantly as $pol_\epsilon = 1$ outputs the same packet and thus the same gossip graph.

For the induction step we use that calling agents exchange everything they know and that this is encoded in the modifications done by pol_{ab}. As an example, consider the $\subseteq$ direction for N. By the induction hypothesis for σ we have

$$N^\sigma \subseteq \{(a,b) \mid \forall \mathsf{pk} \in [\![pol_\sigma]\!](\mathsf{pk}_G) : \mathsf{pk}.Nab = 1\}$$

and want to show for $\sigma; xy$ that

$$N^{\sigma;xy} \subseteq \{(a,b) \mid \forall \mathsf{pk} \in [\![pol_{\sigma;xy}]\!](\mathsf{pk}_G) : \mathsf{pk}.Nab = 1\}$$

Suppose $(a,b) \in N^{\sigma;xy}$. Either $(a,b) \in N^\sigma$, in which case we are done by the induction hypothesis because $\mathsf{pk}.Nab = 1$ is preserved by pol_{xy}, or $(a,b) \notin N^\sigma$. For the latter case, w.l.o.g. we can assume that $a = x$. Thus we know that $(y,b) \in N^\sigma$. From our induction hypothesis we can conclude that $\forall \mathsf{pk} \in [\![pol_\sigma]\!](\mathsf{pk}_G) : \mathsf{pk}.Nyb = 1$. Then in pol_{xy} the field Nxb gets set to 1. Hence we know that $\forall \mathsf{pk} \in [\![pol_{\sigma;xy}]\!](\mathsf{pk}_G) : \mathsf{pk}.Nxb = 1$. □

Proof of Theorem 4

Using soundness and completeness of NetKAT the given syntactic equivalence holds iff we semantically have $[\![pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{success}}]\!] \neq \varnothing$.

Suppose LNS is weakly successful on G. Let us consider input packet pk_G from Definition 10. We have $[\![pol_G]\!](\mathsf{pk}_G) = \{pk_G\}$ by definition. There is at least one call sequence $\sigma \in LNS(G)$ that is successful. By Theorem 3 we know that σ corresponds to an execution of pol_{LNS} on input pk_G and there is a packet pk such that $\mathsf{pk} \in [\![pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk}_G)$ and its fields call_m encode σ. Because σ is successful we also have $[\![pol_{\text{success}}]\!](\mathsf{pk}) = \{\mathsf{pk}\}$. Hence we can conclude $\mathsf{pk} \in [\![pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{success}}]\!](\mathsf{pk}_G)$ and thus that $[\![pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{success}}]\!] \neq \varnothing$.

The other direction is similar, so we omit the proof here. □

Proof of Theorem 5

By soundness and completeness of NetKAT the equivalence holds iff we have $[\![pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!] = [\![pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{success}}]\!]$.

Suppose LNS is strongly successful on G. We immediately have $\supseteq$ because any packet passing the success check also passes the test that LNS has finished.

To show $\subseteq$, take any pk and pk' such that $\mathsf{pk}' \in [\![pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk})$. We then know that $\mathsf{pk} = \mathsf{pk}_G$ because pk passed the test pol_G. Hence $\mathsf{pk}' \in [\![pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk}_G)$. As $[\![pol_G]\!](\mathsf{pk}_G) = \{\mathsf{pk}_G\}$, we get that $\mathsf{pk}' \in [\![pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk}_G)$. From Theorem 3 we know that every output pk' of $[\![pol_{\text{LNS}} \cdot pol_{\text{stop}}]\!](\mathsf{pk}_G)$ corresponds to a call sequence $\sigma \in LNS(G)$. From the

assumption that LNS is strongly successful on G we get that all of these σ are successful call sequences. We thus know that $[\![pol_{\text{success}}]\!](\mathsf{pk}') = \{\mathsf{pk}'\}$ and thereby $\mathsf{pk}' \in [\![pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{success}}]\!](\mathsf{pk})$.

The other direction is similar. □

Proof of Theorem 6

By soundness and completeness of NetKAT, the statement is equivalent to:

$$\left[\!\!\left[pol_G \cdot \prod_{i \in A} (\mathsf{sw} \leftarrow i \cdot \mathsf{pt} \leftarrow home_i \cdot (pol_{Ni} + pol_{\text{self}_i})) \right]\!\!\right] \neq \varnothing$$

$\Rightarrow$ Take the packet pk_G. We know that $[\![pol_G]\!](\mathsf{pk}_G) = \{\mathsf{pk}_G\}$. As G is a sun graph, we know that for each agent i either $[\![pol_{Ni}]\!](\mathsf{pk}_G) = \{\mathsf{pk}_G\}$ or $[\![pol_{\text{self}_i}]\!](\mathsf{pk}_G) = \{\mathsf{pk}_G\}$. Hence we have

$$\left[\!\!\left[\prod_{i \in A} (\mathsf{sw} \leftarrow i \cdot \mathsf{pt} \leftarrow home_i \cdot (pol_{Ni} + pol_{\text{self}_i})) \right]\!\!\right] (\mathsf{pk}_G) = \{\mathsf{pk}\}$$

for some packet pk. Thus we can conclude:

$$\mathsf{pk} \in \left[\!\!\left[pol_G \cdot \prod_{i \in A} (\mathsf{sw} \leftarrow i \cdot \mathsf{pt} \leftarrow home_i \cdot (pol_{Ni} + pol_{\text{self}_i})) \right]\!\!\right] (\mathsf{pk}_G)$$

$\Leftarrow$ is similar. □

Proof of Corollary 1

Fix some G. By Theorem 5 LNS is strongly successful on G if and only if

$$pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{stop}} \equiv pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{success}}$$

and from Theorem 6 that G is a sun graph if and only if

$$pol_G \cdot \prod_{i \in A} (\mathsf{sw} \leftarrow i \cdot \mathsf{pt} \leftarrow home_i \cdot (pol_{Ni} + pol_{\text{self}_i})) \not\equiv 0$$

Now suppose we have $pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{stop}} \equiv pol_G \cdot pol_{\text{LNS}} \cdot pol_{\text{success}}$. From Theorem 5 we then know that LNS is strongly successful on G. By Theorem 2 it follows that G is a sun graph. From Theorem 6 we know this is equivalent to

$$pol_G \cdot \prod_{i \in A} (\mathsf{sw} \leftarrow i \cdot \mathsf{pt} \leftarrow home_i \cdot (pol_{Ni} + pol_{\text{self}_i})) \not\equiv 0$$

The other direction is similar. □