Explaining Safety Failures in NetKAT

Georgiana Caltais[a], Hünkar Can Tunç[a]

[a] *Department for Computer and Information Science, University of Konstanz, Germany*

Abstract

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 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, or program, which does not enable forwarding packets from an ingress i to an undesirable egress e. We show how explanations for safety violations can be derived in an equational fashion, according to a modification of the existing NetKAT axiomatisation. We propose an approach based on the Maude system for actually computing the undesired behaviours witnessing the forwarding of packets from i to e as above. *SDN−SafeCheck* is a tool based on Maude equational theories satisfying important properties such as Church-Rosser and termination. *SDN−SafeCheck* automatically identifies all the undesired behaviours leading to e, covering forwarding paths up to a user specified size.

Keywords: software defined networks, NetKAT, safety, failure analysis, axiomatisations, the Maude system

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) [1], for instance, have been proposed and widely

Email addresses: `gcaltais@gmail.com` (Georgiana Caltais), `hcantunc@gmail.com` (Hünkar Can Tunç)

used by reliability engineers in order to understand how systems can fail, and for debugging purposes.

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 programmable, and more flexible to change. SDN proposes open standards such as the OpenFlow [2] API defining, for instance, low-level languages for handling switch configurations. Typically, 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 [3, 4, 5].

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 [6, 7] –a formal framework for specifying and reasoning about networks, integrated within the Frenetic suite of network management tools [3]. In this work we exploit the sound and complete axiomatisation of NetKAT in [6] and derive explanations of safety failures in a purely equational fashion.

From a more practical perspective, we introduce *SDN–SafeCheck*, a tool based on the Maude system [8], aiming at automatically computing the explanations for undesired behaviours within a NetKAT program that forwards packets from an ingress i to an egress e. *SDN–SafeCheck* is based on Maude confluent and terminating equational specifications, and computes the explanations for all the undesired behaviours covering forwarding paths up to a user specified size.

Related to the current work, the authors of NetKAT [6] show that checking certain properties about networks, including reachability properties, can be reduced to equivalence checking problems in NetKAT by utilizing its sound and complete axiomatisation. NetKAT is also equipped with a practical tool which can check the equivalence of NetKAT policies [7]. The main focus of the tool proposed in [7] is to check whether a property holds in the network. This differs from our focus that we aim on discovering all possible ways a reachability property can be violated, and provide explanations that may be instructive for debugging purposes.

The results in [9] introduce a framework for automated failure localisation in NetKAT. The approach in [9] 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 provides explanations for possible failures irrespective of particular input packets.

The work in [10] was the first to utilize a rewrite engine to manipulate NetKAT expressions in order to verify network properties. The authors of [10] propose an operational semantics for NetKAT and implement their formal specification in Maude. By utilizing the proposed operational semantics, the authors mainly follow three different techniques for automated reasoning in NetKAT: model checking of invariants, linear temporal logic based model checking, and normalization. The proposed formulations of the model checking procedures do not provide an explicit counterexample in case of a failure, hence these methods are unsuitable in our context. The normalization method is a different formulation of the equivalence checking approach that was proposed in [6] for verifying network properties. The normalization method assesses whether NetKAT policies can be converted into the same normal form. This is a relevant method in our setting as well, however, the experimental evaluation in [10] shows that the proposed specification for the normalization approach fails to scale even for networks of moderate size.

Our contributions. This paper is an extension of our previous work in [11]. In [11] we introduced 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, in the context of the so-called "port-based hop-by-hop" switch policies allowing only tests and port modifications. Then, we proposed a notion of *safety failure explanation* which, intuitively, represents the set of finite paths within the network, leading to the hazardous egress. Eventually, we provided a modified version of the original axiomatisation of NetKAT exploited in order to *automatically compute the safety failure explanations*, if any. The axiomatisation employed a proposed *star-elimination construction* which enabled the sound extraction of explanations from Kleene $*$-free NetKAT programs.

The current revised version of the paper extends the results in [11] as follows.

1. We propose a notion of safety in the context of more general switch policies defined as arbitrary expressions over the *-free, **dup**-free frag-

ment of NetKAT.

2. We show that a NetKAT network behaviour is “safe” whenever it can be proven so according to the proposed equational system used to derive safety failure explanations (see Corollary 1).
3. We formalize a concept of minimal, or relevant explanations for safety failures in NetKAT, based on a notion of “normal forms for safety” (see Section 3.2).
4. We introduce *SDN−SafeCheck*, a practical tool for automatically computing safety failure explanations (see Section 4). To the best of our knowledge, this tool is the first to provide automated failure explanations in NetKAT.
5. We provide experimental evaluations for *SDN−SafeCheck* based on the Topology Zoo dataset [12].

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 and we introduce the notion of (minimal) safety failure explanation and the axiomatisation which can be exploited in order to compute such explanations. In Section 4 we introduce the Maude-based tool *SDN−SafeCheck*. Experimental evaluation is discussed in Section 5. In Section 6 we draw the conclusions and pointers to future work.

2. Preliminaries

As pointed out in [6], 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 ($p \cdot q. \dots$), a set of paths is encoded as a union of paths ($p + q + \dots$) whereas iterated processing is encoded using Kleene $*$. This paves the way to reasoning about properties of networks using Kleene Algebra with Tests (KAT) [13]. KAT incorporates both Kleene Algebra [14] 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 1.

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

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

Figure 1: NetKAT Syntax and Semantics [6]

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 $p \cdot q$, repeated application of policies p^* (the Kleene $*$) and duplication **dup** (that saves the current packet at the beginning of the history list). At this point, it might be worth mentioning that **dup** plays a role in building the NetKAT language model but, as we shall later see, it is not necessary in our syntactic approach to failure analysis.

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 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 $a \cdot b$ denotes their conjunction.

Let H be the set of all histories, and $\mathcal{P}(H)$ be the power set of H. In Figure 1, 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 $\mathcal{P}(H)$. 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

$p + (q + r) \equiv (p + q) + r$	KA-PLUS-ASSOC	$a + (b \cdot c) \equiv (a + b) \cdot (a + c)$	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	$a \cdot b \equiv b \cdot a$	BA-SEQ-COMM
$p \cdot (q \cdot r) \equiv (p \cdot q) \cdot r$	KA-SEQ-ASSOC	$a \cdot \neg a \equiv 0$	BA-CONTRA
$1 \cdot p \equiv p$	KA-ONE-SEQ	$a \cdot a \equiv a$	BA-SEQ-IDEM
$p \cdot 1 \equiv p$	KA-SEQ-ONE		
$p \cdot (q + r) \equiv p \cdot q + p \cdot r$	KA-SEQ-DIST-L	$f \leftarrow n \cdot f' \leftarrow n' \equiv f' \leftarrow n' \cdot f \leftarrow n, \text{if } f \neq f'$	PA-MOD-MOD-COMM
$(p + q) \cdot r \equiv p \cdot r + q \cdot r$	KA-SEQ-DIST-R	$f \leftarrow n \cdot f' = n' \equiv f' = n' \cdot f \leftarrow n, \text{if } f \neq f'$	PA-MOD-FILTER-COMM
$0 \cdot p \equiv 0$	KA-ZERO-SEQ	$\mathbf{dup} \cdot f = n \equiv f = n \cdot \mathbf{dup}$	PA-DUP-FILTER-COMM
$p \cdot 0 \equiv 0$	KA-ZERO-SEQ	$f \leftarrow n \cdot f = n \equiv f \leftarrow n$	PA-MOD-FILTER
$1 + p \cdot p^* \equiv p^*$	KA-UNROLL-L	$f = n \cdot f \leftarrow n \equiv f = n$	PA-FILTER-MOD
$1 + p^* \cdot p \equiv p^*$	KA-UNROLL-R	$f \leftarrow n \cdot f \leftarrow n' \equiv f \leftarrow n'$	PA-MOD-MOD
$q + p \cdot r \leq r \Rightarrow p^* \cdot q \leq r$	KA-LFP-L	$f = n \cdot f = n' \equiv 0, \text{if } n \neq n'$	PA-CONTRA
$p + q \cdot r \leq q \Rightarrow p \cdot r^* \leq q$	KA-LFP-R	$\Sigma_i f = i \equiv 1$	PA-MATCH-ALL

Figure 2: NetKAT Axiomatisation [6]

sequential composition of policies $[\![p \cdot q]\!]$ denotes the Kleisli composition $\bullet$ of the functions $[\![p]\!]$ and $[\![q]\!]$.

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 $i \in \mathbb{N}$.

In Figure 2 we recall the sound and complete axiomatisation of NetKAT. The Kleene Algebra with Tests axioms in Figure 2, have been formerly introduced in [13]. Completeness of NetKAT results from the packet algebra axioms in Figure 2. 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 3. The network topology can be given by the NetKAT expression:

$$t \triangleq \begin{array}{l} pt = 5 \cdot pt \leftarrow 6 + pt = 6 \cdot pt \leftarrow 5\, + \\ pt = 1 + pt = 2 + pt = 3 + pt = 4 \end{array} \tag{1}$$

For an intuition, in (1), the expression $pt = 5 \cdot pt \leftarrow 6 + pt = 6 \cdot pt \leftarrow 5$ 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.

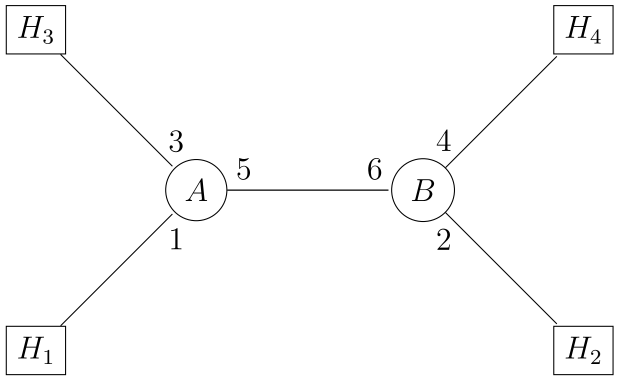

Figure 3: A Simple Network

Furthermore, assume a programmer P_1 as in [6] 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.

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

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

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

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 $(p_1 \cdot t)^*$).

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:

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

Note that, unlike the end-to-end NetKAT network behaviour in [6], the policy in (4) does not contain **dup**. As discussed in more detail in Section 3.1, our (syntactic) approach looks at each operation within a NetKAT expression, hence there is no need to use **dup** in order to record the individual "hops" that packets take as they go through the network.

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

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

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

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

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

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

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

with e a NetKAT expression. Observe that $pt = 1 \cdot pt \leftarrow 2$ cannot be reduced further. Hence, the inequality in (5) holds, as $pt = 1 \cdot pt \leftarrow 2 \not\equiv 0$. 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 and Failures 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).

Firstly, observe that our approach is syntactic in nature and it does not require recording individual packet modifications, or simulating actual "moves" in the NetKAT corresponding automata. Hence, it suffices to consider **dup**-free NetKAT expressions. As we shall later see, this also contributes to deriving more concise, **dup**-free failure explanations.

Secondly, observe that from a more practical perspective, the Kleene-$*$ is mainly used for ensuring a "looping" structure to allow packet moves along the hops. Thus, in our work, we consider ingress (*in*), egress (*out*), switch policies (p) and topologies (t) encoded in terms of **dup**-free, $*$-free NetKAT expressions, while the overall behaviour of a network is given as $in \cdot (p \cdot t)^* \cdot out$.

We call NetKAT$^{\textbf{-dup,}*}$ the **dup**-free, $*$-free fragment of NetKAT. We further proceed by formalizing a safety concept in NetKAT.

Definition 1 (In-Out Safe)**.** *Assume the* NetKAT$^{\textbf{-dup,}*}$ *expressions defining a network topology t, a switch policy p, an ingress policy in, and an egress policy out, the latter encoding the hazard, or the "bad thing". The end-to-end network behaviour is* in-out safe *whenever the following holds:*

$$\vdash in \cdot (p \cdot t)^* \cdot out \equiv 0. \tag{8}$$

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.

We call the *size of the network* the number of forwarding links within the network.

Remark 1. *A notion of reachability within NetKAT-definable networks was proposed in [6] based on the existence of a non-empty packet history that, in* *essence, records all the packet modifications produced by the policy* $in \cdot (p \cdot t)^* \cdot out$. *This is more like a model-checking-based technique that enables identifying* one *counterexample witnessing the violation of the property* $in \cdot (p \cdot t)^* \cdot out \equiv 0$. *As we shall later see, in our setting, we are interested in identifying* all *(minimal) 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 3, P_2 encodes the HbH switch policy:

$$p_2 \triangleq pt = 3 \cdot pt \leftarrow 5 + pt = 6 \cdot pt \leftarrow 4 \tag{9}$$

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

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

$$\vdash (pt = 3) \cdot (p_2 \cdot t)^* \cdot (pt = 4) \not\equiv 0 \tag{10}$$

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

$$\vdash (pt = 3) \cdot (p_2 \cdot t)^* \cdot (pt = 1 + pt = 2) \equiv 0. \tag{11}$$

Nevertheless, it is easy to show that the composed policies p_1 in (2) and p_2 in (9) do not guarantee a safe behaviour. Namely, in the context of the HbH policy $p_1 + p_2$, packets at port 1 can reach port 4, and packets at port 3 can reach port 2. This violates the correctness properties in (6) and (11), respectively:

$$\vdash (pt = 1) \cdot ((p_1 + p_2) \cdot t)^* \cdot (pt = 3 + pt = 4) \not\equiv 0 \tag{12}$$

$$\vdash (pt = 3) \cdot ((p_1 + p_2) \cdot t)^* \cdot (pt = 1 + pt = 2) \not\equiv 0 \tag{13}$$

In the next section, we provide a framework for explaining the failure of network safety as expressed in (12) and (13).

3.1. 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 $(pt = 1) \cdot ((p_1 + p_2) \cdot t)^* \cdot (pt = 3 + pt = 4)$ in (12). The axiomatisation leads to the following equivalence:

$$(pt = 1) \cdot ((p_1 + p_2) \cdot t)^* \cdot (pt = 3 + pt = 4) \equiv (pt = 1 \cdot pt \leftarrow 4) + e \quad (14)$$

where e is a NetKAT expression containing the Kleene $*$. A counterexample can be immediately spotted, namely: $pt = 1 \cdot pt \leftarrow 4$. 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 and KA-UNROLL-L in Figure 2.

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 removing the PA-MOD-MOD and PA-FILTER-MOD axioms in Figure 2. 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 minimal forwarding paths within the network topology that lead from *in* to *out*. (2) Traversing the same path twice 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 path. This is also in accordance with the minimality criterion invoked in the seminal work on causal reasoning in [15], for instance. It is intuitive to see that given a NetKAT program $in \cdot (p \cdot t)^* \cdot out$ there is a sufficient number of $*$-unfoldings that can reveal all the relevant paths from *in* to *out*. As shown by our experimental evaluation, in most of the practical cases, it suffices to analyze paths of length equal with the size n of the network.

Theorem 1 states that safety in NetKAT programs reduces to showing that there are no paths from *in* to *out* for any hop-by-hop forwarding strategy on individual switches complying to a switch policy p. The result in Theorem 1 follows straightforwardly by Lemma 1 and Lemma 2.

Given a NetKAT policy q and a natural number m, we write q^m to denote the repeated application of q for m times:

$$q^m = \begin{cases} 1, & \text{if } m = 0 \\ q \cdot q^{m-1}, & \text{if } m \geq 1. \end{cases}$$

We call *repetitions* expressions of shape p^m.

Lemma 1. *Let p, t be two NetKAT policies. The following holds:*

$$\forall n \in \mathbb{N}.\ (1 + p \cdot t)^n \ \equiv\ 1 + p \cdot t + (p \cdot t)^2 + \ldots + (p \cdot t)^n \tag{15}$$

Proof. The proof follows immediately, by induction on n and by the Kleene Algebra axioms in Figure 2.
Base case: $n = 0$. If $n = 0$ then $(1 + (p \cdot t))^0 = 1$, inferred based on the definition of Kleisli composition.
Induction step: Assume (15) holds for all k such that $0 \leq k \leq n$. It follows that:

$$\begin{aligned}
(1 + p \cdot t)^{n+1} &\equiv_{\text{(Kleisli comp.)}} \\
(1 + p \cdot t)^n \cdot (1 + p \cdot t) &\equiv_{\text{(ind. hypo.)}} \\
(1 + p \cdot t + (p \cdot t)^2 + \ldots + (p \cdot t)^n) \cdot (1 + p \cdot t) &\equiv_{\text{(KA-SEQ-DIST-L/R, KA-PLUS-IDEM)}} \\
1 + p \cdot t + (p \cdot t)^2 + \ldots + (p \cdot t)^n + & \\
p \cdot t + (p \cdot t)^2 + \ldots + (p \cdot t)^n + (p \cdot t)^{n+1} &\equiv_{\text{(KA-PLUS-IDEM)}} \\
1 + p \cdot t + (p \cdot t)^2 + \ldots + (p \cdot t)^n + (p \cdot t)^{n+1} &
\end{aligned}$$

Hence, (15) holds. □

Lemma 2. *Let p, t, in, out be NetKAT policies. The following holds:*

$$\forall n \in \mathbb{N}.\ in \cdot (1 + p \cdot t)^n \cdot out \ \leq\ in \cdot (p \cdot t)^* \cdot out \tag{16}$$

Proof. Consider $n \in \mathbb{N}$. First, observe that

$$\begin{aligned}
&in \cdot (p \cdot t)^* \cdot out \equiv \\
&in \cdot (1 + p \cdot t + (p \cdot t)^2 + \ldots + (p \cdot t)^n + (p \cdot t)^{n+1} \cdot (p \cdot t)^*) \cdot out
\end{aligned} \tag{17}$$

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

$$in \cdot (p \cdot t)^* \cdot out \equiv in \cdot (1 + p \cdot t)^n \cdot out + in \cdot (p \cdot t)^{n+1} \cdot (p \cdot t)^* \cdot out \quad (18)$$

Therefore,

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

holds by the definition of the partial order relation $\leq$. □

Theorem 1. *(Approximation Principle for Safety) Assume a network topology t, a switch policy p, an ingress policy in, and an egress policy out encoding the hazard. The following holds:*

$$\vdash in \cdot (p \cdot t)^* \cdot out \equiv 0 \;\text{ iff }\; \forall n \in \mathbb{N}. \vdash in \cdot (1 + p \cdot t)^n \cdot out \equiv 0 \quad (19)$$

Proof. The "if" case follows immediately, as by Lemma 2, the hypothesis $in \cdot (p \cdot t)^* \cdot out \equiv 0$ and the fact that $0 \leq q$ for all NetKAT policies q, the following holds:

$$\forall n \in \mathbb{N}.\; 0 \leq in \cdot (1 + p \cdot t)^n \cdot out \;\; \leq \;\; in \cdot (p \cdot t)^* \cdot out \equiv 0.$$

For the "only if" case we proceed by reductio ad absurdum.
Assume $\forall n \in \mathbb{N}. \vdash in \cdot (1 + p \cdot t)^n \cdot out \equiv 0$ and

$$in \cdot (p \cdot t)^* \cdot out \not\equiv 0. \quad (20)$$

By the definition of the Kleene $*$ and the assumption in (20), it follows that there exists $m \in \mathbb{N}$ such that:

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

By Lemma 1, we can see that the latter contradicts the hypothesis. Hence, our assumption is false. □

Remark 2 (Construction of $\vdash_s$)**.** *With these ingredients at hand, in accordance with Q_1 and Q_2, we consider an alteration of the NetKAT axiomatisation. Recall that our NetKAT policies do not use* **dup***. Our approach is purely syntactic (it does not involve network packet analysis) and it looks at each operation within a NetKAT expression, in a "small-step" fashion. This can be achieved by removing the axioms PA-MOD-MOD and PA-FILTER-MOD.*

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

Remark 3. *Note that $\vdash_s$ is no longer complete. Nevertheless, the purpose of $\vdash_s$ is not to prove equivalence of arbitrary* NetKAT$^{\textbf{-dup,*}}$*, but to identify safety failure violations and corresponding explanations. In what follows, we show a series of useful/interesting properties of $\vdash_s$.*

Theorem 2 (Consistency of $\vdash_s$)**.** *Assume a* NetKAT$^{\textbf{-dup,*}}$ *policy p. The following holds:*

$$\vdash p \equiv 0 \text{ iff } \vdash_s p \equiv 0 \tag{21}$$

Proof. The key observation behind this proof is that 0-terms can only be derived according to the BA/PA-CONTRA axioms:

$$\begin{aligned} a \cdot \neg a &\equiv 0 \\ f = n \cdot f = n' &\equiv 0 \quad \text{if } n \neq n' \end{aligned}$$

The removed axiom PA-MOD-MOD

$$f \leftarrow n \cdot f \leftarrow n' \equiv f \leftarrow n'$$

can only involve tests when used in combination with the PA-MOD-FILTER axiom:

$$f \leftarrow n \cdot f = n \equiv f \leftarrow n$$

This implies:

$$f \leftarrow n \cdot f \leftarrow n' \equiv f \leftarrow n \cdot f = n \cdot f \leftarrow n' \cdot f = n'$$

Nevertheless, the right hand side of the above reduction can never be evaluated to 0 as commutativity of $\leftarrow$ and $=$ is only allowed in the context of different fields, according to small PA-MOD-FILTER-COMM:

$$f \leftarrow n \cdot f' = n' \equiv f' = n' \cdot f \leftarrow n \;\; \text{if } f \neq f'$$

Moreover, it is straightforward to see that PA-FILTER-MOD

$$f = n \cdot f \leftarrow n \equiv f = n$$

has no influence on the evaluation to 0-terms, as tests are not removed by this axiom.

It is, therefore, safe to conclude that (21) holds. □

Hence, according to Theorem 1 and Theorem 2, we can conclude that a network behaviour is "in-out-safe" whenever it can be proven so according to $\vdash_s$:

Corollary 1 (Safety Sound & Complete). *Assume the* NetKAT$^{\textbf{-dup,*}}$ *policies encoding a network topology* t*, a switch policy* p*, an ingress policy* in*, and an egress policy* out *encoding the hazard. The following holds:*

$$\vdash in \cdot (p \cdot t)^* \cdot out \equiv 0 \text{ iff } \forall n \in \mathbb{N}.\ \vdash_s in \cdot (1 + p \cdot t)^n \cdot out \equiv 0 \tag{22}$$

As previously stated, our experimental evaluation showed that in most of the cases it suffices to consider a limited number of $*$-unfoldings equal to the size n of the network, in order to reveal all the possible ways of reaching a hazardous egress out from a given ingress in. In accordance, we introduce a notion of so-called n-safety failure explanations.

Definition 2 (n-Safety Failure Explanations). *Assume the* NetKAT$^{\textbf{-dup,*}}$ *policies encoding a network topology* t*, a switch policy* p*, an ingress policy* in*, and an egress policy* out *encoding the hazard. An* n-safety failure explanation *is a policy* $expl \not\equiv 0$ *such that, for* $n \in \mathbb{N}$*:*

$$\vdash_s in \cdot (1 + p \cdot t)^n \cdot out \equiv expl. \tag{23}$$

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 3. 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 $\vdash$. Note that the network consists of 6 forwarding links. Hence, 6 unfoldings were sufficient for the new axiomatisation to entail the following explanation:

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

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

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

shows how packets at port 3 can reach port 2.

Remark 4. *The work in [6] proposes a "star elimination" method for switch policies not containing* **dup** *and switch assignments. The procedure in [6] 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 [6] "forget" the intermediate sequences of assignments and tests, and reduce policies to sums of expressions of shape* $(f_1 = v_1. \dots .f_n = v_n)\cdot(f_1 \leftarrow v_1'. \dots .f_n \leftarrow v_n')$ *where* $f_1, \dots, f_n$ *are the packet fields. Hence, the normal forms exploited by the star elimination in [6] can not serve as comprehensive failure explanations.*

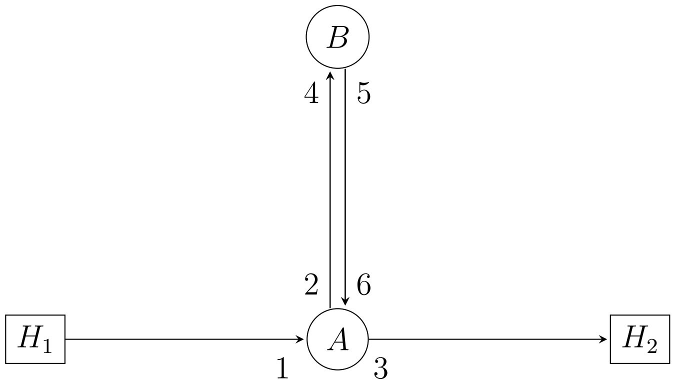

Figure 4: A Firewall

We next provide an additional firewall example to better illustrate the ideas in Remark 4. Consider a scenario where there are two hosts H_1 and H_2, a switch A, and a firewall B, as displayed in Figure 4. In this setting the packets that reach A are first forwarded to the firewall, and then to their destination, and the firewall blocks all non-SSH traffic. The policy and the topology are defined as follows:

$$\begin{aligned} p \triangleq\ & sw = A \cdot (dst = H_2 \cdot firewalled = 0 \cdot pt \leftarrow 2 + \\ & \qquad\qquad dst = H_2 \cdot firewalled = 1 \cdot pt \leftarrow 3) + \\ & sw = B \cdot (typ = SSH \cdot firewalled \leftarrow 1 \cdot pt \leftarrow 5) \end{aligned}$$

$$\begin{aligned} t \triangleq\ & sw = A \cdot (pt = 2 \cdot sw \leftarrow B \cdot pt \leftarrow 4 + pt = 1 + pt = 3) + \\ & sw = B \cdot pt = 5 \cdot sw \leftarrow A \cdot pt \leftarrow 6 \end{aligned}$$

Assume that packets from H_1 reaching to H_2 constitutes a safety violation. The *in* and *out* are defined as follows:

$$\begin{aligned} in &\triangleq sw = A \cdot pt = 1 \cdot dst = H_2 \cdot firewalled = 0 \\ out &\triangleq sw = A \cdot pt = 3 \end{aligned}$$

Generally speaking, we are interested to check whether $in \cdot (p \cdot t)^* \cdot out$ reduces to 0 (indicating the absence of the hazard) or not. Based on the framework devised in this paper, this reduces to checking the aforementioned equalities after unfolding the expression $(p \cdot t)^*$ for a number of times equal to the number of (oriented) links in the network. It is clear that in our case we are interested to check whether $in \cdot (p \cdot t)^4 \cdot out \equiv 0$ or not. Our framework gives the following counterexample:

$$\begin{aligned} & sw = A \cdot pt = 1 \cdot dst = H_2 \cdot firewalled = 0 \cdot \\ & pt \leftarrow 2 \cdot sw \leftarrow B \cdot pt \leftarrow 4 \cdot \\ & typ = SSH \cdot firewalled \leftarrow 1 \cdot pt \leftarrow 5 \cdot sw \leftarrow A \cdot pt \leftarrow 6 \cdot \\ & pt \leftarrow 3 \end{aligned}$$

Remark 5. *In [6], the completeness theorem of NetKAT is based on a language model:*

$$\alpha \cdot \pi_0 \cdot \ \mathbf{dup} \ \cdot \pi_1 \cdot \ \mathbf{dup} \ \ldots \ \mathbf{dup} \cdot \pi_n \tag{24}$$

where $\alpha \triangleq f_1 = n_1 \ldots f_k = n_k$ *is called a complete test and* $\pi \triangleq f_1 \leftarrow n_1 \ldots f_k \leftarrow n_k$ *is called a complete assignment. Note that the axiom that we removed, PA-MOD-MOD, plays an important role in bringing the expressions into this form. If we had strictly followed the approach in [6], then for the above firewall example we would have obtained a counterexample of the following shape:*

$$\begin{aligned} & (sw = A \cdot pt = 1 \cdot dst = H_2 \cdot typ = SSH \cdot firewalled = 0) \cdot \\ & (sw \leftarrow A \cdot pt \leftarrow 1 \cdot dst \leftarrow H_2 \cdot typ \leftarrow SSH \cdot firewalled \leftarrow 0) \cdot \mathbf{dup} \cdot \\ & (sw \leftarrow B \cdot pt \leftarrow 4 \cdot dst \leftarrow H_2 \cdot typ \leftarrow SSH \cdot firewalled \leftarrow 0) \cdot \mathbf{dup} \cdot \\ & (sw \leftarrow A \cdot pt \leftarrow 6 \cdot dst \leftarrow H_2 \cdot typ \leftarrow SSH \cdot firewalled \leftarrow 1) \cdot \mathbf{dup} \cdot \\ & (sw \leftarrow A \cdot pt \leftarrow 3 \cdot dst \leftarrow H_2 \cdot typ \leftarrow SSH \cdot firewalled \leftarrow 1) \end{aligned} \tag{25}$$

Observe that a more concise, **dup***-free counterexample is obtained from our approach, which we believe is better suitable in the context of causality checking. Furthermore, certain information has been lost in the expression in (25), i.e. the assignments* $pt \leftarrow 2$ *and* $pt \leftarrow 5$ *do not appear in the counterexample. More generally, if there exist more than one assignment to a field inside* $p \cdot t$*, then only the last assignment is preserved. We believe this is not favorable for causality checking.*

3.2. Minimal Explanations

Note that the safety failure explanations in Definition 2 are not minimal. For an example, there might be cases in which two explanation paths of shape

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

are identified. In this case, we consider e_1 as more "expressive" than e_2. In this section we introduce a notion of minimality, inspired by the seminal works on causal reasoning in [15, 16]. We define minimality based on a notion of NetKAT *normal forms for safety* (NFS). These normal forms are derived based on the additional equalities in Theorem 3.

Theorem 3 (Distribution of ¬). *Let a, b and $f = n_i$ for $i \in \{1, \ldots, m\}$ stand for NetKAT predicates as in Figure 1. The following hold:*

$$\begin{array}{rcll}
\neg 1 & \equiv & 0 & \text{NEG-ONE} \\
\neg 0 & \equiv & 1 & \text{NEG-ZERO} \\
\neg(\neg a) & \equiv & a & \text{NEG-NEG} \\
\neg(f = n_i) & \equiv & \Sigma_{j \neq i} f = n_j & \text{NEG-ELIM} \\
\neg(a + b) & \equiv & (\neg a) \cdot (\neg b) & \text{DIST-NEG-DISJ} \\
\neg(a \cdot b) & \equiv & (\neg a) + (\neg b) & \text{DIST-NEG-CONJ}
\end{array}$$

Proof Sketch. All the above equivalences follow according to the NetKAT semantics in Figure 1. Consider, for instance, NEG-ONE. The following holds:

$$\begin{array}{rl}
\forall h \in \mathrm{H}: & [\![\neg 1]\!]h =_{(def.of \neg)} \\
& \{h\} \setminus ([\![1]\!]h) =_{(def.of\,1)} \\
& \{h\} \setminus \{h\} = \\
& \{\} =_{(def.of\,0)} \\
& [\![0]\!]h.
\end{array}$$

□

Definition 3 (Token). *We call a* token *the identity policy* 1*, the drop policy* 0*, a test $(f = n)$, or a field modification $f \leftarrow n$.*

Definition 4 (NFS). *A NetKAT policy p is in* NFS *if*

$$p \triangleq \Sigma_{i \in \{1,\ldots,m\}} \, \Pi_{j \in \{1,\ldots,n\}} tk_{i,j}$$

with $tk_{i,j}$ a token, for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

Theorem 4 (NFS reduction). *All policies defined over* NetKAT$^{\textbf{-dup,*}}$ *and repetitions can be reduced to equivalent policies in NFS.*

Proof Sketch. Let p^u denote the repetition-free policy obtained from p by performing all corresponding unfoldings, if any. It can be shown by induction on the structure of p^u that an NFS can be obtained by applying the NetKAT axioms in Figure 2, together with the equalities in Theorem 3 (in particular, KA-SEQ-DIST-L and KA-SEQ-DIST-R). □

Definition 5 ($\sqsubseteq$ / $\sqsubset$). *Let p_i and $p_j^{'}$ be NetKAT policies in NFS. We write $p_i \sqsubseteq p_j^{'}$ whenever p_i can be obtained from $p_j^{'}$ by deleting k atoms at arbitrary positions in $p_j^{'}$, with $k \geq 0$. We write $p_i \sqsubset q_i$ whenever $k > 0$.*

Definition 6 (Minimality). *We call a policy in NFS* minimal*, with*

$$p \triangleq \Sigma_{i \in \{1,\dots,n\}} p_i$$

whenever for all p_j there is no p_k, with $j, k \in \{1, \dots, n\}$ such that $p_j \sqsubset p_k$.

Assume p is in NFS, but is not minimal. We write $min(p)$ for the NFS policy obtained by removing all p_k, with $k \in \{1, \dots, n\}$, such that there exists p_j, with $j \in \{1, \dots, n\}$, satisfying $p_j \sqsubset p_k$.

Assume an explanation $expl \not\equiv 0$ as in (23). Let $expl^{NFS}$ be $expl$ reduced to its NFS. The minimal explanation with respect to the violation of a safety property in NetKAT is represented by $min(expl^{NFS})$.

4. Tools for Explaining NetKAT Safety Failures

In this section we introduce *SDN–SafeCheck*, a tool based on Maude [8], for automatically computing relevant explanations for failures of NetKAT programs. Maude has been proven particularly suitable for defining semantics of programming languages and reasoning about their properties. The Maude tools encompass, amongst others, a suite of model checkers and the so-called Maude Formal Environment (MFE) [17] which includes the Church-Rosser checker and the termination tool. In short, *SDN–SafeCheck* is based on Maude equational theories and it satisfies important properties such as Church-Rosser (which guarantees uniqueness of results) and termination. *SDN–SafeCheck* provides all the explanations for NetKAT safety failures.

4.1. A Brief Overview of the Maude System

Maude specifications come in two flavours: (1) as functional modules, that define data types and associated operations by means of equational theories, or (2) as system modules, or rewrite theories, that specify concurrent transitions given as a set of rewrite rules, or “oriented” equations. Such rules are triggered whenever the rule’s left hand side matches a fragment of the system state and the rule’s condition is satisfied. In this work we utilize Maude functional modules and in the following we discuss the main aspects of Maude functional modules. We then continue with a brief overview of the MFE.

Functional modules. For an intuitive example, we next provide a Maude equational theory specifying NetKAT predicates. First, note that a functional module is specified using the following syntax:

$$\texttt{fmod}\ \mathit{ModuleName}\ \texttt{is}\ \mathit{DeclarationsAndStatements}\ \texttt{endfm} \tag{26}$$

In our case, the module name is `PREDICATE`, whereas the *DeclarationsAndStatements* includes, amongst others, the operators defined according to the syntax in Figure 1, and the associated axioms in Figure 2. Operators are specified over types, or Maude *sorts*, defined within the current module via the keyword `sort`, or imported (possibly in a “protected” fashion) from other modules. Properties such as associativity (`assoc`), commutativity (`comm`), idempotency (`idem`), neutral elements (`id:`) and precedence (`prec`) can be specified as attributes of operators. Note that associativity and idempotency cannot be used together in any combination of attributes. Operators that play the role of constructors (`ctor`) for a certain type can also be specified; this is the case of all the operators defining Predicates in Figure 1. Variables (`var`) of a certain sort can also be declared. Possibly conditional equations are introduced using `eq` or `ceq`, respectively. Identifiers can be specified for equations as well. Comments are preceded by `---`.

A Maude equational theory specifying NetKAT predicates and the additional boolean algebra axioms is given in Figure 5.

The *identity* and *drop* NetKAT policies are defined in terms of two constants (or operators with arity 0), namely, the constructors `one` and `zero`, respectively. *Tests*, *disjunction* and, respectively, *conjunction* are straightforwardly implemented as the Maude binary operators `_=_`, `_+_` and, respectively, `_._`.

Note that conjunction and disjunction are declared as associative and commutative as well. This is in accordance with the NetKAT axioms KA-PLUS-ASSOC, KA-SEQ-ASSOC, KA-PLUS-COMM and BA-SEQ-COMM in Figure 2. The advantage of using operator attributes is that Maude will efficiently perform equational reasoning modulo these attributes. *Negation* is given as the unary operator `~_`. The remaining predicate axioms are specified via the equations in Figure `[BA-PLUS-ONE]`, `[KA-PLUS-ONE]`, `[KA-ONE-SEQ]`, `[KA-ZERO-SEQ]`, `[BA-EXCL-MID]`, `[BA-CONTRA]` and `[BA-SEQ-IDEM]`. Note that KA-SEQ-ONE and KA-SEQ-ZERO in Figure 2 hold implicitly, due to the commutativity of sequential composition of NetKAT predicates.

Fields and their (natural) values are data structures defined within the corresponding Maude functional modules `FIELD` and `NATVAL`, which `PREDICATE` is importing in a protected manner.

The MFE. In our approach, we are using: Maude 2.7.1 for Linux64[1], MFE 1.0b[2] including the Church-Rosser Checker (CRC) 3p, and the Maude Termination Tool (MTT) 1.5j, and AProVE [18].

CRC plays a crucial role in resolving possibly different evaluations of a certain term by suggesting a series of so-called critical pairs. Intuitively, the latter are lemmas which, if proven correct, lead to a confluent equational specification. For instance, `PREDICATE` is Church-Rosser because the following lemmas were soundly added to the specification of NetKAT predicates in Figure 5, according to the additional equalities in Theorem 3:

```
eq ~ one = zero .     eq ~ zero = one .
```

4.2. Immediate Challenges and Observations

In Figure 5 we presented a straightforward implementation of NetKAT predicates in Maude. Next, we wanted to follow a similar approach and devise a Maude equational specification of NetKAT programs $in \cdot (1 + p \cdot t)^n \cdot out$ as in (23). Recall that such programs are expressions defined over NetKAT$^{\textbf{-dup,*}}$ and repetitions $(-)^n$.

Typically, specifying such NetKAT policies would consist in the following straightforward steps:

1. Define a new sort `Policy` as a suprasort of `Predicate`.

[1]`http://maude.cs.illinois.edu/w/index.php/All_Maude_2_versions`

[2]`https://github.com/maude-team/MFE/wiki/How-to-use-the-tool`

```
(fmod PREDICATE is
protecting FIELD .
protecting NATVAL .

sort Predicate .
var A : Predicate .

op one : -> Predicate [ctor] .
op zero : -> Predicate [ctor] .

op _=_ : Field NatVal -> Predicate [ctor prec 39] .
op _+_ : Predicate Predicate -> Predicate
                        [ctor assoc comm prec 43] .
op _._ : Predicate Predicate -> Predicate
                        [ctor assoc comm prec 40] .
op ~_ : Predicate -> Predicate [ctor prec 39] .

eq [BA-PLUS-ONE] : A + one = one .
eq [KA-PLUS-ZERO] : A + zero = A .
eq [KA-ONE-SEQ] : one . A = A .
eq [KA-ZERO-SEQ] : zero . A = zero .
eq [BA-EXCL-MID] : A + ~ A = one .
eq [BA-CONTRA] : A . ~ A = zero .
eq [BA-SEQ-IDEM] : A . A = A .

eq ~ one = zero .
eq ~ zero = one .
endfm)
```

Figure 5: Equational Theory of NetKAT Predicates.

2. Lift the signatures of $+$ and $\cdot$ to `Policy`.
3. Define $\leftarrow$ and the repetition operator $(-)^n$ accordingly.
4. Add the relevant set of axioms in Figure 2 as Maude equations. (Recall that our approach for explaining safety failures discards the axioms for $*$, **dup**, PA-MOD-MOD and PA-FILTER-MOD.)

Unfortunately, the recipe above was not successful. We proceed by describing the main difficulties we encountered.

Commutativity of $\cdot$ *.* Note that, on the one hand, the NetKAT $\cdot$ operator plays the role of conjunction in the context of predicates and is, therefore, commutative. On the other hand, $\cdot$ in the context of policies denotes sequential composition, which is not commutative. Nevertheless, the packet algebra axioms in Figure 2 use $\cdot$ in a uniform fashion, thus, implicitly lifting $\cdot$ to the setting of policies as in step 2 above. Consequently, defining in Maude two operators capturing the two different semantics of $\cdot$, and straightforwardly translating the axioms in Figure 2 into equation is not an option.

Negation. The CRC returned a large number of critical pairs that involved the negation operator. Some of the pairs indicated the necessity of distributing negation over disjunction and conjunction as in Theorem 3. In accordance, we considered:

$$\begin{array}{rcll} \neg(a+b) & \equiv & (\neg a)\cdot(\neg b) & \text{DIST-NEG-DISJ} \\ \neg(a\cdot b) & \equiv & (\neg a)+(\neg b) & \text{DIST-NEG-CONJ} \end{array} \tag{27}$$

Nevertheless, this did not help us eliminate all critical pairs either. Hence, we decided to apply a preprocessing step that reduces arbitrary NetKAT policies to equivalent negation-free policies in two steps. First, negations are pushed to the level of NetKAT predicates $f = n_i$ according to (27). Then, each negated predicate $\neg(f = n_i)$ is soundly replaced according to:

$$\neg(f = n_i) \equiv \Sigma_{j\neq i} f = n_j \qquad \text{NEG-ELIM} \tag{28}$$

As in [6], field values are drawn from finite domains.

Distributivity. We also noticed that the distributivity axioms BA-PLUS-DIST, KA-SEQ-DIST-L and KA-SEQ-DIST-R contribute to the violation of the Church-Rosser property when used together within the equational theory of policies. For instance,

$$(a+b)\cdot(a+c)$$

can be reduced according to BA-PLUS-DIST to:

$$a + b \cdot c \tag{29}$$

and it can be reduced according to KA-SEQ-DIST-R and BA-SEQ-IDEM, to:

$$a + b \cdot a + a \cdot c + b \cdot c. \tag{30}$$

From the perspective of safety failure explanations, the policy in (30) subsumes its counterpart in (29). Hence, BA-PLUS-DIST can be discarded as well.

4.3. Equational Specifications for Explaining Failures

In this section we introduce *SDN−SafeCheck*, a tool for explaining NetKAT safety failures. *SDN−SafeCheck* is based on the Maude equational specification NetKAT$^{\textbf{-dup,*}}$, implemented in a manner that enables accommodating the ideas in Section 4.2. The functional modules behind *SDN−SafeCheck* are proven Church-Rosser and terminating. Hence, *SDN−SafeCheck* provides the unique solution encoding all relevant explanations on how packets can travel from a specified ingress to the undesired egress.

Assume the NetKAT$^{\textbf{-dup,*}}$ policies encoding a network topology t, a switch policy p, an ingress policy in, and an egress policy out encoding an undesired property. Let $P \triangleq in \cdot (1 + p \cdot t)^n \cdot out$ be the corresponding NetKAT program to be analyzed for safety failures. *SDN−SafeCheck* works in three steps.

(I) Firstly, the tool recursively unfolds the policy $(1 + p \cdot t)^n$ into a term U. Then, U is reduced to a term F uniquely expressed as a sum of policies that are union-free and negation-free. This is achieved in accordance with the equivalences (27) and (28) in Section 4.2, and with the distributivity axioms KA-SEQ-DIST-L and KA-SEQ-DIST-R, respectively.

(II) Next, F is reduced to F' according to the relevant NetKAT axioms implemented in Maude in a slightly modified fashion, due to the issues related to the commutativity of $\cdot$, as discussed in Section 4.2.

For an intuition, consider a (possibly conditional) NetKAT axiom generically denoted by $l \cdot r \equiv t$ (if C). With a commutative $\cdot$, it might be the case that F can be equivalently represented as a term F' within which $l \cdot r$ can be matched (whenever C holds). Nevertheless, given that a commutative $\cdot$ could not be considered in the Maude specification of NetKAT policies, it might be the case that $l \cdot r$ does not match in F' (even if C holds). Consequently, the aforementioned axiom might not be employed by the Maude equational reduction procedure, when starting with F'.

The solution is to enable sound reductions according to $l \cdot r \equiv t$ (if C), in all possible contexts. More precisely, each such axiom is implemented via a set of equations of shape:

$$\begin{aligned} l \cdot r &\equiv t \text{ (if } C) \\ l \cdot M \cdot r &\equiv t \text{ (if } C \text{ and } C_s) \end{aligned}$$

where M is a policy term and C_s is a condition that ensures the sound application of the newly introduced equations. For an example, we next provide a corresponding Maude implementation of the PA-CONTRA.

```
ceq (F1 = I1) . (F1 = I2) = zero if I1 =/= I2 .
ceq (F1 = I1) . M . (F1 = I2) = zero if I1 =/= I2 /\ not (F1 <- I2 occursInner M)  .
```

Intuitively, `(F1 <- I2 occursInner M)` checks whether the field modification `F1 <- I2` occurs within the policy `M`. `(F1 <- I2 occursInner M)` is evaluated to `true` whenever the field modification `F1 <- I2` occurs within `M`. Otherwise, `(F1 <- I2 occursInner M)` is evaluated to `false`. We negate the result obtained from performing this check and this way, the second equation soundly equates its left-hand side to `zero`, as the field `F1` is never modified with the value `I2` within `M` and the initial value of the field `F1` is different than `I2`, hence the test `F1 = I2` will always fail.

We then apply certain axioms in order to simplify the expressions. For an example, we provide the implementation of BA-SEQ-IDEM axiom.

```
eq A . A = A .
ceq (F1 = I1) . M . (F1 = I1) = (F1 = I1) . M if M ? F1 .
```

where `A` is of sort predicate. The operator `?` works in a similar fashion to the operator `occursInner`. Intuitively, `occursInner` checks whether a specific term occurs inside a given policy, whereas the operator `?` only checks whether there exist an assignment to a field in a given policy. The term `M ? F1` is evaluated to `true` whenever `F1` is not modified within `M`. Otherwise, `M ? F1` is evaluated to `false`. This way, it is ensured that the term `F1 = I1` can commute inside the terms in `M` as `F1` is not modified within `M`, and then BA-SEQ-IDEM axiom can be applied.

Another phase in this step is to define a total order between the fields and reorder the terms according to this total order. This phase is needed to obtain canonical forms. We introduce the operator `<` to define the total order and we then apply the following equations to bring the expressions into a canonical form.

```
ceq (F1 <- I1) . (F2 <- I2) = (F2 <- I2) . (F1 <- I1) if F1 < F2 .
ceq (F1 = I1) . (F2 = I2) = (F2 = I2) . (F1 = I1) if F1 < F2 .
```

(III) Last, but not least, if the reduction at step (II) returns the unique term $F'' \not\equiv 0$ encoding all safety failure explanations, then *SDN−SafeCheck* computes all relevant explanations when starting with F'', according to the minimization procedure in Section 3.2.

The full implementation of *SDN−SafeCheck* can be downloaded at: `https://gitlab.inf.uni-konstanz.de/huenkar.tunc/sdn-safecheck`.

5. Experimental Evaluation

We performed experiments to evaluate the performance of our implementation on the publicly available Topology Zoo dataset [12] which consist of 261 real-world network topologies. Given that, in essence, safety failure analysis reduces to reachability analysis, in our experiments we analyzed the time required to check for reachability within these topologies. More precisely, we checked point-to-point reachability between the two nodes in the longest path within the network. If there were more than one such paths, then an arbitrary choice was made. We encoded the topologies in the dataset into NetKAT and generated a destination-based shortest path policy to connect each node with every other node by using an automated procedure similar to the one in [19]. The encoded topologies are made available in the link above alongside the implementation of the tool. All the experiments were performed on a computer running Ubuntu 18.04 LTS with 8 core 3.7GHz AMD Ryzen 7 2700x processors and 32 GB RAM.

A scatter plot of the obtained execution times is sketched in Figure 6. We set a time limit of 12000 seconds for checking the reachability property. For three topologies the computation did not finish under this time limit. The networks for which the computation timed out consist of 754, 197 and 153 nodes, and correspond to first, second and fourth largest network in the Topology Zoo dataset, respectively. The results show that for networks up to 70 switches a result is obtained under 60 seconds in most cases. For networks with more than 70 switches the variance of the obtained execution times is

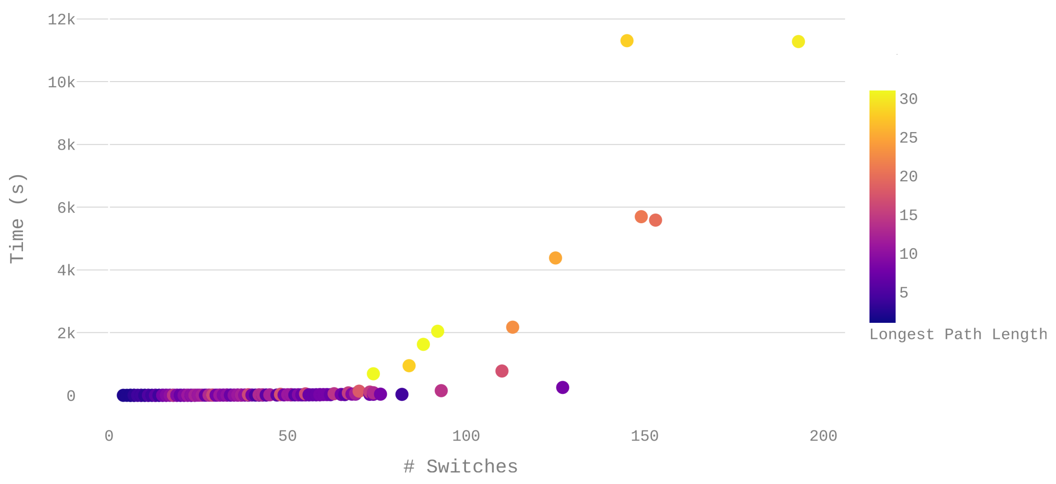

Figure 6: Experimental results

higher. We observe that the longest path length plays a significant role in determining the running time of *SDN−SafeCheck* as networks grow in size.

The execution time can be divided into two categories: IO time and analysis time. The IO time corresponds to the time frame in which the expressions are written into a file and loaded into Maude. Analysis time corresponds to the time frame in which the rewriting and the failure analysis is performed. In Figure 7 we display a comparison between the time taken for IO and the time taken for performing the analysis. We observe that the IO time dominates the total execution time.

6. Conclusions

In this paper we formulate a notion of safety in the context of NetKAT programs [6] 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, as shown by the experimental evaluation, is parametric on the size of the underlying network topology. The new equational system is not complete, as some of the original NetKAT axioms have been removed to enable more comprehensive failure explanations. Nevertheless, the purpose of our framework is not to reason about equivalence, but to identify safety failure violations and corresponding explanations.

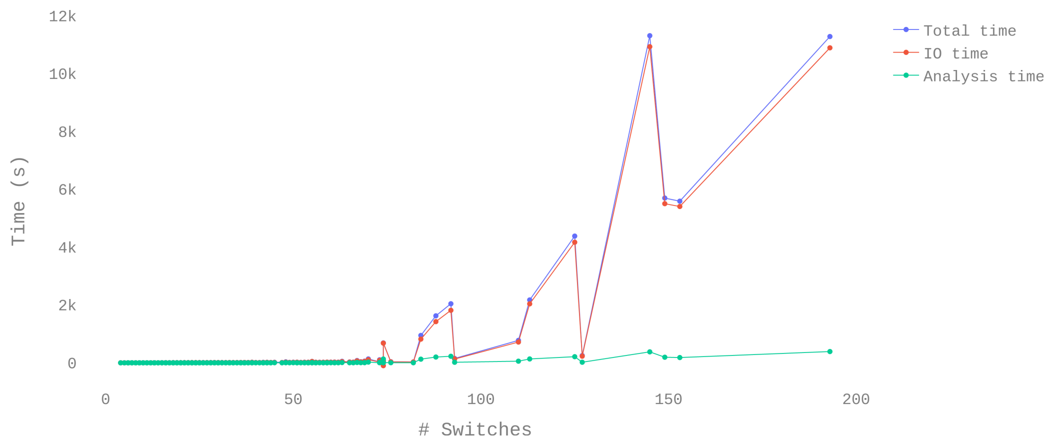

Figure 7: Time comparisons

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, 21], for instance. The Maude system was exploited for implementing *SDN−SafeCheck* tool for automatically computing safety failure explanations. Corresponding experimental evaluation based on the Topology Zoo dataset [12] is also provided.

The results in this paper are part of a larger project on (counterfactual) causal reasoning on NetKAT. In [22], 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 of 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 authors are grateful to Francisco Durán, Steven Eker and the Maude/RL community for their useful comments on using the Maude Formal Environment, and to the reviewers of FROM 2019, for their feedback and observations. Special thanks are addressed to Marcello Bonsangue and Tobias Kappé, for their insight into the formal foundations of NetKAT. Many thanks to Hossein Hojjat and Dang Mai for their insight into the behaviour of SDNs and associated programming languages. This work was supported by the DFG project "CRENKAT", proj. no. 398056821.

References

[1] C. Buckl, A. Knoll, I. Schieferdecker, J. Zander, Model-based analysis and development of dependable systems, in: H. Giese, G. Karsai, E. Lee, B. Rumpe, B. Schätz (Eds.), Model-Based Engineering of Embedded Real-Time Systems - International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. Revised Selected Papers, Vol. 6100 of Lecture Notes in Computer Science, Springer, 2007, pp. 271–293. `doi:10.1007/978-3-642-16277-0\_10`.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, J. S. Turner, OpenFlow: enabling innovation in campus networks, Computer Communication Review 38 (2) (2008) 69–74. `doi:10.1145/1355734.1355746`.

[3] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, D. Walker, 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, 2011, pp. 279–291. `doi:10.1145/2034773.2034812`.

[4] A. Voellmy, P. Hudak, Nettle: A Language for Configuring Routing Networks, in: W. M. Taha (Ed.), Domain-Specific Languages, IFIP TC 2 Working Conference, DSL 2009, Oxford, UK, July 15-17, 2009, Proceedings, Vol. 5658 of Lecture Notes in Computer Science, Springer, 2009, pp. 211–235. `doi:10.1007/978-3-642-03034-5_11`.

[5] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, P. Hudak, Maple: simplifying SDN programming using algorithmic policies, in: ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013, 2013, pp. 87–98. doi:10.1145/2486001.2486030.

[6] C. J. Anderson, N. Foster, A. Guha, J. Jeannin, D. Kozen, C. Schlesinger, D. Walker, 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, 2014, pp. 113–126. doi:10.1145/2535838.2535862.

[7] N. Foster, D. Kozen, M. Milano, A. Silva, L. Thompson, 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, 2015, pp. 343–355. doi:10.1145/2676726.2677011.

[8] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. L. Talcott, The Maude 2.0 System, in: R. Nieuwenhuis (Ed.), Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings, Vol. 2706 of Lecture Notes in Computer Science, Springer, 2003, pp. 76–87. doi:10.1007/3-540-44881-0_7.

[9] I. Pelle, A. Gulyás, An extensible automated failure localization framework using NetKAT, Felix, and SDN traceroute, Future Internet 11 (5) (2019). doi:10.3390/fi11050107.

[10] Y. Deng, M. Zhang, G. Lei, An Algebraic Approach to Automatic Reasoning for NetKAT Based on Its Operational Semantics, in: Z. Duan, L. Ong (Eds.), Formal Methods and Software Engineering - 19th International Conference on Formal Engineering Methods, ICFEM 2017, Xi'an, China, November 13-17, 2017, Proceedings, Vol. 10610 of Lecture Notes in Computer Science, Springer, 2017, pp. 464–480. doi:10.1007/978-3-319-68690-5_28.

[11] G. Caltais, Explaining SDN Failures via Axiomatisations, in: M. Marin, A. Craciun (Eds.), Proceedings Third Symposium on Working Formal Methods, FROM 2019, Timişoara, Romania, 3-5 September 2019, Vol. 303 of EPTCS, 2019, pp. 48–60. doi:10.4204/EPTCS.303.4.

[12] P. Gill, M. F. Arlitt, Z. Li, A. Mahanti, The flattening internet topology: Natural evolution, unsightly barnacles or contrived collapse?, in: M. Claypool, S. Uhlig (Eds.), Passive and Active Network Measurement, 9th International Conference, PAM 2008, Cleveland, OH, USA, April 29-30, 2008. Proceedings, Vol. 4979 of Lecture Notes in Computer Science, Springer, 2008, pp. 1–10. `doi:10.1007/978-3-540-79232-1\_1`.

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

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

[15] J. Y. Halpern, Causality, Responsibility, and Blame: A Structural-Model Approach, in: S. Benferhat, J. Grant (Eds.), Scalable Uncertainty Management - 5th International Conference, SUM 2011, Dayton, OH, USA, October 10-13, 2011. Proceedings, Vol. 6929 of Lecture Notes in Computer Science, Springer, 2011, p. 1. `doi:10.1007/978-3-642-23963-2\_1`.

[16] J. Y. Halpern, A Modification of the Halpern-Pearl Definition of Causality, in: Q. Yang, M. J. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, 2015, pp. 3022–3033.
URL `http://ijcai.org/Abstract/15/427`

[17] F. Durán, C. Rocha, J. M. Álvarez, Towards a Maude Formal Environment, in: G. Agha, O. Danvy, J. Meseguer (Eds.), Formal Modeling: Actors, Open Systems, Biological Systems - Essays Dedicated to Carolyn Talcott on the Occasion of Her 70th Birthday, Vol. 7000 of Lecture Notes in Computer Science, Springer, 2011, pp. 329–351. `doi:10.1007/978-3-642-24933-4\_17`.

[18] J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, R. Thiemann, Analyzing Program Termination and Complexity Automatically with AProVE, J. Autom. Reasoning 58 (1) (2017) 3–31. `doi:10.1007/s10817-016-9388-y`.

[19] R. Beckett, M. Greenberg, D. Walker, Temporal NetKAT, in: C. Krintz, E. Berger (Eds.), Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016, ACM, 2016, pp. 386–401. doi:10.1145/2908080.2908108.

[20] F. Leitner-Fischer, S. Leue, Causality Checking for Complex System Models, in: R. Giacobazzi, J. Berdine, I. Mastroeni (Eds.), Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings, Vol. 7737 of Lecture Notes in Computer Science, Springer, 2013, pp. 248–267. doi:10.1007/978-3-642-35873-9_16.

[21] G. Caltais, S. L. Guetlein, S. Leue, Causality for General LTL-definable Properties, in: B. Finkbeiner, S. Kleinberg (Eds.), Proceedings 3rd Workshop on formal reasoning about Causation, Responsibility, and Explanations in Science and Technology, CREST@ETAPS 2018, Thessaloniki, Greece, 21st April 2018., Vol. 286 of EPTCS, 2018, pp. 1–15. doi:10.4204/EPTCS.286.1.

[22] D. Lewis, Causation, Journal of Philosopy 70 (1973) 556–567.