Explaining SDN Failures via Axiomatisations

Georgiana Caltais *
University of Konstanz, Germany
Georgiana.Caltais@uni-konstanz.de

This work introduces a concept of explanations with respect to the violation of safe behaviours within software defined networks (SDNs) expressible in NetKAT. The latter is a network programming language that is based on a well-studied mathematical structure, namely, Kleene Algebra with Tests (KAT). Amongst others, the mathematical foundation of NetKAT gave rise to a sound and complete equational theory. In our setting, a safe behaviour is characterised by a NetKAT policy which does not enable forwarding packets from ingress to an undesirable egress. Explanations for safety violations are derived in an equational fashion, based on a modification of the existing NetKAT axiomatisation.

1 Introduction

Explaining systems failure has been a topic of interest for many years now. Techniques such as Fault tree analysis (FTA) and Failure mode and effects analysis (FMEA) [5], for instance, have been proposed and widely used by reliability engineers in order to understand how systems can fail, and for debugging purposes.

In the philosophy of science there is a considerable amount of research on counterfactual causal reasoning which, ultimately, can be exploited in explaining system failures as well. A notion of causality that is frequently used in relation to technical systems relies on counterfactual reasoning. The results in [21] formulate the counterfactual argument, which defines when an event is considered a cause for some effect (or hazardous situation) in the following way: a) whenever the event presumed to be a cause occurs, the effect occurs as well, and b) when the presumed cause does not occur, the effect will not occur either. Nevertheless, this formulation of causality is considered too simple for explaining complex logical relationships leading to undesired situations. Consequently, the work in [15] introduces a notion of complex logical events based on boolean equation systems and proposes a number of conditions under which an event can be considered causal for an effect.

The seminal work in [15] has been adopted in various settings. Closely related to the explanation of failures based on adoptions of [15] are the results in [3, 20, 7], for instance. All these works adjust the definition of causality in [15] to the setting of system executions leading to a failure. The approach in [3] considers one such execution at a time, and uses counterfactual causal reasoning for identifying the points in the trace that are relevant for the failure. The results in [20, 7], on the other hand, aim at discovering the causal explanations for all failures in a system, and strongly rely on model-checking based techniques [2].

In this paper we focus on explaining violations of safe behaviours in software defined networks (SDNs). Software defined networking is an emerging approach to network programming in a setting where the network control is decoupled from the forwarding functions. This makes the network control directly programable, and more flexible to change. SDN proposes open standards such as the OpenFlow [22] API defining, for instance, low-level languages for handling switch configurations. Typically,

*This work was supported by the DFG project "CRENKAT", proj. no. 398056821.

Submitted to:
FROM 2019

this kind of hardware-oriented APIs are not intuitive to use in the development of programs for SDN platforms. Hence, a suite of network programming languages raising the level of abstraction of programs, and corresponding verification tools have been recently proposed [10, 26, 27]. It is a known fact that formal foundations can play an important role in guiding the development of programming languages and associated verification tools, in accordance with an intended semantics obeying essential (behavioural) laws. Correspondingly, the current paper is targeting NetKAT [1, 11] –a formal framework for specifying and reasoning about networks, integrated within the Frenetic suite of network management tools [10]. More precisely, we will exploit the sound and complete axiomatisation of NetKAT in [1] in order to derive explanations of safety failures in a purely equational fashion. It is well known that equational reasoning could alleviate the state explosion issue characteristic to model-checking, by equating terms equivalent modulo associativity, commutativity and indepotnecy, for instance.

Related to the current work, the results in [24] introduce a framework for automated failure localisation in NetKAT. The approach in [24] relies on the generation of test cases based on the network specification, further used to monitor the network traffic accordingly and localise faults whenever tests are not satisfied. In contrast, our approach falls under the umbrella of equivalence checking, and it provides an explanation for all possible failures, irrespective of particular input packets.

Our contributions. In this paper we introduce a concept of *safety in NetKAT* which, in short, refers to the impossibility of packets to travel from a given ingress to a specified hazardous egress. Then, we propose a notion of *safety failure explanation* which, intuitively, represents the set of finite paths within the network, leading to the hazardous egress. Eventually, we provide a modified version of the original axiomatisation of NetKAT exploited in order to *automatically compute the safety failure explanations*, if any. The axiomatisation employs a proposed *star-elimination construction* which enables the sound extraction of explanations from Kleene $*$-free NetKAT programs.

Structure of the paper. In Section 2 we provide an overview of NetKAT and the associated sound and complete axiomatisation. In Section 3 we define the concept of safety in NetKAT. In Section 4 we introduce the notion of safety failure explanation and the axiomatisation which can be exploited in order to compute such explanations. In Section 5 we draw the conclusions and pointers to future work.

2 Preliminaries

As pointed out in [1], a network can be interpreted as an automaton that forwards packets from one node to another along the links in its topology. This lead to the idea of using regular expressions –the language of finite automata–, for expressing networks. A path is encoded as a concatenation of processing steps ($p.q.\ldots$), a set of paths is encoded as a union of paths ($p+q+\ldots$) whereas iterated processing is encoded using Kleene $*$. This paves the way to reasoning about properties of networks using Kleene Algebra with Tests (KAT) [19]. KAT incorporates both Kleene Algebra [18] for reasoning about network structure and Boolean Algebra for reasoning about the predicates that define switch behaviour.

NetKAT *packets pk* are encoded as sets of fields f_i and associated values v_i as in Figure 1. *Histories* are defined as lists of packets, and are exploited in order to define the semantics of NetKAT policies/programs as in Figure 2. NetKAT *policies* are recursively defined as: predicates, field modifications $f \leftarrow n$, union of policies $p+q$ ($+$ plays the role of a multi-casting like operator), sequencing of policies $p.q$, repeated application of policies p^* (the Kleene $*$) and duplication **dup** (that saves the current packet at the beginning of the history list). *Predicates*, on the other hand, can be seen as filters. The constant predicate 0 drops all the packets, whereas its counterpart predicate 1 retains all the packets. The test

$$
\begin{array}{rrll}
\text{Fields} & f & ::= & f_1 \mid \cdots \mid f_k \\
\text{Packets} & pk & ::= & \{f_1 = v_1, \cdots, f_k = v_k\} \\
\text{Histories} & h & ::= & pk{::}\langle\rangle \mid pk{::}h \\
\text{Predicates} & a, b & ::= & \mathsf{1} \qquad \textit{Identity} \\
 & & \mid & \mathsf{0} \qquad \textit{Drop} \\
 & & \mid & f = n \quad \textit{Test} \\
 & & \mid & a + b \quad \textit{Disjunction} \\
 & & \mid & a \,.\, b \quad \textit{Conjunction} \\
 & & \mid & \neg a \quad \textit{Negation} \\
\text{Policies} & p, q & ::= & a \qquad \textit{Filter} \\
 & & \mid & f \leftarrow n \quad \textit{Modification} \\
 & & \mid & p + q \quad \textit{Union} \\
 & & \mid & p \,.\, q \quad \textit{Sequential composition} \\
 & & \mid & p^* \quad \textit{Kleene star} \\
 & & \mid & \mathsf{dup} \quad \textit{Duplication}
\end{array}
$$

Figure 1: NetKAT syntax [1]

$$
\begin{array}{rl}
[\![p]\!] & \in \mathsf{H} \to \mathcal{P}(\mathsf{H}) \\
[\![\mathsf{1}]\!]\ h & \triangleq \{h\} \\
[\![\mathsf{0}]\!]\ h & \triangleq \{\} \\
[\![f = n]\!]\ (pk{::}h) & \triangleq \begin{cases} \{pk{::}h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases} \\
[\![\neg a]\!]\ h & \triangleq \{h\} \setminus ([\![a]\!]\ h) \\
[\![f \leftarrow n]\!]\ (pk{::}h) & \triangleq \{pk[f := n]{::}h\} \\
[\![p + q]\!]\ h & \triangleq [\![p]\!]\ h \cup [\![q]\!]\ h \\
[\![p \,.\, q]\!]\ h & \triangleq ([\![p]\!] \bullet [\![q]\!])\ h \\
[\![p^*]\!]\ h & \triangleq \bigcup_{i \in \mathbb{N}} F^i\ h \\
\text{where } F^0\ h \triangleq \{h\} \text{ and } F^{i+1}\ h & \triangleq ([\![p]\!] \bullet F^i)\ h \\
[\![\mathsf{dup}]\!]\ (pk{::}h) & \triangleq \{pk{::}(pk{::}h)\}
\end{array}
$$

Figure 2: NetKAT semantics [1]

predicate $f = n$ drops all the packets whose field f is not assigned value n. Moreover, $\neg a$ stands for the negation of predicate a, $a + b$ represents the disjunction of predicates a and b, whereas $a.b$ denotes their conjunction.

Let H be the set of all histories, and $\mathscr{P}(H)$ be the powerset of H. In Figure 2, the semantic definition of a NetKAT policy p is given as a function $[\![p]\!]$ that takes a history $h \in H$ and produces a (possibly empty) set of histories in $\mathscr{P}(H)$. Some intuition on the semantics of policies was already provided in the paragraph above. In addition, note that negated predicates drop the packets not satisfying that predicate: $[\![\neg a]\!]h = \{h\} \setminus [\![a]\!]h$. The sequential composition of policies $[\![p.q]\!]$ denotes the Kleisli composition $\bullet$ of the functions $[\![p]\!]$ and $[\![q]\!]$ defined as:

$$(f \bullet g)x \triangleq \bigcup \{g\, y \mid y \in f\, x\}.$$

The repeated iteration of policies is interpreted as the union of $F^i\, h$, where the semantics of each F^i coincides with the semantics of the policy resulted by concatenating p with itself for i times, for $i \in \mathbb{N}$.

$p+(q+r) \equiv (p+q)+r$	KA-PLUS-ASSOC	$a+(b.c) \equiv (a+b).(a+c)$	BA-PLUS-DIST
$p+q \equiv q+p$	KA-PLUS-COMM	$a+1 \equiv 1$	BA-PLUS-ONE
$p+0 \equiv p$	KA-PLUS-ZERO	$a+\neg a \equiv 1$	BA-EXCL-MID
$p+p \equiv p$	KA-PLUS-IDEM	$a.b \equiv b.a$	BA-SEQ-COMM
$p.(q.r) \equiv (p.q).r$	KA-SEQ-ASSOC	$a.\neg a \equiv 0$	BA-CONTRA
$1.p \equiv p$	KA-ONE-SEQ	$a.a \equiv a$	BA-SEQ-IDEM
$p.1 \equiv p$	KA-SEQ-ONE		
$p.(q+r) \equiv p.q+p.r$	KA-SEQ-DIST-L		
$(p+q).r \equiv p.r+q.r$	KA-SEQ-DIST-R		
$0.p \equiv 0$	KA-ZERO-SEQ		
$p.0 \equiv 0$	KA-ZERO-SEQ		
$1+p.p^* \equiv p^*$	KA-UNROLL-L		
$1+p^*.p \equiv p^*$	KA-UNROLL-R		
$q+p.r \leq r \Rightarrow p^*.q \leq r$	KA-LFP-L		
$p+q.r \leq q \Rightarrow p.r^* \leq q$	KA-LFP-R		

Figure 3: Kleene Algebra Axioms & Boolean Algebra Axioms [1]

$f \leftarrow n.f' \leftarrow n' \equiv f' \leftarrow n'.f \leftarrow n, \text{if } f \neq f'$	PA-MOD-MOD-COMM
$f \leftarrow n.f' = n' \equiv f' \leftarrow n'.f = n, \text{if } f \neq f'$	PA-MOD-FILTER-COMM
$\mathbf{dup}.f = n \equiv f = n.\mathbf{dup}$	PA-DUP-FILTER-COMM
$f \leftarrow n.f = n \equiv f \leftarrow n$	PA-MOD-FILTER
$f = n.f \leftarrow n \equiv f = n$	PA-FILTER-MOD
$f \leftarrow n.f \leftarrow n' \equiv f \leftarrow n'$	PA-MOD-MOD
$f = n.f = n' \equiv 0, \text{if } n \neq n'$	PA-CONTRA
$\Sigma_i f = i \equiv 1$	PA-MATCH-ALL

Figure 4: Packet Algebra Axioms [1]

In Figure 3 and Figure 4 we recall the sound and complete axiomatisation of NetKAT. The Kleene Algebra with Tests axioms in Figure 3, have been formerly introduced in [19]. Completeness of NetKAT results from the packet algebra axioms in Figure 4. The axiom PA-MOD-MOD-COMM stands for the commutativity of different field assignments, whereas PA-MOD-FILTER-COMM denotes the commutativity of different field assignments and tests, for instance. PA-MOD-MOD states that two subsequent modifications of the same field can be reduced to capture the last modification only. The axiom PA-CONTRA states that the same field of a packet cannot have two different values, etc.

We write $\vdash e \equiv e'$, or simply $e \equiv e'$, whenever the equation $e \equiv e'$ can be proven according to the NetKAT axiomatisation.

Assume, for an example, a simple network consisting four hosts H_1, H_2, H_3 and H_4 communicating with each other via two switches A and B, via the uniquely-labeled ports $1, 2, \ldots, 6$, as illustrated in

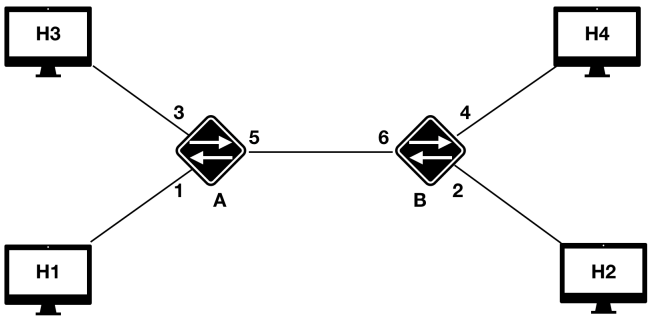

Figure 5: A simple network

Figure 5. The network topology can be given by the NetKAT expression:

$$t \triangleq pt = 5.pt \leftarrow 6 + pt = 6.pt \leftarrow 5 + pt = 1 + pt = 2 + pt = 3 + pt = 4 \tag{1}$$

For an intuition, in (1), the expression $pt = 5.pt \leftarrow 6 + pt = 6.pt \leftarrow 5$ encodes the internal link $5-6$ by using the sequential composition of a filter that keeps the packets at one end of the link and a modification that updates the *pt* fields to the location at the other end of the link. A link at the perimeter of the network is encoded as a filter that returns the packets located at the ingress port.

Furthermore, assume a programmer P_1 as in [1] which has to encode a switch policy that only enables transferring packets from H_1 to H_2. P_1 might define the "hop-by-hop" policy in (2), where each summand stands for the forwarding policy on switch A and B, respectively.

$$p_1 \triangleq pt = 1.pt \leftarrow 5 + pt = 6.pt \leftarrow 2 \tag{2}$$

In the expression above, the NetKAT expression $pt = 1.pt \leftarrow 5$ sends the packets arriving at port 1 on switch A, to port 5, whereas $pt = 6.pt \leftarrow 2$ sends the packets at port 6 on switch B, to port 2.

At this point, from P_1's perspective, the end-to-end behaviour of the network is defined as:

$$(pt = 1).(p_1.t)^*.(pt = 2) \tag{3}$$

In words: packets situated at ingress port 1 (encoded as $pt = 1$) are forwarded to egress port 2 (encoded as $pt = 2$) according to the switch policy p_1 and topology t (encoded as $(p_1.t)^*$).

More generally, assuming a switch policy p, topology t, ingress *in* and egress *out*, the *end-to-end behaviour* of a network is defined as:

$$in.(p.t)^*.out \tag{4}$$

Hence, based on (3), in order to asses the correctness of P_1's program, one has to show that:

1. packets at port 1 reach port 2, i.e.,

$$\vdash (pt = 1).(p_1.t)^*.(pt = 2) \not\equiv 0 \tag{5}$$

2. no packets at port 1 can reach ports 3 or 4, i.e.,

$$\vdash (pt = 1).(p_1.t)^*.(pt = 3 + pt = 4) \equiv 0. \tag{6}$$

By applying the NetKAT axiomatisation, the inequality in (5) can be equivalently rewritten as:

$$\vdash pt = 1.pt \leftarrow 2 + e \not\equiv 0 \tag{7}$$

with e a NetKAT expression. Observe that $pt = 1.pt \leftarrow 2$ cannot be reduced further. Hence, the inequality in (5) holds, as $pt = 1.pt \leftarrow 2 \not\equiv 0$. In other words, the packets located at port 1 reach port 2. Showing that no packets at port 1 can reach port 3 or 4 follows in a similar fashion.

3 Safety in NetKAT

As discussed in the previous section, arguing on equivalence of NetKAT programs can be easily performed in an equational fashion. One interesting way of further exploiting the NetKAT framework is to formalise and reason about well-known notions of program correctness such as safety, for instance. Intuitively, a safety property states that "something bad never happens". Ideally, the framework would provide a positive answer whenever a certain safety property is satisfied by the program, and an explanation of what went wrong in case the property is violated.

Consider the example of programmer P_1. The "bad" thing that could happen is that his switch policy enabled packets to reach ports 3 or 4. One can encode such a hazard via the egress policy $out \triangleq pt = 3 + pt = 4$, and the whole safety requirement as in (6). As previously discussed, the NetKAT axiomatisation provides a positive answer with respect to the satisfiability of the safety requirement in (6).

We further proceed by formalising a notion of *port-based hop-by-hop* policy and a *safety* concept in NetKAT. Let $f_1, \ldots, f_n$ be the list of fields defining a packet, including the port field pt. Moreover, for simplicity, and without loss of generality, assume no two different ports have the same value/identifier in the network. Let $\overleftarrow{A_{pt}}$ represent the set of all possible port assignments $pt \leftarrow v$ for some value v. Let $\overline{\overline{T}}$ denote the set of all possible tests of the form $f = v$ for some field $f \in \{f_1, \ldots, f_n\}$ and value v. We write:

- $\overline{\overline{T}}^*$ to represent the set of all sequences of tests in $\overline{\overline{T}}$
- $\Sigma_{\overline{\overline{T}}^*}$ for the set of all tests $\bar{t} = \Sigma_{i=\{i,\ldots,m\}} \bar{\bar{t}}_i^*$ with $\bar{\bar{t}}_i^* \in \overline{\overline{T}}^*$, for all $i \in \{1, \ldots m\}$

Definition 1 (HbH Switch Policy)**.** *A switch policy p is called* port-based hop-by-hop (HbH) *if it is defined as:*

$$p \triangleq \Sigma_{i \in \{1,\ldots,m\}} \bar{\bar{t}}_i^* \cdot \overleftarrow{pt}_i \tag{8}$$

where $\bar{\bar{t}}_i^* \in \overline{\overline{T}}^*$ and $\overleftarrow{pt}_i \in \overleftarrow{A_{pt}}$, for all $i \in \{1, \ldots, m\}$.

We call m the size *of the HbH policy p.*

Definition 2 (In-Out Safe)**.** *Assume a network topology t, a HbH switch policy p, an ingress policy $in \in \Sigma_{\overline{\overline{T}}^*}$, and an egress policy $out \in \Sigma_{\overline{\overline{T}}^*}$, the latter encoding the hazard, or the "bad thing". The end-to-end network behaviour is* in-out safe *whenever the following holds:*

$$\vdash in.(p.t)^*.out \equiv 0. \tag{9}$$

Intuitively, none of the packages at ingress *in* can reach the "hazardous" egress *out* whenever forwarded according to the switch policy p, across the topology t.

Remark 1. *A notion of reachability within NetKAT-definable networks was proposed in [1] based on the existence of a non-empty packet history that, in essence, records all the packet modifications produced by the policy $in.(p.t)^*.out$. This is more like a model-checking-based technique that enables identifying*

one *counterexample witnessing the violation of the property* $in.(p.t)^*.out \equiv 0$. *As we shall later see, in our setting, we are interested in identifying* all *(loop-free) counterexamples. Hence, we propose a notion of in-out safe behaviour for which, whenever violated, we can provide all relevant bad behaviours.*

Going back to the example in Section 2, assume a new programmer P_2 which has to enable traffic only from H_3 to H_4. Assuming the network in Figure 5, P_2 encodes the HbH switch policy:

$$p_2 \triangleq pt = 3.pt \leftarrow 5 + pt = 6.pt \leftarrow 4 \tag{10}$$

The end-to-end behaviour can be proven correct, by showing that:

1. packets at port 3 reach port 4, i.e.,

$$\vdash (pt = 3).(p_2.t)^*.(pt = 4) \not\equiv 0 \tag{11}$$

2. no packets at port 3 can reach ports 1 or 2, i.e.,

$$\vdash (pt = 3).(p_2.t)^*.(pt = 1 + pt = 2) \equiv 0. \tag{12}$$

Nevertheless, it is easy to show that the composed policies p_1 in (2) and p_2 in (10) do not guarantee a safe behaviour. Namely, in the context of the HbH policy $p_1 + p_2$, packets at port 1 can reach ports 3 or 4, and packets at port 3 can reach ports 1 or 2. This violates the correctness properties in (6) and (12), respectively:

$$\vdash (pt = 1).((p_1 + p_2).t)^*.(pt = 3 + pt = 4) \not\equiv 0 \tag{13}$$

$$\vdash (pt = 3).((p_1 + p_2).t)^*.(pt = 1 + pt = 2) \not\equiv 0 \tag{14}$$

In the next section, we would like to provide the explanation for the failure of the network safety, as expressed in (13) and (14).

4 Explaining Safety Failures

Naturally, the first attempt to explain safety failures is to derive the counterexamples according to the NetKAT axiomatisation. Take, for instance, the end-to-end behaviour $(pt = 1).((p_1 + p_2).t)^*.(pt = 3 + pt = 4)$ in (13). The axiomatisation leads to the following equivalence:

$$(pt = 1).((p_1 + p_2).t)^*.(pt = 3 + pt = 4) \equiv (pt = 1.pt \leftarrow 4) + e \tag{15}$$

where e is a NetKAT expression containing the Kleene $*$. A counterexample can be immediately spotted, namely: $pt = 1.pt \leftarrow 4$. Nevertheless, the information it provides is not intuitive enough to serve as an explanation of the failure. Moreover, e can hide additional counterexamples revealed after a certain number of $*$-unfoldings according to KA-UNROLL-R/L in Figure 4.

In what follows, the focus is on the following two questions:

Q_1: Can we reveal *more information* within the counterexamples witnessing safety failures?

Q_2: Can we reveal *all the counterexamples* hidden within NetKAT expressions containing $*$?

The answer to Q_1 is relatively simple: yes, we can reveal more information on how the packets travel across the topology by inhibiting the PA-MOD-MOD and PA-FILTER-MOD axioms in Figure 4. Recall that, intuitively, this axiom records only the last modification from a series of modifications of the same field.

The answer to Q_2 lies behind the following two observations. (1) From a practical perspective, in order to explain failures it suffices to look at loop-free forwarding paths within the network topology. Reaching the same port twice along a path does not add insightful information about the reason behind the violation of a safety property, as the network behaviour is preserved in the context of that port. Considering loop-free paths is also in accordance with the minimality criterion invoked in the seminal work on causal reasoning in [15], for instance. (2) A HbH switch policy p of size n entails loop-free paths from *in* to *out* crossing at most n switches within a topology t. Hence, in order to determine all loop-free paths from *in* to *out* it suffices to apply the HbH policy p along t for n times. Showing that the suggested approximation is sound reduces to showing:

$$\vdash in.(p.t)^*.out \equiv 0 \text{ iff } \vdash in.(1+p.t)^n.out \equiv 0 \tag{16}$$

Lemma 1 and Lemma 2 are needed in order to prove the equivalence in (16).

Lemma 1. *Let p, t be two NetKAT policies. The following holds, for all natural numbers n:*

$$(1+p.t)^n \equiv 1+p.t+(p.t)^2+\ldots+(p.t)^n \tag{17}$$

Proof. The proof follows by induction on n.
Base case: $n=0$. If $n=0$ then $(1+(p.t))^0=1$, inferred based on the definition of Kleisli composition.
Induction step: Assume (17) holds for all k such that $0 \leq k \leq n$. It follows that:

$$\begin{array}{lcll}
(1+p.t)^{n+1} & \equiv & (1+p.t)^n.(1+p.t) & \text{(Kleisli composition)} \\
& \equiv & (1+p.t+(p.t)^2+\ldots+(p.t)^n).(1+p.t) & \text{(ind. hypothesis)} \\
& \equiv & 1+p.t+(p.t)^2+\ldots+(p.t)^n+ & \\
& & \quad p.t+(p.t)^2+\ldots+(p.t)^n+(p.t)^{n+1} & \text{(KA-SEQ-DIST-L/R, KA-ONE-SEQ)} \\
& \equiv & 1+p.t+(p.t)^2+\ldots+(p.t)^n+(p.t)^{n+1} & \text{(KA-PLUS-IDEM)}
\end{array}$$

Hence, (17) holds. □

Lemma 2. *Let p, t, in, out be NetKAT policies. The following holds, for all natural numbers n:*

$$in.(1+p.t)^n.out \leq in.(p.t)^*.out \tag{18}$$

Proof. First, observe that

$$in.(p.t)^*.out \equiv in.(1+p.t+(p.t)^2+\ldots+(p.t)^n+(p.t)^{n+1}.(p.t)^*).out \tag{19}$$

by KA-UNROLL-L/R, KA-PLUS-IDEM and KA-SEQ-DIST-L/R. Consequently, by Lemma 1, the following also holds:

$$in.(p.t)^*.out \equiv in.(1+p.t)^n.out + in.(p.t)^{n+1}.(p.t)^*.out \tag{20}$$

Therefore,

$$in.(1+p.t)^n.out \leq in.(p.t)^*.out$$

holds as well. □

Theorem 1. *(Star Elimination for Safety) Assume a network topology t, a HbH switch policy p of size n, an ingress policy in* $\in \Sigma_{\overline{\overline{T}}^*}$*, and an egress policy out* $\in \Sigma_{\overline{\overline{T}}^*}$ *encoding the hazard. The following holds:*

$$\vdash in.(p.t)^*.out \equiv 0 \text{ iff } \vdash in.(1+p.t)^n.out \equiv 0 \tag{21}$$

Proof. The "if" case follows immediately, as by Lemma 2 and the hypothesis the following holds:

$$0 \leq in.(1+p.t)^n.out \leq in.(p.t)^*.out \equiv 0.$$

For the "only if" case we proceed by reductio ad absurdum.

Assume:

$$in.(p.t)^*.out \not\equiv 0. \tag{22}$$

By the hypothesis, the definition of Kleene $*$ and the assumption in (22), it follows that there exists $N > n$ such that:

$$in.(p.t)^N.out \not\equiv 0.$$

Consider:

$$\begin{array}{rcl} p & \triangleq & \Sigma_{i\in\{1,\ldots,n\}} \overline{\overline{t}}^*_i \,.pt \leftarrow v_i \\ t & \triangleq & \Sigma_{j\in\{1,\ldots,m\}} pt = v_j \,.pt \leftarrow v'_j \end{array} \tag{23}$$

and assume $N = n+k$. Given the shape of the HbH switch policy p and topology t, it follows that there exist policies $e_1 \not\equiv 0$, e_2 such that:

$$\vdash in.(p.t)^N.out \equiv e_1 + e_2$$

with $e_1 \triangleq in.p_N.out$ and

$$p_N \triangleq (\overline{\overline{t}}^*_{i_1} .pt \leftarrow v_{i_1} .pt = v_{i_1}.pt \leftarrow v'_{j_1}). \ \ldots \ .(\overline{\overline{t}}^*_{i_{n+k}} .pt \leftarrow v_{i_{n+k}} .pt = v_{i_{n+k}}.pt \leftarrow v'_{j_{n+k}}) \tag{24}$$

where $\overline{\overline{t}}^*_{i_l}, v_{i_l}, v'_{j_l}$ range over $\overline{\overline{t}}^*_i, v_i, v'_j$ as in (23).

Note that e_1 denotes a path that goes through $N > n$ switches from *in* to *out*. Recall that the switch policy p can define loop-free paths traversing at most n switches (the size of p). Hence, we conclude that the identified path is not loop-free, i.e.:

$$\begin{array}{rcl} e_1 & \triangleq & in. \\ & & (\overline{\overline{t}}^*_{i_1} .pt \leftarrow v_{i_1} .pt = v_{i_1}.pt \leftarrow v'_{j_1}). \\ & & \ldots \\ & & (\overline{\overline{t}}^*_{i_\alpha} .pt = \mathbf{v_{i_\alpha}} .pt \leftarrow v'_{i_\alpha}). \\ & & \ldots \\ & & (\overline{\overline{t}}^*_{i_{\alpha+\beta}} .pt = \mathbf{v_{i_\alpha}} .pt \leftarrow v'_{i_{\alpha+\beta}}). \\ & & \ldots \\ & & (\overline{\overline{t}}^*_{i_{n+k}} .pt \leftarrow v_{i_{n+k}} .pt = v_{i_{n+k}}.pt \leftarrow v'_{j_{n+k}}). \\ & & out \end{array} \tag{25}$$

with $\beta \geq k$. Based on e_1, we can devise a policy defining a loop-free path that crosses at most n switches from *in* to *out*. Consider a policy $p_N{\downarrow}$ that stands for p_N without the loop of size β from (25):

$$\begin{array}{rcl} p_N{\downarrow} & \triangleq & \overline{\overline{t}}^*_{i_1} .pt \leftarrow v_{i_1} .pt = v_{i_1}.pt \leftarrow v'_{j_1}. \\ & & \ldots \\ & & \overline{\overline{t}}^*_{i_\alpha} . pt = v_{i_\alpha} .pt \leftarrow v'_{i_\alpha}. \\ & & \overline{\overline{t}}^*_{i_{\alpha+\beta}} . pt = v_{i_{\alpha+\beta+1}} .pt \leftarrow v'_{i_{\alpha+\beta+1}}. \\ & & \ldots \\ & & \overline{\overline{t}}^*_{i_{n+k}} .pt \leftarrow v_{i_{n+k}} .pt = v_{i_{n+k}}.pt \leftarrow v'_{j_{n+k}}. \end{array} \tag{26}$$

By the construction of $p_N{\downarrow}$ and the fact that $e_1 \not\equiv 0$ it follows that:

$$in.p_N{\downarrow}.out \not\equiv 0. \tag{27}$$

In words, we identified a path that traverses the topology from *in* to *out* and crosses at most n switches. Moreover, by the hypothesis and Lemma 1, the following hold:

$$\begin{array}{rcl} in.out & \equiv & 0 \\ in.(p.t)^i.out & \equiv & 0 \quad \text{for all } i \in \{1,\ldots,n\}. \end{array} \tag{28}$$

Hence, (27) contradicts (28). We conclude that the "only if" case holds as well. □

With these ingredients at hand, in accordance with Q_1 and Q_2, consider an alteration of the axiomatisation as follows. Firstly, we inhibit the axioms PA-MOD-MOD, PA-FILTER-MOD and KA-UNROLL-R/L. Then, we add the star elimination axiom corresponding to (21) in Theorem 1:

$$in.(p.t)^*.out \equiv in.(1+p.t)^n.out. \tag{29}$$

Let $\vdash_s$ be the entailment relation over the modified axiomatisation.

Definition 3 (Safety Failure Explanations)**.** *Assume a network topology t, a HbH switch policy p of size n, an ingress policy $in \in \Sigma_{\overline{T}^*}$, and an egress policy $out \in \Sigma_{\overline{T}^*}$ encoding the hazard. A* safety failure explanation *is a policy $expl \not\equiv 0$ in canonical form (i.e., it cannot be reduced further) such that:*

$$\vdash_s in.(p.t)^*.out \equiv expl. \tag{30}$$

For an example, we refer to the case of the two programmers providing switch policies p_1 and p_2 forwarding packets from host H_1 to H_2, and from H_3 to H_4 within the network in Figure 5. As previously discussed, the end-to-end network behaviour defined over each of the aforementioned policies can be proven correct using the NetKAT axiomatisation. Nevertheless, a comprehensive explanation of what caused the erroneous behaviour over the unified policy $p_1 + p_2$ could not be derived according $\vdash$. The new axiomatisation, however, entails the following explanation:

$$\vdash_s (pt = 1).((p_1 + p_2).t)^*.(pt = 3 + pt = 4) \equiv pt = 1.pt \leftarrow 5.pt \leftarrow 6.pt \leftarrow 4$$

showing how packets at port 1 can reach port 4. Similarly,

$$\vdash_s (pt = 3).((p_1 + p_2).t)^*.(pt = 1 + pt = 2) \equiv pt = 3.pt \leftarrow 5.pt \leftarrow 6.pt \leftarrow 2$$

shows how packets at port 3 can reach port 2.

Remark 2. *The work in [1] proposes a "star elimination" method for switch policies not containing* **dup** *and switch assignments. The procedure in [1] employs a notion of normal form to which each NetKAT policy can be reduced. The reason for not using the aforementioned star elimination in our context is that the normal forms in [1] "forget" the intermediate sequences of assignments and tests, and reduce policies to sums of expressions of shape $(f_1 = v_1.\ldots.f_n = v_n).(f_1 \leftarrow v'_1.\ldots.f_n \leftarrow v'_n)$ where $f_1,\ldots,f_n$ are the packet fields. Hence, the normal forms exploited by the star elimination in [1] can not serve as comprehensive failure explanations.*

Remark 3. *Note that the safety failure explanations in Definition 3 are not minimal. There might be cases in which two explanation paths*

$$e_1 \triangleq p'.p'' \qquad e_2 \triangleq p'.\tilde{p}.p''$$

are identified. Nevertheless, minimality in this context is easy to achieve in a post-processing step that pattern-matches expressions like e_1 and e_2 and keeps only e_1 as relevant explanation.

5 Discussion

In this paper we formulate a notion of safety in the context of NetKAT programs [1] and provide an equational framework that computes all relevant explanations witnessing a bad, or an unsafe behaviour, whenever the case. The proposed equational framework is a slight modification of the sound and complete axiomatisation of NetKAT, and is parametric on the size of the considered hop-by-hop switch policy. Our approach is orthogonal to related works which rely on model-checking algorithms for computing all counterexamples witnessing the violation of a certain property, such as [20, 6], for instance. A corresponding tool for automatically computing the explanations can be straightforwardly implemented in a programming language like Maude [8], for instance; we leave this exercise as future work. We consider assessing the complexity of the procedure for real case scenarios, on top of benchmarks as in [16, 17], for instance.

The results in this paper are part of a larger project on (counterfactual) causal reasoning on NetKAT. In [21], Lewis formulates the counterfactual argument, which defines when an event is considered a cause for some effect (or hazardous situation) in the following way: a) whenever the event presumed to be a cause occurs, the effect occurs as well, and b) when the presumed cause does not occur, the effect will not occur either. The current result corresponds to item a) in Lewis' definition, as it describes the events that have to happen in order for the hazardous situation to happen as well. The next natural step is to capture the counterfactual test in b). This reduces to tracing back the explanations to the level of the switch policy, and rewrite the latter so that it disables the generation the paths leading to the undesired egress. The generation of a "correct" switch policy can be seen as an instance of program repair.

In the future we would be, of course, interested in defining notions of causality (and associated algorithms) with respect to the violation of other relevant properties such as liveness, for instance. We would also like to explain and eventually disable routing loops (i.e., endlessly looping between A and B) from occurring. Or, we would like to identify the cause of packets being not correctly filtered by a certain policy.

Acknowledgements The author is grateful to the reviewers of FROM 2019, for their feedback and observations. Special thanks are addressed to Tobias Kappé, for his useful comments and insight into the formal foundations of NetKAT.

References

[1] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger & David Walker (2014): *NetKAT: semantic foundations for networks*. In: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pp. 113–126, doi:10.1145/2535838.2535862.

[2] Christel Baier & Joost-Pieter Katoen (2008): *Principles of model checking*. MIT Press.

[3] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni & Richard J. Trefler (2012): *Explaining counterexamples using causality*. *Formal Methods in System Design* 40(1), pp. 20–40, doi:10.1007/s10703-011-0132-2.

[4] Salem Benferhat & John Grant, editors (2011): *Scalable Uncertainty Management - 5th International Conference, SUM 2011, Dayton, OH, USA, October 10-13, 2011. Proceedings*. *Lecture Notes in Computer Science* 6929, Springer, doi:10.1007/978-3-642-23963-2.

[5] Christian Buckl, Alois Knoll, Ina Schieferdecker & Justyna Zander (2007): *Model-Based Analysis and Development of Dependable Systems*. In Giese et al. [13], pp. 271–293, doi:10.1007/978-3-642-16277-0_10.

[6] Georgiana Caltais, Sophie Linnea Guetlein & Stefan Leue (2018): *Causality for General LTL-definable Properties*. In Finkbeiner & Kleinberg [9], pp. 1–15, doi:10.4204/EPTCS.286.1.

[7] Georgiana Caltais, Stefan Leue & Mohammad Reza Mousavi (2016): *(De-)Composing Causality in Labeled Transition Systems*. In Gößler & Sokolsky [14], pp. 10–24, doi:10.4204/EPTCS.224.3.

[8] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer & Jose F. Quesada (1999): *The Maude System*. In Narendran & Rusinowitch [23], pp. 240–243, doi:10.1007/3-540-48685-2_18.

[9] Bernd Finkbeiner & Samantha Kleinberg, editors (2019): *Proceedings 3rd Workshop on formal reasoning about Causation, Responsibility, and Explanations in Science and Technology, CREST@ETAPS 2018, Thessaloniki, Greece, 21st April 2018*. *EPTCS* 286, doi:10.4204/EPTCS.286.

[10] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story & David Walker (2011): *Frenetic: a network programming language*. In: *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pp. 279–291, doi:10.1145/2034773.2034812.

[11] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva & Laure Thompson (2015): *A Coalgebraic Decision Procedure for NetKAT*. In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pp. 343–355, doi:10.1145/2676726.2677011.

[12] Roberto Giacobazzi, Josh Berdine & Isabella Mastroeni, editors (2013): *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*. *Lecture Notes in Computer Science* 7737, Springer, doi:10.1007/978-3-642-35873-9.

[13] Holger Giese, Gabor Karsai, Edward Lee, Bernhard Rumpe & Bernhard Schätz, editors (2011): *Model-Based Engineering of Embedded Real-Time Systems - International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. Revised Selected Papers*. *Lecture Notes in Computer Science* 6100, Springer, doi:10.1007/978-3-642-16277-0.

[14] Gregor Gößler & Oleg Sokolsky, editors (2016): *Proceedings First Workshop on Causal Reasoning for Embedded and safety-critical Systems Technologies, CREST@ETAPS 2016, Eindhoven, The Netherlands, 8th April 2016*. *EPTCS* 224, doi:10.4204/EPTCS.224.

[15] Joseph Y. Halpern (2011): *Causality, Responsibility, and Blame: A Structural-Model Approach*. In Benferhat & Grant [4], p. 1, doi:10.1007/978-3-642-23963-2_1.

[16] Peyman Kazemian, George Varghese & Nick McKeown (2012): *Header Space Analysis: Static Checking for Networks*. In: *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pp. 113–126. Available at `https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/kazemian`.

[17] Simon Knight, Hung X. Nguyen, Nick Falkner, Rhys Alistair Bowden & Matthew Roughan (2011): *The Internet Topology Zoo*. *IEEE Journal on Selected Areas in Communications* 29(9), pp. 1765–1775, doi:10.1109/JSAC.2011.111002.

[18] Dexter Kozen (1994): *A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events*. *Inf. Comput.* 110(2), pp. 366–390, doi:10.1006/inco.1994.1037.

[19] Dexter Kozen (1997): *Kleene Algebra with Tests*. *ACM Trans. Program. Lang. Syst.* 19(3), pp. 427–443, doi:10.1145/256167.256195.

[20] Florian Leitner-Fischer & Stefan Leue (2013): *Causality Checking for Complex System Models*. In Giacobazzi et al. [12], pp. 248–267, doi:10.1007/978-3-642-35873-9_16.

[21] D. Lewis (1973): *Causation*. *Journal of Philosopy* 70, pp. 556–567, doi:10.2307/2025310.

[22] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru M. Parulkar, Larry L. Peterson, Jennifer Rexford, Scott Shenker & Jonathan S. Turner (2008): *OpenFlow: enabling innovation in campus networks*. *Computer Communication Review* 38(2), pp. 69–74, doi:10.1145/1355734.1355746.

[23] Paliath Narendran & Michaël Rusinowitch, editors (1999): *Rewriting Techniques and Applications, 10th International Conference, RTA-99, Trento, Italy, July 2-4, 1999, Proceedings*. Lecture Notes in Computer Science 1631, Springer, doi:10.1007/3-540-48685-2.

[24] István Pelle & András Gulyás (2019): *An Extensible Automated Failure Localization Framework Using NetKAT, Felix, and SDN Traceroute*. Future Internet 11(5), doi:10.3390/fi11050107. Available at `https://www.mdpi.com/1999-5903/11/5/107`.

[25] Walid Mohamed Taha, editor (2009): *Domain-Specific Languages, IFIP TC 2 Working Conference, DSL 2009, Oxford, UK, July 15-17, 2009, Proceedings*. Lecture Notes in Computer Science 5658, Springer, doi:10.1007/978-3-642-03034-5.

[26] Andreas Voellmy & Paul Hudak (2009): *Nettle: A Language for Configuring Routing Networks*. In Taha [25], pp. 211–235, doi:10.1007/978-3-642-03034-5_11.

[27] Andreas Voellmy, Junchang Wang, Yang Richard Yang, Bryan Ford & Paul Hudak (2013): *Maple: simplifying SDN programming using algorithmic policies*. In: ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013, pp. 87–98, doi:10.1145/2486001.2486030.