Cost InterNetKAT: Basics of Algebraic Network Routing

Thesis submitted by

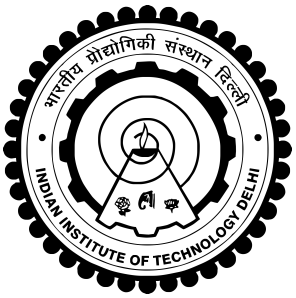
Avaljot Singh
2016CS50389

under the guidance of

Prof. Sanjiva Prasad

in partial fulfilment of the requirements
for the award of the degree of

Bachelor and Master of Technology



Department Of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI

January 2021

THESIS CERTIFICATE

This is to certify that the thesis titled **Cost InterNetKAT: Basics of Algebraic Network Routing**, submitted by **Avaljot Singh**, to the Indian Institute of Technology Delhi for the award of the degrees of **Bachelor and Master of Technology**, is a *bona fide* record of the research work done by him under my supervision. The contents of this thesis, in full or in part, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Sanjiva Prasad
Professor
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
New Delhi 110016 INDIA

Place: New Delhi

Date: January 2021

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor **Prof. Sanjiva Prasad** for his immense support, guidance and motivation during this project. His impeccable knowledge of the subject and his humility and approachable attitude has made this project a delight. I appreciate all the time and effort he invested in teaching and guiding me.

This project was a joint effort and it has also been a great pleasure to work with **Mankaran Singh** as a collaborator - who has taught and helped me a great deal through our stimulating interactions. I would also like to acknowledge **Arun Shankar** for forming a part the basis of this project.

I am also obliged to **Prof. Subodh Sharma** for his immense support in grooming me as a researcher. He has been a great source of support in giving me feedback on my ideas and research work and also, teaching me a great deal about a systematic approach to research for which I will be ever grateful. Further, I would like to thank **Prof. S Arun Kumar** for his valuable insights into this project.

Lastly, I would like to thank my family for being the backbone of round-the-clock support and motivation.

Avaljot Singh

ABSTRACT

KEYWORDS: Cost-InterNetKAT ; Kleene Algebra with Tests; Cost Algebra; NetKAT homomorphisms.

With the increasing use of SDNs in the modern world, there is also an increasing demand of a high level programming language that can abstract away the low level details of the network and help the programmer to write and verify the network policies and routing topologies. NetKAT, a new network programming language is based on a sound and complete mathematical foundation. It is essentially a Kleene Algebra with Tests (KAT) with primitives for modifying and testing packet headers, and encoding network topologies along with the axioms for reasoning about those constructs. Its applications include answering some important fundamental questions such as reachability, waypointing etc.

However, there are several ways in which we can improve the language and its expressiveness. NetKAT does not talk about the costs involved in routing packets along a certain path. Therefore, we wish to refine the semantics of NetKAT with a cost metric. We take into consideration that the costs come from the underlying topology and switching costs. While the tests and the packet modifications usually are of negligible cost, switching and routing along links on the other hand, are actually costly operations. The costs may vary with the routing device and the routing cost. Moreover, we wish to formulate policies that take into account the costs along different paths. Second, it does not talk about the NetKAT homomorphisms which can allow us to abstract out or refine certain details in one network to represent the routing details in another network. We may be able to use some of the reachability properties in the first network to define the reachability properties in the second instead of redefining them all over again. Third, NetKAT in its current equational theory does not allow the functionality of defining cross-network routing. Rather, it assumes a uniform network throughout in order to write routing policies.

In this work, we try to address some of these issues and propose the appropriate equational theory for Cost InterNetKAT.

Contents

List of Figures

ABBREVIATIONS

BGP	Border Gateway Protocol
IGP	Internal Gateway Protocol
EGP	External Gateway Protocol
SDN	Software Defied Network
DSL	Domain Specific Language
KAT	Kleene Algebra with Tests
ASE	Abstract Switching Element
BDD	Binary Decision Diagram
FDD	Forwarding Decision Diagram
IITD	Indian Institute of Technology, Delhi

NOTATION

f	Non-cost fields
c	Cost fields
pk	NetKAT packets
p, q	NetKAT policies
$\mathtt{cp}$	Checkpoint
α	Complete Test
π	Complete Assignment
β	Complete Non-cost Test
ϵ	Complete Local cost Test
θ	Complete Non-cost Assignment
κ	Complete Cost Assignment
A	Set of Complete tests
Π	Set of Complete assignments
$\mathbb{N}$	Natural numbers

Chapter 1

Introduction

1.1 Need for Algebraic Network Routing

The modern day internet is much more than just a graph of nodes and edges. Although originally developed as a simple graph structure, technological advances and growing needs of connectivity and security have increased its complexity. It now has a wide variety of special purpose devices such as routers, switches, repeaters, bridges, and many more. On one hand, this makes it very difficult for the network operators to come up with new routing protocols or even modify the existing ones to adjust them to their business needs and on the other hand, the *ad hoc* solutions to these problems do not offer any correctness guarantees. Even the industry standard network routing protocols like BGP often lack good behaviour in the sense that they do not offer convergence guarantees in asynchronous settings. As a result, the network developers fall back to solutions like using EGP protocols like BGP for IGP purposes. All these problems and many others have made the network outages very common. In the following sections, we briefly describe the existing infrastructure that is used for routing the network packets from one point in the network to another and the problems that are associated with these existing techniques.

1.2 Existing Infrastructure and related problems

1.2.1 Decentralised Network Routing

The fundamental tool used for network communications are the routing tables. Each network device uses its locally calculated routing table to determine the next hop for every incoming packet. Each device shares its own routing information with its neighbours and vice versa. The information thus received is used to update its own routing table. This decentralised way of working out the routing information reaches a fixed point and converges under ideal conditions. However, problems arise because the physical network links in the underlying topology go down quite frequently. Further, in modern networks, various factors other than the cost due to path length play an important role in deciding the network communications. For example, consider the network topology in Figure 1.1 Owing to some security issues, irrespective of any costs, C is required to route all the packets from source A to destination D and from source B to

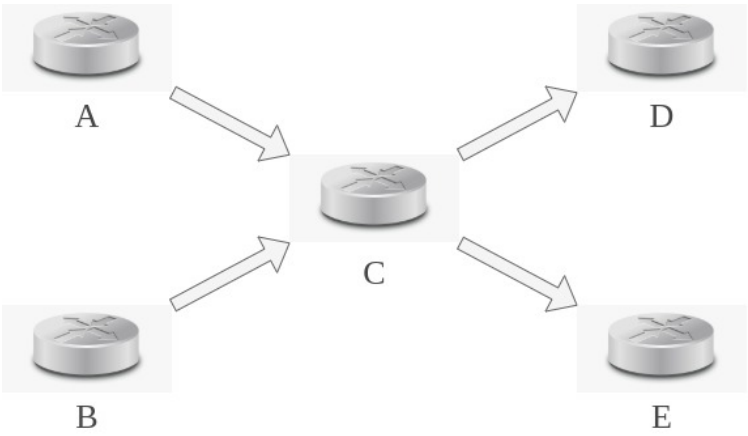


Figure 1.1: Example Network Topology

destination E. Traditionally, such *intentions* are fulfilled by manually writing the corresponding specifications in the configuration files of each device separately. Manually doing all the configurations can be a very tedious task. Further, owing to human errors, there usually arise semantic gaps between the network developer's intent and the actual implementation. It is difficult to translate the global intentions to local implementations. For example, it may be desired that every path from a device at IIT Kanpur to IIT Bombay must pass through IIT Delhi. But how exactly to achieve such a behaviour and what configurations are required becomes a difficult question to answer if done manually.

1.2.2 Centralised Network Routing

Quite recently however, the advancement of Software Defined Networks (SDNs) has replaced the decentralised way of calculating the routing information with a centralised one. The *control plane* acts as the brain of the network. Taking into consideration the underlying topology and desired network behaviour, the control plane calculates all the information that is required by all the devices in the network to appropriately route the packets. The devices that actually route the packets in the network form the *data plane*. The controller in a sense has a global view of the network and has all the network intelligence which alleviates some of the problems and hurdles described above. For instance, in such scenario, all the policy configurations need to be updated at just the controller instead of doing so with all the devices in the network. The controller sends the routing information at the devices in the data plane using protocols written in OpenFlow [MAB+08] or P4 [BDG+14]. However, it is still a very daunting task to directly write correct network programs at a low level of abstraction.

1.3 Network Verification and Debugging

To further add to these problems, networks act like black boxes in a way. The dropping of a packet opens many questions like when, where and how exactly was the packet dropped. Unfortunately, one has to rely on very low level tools like `traceroute` or `ping` to debug the networks. This debugging process is very time consuming, error prone and often inconclusive. In fact, the network administrators are usually expected to reason about the properties of each and every packet that can exist in the network. But this is practically impossible in a setting where the configurations are done manually simply because of the vast size of the state space. The IPv4 packet headers contain a minimum 160 bits and can go up to 480 bits. So, there exist at least 2^{160} distinct IPv4 packet headers. So, it is difficult to establish any global properties like *"Hosts A and B must not be able to communicate with each other"*. One can easily inject a packet at source A and check whether it is able to reach the destination B using `traceroute`. But that is just one packet out of total 2^{160} possible packets.

1.4 Algebraic Routing Related work

All in all, there are two main issues with network programming: (a) the difficulty of coming up with new network protocols in the first place, and (b) verifying the correctness of the programs implementing those protocols with respect to the desired properties so as to ensure the absence of any semantic gaps. In the recent past, many network researchers have tried to solve these problems by coming up with appropriate abstractions and develop domain specific languages (DSLs) for networks and also using a strong mathematical foundations for defining the semantics of these DSLs in order to ensure correctness properties. Following this idea, Griffin and Sobrinho in their paper [GS05] used a high level declarative language framework to define and prove the convergence properties of routing protocols. The base algebras defining the basic routing can be combined using a Routing Algebra Metalanguage to define new complex routing protocols, where the convergence properties are relatively straightforward to prove.

Further, in an attempt to formalise the routing protocols across the layers of the internet architecture stack, and model route redistribution across distinct routing protocols, Griffin and Billings introduced the idea of using semi-modules [BG09]. This idea of algebraic composition works elegantly in a structured form of the internet routing which can be modelled using idempotent semiring structures.

Not all the existing network protocols satisfy *distributivity*, an important property needed for any semiring structure. Such protocols that do not necessarily follow this property are called "policy rich protocols". In fact, most of the protocols currently in use like BGP or RIR are policy rich. Researchers have come up with certain conditions that the protocols must follow in order to guarantee convergence. There are several factors which can be used to decide how good

these conditions are. The paper [DGG18] very succinctly summarises these factors and gives a general enough solution for the convergence guarantees for policy-rich routing protocols.

Adding to these problems, there are terms in the internet that haven't been properly formalised. Similarly, there are several other diversities in the network that make them difficult to analyse. In the paper [KKPB07], the authors attempt to model the internet's forwarding mechanisms and communications in an axiomatic formulation. Based on a "leads-to" relation, they are able to define using 4 basic axioms, fundamental communication concepts such as names, addresses, peers, tunneling etc., as well as forwarding operations quite easily. Further this concept is extended to include control mechanisms within a namespace and also combining namespaces. A Hoare style formal semantics of the primitives along with the leads to relation makes this formulation quite convincing. Further extensions of the work use temporal logic in order to address the network changes and reconfigurations over time by introducing a temporal order [ZP15]. This axiomatisation is able to address issues like IP mobility which are an example of the ever-changing dynamic nature of the internet. Further, the security issues such as those in TOR networks can also be modelled in this formulation.

Finally, the main concept that we build upon in this dissertation is NetKAT [AFG$^+$14, FKM$^+$15], a network programming language based on Kleene Algebra with Tests (KAT). It enjoys a complete and sound equational theory (with respect to the operational behaviour). According to the abstractions in this language, the network packets are a characterised by a set of field-value pairs (a finite set). The primitive programs can either filter the packets on the basis the value of a particular field, or can even modify the the value of a field. Programs can also be extended from these primitive ones either by concatenating programs using the Kleisli composition or by taking the union of the programs. The filters from a Boolean subalgebra of the KAT. Finally, the Kleene-star operation defines the overall network program. The NetKAT compiler converts these abstract programs into match action tables that can be installed onto the devices in the data plane. The next chapter briefly describes NetKAT and its equational theory.

1.5 Main Contributions

Although NetKAT provides an excellent way of writing network programs and also verifying them, yet we are far from completely simulating the modern networks. The values of every field in the NetKAT packets come from discrete sets. So NetKAT, in its current equational theory, does not talk about the costs incurred in routing a packet along a particular network path, allowing just two extremes – either reachable or unreachable. We propose to (conservatively) incorporate costs from a suitable cost algebra, thus allowing cost-dependent policies, such as cost-bounded reachability.

Second, NetKAT assumes a single uniform network throughout, in order to write routing policies, and does not allow the functionality of defining cross-network routing. So, we add

new functionality to NetKAT using NetKAT homomorphisms in order to allow a more layered structure. This would also enable one to compose policies from two distinct networks. Also, one can talk about shortest path routes from a point in one network to a point in another network by composing the two network policies using semimodules.

Chapter 2

NetKAT

Note : Readers already familiar with NetKAT may skip this chapter

2.1 Overview

NetKAT is a Domain Specific Language for writing network programs. It is based on Kleene Algebra with Tests (KAT). KATs are quite extensively studied and are known to have a sound and complete theory along with a PSPACE decision procedure. So this makes it easy to check for programs equivalence and also verify the NetKAT programs for specific properties like reachability, waypointing etc.

The data plane consists of a variety of switching elements like switches, routers, bridges, gateways etc. Although these devices have a very efficient implementation with respect to the hardware, yet they are essentially "dumb" in the sense that they do not have any built-in domain- and configuration-specific intelligence. What all of these have in common is that they maintain a match-action table. Whenever a packet arrives at a device, they match the packet header with one of the entries in the match-action table, and take the corresponding action. They can match on the basis of one or more fields. The actions consist of changing the value of a field, or forwarding the packet to a specific port etc. This switching functionality, independent of the type of device, is what we are the most interested in, and so we abstract all these devices and refer to them as *abstract switching elements* (ASEs) as in [KKPB07].

At the level of an ASE, a network program has two fundamental functionalities: packet modification and classification based on packet headers. At a global level, one must be able to write programs to forward the packets along a path according to a policy. NetKAT very succinctly provides all of these abstractions. $(f = v)$ are the primitive tests that classify the packets on the basis of the value of a particular field. For example, a NetKAT program $p \equiv (src = IITD)$ would drop all the packets not originating from IITD. Therefore, $(f = v)$ form the domain specific tests as a part of the Boolean subalgebra of the KAT (NetKAT). An assignment $(f \leftarrow v)$ is another primitive NetKAT program. It modifies a particular field of the packet header to a specific value. For example, $p \equiv (port := 5)$ would change the port of any incoming packet to 5.

Fields	$f ::= f_1 \mid \cdots \mid f_k$	
Packets	$pk ::= \{f_1 = v_1, ..., f_k = v_k\}$	
History	$h ::= \langle pk \rangle \mid pk :: h$	
Predicates	$a, b ::= 1$	*Identity*
	$\mid 0$	*Drop*
	$\mid f = n$	*Test*
	$\mid a + b$	*Disjunction*
	$\mid a.b$	*Conjunction*
	$\mid \neg a$	*Negation*
Policies	$p, q ::= a$	*Filter*
	$\mid f \leftarrow n$	*Assignment*
	$\mid p + q$	*Union*
	$\mid p \cdot q$	*Sequential composition*
	$\mid p^*$	*Kleene star*
	$\mid$ `cp`	*Checkpoint*

Figure 2.1: NetKAT syntax

2.2 NetKAT Syntax and Semantics

More formally, NetKAT packets are field-value pairs. NetKAT programs are either boolean predicates or policies. `true`,`false`,$(f = v)$ form the primitive boolean predicates. $+, \cdot, \neg$ represent the usual disjunction, conjunction and negation operations respectively in thhe Boolean subalgebra. Policies comprise these boolean filters and primitive assignments as described earlier. $+, \cdot, ^*$ represent the concatenation, union, and iteration operations respectively, as in any Kleene algebra such as regular languages..

A NetKAT packet in a way represents the current state of a network packet and contains entire information that is required to determine its location. The current switch and the port also form a part of the NetKAT packet header. NetKAT follows a trace semantics, *i.e.*, NetKAT programs actually work with *packet histories*. A history is defined as a list of NetKAT packets. A `cp` [1] operation checkpoints a packet into this history, *i.e.*, it just duplicates the first packet at the front of the list. All the other programs either filter the histories based on the topmost packet or change the field of the topmost packet, leaving all packets below the top in the history unchanged. A NetKAT program may produce one or more histories from a given history, due to the union operation. Therefore, a NetKAT programs maps an input history to an output set of histories. The union operation (+) simply takes the union of the sets of histories produced

[1] In the original NetKAT paper, this is written as `dup`

$$[\![p]\!] \in \mathrm{H} \to 2^{\mathrm{H}}$$
$$[\![1]\!]h = \{h\}$$
$$[\![0]\!]h = \{\}$$
$$[\![f = n]\!](pk :: h) = \begin{cases} \{pk :: h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases}$$
$$[\![\neg a]\!]h = \{h\} \setminus ([\![a]\!]h)$$
$$[\![f \leftarrow n]\!](pk :: h) = \{pk[f := n] :: h\}$$
$$[\![p + q]\!]h = [\![p]\!]h \cup [\![q]\!]h$$
$$[\![p.q]\!]h = ([\![p]\!] \bullet [\![q]\!])h$$
where $\bullet$ is the Kleisli composition
$$[\![p^*]\!]h = \bigcup_i F_p^i h$$
where $F_p^0 h = \{h\}$ and $F_p^{i+1} h = ([\![p]\!] \bullet F_p^i)h$
$$[\![\mathsf{cp}]\!](pk :: h) = \{pk :: (pk :: h)\}$$

Figure 2.2: NetKAT semantics

by the two subprograms, *i.e.*, $[\![p + q]\!]h = [\![p]\!]h \cup [\![q]\!]h$. the $\cdot$ operation is treated as a Kleisli composition, and the * as the iteration.

The boolean predicates very beautifully separate the local programs at each ASE. All of the local programs can be just combined using the union operator to simulate the overall network program p. Further, the topology, t can be encoded in a similar fashion. Finally, the complete program can be written as $(p.t)^*$. For example consider the network in Fig. 2.3.

$$p_R \triangleq \mathsf{dst} = \mathsf{A}\cdot\mathsf{pt} := 1 + \mathsf{dst} = \mathsf{B}\cdot\mathsf{pt} := 2 + \mathsf{dst} = \mathsf{C}\cdot\mathsf{pt} := 3$$

$$p_S \triangleq \mathsf{dst} = \mathsf{A}\cdot\mathsf{pt} := 1 + \mathsf{dst} = \mathsf{B}\cdot\mathsf{pt} := 1 + \mathsf{dst} = \mathsf{C}\cdot\mathsf{pt} := 2$$

$$p \triangleq p_R + p_S$$

$$\begin{aligned} t \triangleq\ & \mathsf{sw} = \mathsf{R}\cdot\mathsf{pt} = 1\cdot\mathsf{sw} := \mathsf{A} + \\ & \mathsf{sw} = \mathsf{R}\cdot\mathsf{pt} = 2\cdot\mathsf{sw} := \mathsf{A} + \\ & \mathsf{sw} = \mathsf{R}\cdot\mathsf{pt} = 3\cdot\mathsf{sw} := \mathsf{S}\cdot\mathsf{pt} := 1 + \\ & \mathsf{sw} = \mathsf{S}\cdot\mathsf{pt} = 1\cdot\mathsf{sw} := \mathsf{R}\cdot\mathsf{pt} := 3 + \\ & \mathsf{sw} = \mathsf{S}\cdot\mathsf{pt} = 2\cdot\mathsf{sw} := \mathsf{C} \end{aligned}$$

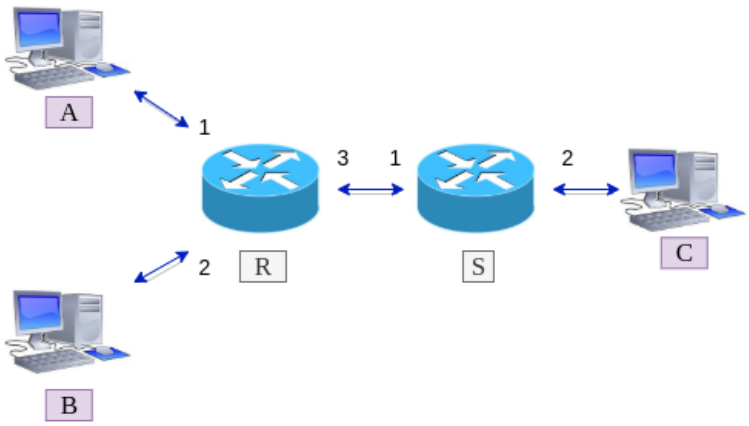


Figure 2.3: Example Network

$$p_{net} \triangleq (p \cdot t)^*$$

In fact, one can also define complete end-to-end policies. Let's say a network operator wants to send each message originating at Host A to Host C check pointing at every step in the path:

$$p \triangleq \mathsf{sw} = \mathsf{A} \cdot \mathsf{cp} \cdot \mathsf{sw} := \mathsf{R} \cdot \mathsf{pt} := 1 \cdot \mathsf{cp} \cdot \mathsf{pt} := 3 \cdot \mathsf{cp} \cdot \mathsf{sw} := \mathsf{S} \cdot \mathsf{pt} := 1 \cdot \mathsf{cp} \cdot \mathsf{pt} := 2 \cdot \mathsf{cp} \cdot \mathsf{sw} := \mathsf{C}$$

2.3 Correctness and Reachability properties

Along with the standard KAT axioms, NetKAT has some additional domain-specific Packet Algebra axioms as a part of its equational theory. Kindly refer to the NetKAT papers [FKM+15] [AFG+14] for the detailed equational theory of NetKAT. Proving soundness of NetKAT relies on set-theoretic semantics of NetKAT programs. Each program can be thought of as a relation between NetKAT histories.

$$(h_1, h_2) \in [p] \triangleq h_2 \in [\![p]\!] h_1$$

This notation simplifies proving the soundness of each of the axioms. For example, according to one of the axioms, $(f := v \cdot f = v) \equiv (f := v)$. In order to prove this equivalence, it suffices to show that $\forall (h_1, h_2) \in [f := v \cdot f = v], (h_1, h_2) \in [f := v]$ and vice-versa. Similarly, for each of the axioms of the type $p_1 \equiv p_2$, proving the soundness is same as showing $[p_1] \subseteq [p_2]$ and $[p_2] \subseteq [p_1]$.

Proving the completeness is a bit involved. This proof works by representing the NetKAT programs into an equivalent canonical form. Let α and π represent the complete tests and complete assignments respectively, *i.e.*, a *complete test* is one that tests the value of *each* field in the packet header, and a *complete assignment* assigns a value to *each* of the fields in the packet header.

$$\alpha \triangleq (f_1 = v_1) \cdot (f_2 = v_2) \cdots (f_n = v_n)$$

$$\pi \triangleq (f_1 := v_1)\cdot(f_2 := v_2)\cdots(f_n := v_n)$$

Let At and Pi represent the sets of complete tests and complete assignments respectively. Every (non-complete) test can be written equivalent to a disjunction (sum) of finitely many complete tests, and every (non-complete) assignment can be written as a sum of *guarded complete assignments*.

$$\begin{aligned}(x = v) &\equiv \top \cdot (x = v)\\ &\equiv (\sum_{\alpha \in \mathsf{At}} \alpha) \cdot (x = v)\\ &\equiv \sum_{\alpha \in \mathsf{At}} \alpha \cdot (x = v)\\ &\equiv \sum_{\alpha' \leq x = v} \alpha'\end{aligned} \tag{2.1}$$

$$\begin{aligned}(x \leftarrow v) &= \top \cdot (x \leftarrow v)\\ &= (\sum_{\alpha \in A} \alpha \cdot \pi_\alpha) \cdot (x \leftarrow v)\\ &= (\sum_{\alpha \in A} \alpha \cdot \pi_{\alpha[x \leftarrow v]})\end{aligned} \tag{2.2}$$

Further, a language model is defined over *canonical forms* where all the complete tests in the program are brought to the front. This set of canonical form strings is called the normalised form or *reduced string form* of the NetKAT program. After converting into the normalised form it is easy to see the completeness.

Theorem 2.3.1 (Soundness)**.** $\vdash p \equiv q \implies [\![p]\!] = [\![q]\!]$

Theorem 2.3.2 (Completeness)**.** $[\![p]\!] = [\![q]\!] \implies \vdash p \equiv q$

For more details on the proofs, please refer to the NetKAT papers [AFG+14, FKM+15].

With NetKAT syntax and semantics, it becomes very easy to encode and verify properties like reachability or waypointing. The reachability of b from a can be checked by checking if p defined below is equivalent to dropping the packet or not.

$$p_r \triangleq a\cdot\mathtt{cp}\cdot(p\cdot t)^*\cdot\mathtt{cp}\cdot b$$

If $p_r \not\equiv \mathtt{drop}$, b is reachable from a.

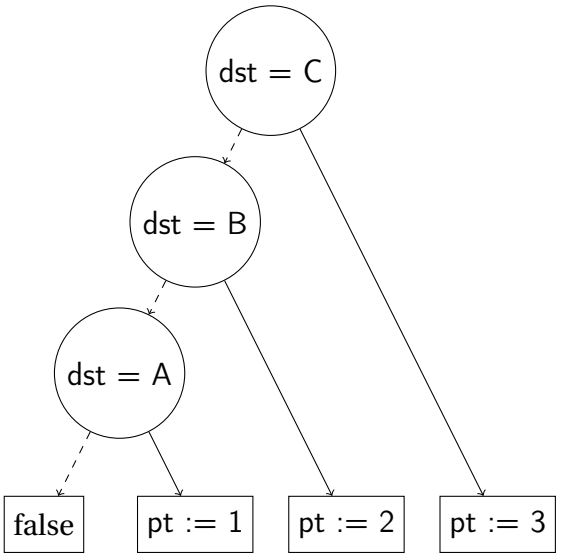
Similarly, one can write program to verify if each path from a to b passes through c.

$$a\cdot\mathtt{cp}\cdot(p\cdot t\cdot\mathtt{cp})^*\cdot b \leq a\cdot\mathtt{cp}\cdot(\neg b\cdot p\cdot t\cdot\mathtt{cp}) * \cdot c\cdot(\neg a\cdot p\cdot t\cdot\mathtt{cp})^*\cdot b$$

2.4 Compilation

The control plane has the actual network intelligence but the ASEs in the data plane just need to know the routing tables, as determined by the control plane. So the controller must supply them with the match-action tables Further, these tables need to be generated for each of the ASEs separately.

On closer inspection, one may realise that all the NetKAT tests are the match fields and the assignments are the actions in these tables. The tables for concatenation and the union can be defined recursively. But this method of generating tables has an exponential time complexity. The paper [SEFG15] describes an efficient method for compiling a NetKAT programs using Binary Decision Diagrams (BDD) and adapting them into a similar domain specific data structure called Forwarding Decision Diagrams (FDD). The compilation pipeline first converts all the global programs into a disjunctive union (+) of local programs using symbolic NetKAT automata. These local programs are then converted into FDDs. These FDDs are then converted into the match-action tables. Figure 2.4 describes the FDD and the corresponding table for the switch *R* in Figure 2.3



(a) FDD

Match	Action
dst = A	pt := 1
dst = B	pt := 2
dst = C	pt := 3

(b) Match Action table

Figure 2.4: Local compilation for ASE at R

2.5 Extensions

The abstractions provided by NetKAT provide a strong basis for network routing. There are various extensions already proposed to make NetKAT even versatile and equip it with sufficient functionality with a target of completely capturing the needs of modern networks. In this dissertation, we propose to conservatively extend the equational theory with a suitable *cost*

algebra.

A recent paper by Beckett et. al. [BGW16] extended the language by adding a linear time (past) temporal logic (ptLTL) to NetKAT, enabling one to ask and verify the network program properties relating to path-based queries. It combines the equational theories of LTL and KAT along with the soundness and completeness proofs and also extends the compiler to incorporate the extensions. SNAP [AKG+16], Event-Driven Network Programming [MHFv16], [BFH+17] Propane [BMM+19] are some other extensions of NetKAT in various domains. Probabilistic programming is also currently gaining a lot of attention of various research labs. The development of Probabilistic NetKAT [FKM+16, SKF+17, SKK+19] at Cornell, ProFoundNet project at UCL, and [VS19] are some examples.

Further, NetKAT programs inherently assume that all the ASEs in the underlying technology are fully programmable. However, this is not always the case. Some of the legacy devices have a fixed functionality which create an obstacle for the network developers to write arbitrary NetKAT programs. It does not sound an economically feasible solution to simply replace all such devices with the programmable ones. This problem has opened another line of research to use heuristic guided search to translate arbitrary NetKAT programs into the ones that can work with the devices in the underlying topology [ave21].

Chapter 3

Cost Algebra and Cost NetKAT

3.1 Introduction

Normally there are multiple paths between any two nodes in a network. One can route messages through any one of them to reach the final destination. Most routing protocols followed in networking deploy the most fundamental tie-breaker criteria, that is the shortest path algorithm. Traditionally, for all the ASEs in the network, the distance vector routing protocols determine the distance of the node from all other nodes in the network. This information is then used to determine the next hop table for each node based on the final destination. Be it a decentralised way or a centralised way as in SDNs, most of the protocols focus on finding the single source shortest tree for each node in the graph. These costs need not be just the number of hops or the geometric distance, but can be a complex tuple structure cost with many fields.

The famous Dijkstra's shortest path algorithm [Dij59] is a solution to this problem. The algorithm proceeds by maintaining two sets of nodes – one for which the shortest path has been discovered and the other for the remaining nodes. This creates a recursive problem of a smaller size, thus reaching a fixed point after a finite number of iterations. [HM12] shows a very clean relation of this algorithm with Kleene Algebra with Tests and *derives* the same algorithm and also proves its correctness in a purely algebraic way. So it seems only natural to use KATs as the fundamental algebra for implementing the network routing.

The original NetKAT theory defines each *point* in the state space as a *complete test*. So each action (complete or not) just defines a transition from one point to another. A NetKAT policy, defines edges between these states in the state space. In a matrix representation, consider a matrix of size $n \times n$, where n is the size of the set of all complete tests, *i.e.*, $n = |At|$. The matrix entry $M(i, j) = 1$ represents that according the NetKAT policy, point j is reachable from i by some trajectory, whereas $M(i, j) = 0$ represents otherwise.

However, NetKAT in its current semantics does not use any notion of the costs incurred due to forwarding the packets along any path. The values of all the fields in a NetKAT packet header are drawn from discrete sets that do not follow any ordering. For example, an IP address a_1 can neither be considered less than or even greater than another IP address a_2. There simply is no ordering between the elements of these sets, and therefore no notion such as "shortest path" using which we can compare prioritise trajectories according to a policy. Further, the assignment operations in NetKAT simply replace the original value of a field with a new value.

However, in network routing policies, just talking about reachability is not enough. "Reachable" or "unreachable" are two extremes of a complete spectrum of values in a suitable algebraic/ordering structure. What is required in order to simulate real world networks is a *cost* associated with the transition from one point in the state space to another. In other words, the matrix entry $M(i,j)$ should represent the cost associated along the trajectory from i to j. In order to associate any notion of costs, we first need to develop a *cost algebra* general enough to represent costs of any nature and complexity, but conservatively extend our existing NetKAT theory.

3.2 Cost Algebra: Terminology and Properties

A *cost algebra* is an idempotent semiring $\mathscr{C} = \langle \Sigma, \vee, \circ, \mathbf{1}, \mathbf{e} \rangle$ where the costs are the elements drawn from Σ.

The $\vee$ operation is a commutative, associative function that maps the elements of $\Sigma \times \Sigma$ to Σ.

$$\forall c_1, c_2 \in \Sigma,\ (c_1 \vee c_2) = (c_2 \vee c_1)$$

$$\forall c_1, c_2, c_3 \in \Sigma,\ (c_1 \vee c_2) \vee c_3 = c_1 \vee (c_2 \vee c_3)$$

$$\forall c \in \Sigma,\ c \vee \mathbf{1} = c = \mathbf{1} \vee c$$

We can define a partial ordering relation ($\preceq$) amongst the elements of Σ: $c \preceq d$ if $cc \vee d = d$. Equivalently, we write $(c \succeq d)$ if $(c \vee d = c)$, where $c \succeq d$ means that c is a *better* or *more preferred* cost than d.

$$\forall c \in \Sigma,\ c \succeq \mathbf{1}$$

The continuation function, $\circ$ is an associative function that maps $\Sigma \times \Sigma$ to Σ. $\mathbf{e}$ is the identity element of the continuation function.

$$\forall c_1, c_2, c_3 \in \Sigma,\ (c_1 \circ c_2) \circ c_3 \ = \ c_1 \circ (c_2 \circ c_3)$$

$$\forall c \in \Sigma,\ c \circ \mathbf{e} \ = \ c \ = \ \mathbf{e} \circ c$$

Further, $\circ$ distributes over $\vee$. $\mathbf{1}$ is an annihilator for $\circ$.

$$\forall c_1, c_2, c_3 \in \Sigma,\ c_1 \circ (c_2 \vee c_3) = (c_1 \circ c_2) \vee (c_1 \circ c_3)$$

$$\forall c \in \Sigma,\ \mathbf{1} \circ c \ = \ \mathbf{1} \ = \ c \circ \mathbf{1}$$

We see that the $\mathbf{1}$ element, *i.e.*, the identity element of $\vee$ is the least preferred cost, whereas the identity of $\circ$ $\mathbf{e}$ is the most preferred. In a cost algebra for networks, we will require that

the cost of any trajectory from a point i to itself is $\mathbf{e}$. Further, $\circ$ is a monotonically contracting function.

$$\forall c, d \in \Sigma,\ c \vee (c \circ d) = c$$

In nutshell, $\mathscr{C} = \langle \Sigma, \vee, \circ, \mathbf{1}, \mathbf{e} \rangle$ is a valid cost algebra if it satisfies the following properties:

1. $\langle \Sigma, \circ, \mathbf{e} \rangle$ is a monoid
2. $\langle \Sigma, \vee, \mathbf{1} \rangle$ is a commutative monoid
3. $\circ$ distributes over $\vee$
4. $\mathbf{e}$ is the annihilator of $\vee$
5. $x \vee x = x$
6. $\circ$ is a monotonic function over the elements of Σ

The most common example of this cost algebra is the $\mathscr{N} = \langle \mathbb{N}, \min, +, \infty, 0 \rangle$ algebra. The elements of this algebra are the drawn from the set of natural numbers, $\mathbb{N}$. The elements of this set ordered with the well known algebraic ordering with <= as the algebraic ordering relation. $n_1 <= n_2 \equiv n_1 \succeq n_2$. Further, 0 and ∞ are the minimum and maximum elements. The ∞ can in fact be replaced by any practically suitable large enough value such that any message with a cost greater than this upper bound will just be dropped. In case of networks packets, most of the header fields have a maximum width allowed. Let the maximum length allowed be b bits. So, the upper bound can be set to 2^b.

1. $\langle \mathbb{N}, +, 0 \rangle$ is a monoid.
   (a) 0 is the identity element. $\forall n \in \mathbb{N},\ (n+0) = n$.
   (b) $+$ is associative. $\forall n_1, n_2, n_3 \in \mathbb{N},\ (n_1 + n_2) + n_3 = n_1 + (n_2 + n_3)$.
2. $\langle \mathbb{N}, \min, \infty \rangle$ is a commutative monoid.
   (a) min is a commutative function. $\forall n_1, n_2 \in \mathbb{N},\ \min(n_1, n_2) = \min(n_2, n_1)$.
   (b) ∞ is the identity element. $\forall n \in \mathbb{N},\ \min(n, \infty) = n$.
   (c) min is associative. $\forall n_1, n_2, n_3 \in \mathbb{N},\ \min(\min(n_1, n_2), n_3) = \min(n_1, \min(n_2, n_3))$.
3. $+$ distributes over min. $\forall n_1, n_2, n_3 \in \mathbb{N},\ n_1 + \min(n_1, n_2) = \min(n_1 + n_2, n_1 + n_3)$.
4. 0 acts as the annihilator of min. $\forall n \in \mathbb{N},\ \min(n, 0) = 0$.
5. $\forall n \in \mathbb{N},\ \min(n, n) = n$
6. $+$ is a monotonic function. $\forall n, n' \in \mathbb{N},\ \min(n, n + n') = n$.

So, $\mathcal{N} = \langle \mathbb{N}, \min, +, \infty, 0 \rangle$ is a valid cost algebra. Many cost fields in the packet actually use $\mathcal{N}$ as the underlying algebra. For example, geometrical distance between two hosts, total number of hops, timestamp of the packet etc. A path with lesser number of hops usually preferred. Also, a path that covers a lesser geometrical distance between the source and the destination is a natural choice and is the solution of the Dijkstra's shortest path algorithm. Further, if a packet exceeds a specific value of timestamp, it is dropped from the network, and therefore, needs to be resent by the source.

Another example of this cost algebra is $\mathcal{M} = \langle \mathbb{N}, \min, \max, \infty, 0 \rangle$

1. $\langle \mathbb{N}, \max, 0 \rangle$ is a monoid.
   (a) 0 is the identity element. $\forall n \in \mathbb{N}$, $\max(n, 0) = n$.
   (b) max is associative. $\forall n_1, n_2, n_3 \in \mathbb{N}$, $\max(\max(n_1, n_2), n_3) = \max(n_1, \max(n_2, n_3))$.
2. $\langle \mathbb{N}, \min, \infty \rangle$ is a commutative monoid.
   (a) min is a commutative function. $\forall n_1, n_2 \in \mathbb{N}$, $\min(n_1, n_2) = \min(n_2, n_1)$.
   (b) ∞ is the identity element. $\forall n \in \mathbb{N}$, $\min(n, \infty) = n$.
   (c) min is associative. $\forall n_1, n_2, n_3 \in \mathbb{N}$, $\min(\min(n_1, n_2), n_3) = \min(n_1, \min(n_2, n_3))$.
3. max distributes over min. $\forall n_1, n_2, n_3 \in \mathbb{N}$, $\max(n_1, \min(n_1, n_2)) = \min(\max(n_1, n_2), \max(n_1, n_3)$.
4. 0 acts as the annihilator of min. $\forall n \in \mathbb{N}$, $\min(n, 0) = 0$.
5. $\forall n \in \mathbb{N}$, $\min(n, n) = n$
6. max is a monotonic function. $\forall n, n' \in \mathbb{N}$, $\min(n, \max(n, n')) = n$.

Therefore, $\mathcal{M} = \langle \mathbb{N}, \min, \max, \infty, 0 \rangle$ is a valid cost algebra.

Consider the network in the Figure 3.1 where the hosts A and B wish to establish a connection through the path that is the least prone to dropping the packets. A broadcasts a packet with 0 initial cost into the network to find the minimum cost path. Each link in the network has a value associated that indicates the current congestion. In such a situation, one would want to minimize the maximum congestion along any link in the path.

There can be two paths that can send packets between A and B, namely $\widehat{PQR}$ and $\widehat{PTSR}$. A packet will travel through each of the two paths and the congestion cost field follows the $\mathcal{M}$ algebra. The link QR in the first path, $\widehat{PQR}$ offers the maximum congestion equal to 10 and acts as the bottleneck of the path. Similarly, the bottleneck in the second path $\widehat{PTSR}$ is the link TS which offers a congestion equal to 7. Figure 3.2 shows the packet cost at each step in the two paths. Finally, B receives both of the two packets, and the connection is set up with the path with lower net cost, i.e., $\widehat{PTSR}$.

Now consider the example in Figure 3.3. A network packet header may contain many fields - all of which follow varied cost algebras with different continuation functions or different signatures. Consider a packet header with two fields - c_1 and c_2 - one which follow $\mathcal{N}_1$ as the cost

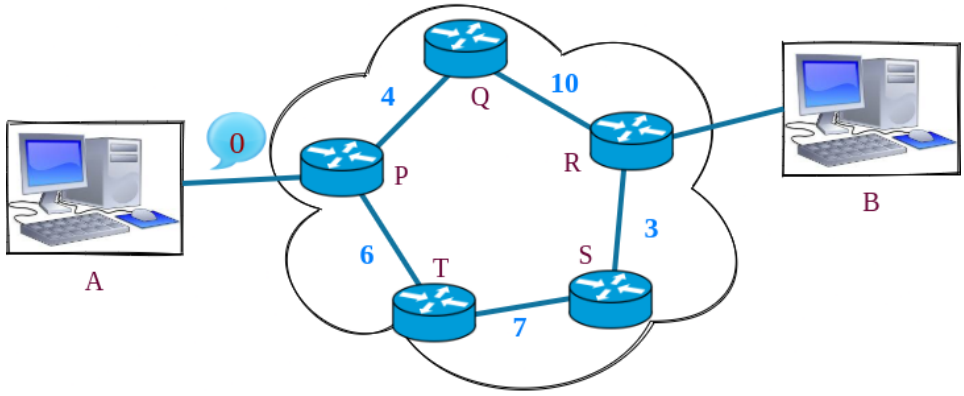


Figure 3.1: Maximum congestion in the network

algebra and the other which follow $\mathcal{N}_2$. At some point in time, a packet reaches a bifurcation and it has two choices of paths to the same final destination. The packet must take a path with the lower overall cost. But it may not always be possible to compare the two costs. In the following example in Figure 3.3, a packet at P can either take a path with two hops and a total path length equal to 7, or it can choose the other path with a single path but a path length equal to 16. In this case, both the costs are pairs of number of hops and the total path length. This example shows that it is not always compulsory for the two elements to be ordered.

We extend the algebra to a Residuated Kleene Lattice with Tests [].

1. A meet operation $\wedge$.
2. Residual \, /
3. Kleene-star operation $*$

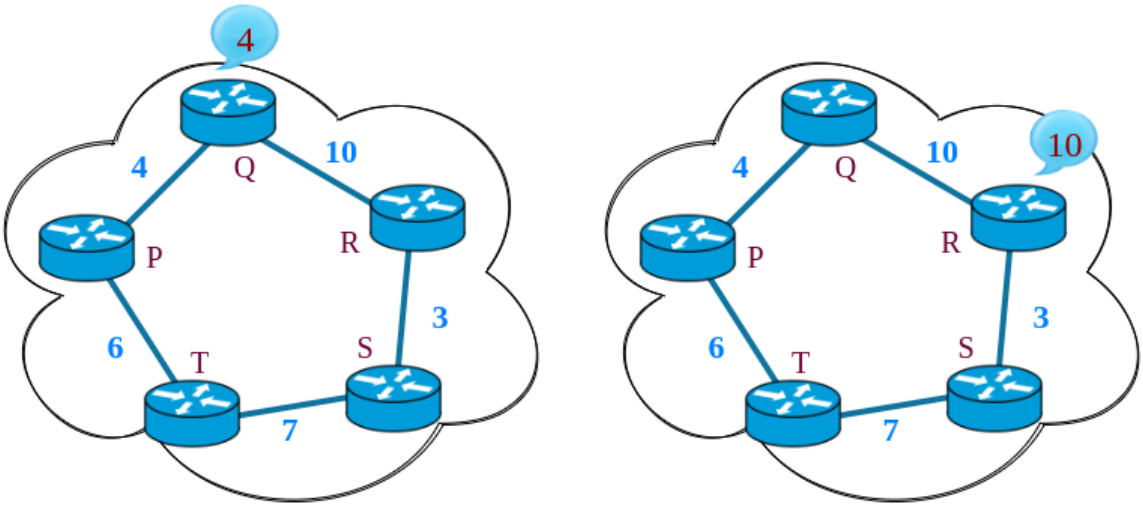
$$x \wedge y = \bigvee \{z : x \succeq z \text{ and } y \succeq z\}$$

$$x \backslash y = \bigvee \{z : y \succeq x \circ z\}$$

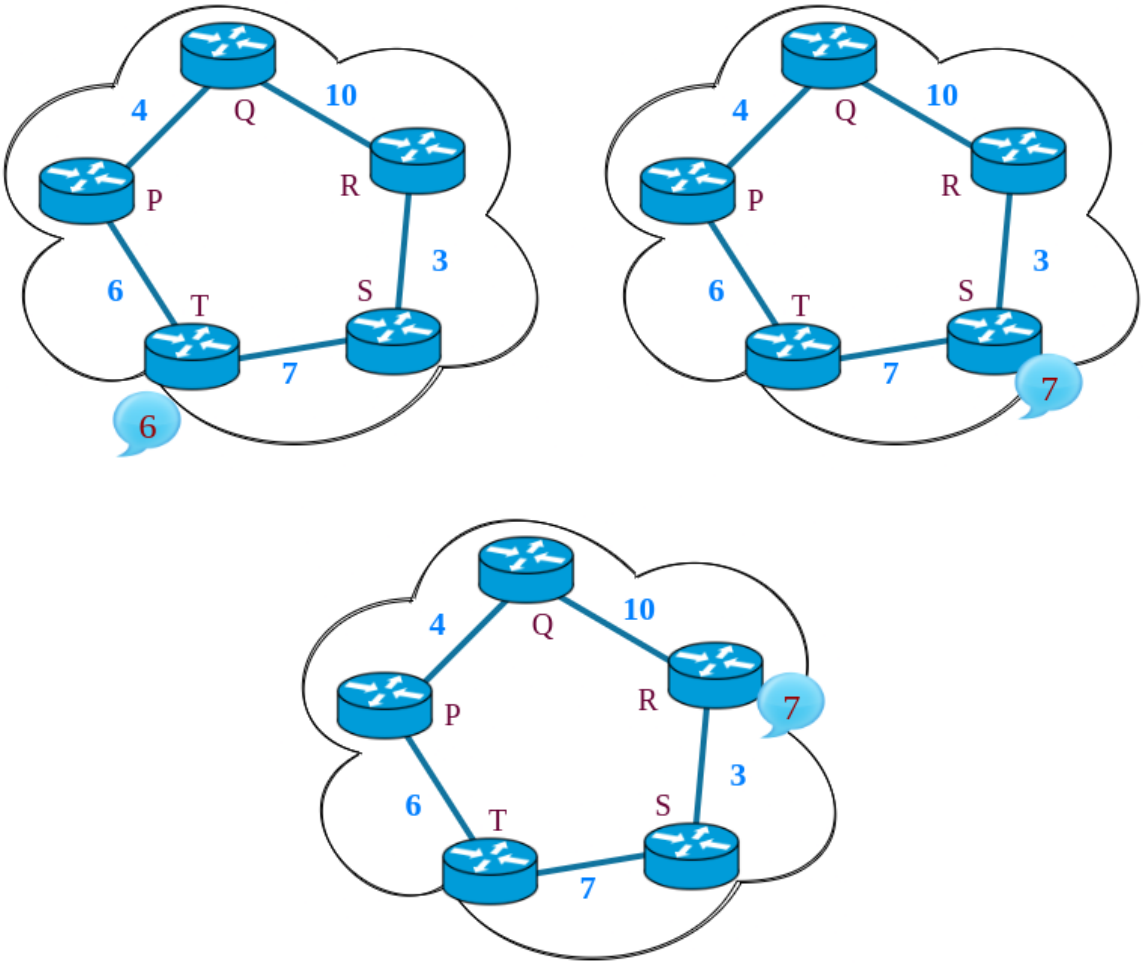
$$x / y = \bigvee \{z : x \succeq z \circ y\}$$

$$x^0 = \mathbf{e} \qquad x^n = x^{n-1} \circ x \qquad x^* = \bigvee_{i \in \mathcal{N}} x^i$$

- The $\vee$ operation gives the lowest upper bound of two elements in Σ. The $\wedge$ on the other had, is the greatest lower bound.
- It is important in network communications about the weakest preconditions that will lead a packet to its final destination. For example, for a packet originating at host A destined to reach the final destination at host B in a network, $\alpha \backslash \beta$ gives the initial values that



(a) Message through path $\widehat{PQR}$



(b) Message through path $\widehat{PTSR}$

Figure 3.2: Comparison of cost along the two paths

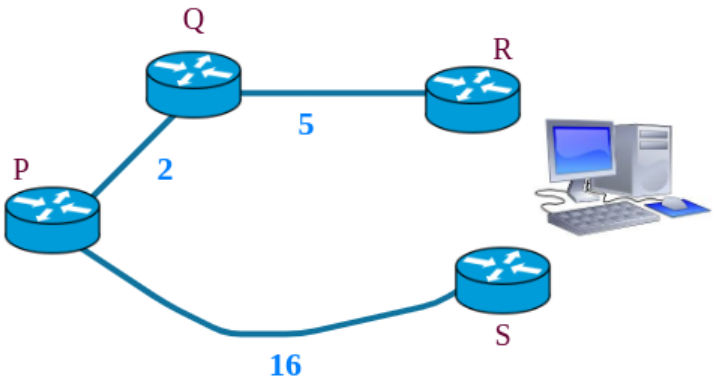


Figure 3.3: Maximum congestion in the network

must be installed in the packet header at A, where α is the condition of the packet being at A, while β is the condition of the packet being at B.
Similarly, x/y gives the all the initial values of the packets that will final reach x by the program y.

- Kleene-star ($*$) is the usual iteration operation (also used in original NetKAT).

3.3 Cost NetKAT - syntax and semantics

Using the cost algebra defined so far, we conservatively extend the syntax and semantics of NetKAT. The values of all the fields of the original NetKAT packet were drawn from a discrete set. Let's call all those as the *non-cost* fields. In Cost NetKAT, we add some fields in the packet header whose values are elements of a cost algebra. Any new cost incurred to these fields won't just replace the original value but will increase the cost defined by the continuation operation. Further, instead of checking for exact equality of these cost fields, it is more suitable to check for a preference over these cost fields. A network developer may be concerned about the cumulative cost incurred by the packet in the entire starting from its source, or they may just care about the cost incurred in the current network. So, instead of having an atomic cost field, we maintain a pair as the cost field. The first element of the pair is the local cost incurred by the packet, while the second element of the field is the cumulative cost incurred starting from its creation. The local cost element must be reset to **e** at every check-pointing operation (cp). Further, whenever a new cost is incurred ($c \twoheadleftarrow \sigma$), both the elements of the cost field must increase by the same amount. Figures 3.4 and 3.5 present the syntax and semantics of Cost NetKAT.

There is a subtle change in the semantics of the union (+) operator. To understand the change, consider the following example

$$p_1 \equiv (\text{hops} \twoheadleftarrow 5)$$

$$p_2 \equiv (\text{hops} \twoheadleftarrow 3)$$

Non-cost Fields	$f ::= f_1 \mid \cdots \mid f_k$	
Cost Fields	$c ::= c_1 \mid \cdots \mid c_l$	
Packets	$pk ::= \{f_1 = v_1, \cdots, f_k = v_k,\ c_1 = \sigma_1, \cdots c_l = \sigma_l\}$	
History	$h ::= \langle pk \rangle \mid pk :: h$	
Predicates	$a, b ::= 1$	*Identity*
	$\mid 0$	*Drop*
	$\mid f = n$	*Non-cost Test*
	$\mid c^{lo} \geq \sigma$	*Local cost Test*
	$\mid c^{cu} \geq \sigma$	*Cumulative cost Test*
	$\mid a + b$	*Disjunction*
	$\mid a.b$	*Conjunction*
	$\mid \neg a$	*Negation*
Policies	$p, q ::= a$	*Filter*
	$\mid f \leftarrow n$	*Modification*
	$\mid c \twoheadleftarrow \sigma$	*Increment*
	$\mid p + q$	*Union*
	$\mid p \cdot q$	*Sequential composition*
	$\mid p^*$	*Kleene star*
	$\mid \texttt{cp}$	*Checkpoint*

Figure 3.4: Cost NetKAT syntax

$$p \equiv p_1 + p_2$$

$$q_1 \equiv (\text{hops} \twoheadleftarrow 5) \cdot (\text{length} \twoheadleftarrow 30)$$

$$q_2 \equiv (\text{hops} \twoheadleftarrow 3) \cdot (\text{length} \twoheadleftarrow 50)$$

$$q \equiv q_1 + q_2$$

Clearly, the program p_1 would increase the hop count of the top packet of any input history by 3 units. Similarly, the program p_2 would increase the hop count by 5 units. Now if the elements 3, and 5 were not ordered, the program p would simply produce an output set of two histories - one which incurred 3 hop counts and the other with 5. In this case however, since the elements are ordered, one would not expect the output to contain the packet with a larger cost. So in fact, $p \equiv p_2$.

In order to mathematically define the semantics of $\uplus$, we need to lift the semantics of the preference relation from a packet header to a trace of packet headers (if it is not already defined). Two packet histories are ordered by the preference relation $\geq$ if all the individual ele-

$$
\begin{aligned}
[\![p]\!] &\in \mathrm{H} \to 2^{\mathrm{H}} \\
[\![1]\!]h &= \{h\} \\
[\![0]\!]h &= \{\} \\
[\![f = n]\!](pk :: h) &= \begin{cases} \{pk :: h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases} \\
[\![c^{lo} \geq \sigma]\!](pk :: h) &= \begin{cases} \{pk :: h\} & \text{if } pk.c^{lo} \geq \sigma \\ \{\} & \text{otherwise} \end{cases} \\
[\![c^{cu} \geq \sigma]\!](pk :: h) &= \begin{cases} \{pk :: h\} & \text{if } pk.c^{cu} \geq \sigma \\ \{\} & \text{otherwise} \end{cases} \\
[\![\neg a]\!]h &= \{h\} \setminus ([\![a]\!]h) \\
[\![f \leftarrow n]\!](pk :: h) &= \{pk[f := n] :: h\} \\
[\![c \twoheadleftarrow \sigma]\!](pk :: h) &= \{pk[c^{lo} \circ \sigma][c^{cu} \circ \sigma] :: h\} \\
[\![p + q]\!]h &= [\![p]\!]h \uplus [\![q]\!]h \\
[\![p.q]\!]h &= ([\![p]\!] \bullet [\![q]\!])h \\
&\text{where } \bullet \text{ is the Kleisli composition} \\
[\![p^*]\!]h &= \biguplus_i F_p^i h \\
&\text{where } F_p^0 h = \{h\} \text{ and } F_p^{i+1} h = ([\![p]\!] \bullet F_p^i)h \\
[\![\mathsf{cp}]\!](pk :: h) &= \{pk[c^{lo} := \mathbf{e}] :: (pk :: h)\}
\end{aligned}
$$

Figure 3.5: Cost NetKAT semantics

ments of both the histories follow the same ordering and they have the same length.

$$\text{Let } s = \langle pk_1 :: pk_2 :: \cdot :: pk_n \rangle$$

$$\text{Let } t = \langle pk'_1 :: pk'_2 :: \cdot :: pk'_n \rangle$$

$$s \succeq t \text{ iff } \forall i, pk_i \succeq pk'_i$$

Using this, now we can define the $\uplus$ operation as follows:

$$S_1 \uplus S_2 = \{x | x \in S_1 \cup S_2 \text{ and } \nexists y \in S_1 \cup S_2 \text{ s.t. } y \succeq x\}$$

So, in the examples shown above, $[\![p_1]\!]h \uplus [\![p_2]\!]h = [\![p_2]\!]h$. However, $[\![q_1]\!]h \uplus [\![q_2]\!]h = [\![q_1]\!]h \cup [\![q_2]\!]h$, if the elements of (hops, length) are not always ordered.

The next section presents some examples for a better understanding of the policies that one

can write using the new syntax and semantics of Cost NetKAT.

3.4 Examples

For all the examples in this section, the costs are natural numbers drawn from the set $\mathbb{N}$. The preference relation is the algebraic ordering between natural numbers, i.e., $\forall n_1, n_2 \in \mathbb{N}\ (n_1 <= n_2) \equiv (n_1 \geq n_2)$. Also, $\neg(c <= n)$ is simply written as (c > n)

3.4.1 Example 1 - Hops vs. Congestion

Consider the example in Figure 3.6. Any packet that reaches the ASE at P, it can follow two different paths - a direct path ($\widehat{PB}$) or a longer path ($\widehat{PQRSB}$). However, the path $\widehat{PB}$ suffers from a larger congestion value. So, the ASE at P sends the packet through this direct link only if the packet has already travelled a longer path than a threshold value. Otherwise, the packet must be routed through the longer path which has a maximum congestion equal to 3.

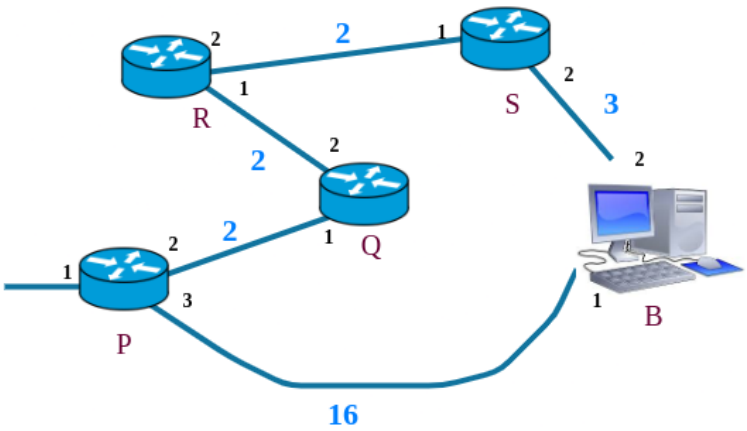


Figure 3.6: Example network showing hops vs. congestion costs

$$
\begin{aligned}
t \triangleq\ & \mathsf{sw} = \mathsf{P} \cdot \mathsf{pt} = 2 \cdot \mathsf{sw} \leftarrow \mathsf{Q} \cdot \mathsf{pt} \leftarrow 1 \cdot \mathsf{hops} \twoheadleftarrow 1 \cdot \mathsf{cong} \twoheadleftarrow 2 + \\
& \mathsf{sw} = \mathsf{P} \cdot \mathsf{pt} = 3 \cdot \mathsf{sw} \leftarrow \mathsf{B} \cdot \mathsf{pt} \leftarrow 1 \cdot \mathsf{hops} \twoheadleftarrow 1 \cdot \mathsf{cong} \twoheadleftarrow 16 + \\
& \mathsf{sw} = \mathsf{Q} \cdot \mathsf{pt} = 2 \cdot \mathsf{sw} \leftarrow \mathsf{R} \cdot \mathsf{pt} \leftarrow 1 \cdot \mathsf{hops} \twoheadleftarrow 1 \cdot \mathsf{cong} \twoheadleftarrow 2 + \\
& \mathsf{sw} = \mathsf{R} \cdot \mathsf{pt} = 2 \cdot \mathsf{sw} \leftarrow \mathsf{S} \cdot \mathsf{pt} \leftarrow 1 \cdot \mathsf{hops} \twoheadleftarrow 1 \cdot \mathsf{cong} \twoheadleftarrow 2 + \\
& \mathsf{sw} = \mathsf{S} \cdot \mathsf{pt} = 2 \cdot \mathsf{sw} \leftarrow \mathsf{B} \cdot \mathsf{pt} \leftarrow 2 \cdot \mathsf{hops} \twoheadleftarrow 1 \cdot \mathsf{cong} \twoheadleftarrow 3
\end{aligned}
$$

$$
\begin{aligned}
p_P \triangleq\ & \mathsf{sw} = \mathsf{P} \cdot \mathsf{pt} = 1 \cdot \mathsf{hops}^{\mathsf{lo}} <= 8 \cdot \mathsf{dst} = \mathsf{B} \cdot \mathsf{pt} \leftarrow 2 + \\
& \mathsf{sw} = \mathsf{P} \cdot \mathsf{pt} = 1 \cdot \mathsf{hops}^{\mathsf{lo}} > 8 \cdot \mathsf{dst} = \mathsf{B} \cdot \mathsf{pt} \leftarrow 3
\end{aligned}
$$

$$p_Q \triangleq \mathsf{sw} = \mathsf{Q} \cdot \mathsf{pt} = 1 \cdot \mathsf{dst} = \mathsf{B} \cdot \mathsf{pt} \leftarrow 2$$

$$p_R \triangleq \mathsf{sw} = \mathsf{R} \cdot \mathsf{pt} = 1 \cdot \mathsf{dst} = \mathsf{B} \cdot \mathsf{pt} \leftarrow 2$$

$$p_S \triangleq \mathsf{sw} = \mathsf{S} \cdot \mathsf{pt} = 1 \cdot \mathsf{dst} = \mathsf{B} \cdot \mathsf{pt} \leftarrow 2$$

$$p \triangleq p_P + p_Q + p_R + p_S$$

$$p_{net} \triangleq (p \cdot t)^*$$

3.4.2 Example 2 - Local vs. Cumulative cost

In this example, we illustrate a case highlighting the difference between local and cumulative cost. At every check-pointing operation (cp), the local cost of each cost field is reset. In Figure 3.7, the ASE at R sends a packet to B only if it has less than a certain number of hops in its own local network. For this to happen, as soon as the packet is received at P, its local cost is reset. The switch at P however, allows only those packets in its network which have not travelled in the entire network for more than a certain time limit.

$$p_P \triangleq \mathsf{sw} = \mathsf{P} \cdot \mathsf{pt} = 1 \cdot (\mathsf{type} = \mathsf{ssh}) \cdot (\mathsf{time}^{\mathsf{cu}} <= 30) \cdot \mathsf{cp} \cdot \mathsf{pt} \leftarrow 2 \cdot \mathsf{hops} \twoheadleftarrow 1 +$$
$$\mathsf{sw} = \mathsf{P} \cdot \mathsf{pt} = 1 \cdot (\mathsf{type} = \mathsf{http}) \cdot (\mathsf{time}^{\mathsf{cu}} <= 30) \cdot \mathsf{cp} \cdot \mathsf{pt} \leftarrow 3 \cdot \mathsf{hops} \twoheadleftarrow 1$$

$$p_R \triangleq \mathsf{sw} = \mathsf{R} \cdot \mathsf{pt} = 1 \cdot (\mathsf{type} = \mathsf{ssh}) \cdot (\mathsf{hops}^{\mathsf{lo}} <= 5) \cdot \mathsf{pt} \leftarrow 3$$

$$p \triangleq p_P + p_R + p_S$$

$$p_{net} \triangleq (p \cdot t)^*$$

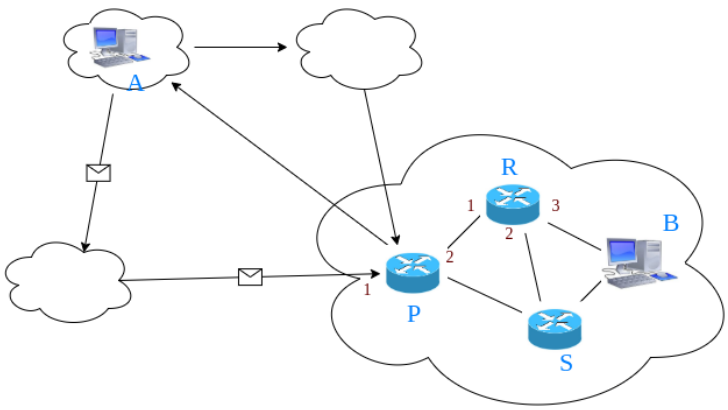


Figure 3.7: Example network showing cumulative vs. global costs

Chapter 4

Axioms, Correctness and Properties of Cost NetKAT

4.1 Soundness

The original NetKAT programs follow some axioms other than the ones presented by the KAT theory. These are the domain specific axioms called the packet algebra axioms. According to [BCG17], these form the client theory, $\mathcal{T}$. In this chapter, we present some additional packet algebra axioms that arise due to the addition of cost fields. All the earlier axioms still hold for the non-cost fields. These axioms are of the form $p_1 \equiv p_2$ presenting the equivalence of two or more Cost NetKAT programs. As done for original NetKAT axioms, we prove the soundness of these axioms by using a relational semantics. This is isomorphic to the trace semantics defines earlier.

$$(h_1, h_2) \in [p] \iff h_2 \in [\![p]\!] h_1$$

Following are the additional Cost NetKAT axioms with their soundness proofs.

Axiom 4.1.1.

$$(c_1^\star \geq \sigma_1) \cdot (c_2^\bullet \geq \sigma_2) \equiv (c_2^\bullet \geq \sigma_2) \cdot (c_1^\star \geq \sigma_1)$$

$$(c_1 \twoheadleftarrow \sigma_1) \cdot (c_2 \twoheadleftarrow \sigma_2) \equiv (c_2 \twoheadleftarrow \sigma_2) \cdot (c_1 \twoheadleftarrow \sigma_1)$$

$$(c_1 \twoheadleftarrow \sigma_1) \cdot (c_2^\bullet \geq \sigma_2) \equiv (c_2^\bullet \geq \sigma_2) \cdot (c_1 \twoheadleftarrow \sigma_1)$$

$$(c \twoheadleftarrow \sigma) \cdot (f = n) \equiv (f = n) \cdot (c \twoheadleftarrow \sigma)$$

$$(c \twoheadleftarrow \sigma) \cdot (f \leftarrow n) \equiv (f \leftarrow n) \cdot (c \twoheadleftarrow \sigma)$$

$$(c \geq \sigma) \cdot (f = n) \equiv (f = n) \cdot (c \twoheadleftarrow \sigma)$$

$$(c \geq \sigma) \cdot (f \leftarrow n) \equiv (f \leftarrow n) \cdot (c \twoheadleftarrow \sigma)$$

$$\star, \bullet \in \{lo, cu\}$$

Axiom 4.1.2.

$$(c^{cu} \geq \sigma) \cdot (c^{lo} \geq \sigma) \equiv c^{cu} \geq \sigma \equiv (c^{lo} \geq \sigma) \cdot (c^{cu} \geq \sigma)$$

Axiom 4.1.3.

$$\textit{If } \sigma_1 \geq \sigma_2, \textit{ then}$$

$$(c^{cu} \geq \sigma_1) + (c^{cu} \geq \sigma_2) \equiv (c^{cu} \geq \sigma_2)$$

$$(c^{lo} \geq \sigma_1) + (c^{lo} \geq \sigma_2) \equiv (c^{lo} \geq \sigma_2)$$

Axiom 4.1.4.

$$\textit{If } \sigma_1 \geq \sigma_2, \textit{ then}$$

$$(c^{cu} \geq \sigma_1) \cdot (c^{cu} \geq \sigma_2) \equiv (c^{cu} \geq \sigma_1)$$

$$(c^{lo} \geq \sigma_1) \cdot (c^{lo} \geq \sigma_2) \equiv (c^{lo} \geq \sigma_1)$$

Axiom 4.1.5.

$$\textit{If } \sigma_1 \geq \sigma_2, \textit{ then}$$

$$(c \twoheadleftarrow \sigma_1) + (c \twoheadleftarrow \sigma_2) \equiv (c \twoheadleftarrow \sigma_1)$$

Axiom 4.1.6.

$$(c \twoheadleftarrow \sigma_1) \cdot (c \twoheadleftarrow \sigma_2) \equiv (c \twoheadleftarrow (\sigma_1 \circ \sigma_2))$$

Axiom 4.1.7.

$$(c^{cu} \geq \sigma_1) \cdot (c \twoheadleftarrow \sigma_2) \leq (c \twoheadleftarrow \sigma_2) \cdot (c^{cu} \geq (\sigma_1 \circ \sigma_2))$$

$$(c^{lo} \geq \sigma_1) \cdot (c \twoheadleftarrow \sigma_2) \leq (c \twoheadleftarrow \sigma_2) \cdot (c^{lo} \geq (\sigma_1 \circ \sigma_2))$$

Axiom 4.1.8.

$$(c \twoheadleftarrow \sigma_1); (c^{lo} \geq \sigma_2) \equiv (c^{lo} \geq \sigma_2/\sigma_1); (c \twoheadleftarrow \sigma_1)$$

$$(c \twoheadleftarrow \sigma_1); (c^{cu} \geq \sigma_2) \equiv (c^{cu} \geq \sigma_2/\sigma_1); (c \twoheadleftarrow \sigma_1)$$

Based on the soundness of the axioms above, we state the soundness theorem.

Theorem 4.1.1. *If* $\vdash p \equiv q$, *then* $[\![p]\!] = [\![q]\!]$

The proof of the soundness of this theorem can be found in Appendix A

4.2 Completeness

The completeness proof of Cost NetKAT with respect to its equational theory is inspired from the completeness proofs in [AFG+14], [BGW16], [BCG17]. This proof proceeds in four steps - defining reduced Cost NetKAT, developing a language model for reduced Cost NetKAT, converting Cost NetAT policies into a normal form, and finally, using the completeness of the reduced form to achieve the Cost NetKAT completeness result

4.2.1 Reduced Cost NetKAT

In reduced Cost NetKAT, every assignment is a complete assignment, ad every test is a complete test. Let $\mathscr{F} = \{f_1, \cdots, f_k, c_1, \cdots c_l\}$ denote the sets of fields Cost NetKAT. In the context of Cost NetKAT, a *primitive test* in $\mathscr{B}$ is a *literal* b_i where $b_i \equiv (f_i = v)$ or $c_i \geq \sigma$, and a *complete test*

is of the form $(f_1 = v_1) \cdot \ldots \cdot (f_k = v_k) \cdot (c_1 \succeq \sigma_1) \cdot \ldots \cdot (c_l \succeq \sigma_l)$ where $f_1, \cdots f_k, c_1, \cdots, c_l$ are all the fields in the packet header in some arbitrary but fixed order. If field x_i can assume n_i distinct values, then there are exactly $n_1 \times \ldots \times n_{(k+l)}$ different complete tests in Cost NetKAT boolean subalgebra $\mathscr{B}$. These complete tests constitute the *atoms* of the Boolean algebra $\mathscr{B}$ because they are minimal nonzero elements of $\mathscr{B}$ generated by the tests. Let A denote the set of complete tests of $\mathscr{B}$. Similarly, in NetKAT, a *primitive assignment* is of the form $f_i \leftarrow v$ or $c_j \twoheadleftarrow \sigma$, and a *complete assignment* is of the form $f_1 \leftarrow v_1 \cdot \ldots \cdot f_k \leftarrow v_k \cdot c_1 \twoheadleftarrow \sigma_1 \cdot \ldots \cdot c_l \twoheadleftarrow \sigma_l$. Let Π denote the set of all complete assignments.

Syntax

$$
\begin{aligned}
\text{Complete non-cost assignment} \quad & \theta \triangleq f_1 \leftarrow v_1 \cdot \ldots \cdot f_k \leftarrow v_k \\
\text{Complete cost assignment} \quad & \kappa \triangleq c_1 \twoheadleftarrow \sigma_1 \cdot \ldots \cdot c_l \twoheadleftarrow \sigma_l \\
\text{Complete assignment} \quad & \pi \triangleq \theta \cdot \kappa
\end{aligned}
$$

$$
\begin{aligned}
\text{Complete non-cost test} \quad & \beta \triangleq f_1 = v_1 \cdot \ldots \cdot f_k = v_k \\
\text{Complete local cost test} \quad & \epsilon \triangleq c_1^{lo} \succeq \sigma_1 \cdot \ldots \cdot c_l^{lo} \succeq \sigma_l \\
\text{Complete cumulative cost test} \quad & \delta \triangleq c_1^{cu} \succeq \sigma_1 \cdot \ldots \cdot c_l^{cu} \succeq \sigma_l \\
\text{complete cost test} \quad & \alpha \triangleq \beta \cdot \epsilon \cdot \delta
\end{aligned}
$$

$$
\begin{aligned}
p, q ::= & \pi \\
| \ & \alpha \\
| \ & p + q \\
| \ & p.q \\
| \ & p^* \\
| \ & \mathsf{cp}
\end{aligned}
$$

(a) Reduced Cost NetKAT syntax

$$
\begin{aligned}
R(\pi) &= \{\pi\} \\
R(\alpha) &= \{\alpha\} \\
R(p+q) &= R(p) \uplus R(q) \\
R(p.q) &= R(p).R(q) \\
R(\mathsf{cp}) &= \{\mathsf{cp}\} \\
R(p^*) &= \biguplus_{n \geq 0} R(p^n)
\end{aligned}
$$

(b) $R(p) \subseteq (\Pi + \mathsf{A} + \mathsf{cp})^*$

Figure 4.1: Reduced Cost NetKAT syntax and regular interpretation

Reduced Cost NetKAT axioms

Before we introduce the axioms, we introduce some new notation based on the elements of A and Π

$$\text{If } \theta = f_1 \leftarrow v_1 \cdot \ldots \cdot f_k \leftarrow v_k, \text{ then, } \beta_\theta \triangleq f_1 = v_1 \cdot \ldots \cdot f_k = v_k$$

$$\text{If } \beta = f_1 = v_1 \cdot \ldots \cdot f_k = v_k, \text{ then, } \theta_\beta \triangleq f_1 \leftarrow v_1 \cdot \ldots \cdot f_k \leftarrow v_k$$

$$\text{If } \delta = c_1^{cu} \succeq \sigma_1 \cdot \ldots \cdot c_l^{cu} \succeq \sigma_l, \text{ then, } \epsilon_\delta \triangleq c_1^{lo} \succeq \sigma_1 \cdot \ldots \cdot c_l^{lo} \succeq \sigma_l$$

$$\epsilon_\infty \triangleq c_1^{lo} \succeq \infty \cdot \ldots \cdot c_l^{lo} \succeq \infty$$

$$\delta_\infty \triangleq c_1^{cu} \succeq \infty \cdot \ldots \cdot c_l^{cu} \succeq \infty$$

$$\kappa_0 \triangleq c_1 \twoheadleftarrow 0 \cdot \ldots \cdot c_l \twoheadleftarrow 0$$

$$\text{If } \pi = \theta \cdot \kappa, \text{ then, } \alpha_\pi \triangleq \beta_\theta \cdot \epsilon_\infty \cdot \delta_\infty$$

$$\text{If } \alpha = \beta \cdot \epsilon \cdot \delta, \text{ then, } \pi_\alpha \triangleq \theta_\beta \cdot \kappa_0$$

Following are axioms relating to A and Π

Axiom 4.2.1.

$$\theta \equiv \theta \cdot \beta_\theta \qquad \kappa \equiv \kappa \cdot \epsilon_\infty \cdot \delta_\infty \qquad \pi \equiv \pi \cdot \alpha_\pi$$

$$\beta \equiv \beta \cdot \theta_\beta \qquad \epsilon \equiv \epsilon \cdot \kappa_0 \qquad \delta \equiv \delta \cdot \kappa_0 \qquad \alpha \equiv \alpha \cdot \pi_\alpha$$

Axiom 4.2.2.

$$cp \cdot (\beta \cdot \epsilon \cdot \delta) \equiv (\beta \cdot \epsilon_\delta \cdot \delta) \cdot cp$$

4.2.2 Language Model, G

$$\begin{aligned} G(\pi) &= \{\beta \cdot \epsilon_\infty \cdot \delta_\infty \cdot \pi | \forall \beta\} \\ G(\alpha) &= \{\alpha \cdot \pi_\alpha\} \\ G(p+q) &= G(p) \uplus G(q) \\ G(p.q) &= G(p) \diamond G(q) \\ G(\mathsf{cp}) &= \{\beta \cdot \epsilon_\infty \cdot \delta_\infty \cdot \theta_\beta \cdot \kappa_0 \cdot \mathsf{cp} \cdot \theta_\beta \cdot \kappa_0 | \forall \beta\} \\ G(p^*) &= \biguplus_{n \geq 0} G(p^n) \end{aligned}$$

Figure 4.2: $G(p) \subseteq I = \mathsf{A}^*(\Pi \cdot \mathsf{cp})^* \cdot \Pi$

Lemma 4.2.1. *For all policies p,* $[\![p]\!] = \biguplus_{x \in G(p)} [\![x]\!]$

Lemma 4.2.2. *If* $x, y \in I$*, then* $[\![x]\!] = [\![y]\!]$ *iff* $x = y$

Lemma 4.2.3. *For all policies* p, q*,* $[\![p]\!] = [\![q]\!]$ *iff* $G(p) = G(q)$

4.2.3 Cost NetKAT Normal form

Definition 4.2.1. *A cost NetKAT policy p is in normal form if* $R(p) \in I$*. A policy p is normalisable if it is provably equivalent to a policy in normal form.*

Lemma 4.2.4. *Every cost-NetKAT policy p is normalisable*

4.2.4 Completeness Proof

Lemma 4.2.5. *If $R(p) \in I$, then $R(p) = G(p)$*

The proofs of all the above lemmas can be found in Appendix B

Theorem 4.2.1. *If $[\![p]\!] = [\![q]\!]$, then $\vdash p \equiv q$*

Proof. Let $\hat{p}, \hat{q}$ be the normal forms of p, q respectively.

$$\vdash p \equiv \hat{p}, \vdash q \equiv \hat{q}$$

$$\Rightarrow [\![p]\!] = [\![\hat{p}]\!],\ [\![q]\!] = [\![\hat{q}]\!]$$

$$\Rightarrow [\![\hat{p}]\!] = [\![\hat{q}]\!]$$

$$\Rightarrow G(\hat{p}) = G(\hat{q})$$

$$G(\hat{p}) = R(\hat{p}), G(\hat{q}) = R(\hat{q})$$

$$\Rightarrow R(\hat{p}) = R(\hat{q})$$

Since $R(\hat{p}), R(\hat{q})$ are regular sets, $\vdash \hat{p} \equiv \hat{q}$ Also, $\vdash p \equiv \hat{p}$, $\vdash q \equiv \hat{q}$

$$\therefore \vdash p \equiv q$$

□

4.3 Reachability properties

We only make an incremental change to state a new reachability test for the cost NetKAT policies.

Definition 4.3.1. *We say b is reachable from a with a cost at least as good as cost predicate c if and only if there exists a trace*

$$\langle pk_1, \cdots, pk_n \rangle \in [\![cp \cdot (p \cdot t \cdot cp)^*]\!]$$

such that $[\![a]\!]\langle pk_n \rangle = \{pk_n\}$ and $[\![b \cdot c]\!]\langle pk_1 \rangle = \{pk_1\}$

Theorem 4.3.1. *For non-cost predicates a and b, cost predicate c, policy p, and topology t, $a \cdot cp \cdot (p \cdot t \cdot cp)^* \cdot b \cdot c \not\equiv 0$, if and only if b is reachable from a with a cost at least as good as c.*

Proof. Translate the cost NetKAT equation into the language model:

$$a \cdot \mathtt{cp} \cdot (p \cdot t \cdot \mathtt{cp})^* \cdot b \cdot c \neq 0$$

$$\Rightarrow \exists \alpha, \pi_n, \cdots, \pi_1 \text{ s.t., } \alpha \cdot \pi_n \cdot \mathtt{cp} \cdots \mathtt{cp} \cdot \pi_1 \in G(a \cdot \mathtt{cp} \cdot (p \cdot t \cdot \mathtt{cp})^* \cdot b \cdot c)$$

Also translate each term in the semantic definition of reachability into the language model:

$$\langle pk_1, \cdots, pk_n \rangle \in [\![\mathtt{cp} \cdot (p \cdot t \cdot \mathtt{cp})^*]\!],$$

$$[\![a]\!]\langle pk_n \rangle = \{pk_n\}$$

$$[\![b \cdot c]\!]\langle pk_1 \rangle = \{pk_1\}$$

$$\Rightarrow \exists \pi'_m, \cdots, \pi'_1 \text{ s.t.,}$$

$$\alpha_{\pi'_m} \cdot \pi'_m \cdot \mathtt{cp} \cdots \mathtt{cp} \cdot \pi'_1 \in G(\mathtt{cp} \cdot (p \cdot t \cdot \mathtt{cp})^*)$$

$$\alpha_{\pi'_m} \in G(a) \text{ and}$$

$$\alpha_{\pi'_1} \in G(b \cdot c)$$

To prove soundness, let $\alpha = \alpha_{\pi'_m}$ and $m = n$ to show that if

$$\alpha \cdot \pi_n \cdot \mathtt{cp} \cdots \mathtt{cp} \cdot \pi_1 \in G(a \cdot \mathtt{cp} \cdot (p \cdot t \cdot \mathtt{cp})^* \cdot b \cdot c)$$

then,

$$\alpha \cdot \pi'_m \cdot \mathtt{cp} \cdots \mathtt{cp} \cdot \pi'_1 \in G(\mathtt{cp} \cdot (p \cdot t \cdot \mathtt{cp})^*)$$

Completeness proofs proceeds in the same manner. □

Similarly, following is a new way-pointing theorem

Definition 4.3.2. *We say w is a waypoint from a to b, if and only if, for all histories $\langle pk_1, \cdots, pk_n \rangle \in [\![cp \cdot (p \cdot t \cdot cp)^*]\!]$ where $[\![a]\!]\langle pk_n \rangle = \{pk_n\}$ and $[\![b]\!]\langle pk_1 \rangle = \{pk_1\}$, there exists a $pk_x \in \langle pk_1, \cdots, pk_n \rangle$ such that:*

- $[\![w]\!]\langle pk_x \rangle = \{pk_x\}$, *and*
- $[\![b]\!]\langle pk_i \rangle = \{\}$ *for all* $1 < i < x$, *and*
- $[\![a]\!]\langle pk_j \rangle = \{\}$ *for all* $x < j < n$

Theorem 4.3.2. *For non-cost predicates a, b, and w, and cost predicate c, $a \cdot cp \cdot (p \cdot t \cdot cp)^* \cdot b \cdot c \leq a \cdot cp \cdot (\neg b \cdot p \cdot t \cdot cp)^* \cdot w \cdot (\neg a \cdot p \cdot t \cdot cp)^* \cdot b$ if and only if all packets from a that reach b with a cost at least as good as c, are waypointed through w.*

Chapter 5

Inter-NetKAT

This work in this section builds on the work already done by Arun Shankar and Sanjiva Prasad.

5.1 Introduction

NetKAT in its current equational theory assumes a uniform network throughout the packet journey. However, there are many scenarios where a packet travels through many different networks in order to reach its final destination. In the path, an ASE (a router) may add an additional packet header on the top of the packet or may even pop some part of it. In fact, according to the traditional OSI model, a network packet travels across the vertical layers, namely, application layer, network layer, data link layer, etc. At the interface of each of these layers, an additional header is installed on the packet and then it is sent forward. After reaching the final destination, these headers are removed at the same corresponding interfaces.

NetKAT on the other hand, assumes packet to be composed of a fixed number of fields and that the packet travels through the same network throughout its journey. In this work, we try to extend the NetKAT syntax to accommodate this functionality of changing the network in between. With this addition, we will be able to treat the interfaces of this vertical stack as nothing but some ASEs in the network. We do this by using NetKAT homomorphisms to define semimodule operations which map the packets in one network to packets in the other. In the following section, we first build the theory only for non-cost fields in NetKAT.

5.2 KAT homomorphisms

Let $\mathscr{K}, \mathscr{K}'$ be Kleene algebras with tests, with $\mathscr{B}, \mathscr{B}'$ respectively being their boolean subalgebras. A KAT-homomorphism is a Kleene algebra homomorphism $h : \mathscr{K} \to \mathscr{K}'$ whose restriction to $\mathscr{B}$ is a boolean algebra homomorphism to $\mathscr{B}'$:

$$\begin{array}{cc} h(1) = 1' & h(x \cdot y) = h(x) \cdot' h(y) \\ h(0) = 0' & h(x + y) = h(x) +' h(y) \\ h(\overline{x}) = \overline{h(x)}' & h(x^*) = h(x)^{*'} \end{array}$$

where the unprimed and primed constants and operations on the left and the right side of the equations refer to the KAT operations in $\mathscr{K}, \mathscr{K}'$ respectively. Kleene algebras with tests and

KAT-homomorphisms form a category.

5.3 Non-cost NetKAT homomorphisms

The extension of KAT-homomorphisms to NetKAT requires a brief discussion. We first consider the properties of NetKAT homomorphisms on the boolean subalgebras. Let $h : \mathscr{B} \to \mathscr{B}'$ be a homomorphism between the boolean sub-algebras of the respective NetKAT structures.

Let $\mathscr{F} = \{x_1, x_2, ..., x_k\}$ and $\mathscr{F}' = \{x'_1, x'_2, ..., x'_m\}$ denote the sets of fields in the respective NetKAT structures. In the context of NetKAT, a *primitive test* in $\mathscr{B}$ is a *literal* b_i where $b_i \equiv (x_i = v)$, and a *complete test* is of the form $(x_1 = v_{1i_1}) \cdot \ldots \cdot (x_k = v_{ki_k})$ where $x_1, ..., x_k$ are all the fields in the packet header in some arbitrary but fixed order. If field x_i can assume n_i distinct values, then there are exactly $n_1 \times \ldots \times n_k$ different complete tests in NetKAT boolean subalgebra $\mathscr{B}$. These complete tests constitute the *atoms* of the Boolean algebra $\mathscr{B}$ because they are minimal nonzero elements of $\mathscr{B}$ generated by the tests. Similarly, if x'_i can assume n'_i distinct values, then there are exactly $n'_1 \times \ldots \times n'_m$ different complete tests in NetKAT algebra $\mathscr{B}'$. Let A and A' denote the set of complete tests of $\mathscr{B}$ and $\mathscr{B}'$ respectively.

A NetKAT homomorphism maps each *complete test* in A to a *test* in $\mathscr{B}'$, or equivalently to a subset of complete tests in At'.

NetKAT homomorphisms on complete tests. NetKAT-homomorphisms, being KAT-homomorphisms, must satisfy two additional properties: *exclusivity* and *exhaustiveness*.

Proposition 5.3.1 (Exclusivity)**.** *If a_i and a_j are two distinct primitive tests on the same field, $h(a_i \cdot a_j) = 0$.*

Here a_i, a_j will be of the form $x = v_i$ and $x = v_j$ respectively for some field x, with $v_i \neq v_j$. From Packet Axiom PA-CONTRA, $(x = v_i) \cdot (x = v_j) \equiv 0$ if $v_i \neq v_j$. Thus, for h to be a valid NetKAT homomorphism, $h(a_i \cdot a_j) = 0$.

Corollary 5.3.1. *For any two different complete tests $\alpha_1, \alpha_2 \in \mathsf{A}$, $h(\alpha_1 \cdot \alpha_2) = 0$.*

In particular, two *different* complete tests, which must differ in at least one particular field, cannot be mapped to the same non-zero element.

Proposition 5.3.2 (Exhaustiveness)**.** *If $\{v_1, v_2, v_3, ..., v_n\}$ is the set of values that a field x can assume, then, $h(\sum_{i=1}^{n} x = v_i) = 1$.*

From the packet algebra axiom PA-MATCH-ALL, $\sum_{i=1}^{n}(x = v_i) = 1$. Thus, for h to be a valid NetKAT homomorphism, *exhaustiveness* must hold.

Corollary 5.3.2. *Let* A *be the set of complete tests in* $\mathscr{B}$*. Then* $h(\sum_{\alpha\in\mathsf{A}}\alpha) = 1$.

From the corollaries 5.3.1 and 5.3.2, it is clear that a homomorphism maps each point in A to a partition (set of points) of At', such that the union is exhaustive. In a matrix interpretation, the homomorphic image of a point (a matrix with a single 1 on the main diagonal) is a set of points along the main diagonal. We will talk more about this in the sequel.

If for complete test $\alpha \in \mathsf{A}$, $h(\alpha) = \sum_i^l \beta_i$, for some $\beta_1, \ldots, \beta_l \in \mathsf{A}'$, then $\beta \in \mathsf{A}' \leq h(\alpha)$ iff β is one of the summands β_i.

NetKAT homomorphisms on Tests. Talking only about the mapping of complete tests is not sufficient; we also need to specify what happens to tests that are not complete. Recall that every primitive test b in $\mathscr{B}$ is equivalent to a sum of complete tests: $b \equiv \sum_{\alpha\in\mathsf{A}\leq b}\alpha$.

Suppose $b \equiv (x = v)$ Let $\mathsf{A}\wr[x = v] \stackrel{\Delta}{=} \{\alpha \in \mathsf{A} \mid \alpha \cdot (x = v) \neq 0\}$. Note that in each $\alpha' \in \mathsf{A}\wr[x = v]$, the value for field $x = v$.

$$\begin{aligned}(x = v) &\equiv 1 \cdot (x = v)\\ &\equiv (\sum_{\alpha\in\mathsf{A}}\alpha)\cdot(x = v)\\ &\equiv \sum_{\alpha\in\mathsf{A}}\alpha\cdot(x = v)\\ &\equiv \sum_{\alpha'\in\mathsf{A}\wr[x=v]}\alpha'\\ &\equiv \sum_{\alpha'\leq x=v}\alpha'\end{aligned} \tag{5.1}$$

Observe that $\alpha \in \mathsf{A} \leq (x = v)$ iff $\alpha' \in \mathsf{A}\wr[x = v]$.

We define the mapping for a primitive test $b \in \mathscr{B}$ as follows:

$$h(b) = h(\sum_{\alpha\leq b}\alpha) = \sum_{\alpha\leq b} h(\alpha) \tag{5.2}$$

In a matrix interpretation, the homomorphic image of a test (a matrix with some 1's on the main diagonal) is a matrix with 1's along the main diagonal which is obtained as the sum of the image matrices, where no two share a 1 in any position.

NetKAT homomorphisms for assignments. Now we characterise the homomorphic mapping for *assignments* in $\mathscr{K}$. Let P, P' denote the *complete assignments* in $\mathscr{K}, \mathscr{K}'$. In NetKAT, a *primitive assignment* is of the form $x_i \leftarrow v$, and a *complete assignment* is of the form $x_1 \leftarrow v_{1i_1}; \cdots; x_k \leftarrow v_{ki_k}$. In shorthand notation, we write the complete tests and assignments as $\vec{x} = \vec{v}$ and $\vec{x} \leftarrow \vec{v}$ respectively.

As we see that the mapping that we defined for NetKAT complete and primitive tests, qual-

ifies to be a valid NetKAT homomprhism. However, for a valid homomorphism, the condition that $h(x \cdot y) = h(x) \cdot h(y)$, makes such a mapping in the case of complete and primitive assignments very restrictive. So instead, we allow a weaker condition for mapping the assignments in one network to programs in another network, i.e., $g(x \cdot y) \leq g(x) \cdot g(y)$. Since we have already established that the mapping we require satisfies to be a valid homomorphism for the boolean subalgebra, the mapping g induces the same homomorphism h for the strongest post conditions of the NetKAT assignments and programs in general. The strongest post-condition of a NetKAT program is defined as follows.

In NetKAT, the complete tests and complete assignments are in 1-1 correspondence according to the values $\vec{v}$. So, if α is an atom (complete test), the corresponding complete assignment is denoted by π_α and if π is a complete assignment, then the corresponding complete test is denoted by α_π.

Definition 5.3.1. *For a NetKAT program p, such that the normal form of the program is* $p \equiv \Sigma_i \alpha_i \cdot \pi_i \cdot \cdots \cdot cp \cdot \pi'_i$*, the the strongest post-condition of p is defined as* $sp(p) \triangleq \Sigma_i \alpha_{\pi'_i}$

We will write $\pi_\beta \leq h(\pi_\alpha)$ if π_β is one of the summands. In a matrix interpretation, the homomorphic image of a complete assignment is a matrix with 1's in each column indexed by an atom in the corresponding image of the complete test.

In general, let $h(\pi) = p_1 + p_2 + \cdots + p_n$, where each p_i is of the form $\alpha \cdot \pi_1 \cdot \mathrm{cp} \cdots \mathrm{cp} \cdot \pi_j$. Since in NetKAT, π_α is an assignment that will bring the packet to the point α, then the homomorphic image of π_α must also bring the packet finally to $h(\alpha)$. This can also be stated and derived mathematically as follows.

Lemma 5.3.1. *If* $h(\pi_\alpha) = \Sigma p_i$*, where each* p_i *is of the form* $p_i = \Sigma \alpha_i \cdot \pi_i \cdot cp \cdots cp \cdot \pi'_i$*, then for each* p_i, $\alpha_{\pi'_i} \leq h(\alpha)$

Proof.

$$\pi_\alpha = \pi_\alpha \cdot \alpha$$

$$\Rightarrow h(\pi_\alpha) = h(\pi_\alpha \cdot \alpha) \leq h(\pi_\alpha) \cdot h(\alpha)$$

$$\text{Assume } p \leq h(\pi_\alpha) \text{ s.t., } p = \alpha' \cdot \pi'_1 \cdot \mathrm{cp} \cdots \mathrm{cp} \cdot \pi'_n \text{ where } \pi'_n \not\leq h(\alpha)$$

$$\Rightarrow h(p) \cdot (\alpha) \equiv 0$$

$$\Rightarrow h(\pi) \not\leq h(\pi_\alpha) \cdot h(\alpha)$$

This contradicts the original statement.

□

Also, $\alpha \cdot \pi_\alpha \equiv \alpha$. This means that if a packet is already at the point α, the we do not need any assignment operation to bring it to α. Thus we can posit the following requirement on homomorphic images of a complete test:

Lemma 5.3.2. *If* $h(\alpha) = \sum_{i=1}^{l} \beta_i$ *then* $\sum_{i=1}^{l} \pi_{\beta_i} \leq h(\pi_\alpha)$

Proof.

$$\begin{aligned} h(\alpha) \cdot' h(\pi_\alpha) &= (\sum_{i=1}^{l} \beta_i) \cdot' (\sum_{j=1}^{l} \pi_{\beta_j}) \\ &= \sum_{i=1}^{l} (\beta_i \cdot' (\sum_{j=1}^{l} \pi_{\beta_j})) \\ &\geq \sum_{i=1}^{l} (\beta_i \cdot' \pi_{\beta_i}) \\ &= \sum_{i=1}^{l} \beta_i \\ &= h(\alpha) \\ &= h(\alpha \cdot \pi_\alpha) \end{aligned} \tag{5.3}$$

□

Once we have defined how homomorphisms behave for complete tests, we can automatically extend the definition for the primitive assignments. The primitive assignments in $\mathscr{K}$ can be written as a sum of complete tests followed by complete assignments. Let $(x \leftarrow v)$ be a primitive assignment. Then,

$$\begin{aligned} (x \leftarrow v) &= 1 \cdot (x \leftarrow v) \\ &= (\sum_{\alpha \in At} \alpha \cdot \pi_\alpha) \cdot (x \leftarrow v) \\ &= (\sum_{\alpha \in At} \alpha \cdot \pi_{\alpha[x \leftarrow v]}) \end{aligned} \tag{5.4}$$

where $\pi_{\alpha[x \leftarrow v]}$ is π_α with the assignment to x replaced by $x \leftarrow v$. Therefore,

$$h(x \leftarrow v) = h(\sum_{\alpha \in At} \alpha \cdot \pi_{\alpha[x \leftarrow v]}).$$

Finally, we complete the definition of NetKAT homomorphisms on cp:

$$h(\mathsf{cp}) = \mathsf{cp}$$

Let $L \subseteq \mathsf{A} \cdot \Pi \cdot (\mathsf{cp} \cdot \Pi)^*$ be a (regular) set of *reduced strings*. Each string $x \in L$ is of the form $\alpha \cdot r_0 \cdot \mathsf{cp} \cdot r_1 \cdots \mathsf{cp} \cdot r_n$, $n \geq 0$, where $\alpha \in \mathsf{A}$, and each $r_i \in \Pi$. Using the properties of homomorphisms, let us define

$$h(x) \triangleq \{y \equiv \beta \cdot s_0 \cdot \mathsf{cp} \cdot s_1 \cdots \mathsf{cp} \cdot s_n \mid \beta \in h(\alpha),\ \forall i : s_i \in h(r_i)\}$$

At the language level, this lifts naturally to the following:

$$h(L) = \bigcup_{x \in L} h(x).$$

5.3.1 Preimage

We deduce from the Exclusivity and Exhaustiveness properties that any NetKAT homomorphism h from A creates a partition in the target set of atoms A'. Therefore, it is quite clear that the cardinality of A' has to be greater than that of A. Thus the observation that each atom in $\mathscr{B}'$ has exactly one inverse mapping in $\mathscr{B}$. This inverse mapping may or may not itself be a homomorphism depending on the cardinalities of A and A'.

Since complete assignments in NetKAT are in one-to-one correspondence with complete tests, for each the complete assignment in P', there is exactly one inverse mapping to a complete assignment in P. This leads us to the definition of the *preimage* h^{-1}

Definition 5.3.2.

$$h^{-1}(\beta) = \alpha \text{ iff } \beta \leq h(\alpha)$$

Proposition 5.3.3. $h^{-1}(\beta)$ *is unique.*

Proof. Let α_1 and α_2 be the preimages of β under h.

$$\Rightarrow h(\alpha_1) = \beta = h(\alpha_2)$$

$$\Rightarrow h(\alpha_1).h(\alpha_2) = \beta$$

This violates the Exclusivity property.

□

Note that however, there can be more than one atoms in $\mathscr{K}'$ that might have the same preimage.
In the packet algebra, this is quite intuitive, in the sense that for every NetKAT network packet in $\mathscr{K}'$, we are certain about the field values that it would have been in the original NetKAT network $\mathscr{K}$.

5.3.2 NetKAT Refinement, Abstraction and Translation

The Exclusivity property and the cardinalities of the sets A and A' lead us to three cases of NetKAT homomorphisms.

NetKAT Refinement Let us consider the first case where the cardinality of the set A is smaller than that of At′, *i.e.*,

$$|\mathsf{A}| < |\mathsf{A}'|$$

Exclusivity says that each complete test of B is mapped to a complete test in B′ or to a union (sum) of more than one complete tests in such a way that two distinct complete tests in B are mapped to disjoint sets of compete tests in B′, *i.e.*, $h(\alpha_1).h(\alpha_2) = 0$ if $\alpha_1 \neq \alpha_2$. In other words, if $h(\alpha_1) = \beta_{i_1} + \beta_{i_2} + ... + \beta_{i_k}$, and $h(\alpha_2) = \beta_{j_1} + \beta_{j_2} + ... + \beta_{j_l}$, then the sets $\{\beta_{i_1}, \beta_{i_2}, ..., \beta_{i_k}\}$ and $\{\beta_{j_1}, \beta_{j_2}, ..., \beta_{j_l}\}$ are disjoint. The homomorphism h partitions At′. Since the cardinality of At′ is larger than A, there must be at least one complete test in A that is mapped to a union of complete tests in A′.

$$\exists \alpha' \in \mathsf{A} : h(\alpha') = \sum_{i=1}^{l} \beta_{j_i}, l > 1$$

In such a case, the inverse mapping from A′ to A will *not* be a homomorphism. This is because if the inverse exists, then $\forall \beta \in \{\beta_{j_1}, ..., \beta_{j_l}\}, h^{-1}(\beta) = \alpha'$. However, Exclusivity does not allow this.

NetKAT Abstraction From the discussion above on NetKAT refinements, it is clear that the inverse mapping of a NetKAT Refinement cannot be a homomorphism. Therefore, abstractions cannot be NetKAT homomorphisms.

NetKAT Translation The last situation is when the cardinalities of sets A and A′ are equal, *i.e.*, $|\mathsf{A}| = |\mathsf{A}'|$. In such a case, each complete test of A will be mapped to exactly one distinct complete test in A′. In this case, the homomorphism will be a total function, and therefore the inverse mapping from A′ to A will also be a homomorphism. This is an interesting situation with many applications and some special properties.

Therefore, NetKAT translation naturally induces a valid *inverse homomorphism* given by the preimage h^{-1} as defined earlier. NetKAT Refinements, however, do not induce any such inverse.

Proposition 5.3.4. *h^{-1} is a valid NetKAT homomorphism iff* $|\mathsf{A}| = |\mathsf{A}'|$

5.3.3 What happens to NetKAT Automaton?

Theorem 5.3.1. *Let $h : \mathscr{K} \to \mathscr{K}'$ be a Non-cost NetKAT homomorphism. Let Σ, B, A, Π denote the set of actions, tests, complete tests, and complete assignments respectively in $\mathscr{K}$, and the corresponding components of $\mathscr{K}'$ be Σ', B', A', Π'.*
If $\mathscr{L} \subseteq \mathsf{A} \cdot \Pi \cdot (cp \cdot \Pi)^*$ is regular, then so is its image h(L).

Theorem 5.3.2. *If $h(\mathscr{L}) \subseteq \mathsf{A}' \cdot \Pi' \cdot (cp \cdot \Pi')^*$ is regular, then so its pre-image $\mathscr{L}$.*

Refer to C for the proofs of the above theorems.

Theorem 5.3.3. *Given a NetKAT homomorphism h,*

1. *If* $L_1, L_2 \subseteq \mathsf{A} \cdot \Pi \cdot (cp \cdot \Pi)^*$ *such that* $L_1 \subseteq L_2$*, then* $h(L_1) \subseteq h(L_2)$
2. *If* $L_1, L_2 \subseteq \mathsf{A} \cdot \Pi \cdot (cp \cdot \Pi)^*$ *such that* $L_1 \cap L_2 = \phi$*, then* $h(L_1) \cap h(L_2) = \phi$
3. *If* $L_1, L_2 \subseteq \mathsf{A} \cdot \Pi \cdot (cp \cdot \Pi)^*$ *such that* $h(L_1) \subseteq h(L_2)$*, then* $L_1 \subseteq L_2$

Note however that $h^{-1}(L_1)$ and $h^{-1}(L_2)$ need not be disjoint even if L_1 and L_2 are disjoint, i.e., $L_1 \cap L_2 = \phi \not\Rightarrow h^{-1}(L_1) \cap h^{-1}(L_2) = \phi$. This holds only when h is a NetKAT Translation.

5.4 Cost NetKAT homomorphisms

Homomorphisms for Cost NetKAT can be defined in a similar fashion as non-cost NetKAT. There are some major differences between both the homomorphisms:

- **Monotonicity:** The homomorphism for cost part must be a monotonic funcion.

$$\text{If } c_1 \succeq c_2, \text{ then } h_c(c_1) \succeq h_c(c_2)$$

- For any two complete tests in the cost part of NetKAT, $a_i \cdot a_j \not\equiv 0$. Hence, exclusivity defined in the previous section does not hold.

The preimage can be generalised to the following:

Definition 5.4.1.

$$h^{-1}(z) = \bigvee \{x \mid z \leq h(x)\}$$

$$h^{-1}(z_1 + z_2) = h^{-1}(z_1) h^{-1}(z_2)$$

$$x_1 \leq x_2 \;\Rightarrow\; h^{-1}(x_1) \leq h^{-1}(x_2) \tag{5.5}$$

$$x \;\leq\; h(h^{-1}(x)) \tag{5.6}$$

$$x \;=\; h^{-1}(h(x)) \tag{5.7}$$

This makes h a **Galvois Insertion**

Finally, in Cost InterNetKAT the homomorphism has two parts - a) non-cost homomorphism, b) cost homomorphism. Composing two networks is best done using semimodules.

5.4.1 Left semimodule

Left semimodule is an operator that maps the elements from $N_1 \times N_2$ to N_2

$$\triangleright : N_1 \times N_2 \to N_2$$

The left semimodule must satisfy the following properties:

1. $x \triangleright (y_1 + y_2) = x \triangleright y_1 + x \triangleright y_2$
2. $(x_1 + x_2) \triangleright y = x_1 \triangleright y + x_2 \triangleright y$
3. $(x_1 \cdot x_2) \triangleright y = x_1 \triangleright (x_2 \triangleright y)$
4. $1 \triangleright y = y$
5. $0 \triangleright y = 0 = x \triangleright 0$

In our case, we define the left-semimodule as :

$$x \triangleright y \triangleq h(x) \cdot y$$

Since the mapping h is a homomorphism on the strngest postconditions and the weakest preconditions of the network programs, and satisfies only a weaker property in general, this semimodule consequently satisfies a weaker property of associativity, i.e., $(x_1 \cdot x_2) \triangleright y \leq x_1 \triangleright (x_2 \triangleright y)$

Proof.

$$\begin{aligned} x \triangleright (y_1 + y_2) &= h(x) \cdot (y_1 + y_2) \\ &= h(x) \cdot y_1 + h(x) \cdot y_2 \\ &= x \triangleright y_1 + x \triangleright y_2 \end{aligned}$$

$$\begin{aligned} (x_1 + x_2) \triangleright y &= h(x_1 + x_2) \cdot y \\ &= (h(x_1) + h(x_2)) \cdot y \\ &= h(x_1) \triangleright y + h(x_2) \triangleright y \\ &= x_1 \triangleright y + x_2 \triangleright y \end{aligned}$$

$$\begin{aligned} (x_1 \cdot x_2) \triangleright y &= h(x_1 \cdot x_2) \cdot y \\ &\leq h(x_1) \cdot h(x_2) \cdot y \\ &= h(x_1) \cdot (x_2 \triangleright y) \\ &= x_1 \triangleright (x_2 \triangleright y) \end{aligned}$$

$$\begin{aligned} 1 \rhd y &= h(1) \cdot y \\ &= 1 \cdot y \\ &= y \end{aligned}$$

$$\begin{aligned} 0 \rhd y &= h(0) \cdot y \\ &= 0 \cdot y \\ &= 0 \\ x \rhd 0 &= h(x) \cdot 0 \\ &= 0 \end{aligned}$$

□

5.4.2 Right semimodule

Right semimodule is an operator that maps the elements from $N_1 \times N_2$ to N_1

$$\lhd : N_1 \times N_2 \to N_1$$

The right semimodule must satisfy the following properties:

1. $x \lhd (y_1 + y_2) = x \lhd y_1 + x \lhd y_2$
2. $(x_1 + x_2) \lhd y = x_1 \lhd y + x_2 \lhd y$
3. $(x_1 \cdot x_2) \lhd y = x_1 \lhd (x_2 \lhd y)$
4. $x \lhd 1 = x$
5. $0 \lhd y = 0 = x \lhd 0$

In our case, we define the left-semimodule only for the boolean subalgebra as :

$$x \rhd y \triangleq x \cdot h^{-1}(y)$$

where x, y are the NetKAT policies. For a NetKAT test, $h^{-1}(y)$ is the preimage of y. For an assignment, $h^{-1}(p) \triangleq h^{-1}(\pi_{\mathtt{sp}(p)})$

Proof.

$$\begin{aligned} x \lhd (y_1 + y_2) &= x \cdot h^{-1}(y_1 + y_2) \\ &= x \cdot (h^{-1}(y_1) + h^{-1}(y_2)) \\ &= x \lhd h^{-1}(y_1) + x \lhd h^{-1}(y_2) \\ &= x \lhd y_1 + x \lhd y_2 \end{aligned}$$

$$\begin{aligned} (x_1 + x_2) \lhd y &= (x_1 + x_2) \cdot h^{-1}(y) \\ &= x_1 \cdot h^{-1}(y) + x_2 \cdot h^{-1}(y) \\ &= x_1 \lhd y + x_2 \lhd y \end{aligned}$$

$$\begin{aligned} (x_1 \cdot x_2) \rhd y &= (x_1 \cdot x_2) \cdot h^{-1}(y) \\ &= x_1 \cdot (x_2 \cdot h^{-1}(y)) \\ &= x_1 \cdot (x_2 \lhd y) \\ &= x_1 \lhd (x_2 \lhd y) \end{aligned}$$

$$\begin{aligned} x \lhd 1 &= x \cdot h^{-1}(1) \\ &= x \cdot 1 \\ &= x \end{aligned}$$

$$\begin{aligned} 0 \lhd y &= 0 \cdot h^{-1}(y) \\ &= 0 \\ x \lhd 0 &= x \cdot h^{-1}(0) \\ &= x \cdot 0 \\ &= 0 \end{aligned}$$

□

5.5 Cost InterNetKAT

The problem with the current version of NetKAT is that it lets the programmer define policies within one network, i.e., the set of the header fields remain fixed. However, present internet is composed various Autonomous Networks with their own policies and protocols. Some may use IP protocol while other may just use Ethernet protocol for routing packets from one host to the

other. The signature of the packet headers and the values are therefore decided according to the protocol in which they operate. An IPv4 packet has a completely different header from that of a IPv6 packet. So, NetKAT program written in a network cannot route messages to another network in its current design. For two separate networks, two separate NetKAT programs will be written, however, there is no way to compose them to write one unified program and as a result, in no way can the packets be exchanged reliably between two networks.

This brings us to defining **Cost InterNetKAT**, where we can write programs that can send messages from one network to the other by changing the set of packet header fields. There are two different ways in which one can think of composing two NetKAT programs. One is the horizontal composition and the second is the vertical composition. In the horizontal composition, a packet simply jumps from one network to another. For example, a message generated in Network N_1 can be sent to a host in adjacent Network N_2 by jumping from N_1 to N_2 somewhere in the path. In case of vertical composition, the message source and destination are both in the same network, but the packet needs to travel through some other network to reach the final destination. For example, in the layered structure, a packet at the Network layer at Host A reaches Network layer of Host B, but in the middle, it travels through the physical layer.

5.5.1 Horizontal Composition

For horizontal composition, we just need a mapping m, which is a valid homomorphism for the boolean subalgebra of the KAT. $\rightsquigarrow$ changes the network of the packet. The semantics now operate not on the list of histories which we call as the **Trace**. i.e., $[\![p]\!] \in \mathrm{T} \rightarrow 2^{\mathrm{T}}$. Figures 5.1 and 5.2 show the syntax for Cost InterNetKAT.

5.6 Future Work

- Proving the correctness of Horizontal Composition
- Finding solution to vertical composition.

Non-cost Fields	f	$::= f_1 \mid \cdots \mid f_k$	
Cost Fields	c	$::= c_1 \mid \cdots \mid c_l$	
Packets	pk	$::= \{f_1 = v_1, \cdots, f_k = v_k,\ c_1 = \sigma_1, \cdots c_l = \sigma_l\}$	
History	h	$::= \langle pk \rangle \mid pk :: h$	
Trace	t	$::= \{\!\{h\}\!\} \mid \{\!\{h\ ,\ t\}\!\}$	
Mapping	m		
Predicates	a, b	$::= 1$	*Identity*
		$\mid 0$	*Drop*
		$\mid f = n$	*Non-cost Test*
		$\mid c^{lo} \geq \sigma$	*Local cost Test*
		$\mid c^{cu} \geq \sigma$	*Cumulative cost Test*
		$\mid a + b$	*Disjunction*
		$\mid a.b$	*Conjunction*
		$\mid \neg a$	*Negation*
Policies	p, q	$::= a$	*Filter*
		$\mid f \leftarrow n$	*Modification*
		$\mid c \twoheadleftarrow \sigma$	*Increment*
		$\mid p + q$	*Union*
		$\mid p \cdot q$	*Sequential composition*
		$\mid p \rightsquigarrow_m q$	*Jump*
		$\mid p^*$	*Kleene star*
		$\mid \mathtt{cp}$	*Checkpoint*

Figure 5.1: Cost InterNetKAT syntax

$$[\![p]\!] \in \mathrm{T} \to 2^{\mathrm{T}}$$

$$[\![1]\!]\, t = \{t\}$$

$$[\![0]\!]\, t = \{\}$$

$$[\![f = n]\!]\{\!|(pk :: h)\ ,\ t|\!\} = \begin{cases} \{\ \{\!|(pk :: h)\ ,\ t|\!\}\ \} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases}$$

$$[\![c^{lo} \geq \sigma]\!]\{\!|(pk :: h)\ ,\ t|\!\} = \begin{cases} \{\ \{\!|(pk :: h)\ ,\ t|\!\}\ \} & \text{if } pk.c^{lo} \geq \sigma \\ \{\} & \text{otherwise} \end{cases}$$

$$[\![c^{cu} \geq \sigma]\!]\{\!|(pk :: h)\ ,\ t|\!\} = \begin{cases} \{\ \{\!|(pk :: h)\ ,\ t|\!\}\ \} & \text{if } pk.c^{cu} \geq \sigma \\ \{\} & \text{otherwise} \end{cases}$$

$$[\![\neg a]\!]\, t = \{t\} \backslash ([\![a]\!]\, t)$$

$$[\![f \leftarrow n]\!]\{\!|(pk :: h)\ ,\ t|\!\} = \{\ \{\!|(pk[c^{lo} \circ \sigma][c^{cu} \circ \sigma] :: h)\ ,\ t|\!\}\ \}$$

$$[\![c \twoheadleftarrow \sigma]\!]\{\!|(pk :: h)\ ,\ t|\!\} = \{\ \{\!|(pk[f := n] :: h)\ ,\ t|\!\}\ \}$$

$$[\![p + q]\!]\, t = [\![p]\!]\, t \uplus [\![q]\!]\, t$$

$$[\![p.q]\!]\, t = ([\![p]\!] \bullet [\![q]\!])\, t$$

where $\bullet$ is the Kleisli composition

$$[\![p \rightsquigarrow_m q]\!]\{\!|(pk :: h)\ ,\ t|\!\} = \bigcup \{\ \{\!|(pk')\ ,\ (pk :: h)\ ,\ t|\!\}\ \}$$

where $pk' \in m(pk)$

$$[\![p^*]\!]\, t = \biguplus_i F_p^i\, t$$

where $F_p^0\, t = \{t\}$ and $F_p^{i+1}\, t = ([\![p]\!] \bullet F_p^i)\, t$

$$[\![\mathsf{cp}]\!]\{\!|(pk :: h)\ ,\ t|\!\} = \{\ \{\!|(pk[c^{lo} := \mathbf{e}] :: pk :: h)\ ,\ t|\!\}\ \}$$

Figure 5.2: Cost InterNetKAT semantics

Appendix A

Soundness of Cost NetKAT

Axiom A.0.1.

$$(c_1^\star \geq \sigma_1) \cdot (c_2^\bullet \geq \sigma_2) \equiv (c_2^\bullet \geq \sigma_2) \cdot (c_1^\star \geq \sigma_1)$$

$$(c_1 \twoheadleftarrow \sigma_1) \cdot (c_2 \twoheadleftarrow \sigma_2) \equiv (c_2 \twoheadleftarrow \sigma_2) \cdot (c_1 \twoheadleftarrow \sigma_1)$$

$$(c_1 \twoheadleftarrow \sigma_1) \cdot (c_2^\bullet \geq \sigma_2) \equiv (c_2^\bullet \geq \sigma_2) \cdot (c_1 \twoheadleftarrow \sigma_1)$$

$$(c \twoheadleftarrow \sigma) \cdot (f = n) \equiv (f = n) \cdot (c \twoheadleftarrow \sigma)$$

$$(c \twoheadleftarrow \sigma) \cdot (f \leftarrow n) \equiv (f \leftarrow n) \cdot (c \twoheadleftarrow \sigma)$$

$$(c \geq \sigma) \cdot (f = n) \equiv (f = n) \cdot (c \twoheadleftarrow \sigma)$$

$$(c \geq \sigma) \cdot (f \leftarrow n) \equiv (f \leftarrow n) \cdot (c \twoheadleftarrow \sigma)$$

$$\star, \bullet \in \{lo, cu\}$$

Proof. The order of two cost assignments or cost filters does not matter if it concerns two different cost fields. □

Axiom A.0.2.

$$(c^{cu} \geq \sigma) \cdot (c^{lo} \geq \sigma) \equiv c^{cu} \geq \sigma \equiv (c^{lo} \geq \sigma) \cdot (c^{cu} \geq \sigma)$$

Proof. This axiom uses the fact that the cumulative cost and the local cost increase by the same amount whenever a new cost is incurred. Also, on a check point operation (cp), the local cost is reset, while the cumulative cost remains unchanged. So, the cumulative cost is always greater than the local cost, i.e., $c^{lo} \geq c^{cu}$

$$\text{Let } (pk :: h, pk :: h) \in [c^{cu} \geq m]$$

$$\Rightarrow pk.c^{cu} \geq m$$

$$\text{Also, } c^{lo} \geq c^{cu}$$

$$\Rightarrow pk.c^{lo} \geq \sigma$$

$$(pk :: h, pk :: h) \in [(c^{cu} \geq \sigma) \cdot (c^{lo} \geq \sigma)]$$

$$(pk :: h, pk :: h) \in [(c^{lo} \geq \sigma) \cdot (c^{cu} \geq \sigma)]$$

The other directions can be proved in a similar fashion.

□

Axiom A.0.3.

$$If\, \sigma_1 \succeq \sigma_2,\ then$$

$$(c^{cu} \succeq \sigma_1) + (c^{cu} \succeq \sigma_2) \equiv (c^{cu} \succeq \sigma_2)$$

$$(c^{lo} \succeq \sigma_1) + (c^{lo} \succeq \sigma_2) \equiv (c^{lo} \succeq \sigma_2)$$

Proof.

$$\text{Let } (pk :: h, pk :: h) \in [c^{cu} \succeq \sigma_2]$$

$$\Rightarrow pk.c^{cu} \succeq \sigma_2$$

$$\text{Also, } \sigma_1 \succeq \sigma_2$$

$$\Rightarrow pk.c^{cu} \succeq \sigma_1$$

$$\Rightarrow (pk :: h, pk :: h) \in [c^{cu} \succeq \sigma_1]$$

$$\Rightarrow (pk :: h, pk :: h) \in [(c^{cu} \succeq \sigma_1) + (c^{cu} \succeq \sigma_2)]$$

The other direction can be proved in a similar fashion.

□

Axiom A.0.4.

$$If\, \sigma_1 \succeq \sigma_2,\ then$$

$$(c^{cu} \succeq \sigma_1) \cdot (c^{cu} \succeq \sigma_2) \equiv (c^{cu} \succeq \sigma_1)$$

$$(c^{lo} \succeq \sigma_1) \cdot (c^{lo} \succeq \sigma_2) \equiv (c^{lo} \succeq \sigma_1)$$

Proof.

$$\text{Let } (pk :: h, pk :: h) \in [c^{cu} \succeq \sigma_1]$$

$$\Rightarrow pk.c^{cu} \succeq \sigma_1$$

$$\text{Also, } \sigma_1 \succeq \sigma_2$$

$$\Rightarrow pk.c^{cu} \succeq \sigma_2$$

$$\Rightarrow (pk :: h, pk :: h) \in [c^{cu} \succeq \sigma_2]$$

$$\Rightarrow (pk :: h, pk :: h) \in [(c^{cu} \succeq \sigma_1) \cdot (c^{cu} \succeq \sigma_2)]$$

The other direction can be proved in a similar fashion.

□

Axiom A.0.5.

$$If\, \sigma_1 \succeq \sigma_2,\ then$$

$$(c \twoheadleftarrow \sigma_1) + (c \twoheadleftarrow \sigma_2) \equiv (c \twoheadleftarrow \sigma_1)$$

Proof. If it is known that the only difference between p_1 and p_2 is regarding the cost incurred, and p_1 is strictly better program than p_2 in terms of cost, then $p_1 + p_2 \equiv p_1$ □

Axiom A.0.6.

$$(c \twoheadleftarrow \sigma_1) \cdot (c \twoheadleftarrow \sigma_2) \equiv (c \twoheadleftarrow (\sigma_1 \circ \sigma_2))$$

Proof. Trivial. □

Axiom A.0.7.

$$(c^{cu} \succeq \sigma_1) \cdot (c \twoheadleftarrow \sigma_2) \leq (c \twoheadleftarrow \sigma_2) \cdot (c^{cu} \succeq (\sigma_1 \circ \sigma_2))$$

$$(c^{lo} \succeq \sigma_1) \cdot (c \twoheadleftarrow \sigma_2) \leq (c \twoheadleftarrow \sigma_2) \cdot (c^{lo} \succeq (\sigma_1 \circ \sigma_2))$$

Proof.

$$\text{Let } (pk :: h, pk' :: h) \in [(c^{cu} \succeq \sigma_1) \cdot (c \twoheadleftarrow \sigma_2)]$$

$$\Rightarrow pk.c^{cu} \succeq \sigma_1$$

$$\Rightarrow pk'.c^{cu} \succeq \sigma_1 \circ \sigma_2$$

$$\Rightarrow (pk :: h, pk' :: h) \in [(c \twoheadleftarrow \sigma_2) \cdot (c^{cu} \succeq (\sigma_1 \circ \sigma_2))]$$

□

Axiom A.0.8.

$$(c \twoheadleftarrow \sigma_1); (c^{lo} \succeq \sigma_2) \equiv (c^{lo} \succeq \sigma_2/\sigma_1); (c \twoheadleftarrow \sigma_1)$$

$$(c \twoheadleftarrow \sigma_1); (c^{cu} \succeq \sigma_2) \equiv (c^{cu} \succeq \sigma_2/\sigma_1); (c \twoheadleftarrow \sigma_1)$$

Proof. By the definition of Residuals. This is an important axiom that will be used in the completeness proof. Let's refer to this axiom as *Push-back*. □

Appendix B

Completeness of Cost NetKAT

Lemma B.0.1. *For all policies p,* $[\![p]\!] = \biguplus_{x \in G(p)} [\![x]\!]$

Proof. By structural induction on p.
Base cases:

$$[\![\pi]\!] = [\![\sum_{\beta} \beta \cdot \epsilon_\infty \cdot \delta_\infty \cdot \pi]\!] = \bigcup_{x \in G(\pi)} [\![x]\!] = \biguplus_{x \in G(\pi)} [\![x]\!]$$

$$[\![\alpha]\!] = [\![\alpha \cdot \pi_\alpha]\!] = \biguplus_{x \in G(\alpha)} [\![x]\!]$$

$$[\![\mathsf{cp}]\!] = [\![\sum_{\beta} \beta \cdot \epsilon_\infty \cdot \delta_\infty \cdot \theta_\beta \cdot \kappa_0 \cdot \mathsf{cp} \cdot \theta_\beta \cdot \kappa_0]\!] = \bigcup_{x \in G(\mathsf{cp})} [\![x]\!] = \biguplus_{x \in G(\mathsf{cp})} [\![x]\!]$$

Induction steps:

$$\begin{aligned}
[\![p + q]\!] &= [\![p]\!] \uplus [\![q]\!] \\
&= (\biguplus_{x \in G(p)} [\![x]\!]) \uplus (\biguplus_{x \in G(q)} [\![x]\!]) \\
&= \biguplus_{x \in G(p) \uplus G(q)} [\![x]\!] \\
&= \biguplus_{x \in G(p+q)} [\![x]\!]
\end{aligned}$$

$$\begin{aligned}
[\![p \cdot q]\!] &= [\![p]\!] \bullet [\![q]\!] \\
&= (\biguplus_{x \in G(p)} [\![x]\!]) \bullet (\biguplus_{y \in G(q)} [\![y]\!]) \\
&= \biguplus_{x \in G(p)} \biguplus_{y \in G(q)} [\![x]\!] \bullet [\![y]\!] \\
&= \biguplus_{x \in G(p)} \biguplus_{y \in G(q)} [\![x \cdot y]\!] \\
&= \biguplus_{x \in G(p)} \biguplus_{y \in G(q)} [\![x \diamond y]\!] \\
&= \biguplus_{z \in G(p \cdot q)} [\![z]\!]
\end{aligned}$$

$$[\![p^*]\!] = \biguplus_{n \geq 0} [\![p^n]\!] = \biguplus_{n \geq 0} \biguplus_{x \in G(p^n)} [\![x]\!] = \biguplus_{x \in \uplus_{n \geq 0} G(p^n)} [\![x]\!] = \biguplus_{x \in G(p^*)} [\![x]\!]$$

□

Lemma B.0.2. *If* $x, y \in I$, *then* $[\![x]\!] = [\![y]\!]$ *iff* $x = y$

Proof. Trivial □

Lemma B.0.3. *For all policies p, q, $[\![p]\!] = [\![q]\!]$ iff $G(p) = G(q)$*

Proof. Reverse direction:

$$[\![p]\!] = \biguplus_{x \in G(p)} [\![x]\!] = \biguplus_{x \in G(q)} [\![x]\!] = [\![q]\!]$$

Forward direction:

$$\begin{aligned}
[\![p]\!] = [\![q]\!] &\Rightarrow \biguplus_{x \in G(p)} [\![x]\!] = \biguplus_{y \in G(q)} [\![y]\!] \\
&\Rightarrow \forall h, \left(\biguplus_{x \in G(p)} [\![x]\!] h = \biguplus_{y \in G(q)} [\![y]\!] h \right) \\
&\Rightarrow \forall h \forall x \in G(p) \left([\![x]\!] h \subseteq \biguplus_{y \in G(q)} [\![y]\!] h \right) \\
&\Rightarrow \forall h \forall x \in G(p) \exists y \in G(q) \left([\![x]\!] h \subseteq \biguplus_{y \in G(q)} [\![y]\!] h \right) \\
&\Rightarrow \forall h \forall x \in G(p) \exists y \in G(q) (x = y) \\
&\Rightarrow G(p) \subseteq G(q)
\end{aligned}$$

Similarly, $G(q) \subseteq G(p)$. Therefore, $G(p) = G(q)$ □

Lemma B.0.4. *Every cost-NetKAT policy p is normalisable*

Proof. Base cases:

$$\begin{aligned}
f \leftarrow n &\equiv \Sigma\, \beta \cdot \epsilon_\infty \cdot \delta_\infty \cdot \theta'_\beta \cdot \kappa_0 \\
c \twoheadleftarrow \sigma &\equiv \Sigma\, \beta \cdot \epsilon_\infty \cdot \delta_\infty \cdot \theta'_\beta \cdot \kappa_\sigma \\
\mathtt{cp} &\equiv \Sigma\, \beta \cdot \epsilon_\infty \cdot \delta_\infty \cdot \theta_\beta \cdot \kappa_0 \cdot \mathtt{cp} \cdot \theta_\beta \cdot \kappa_0 \\
f = n &\equiv \Sigma\, \beta' \cdot \epsilon_\infty \cdot \delta_\infty \cdot \theta_{\beta'} \cdot \kappa_0 \\
c^{lo} \succeq \sigma &\equiv \Sigma\, \beta \cdot \epsilon'_c \cdot \delta_\infty \cdot \theta_\beta \cdot \kappa_0 \\
c^{cu} \succeq \sigma &\equiv \Sigma\, \beta \cdot \epsilon'_c \cdot \delta'_c \cdot \theta_\beta \cdot \kappa_0
\end{aligned}$$

The induction steps are quite complicated, one can refer to [BCG17] for a deeper insight as to how to convert the cost NetKAT policies to a normal form. □

Lemma B.0.5. *If $R(p) \in I$, then $R(p) = G(p)$*

Proof. Let $R(p) \subseteq I$. Since $G(x) = \{x\}$ for $x \in I$,

$$G(p) = \biguplus_{x \in R(p)} G(x) = \biguplus_{x \in R(p)} \{x\} = R(p)$$

□

Appendix C

NetKAT Automata

Theorem C.0.1. *Let $h : \mathscr{K} \to \mathscr{K}'$ be a Non-cost NetKAT homomorphism. Let Σ, B, A, Π denote the set of actions, tests, complete tests, and complete assignments respectively in $\mathscr{K}$, and the corresponding components of $\mathscr{K}'$ be Σ', B', A', Π'.*
If $\mathscr{L} \subseteq \mathsf{A} \cdot \Pi \cdot (cp \cdot \Pi)^*$ is regular, then so is its image h(L).

Proof. Since $\mathscr{L}$ is regular, there is a deterministic NetKAT automaton $\mathscr{A} = \langle S, s, \delta_{\alpha\beta}, \epsilon_{\alpha\beta}, \mathsf{Accept} \rangle$ recognising it. $\mathscr{A}$ is a NetKAT coalgebra with a distinguished start state $s \in S, \delta_{\alpha\beta} : S \to S$ is a continuation map, and $\epsilon_{\alpha\beta} : S \to 2$ is the observation map. Inputs to the automaton are the reduced strings in the set $N = \mathsf{A} \cdot \Pi \cdot (\mathsf{cp} \cdot \Pi)^*$ consisting of strings of the form $\alpha \cdot \pi_0 \cdot \mathsf{cp} \cdot \pi_1 \cdot \mathsf{cp} \cdots \mathsf{cp} \cdot \pi_n$ for some $n \geq 0$. Acceptance is determined by a predicate $\mathsf{Accept} : S \times N \to 2$ defined co-inductively:

$$\mathsf{Accept}(t, \alpha \cdot \pi_\beta \cdot \mathsf{cp} \cdot x) = \mathsf{Accept}(\delta_{\alpha\beta}, \beta x)$$

$$\mathsf{Accept}(t, \alpha\pi_\beta) = \epsilon_{\alpha\beta}(t)$$

A reduced string $x \in N$ is accepted by $\mathscr{A}$ iff $\mathsf{Accept}(s, x)$. Create a new automaton

$$\mathscr{A}' = \langle S, s, \delta'_{\alpha\beta}, \epsilon'_{\alpha\beta}, \mathsf{Accept}' \rangle$$

for $h(L)$ as follows. The set of states and the start state are the same as in $\mathscr{A}$. $\delta'_{\alpha\beta} : S \to S$ is the continuation map, and $\epsilon'_{\alpha\beta} : S \to 2$ is the observation map. Define $\delta'_{\alpha\beta}, \epsilon'_{\alpha\beta}$ and Accept' as follows (note that since the inverses α_h^{-1}'s are unique *atoms* in A, this gives us a deterministic automaton):

$$\delta'_{\alpha\beta}(t) = \delta_{h^{-1}(\alpha)h^{-1}(\beta)}$$

$$\epsilon'_{\alpha\beta}(t) = \epsilon_{h^{-1}(\alpha)h^{-1}(\beta)}$$

We must now prove that $L(\mathscr{A}') = h(L(\mathscr{A}))$. The input to $\mathscr{A}'$ are reduced strings x' of the set $N' = \mathsf{A}' \cdot \Pi' \cdot (\mathsf{cp} \cdot \Pi')^*$. It suffices to prove that $\mathsf{Accept}'(s, x') = \mathsf{Accept}(s, h^{-1}(x'))$ for all $x' \in N'$. The proof is by induction on the number of checkpoints in x'.

Base case: $x = \alpha\pi_\beta \in N$. Then, $x' = \alpha'\pi_{\beta'} \in N'$, where $\alpha' \leq h(\alpha)$, and $\beta' \leq h(\beta)$.

$$\begin{aligned}
\mathsf{Accept}'(t, x') &= \mathsf{Accept}'(t, \alpha'\pi_{\beta'}) \\
&= \epsilon'_{\alpha'\beta'}(t) \\
&= \epsilon_{\alpha\beta}(t) \\
&= \mathsf{Accept}(t, \alpha\pi_\beta) \\
&= \mathsf{Accept}(t, h^{-1}(x'))
\end{aligned} \tag{C.1}$$

Induction hypothesis: Assume $\mathsf{Accept}'(t, x') = \mathsf{Accept}(t, h^{-1}(x'))$ if x' has n $\mathtt{cp}$ for $n \geq 0$

Induction case: $x = \alpha \cdot \pi_\beta \mathtt{cp} x_1$ where x_1 has n $\mathtt{cp}$. Then, $x' \leq h(x)$ is of the form $\alpha' \cdot \pi_{\beta'} \cdot \mathtt{cp} \cdot x'_1$, where $x'_1 \leq h(x_1)$ has n $\mathtt{cp}$, and $\alpha' \leq h(\alpha)$ and $\beta' \leq h(\beta)$

$$\begin{aligned}
\mathsf{Accept}'(t, x') &= \mathsf{Accept}'(t, \alpha'\pi_{\beta'}\mathtt{cp}x'_1) \\
&= \mathsf{Accept}'(\delta'_{\alpha'\beta'}(t), x'_1) \\
&= \mathsf{Accept}'(\delta_{\alpha\beta}(t), x'_1) \\
&= \mathsf{Accept}(\delta_{\alpha\beta}(t), x_1) \\
&= \mathsf{Accept}(t, \alpha\pi_\beta\mathtt{cp}x_1) \\
&= \mathsf{Accept}(t, x) \\
&= \mathsf{Accept}(t, h^{-1}(x'))
\end{aligned} \tag{C.2}$$

Since $\mathsf{Accept}'(t, x') = \mathsf{Accept}(t, h^{-1}(x'))$ for all $t \in S$, it is specifically so for the distinguished start state $s \in S$, i.e., $\mathsf{Accept}'(s, x') = \mathsf{Accept}(s, h^{-1}(x'))$ for all $x' \in N'$. In other words, $L(\mathscr{A}') = h(L(\mathscr{A}))$ □

Theorem C.0.2. *If $h(\mathscr{L}) \subseteq \mathsf{A}' \cdot \Pi' \cdot (cp \cdot \Pi')^*$ is regular, then so its pre-image $\mathscr{L}$.*

Proof. Since $h(\mathscr{L})$ is regular, there is a deterministic NetKAT automaton $\mathscr{A}' = \langle S, s, \delta'_{\alpha\beta}, \epsilon'_{\alpha\beta}, \mathsf{Accept}\rangle$ recognising it. $\delta'_{\alpha'\beta'} : S \to S$ is a continuation map, and $\epsilon_{\alpha'\beta'} : S \to 2$ is the observation map. Here, $\alpha', \beta' \in \mathsf{At}'$. Construct a *non-deterministic* automaton $\mathscr{A} = \langle S, s, \delta_{\alpha\beta}, \epsilon_{\alpha\beta}, \mathsf{Accept}\rangle$ with state space S, start state s, and continuation , and observation maps $\delta_{\alpha\beta} : S \to 2^S$ and $\epsilon_{\alpha\beta} : S \to 2$ defined as follows:

$$\delta_{\alpha\beta}(t) = \{\delta'_{\alpha'\beta'}(t) \mid \alpha' \leq h(\alpha), \beta' \leq h(\beta)\}$$

$$\epsilon_{\alpha\beta}(t) = \bigvee_{\alpha' \leq h(\alpha), \beta' \leq h(\beta)} \epsilon'_{\alpha'\beta'}(t)$$

where $t \in S$. Determinization is effected because there might be multiple atoms α' in $\mathscr{K}'$ s.t. $\alpha' \leq h(\alpha)$. Acceptance is determined by the Accept predicate, exactly as done previously. It is

straightforward to show by induction (on the number of dups) that the language accepted by this automaton is $\mathscr{L}$.

Base case: $x = \alpha'\pi_{\beta'} \in N'$. Then, $x = \alpha\pi_\beta \in N$, where $\alpha' \leq h(\alpha)$, and $\beta' \leq h(\beta)$.

$$\begin{aligned}
\mathsf{Accept}(t, x) &= \mathsf{Accept}(t, \alpha\pi_\beta) \\
&= \epsilon_{\alpha\beta}(t) \\
&= \bigvee_{\alpha' \leq h(\alpha), \beta' \leq h(\beta)} \epsilon'_{\alpha'\beta'}(t) \\
&= \bigvee_{\alpha' \leq h(\alpha), \beta' \leq h(\beta)} \mathsf{Accept}'(t, \alpha'\pi_{\beta'}) \\
&= \mathsf{Accept}'(t, h(x))
\end{aligned} \tag{C.3}$$

Induction hypothesis: Assume $\mathsf{Accept}(t, x) = \mathsf{Accept}'(t, h(x))$ if x has n $\mathtt{cp}$ for $n \geq 0$

Induction case: $x' = \alpha' \cdot \pi_{\beta'} \cdot \mathtt{cp} \cdot x'_1$ where x'_1 has n $\mathtt{cp}$. Then, $x : x' \leq h(x)$ is of the form $\alpha \cdot \pi_\beta \cdot \mathtt{cp} \cdot x_1$, where $x'_1 \leq h(x_1)$ has n $\mathtt{cp}$, and $\alpha' \leq h(\alpha)$ and $\beta' \leq h(\beta)$

$$\begin{aligned}
\mathsf{Accept}(t, x) &= \mathsf{Accept}(t, \alpha \cdot \pi_\beta \cdot \mathtt{cp} \cdot x_1) \\
&= \mathsf{Accept}(\delta_{\alpha\beta}(t), x_1) \\
&= \bigvee_{\alpha' \leq h(\alpha), \beta' \leq h(\beta)} \mathsf{Accept}(\delta'_{\alpha'\beta'}(t), x_1) \\
&= \bigvee_{\alpha' \leq h(\alpha), \beta' \leq h(\beta)} \mathsf{Accept}'(\delta'_{\alpha'\beta'}(t), h(x_1)) \\
&= \bigvee_{\alpha' \leq h(\alpha), \beta' \leq h(\beta)} \mathsf{Accept}'(t), \alpha' \cdot \pi_{\beta'} \cdot \mathtt{cp} \cdot h(x_1)) \\
&= \mathsf{Accept}'(t, h(\alpha) \cdot h(\pi_\beta) \cdot \mathtt{cp} \cdot h(x_1)) \\
&= \mathsf{Accept}'(t, h(x))
\end{aligned} \tag{C.4}$$

Therefore, $L(\mathscr{A}) = \mathscr{L}$ □

Bibliography

[AFG+14] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: semantic foundations for networks. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 113–126. ACM, 2014.

[AKG+16] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. Snap: Stateful network-wide abstractions for packet processing. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 29–43, New York, NY, USA, 2016. Association for Computing Machinery.

[ave21] Avenir: Managing data plane diversity with control plane synthesis. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, April 2021.

[BCG17] Ryan Beckett, Eric Campbell, and Michael Greenberg. Kleene Algebra Modulo Theories. *arXiv e-prints*, page arXiv:1707.02894, July 2017.

[BDG+14] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[BFH+17] S. Basu, N. Foster, H. Hojjat, P. Palacharla, C. Skalka, and X. Wang. Life on the edge: Unraveling policies into configurations. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 178–190, 2017.

[BG09] John N. Billings and Timothy G. Griffin. A model of internet routing using semimodules. In Rudolf Berghammer, Ali Jaoua, and Bernhard Möller, editors, *Relations and Kleene Algebra in Computer Science, 11th International Conference on Relational Methods in Computer Science, RelMiCS 2009, and 6th International Conference on Applications of Kleene Algebra, AKA 2009, Doha, Qatar, November 1-5, 2009. Proceedings*, volume 5827 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2009.

[BGW16] Ryan Beckett, Michael Greenberg, and David Walker. Temporal netkat. In Chandra Krintz and Emery Berger, editors, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 386–401. ACM, 2016.

[BMM+19] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. Don't mind the gap: Bridging network-wide objectives and device-level configurations: Brief reflections on abstractions for network programming. *SIGCOMM Comput. Commun. Rev.*, 49(5):104–106, November 2019.

[DGG18] Matthew L. Daggitt, Alexander J. T. Gurney, and Timothy G. Griffin. Asynchronous convergence of policy-rich distributed bellman-ford routing protocols. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 103–116, New York, NY, USA, 2018. Association for Computing Machinery.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[FKM+15] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for netkat. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 343–355. ACM, 2015.

[FKM+16] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic netkat. In *Proceedings of the 25th European Symposium on Programming Languages and Systems - Volume 9632*, page 282–309, Berlin, Heidelberg, 2016. Springer-Verlag.

[GS05] Timothy Griffin and João Sobrinho. Metarouting. volume 35, pages 1–12, 10 2005.

[HM12] Peter Höfner and Bernhard Möller. Dijkstra, floyd and warshall meet kleene. *Form. Asp. Comput.*, 24(4–6):459–476, July 2012.

[KKPB07] Martin Karsten, Srinivasan Keshav, Sanjiva Prasad, and Mirza Beg. An axiomatic basis for communication. In Jun Murai and Kenjiro Cho, editors, *Proceedings of the ACM SIGCOMM 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, August 27-31, 2007*, pages 217–228. ACM, 2007.

[MAB+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[MHFv16] Jedidiah McClurg, Hossein Hojjat, Nate Foster, and Pavol Černý. Event-driven network programming. *SIGPLAN Not.*, 51(6):369–385, June 2016.

[SEFG15] Steffen Smolka, Spiridon Aristides Eliopoulos, Nate Foster, and Arjun Guha. A fast compiler for netkat. In Kathleen Fisher and John H. Reppy, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*, pages 328–341. ACM, 2015.

[SKF+17] Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. Cantor meets scott: Semantic foundations for probabilistic networks. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 557–571, New York, NY, USA, 2017. Association for Computing Machinery.

[SKK+19] Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. Scalable verification of probabilistic networks. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 190–203, New York, NY, USA, 2019. Association for Computing Machinery.

[VS19] Alexander Vandenbroucke and Tom Schrijvers. P$\lambda\omega$nk: Functional probabilistic netkat. *Proc. ACM Program. Lang.*, 4(POPL), December 2019.

[ZP15] Lenore D. Zuck and Sanjiva Prasad. A switch, in time. In Pierre Ganty and Michele Loreti, editors, *Trustworthy Global Computing - 10th International Symposium, TGC 2015, Madrid, Spain, August 31 - September 1, 2015 Revised Selected Papers*, volume 9533 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2015.