Journal of Computer Security 0 (2018) 1–48
IOS Press

Reasoning About Firewall Policies Through Refinement and Composition

Ultan Neville [a,*], Simon N. Foley [b]
[a] *Department of Computer Science, University College Cork, Ireland*
E-mail: u.neville@cs.ucc.ie
[b] *IMT Atlantique, Rennes, France*
E-mail: simon.foley@imt-atlantique.fr

Abstract. Network and host-based access controls, for example, firewall systems, are important points of security-demarcation, operating as a front-line defence for networks and networked systems. A firewall policy is conventionally defined as a sequence of order-dependant rules, and when a network packet matches with two or more policy rules, the policy is anomalous. Policies for access-control mechanisms may consist of thousands of access-control rules, and correct management is complex and error-prone. We argue that a firewall policy should be anomaly-free by construction, and as such, there is a need for a firewall policy language that allows for constructing, comparing, and composing anomaly-free policies. In this paper, an algebra is proposed for constructing and reasoning about anomaly-free firewall policies. Based on the notion of refinement as safe replacement, the algebra provides operators for sequential composition, union and intersection of policies. The effectiveness of the algebra is demonstrated by its application to anomaly detection, and standards compliance. The effectiveness of the approach in practice is evaluated through a mapping to/from iptables. The algebra is used to specify and reason about iptables firewall policy configurations. A prototype policy management toolkit has been implemented.

Keywords: Firewalls, Algebra, iptables, Anomalies, Policy-composition

1. Introduction

Firewall policy management is complex and error-prone. A misconfigured policy may permit accesses that were intended to be denied and/or vice-versa. We regard the specification of a firewall policy as a process that evolves, as threats to, and access requirements for, resources behind a firewall do not usually remain static, and over time, a policy or distributed policy configuration may be updated on an ad-hoc basis, possibly by multiple specifiers/administrators. This can be problematic and may introduce anomalies; whereby the intended semantics of the specified access controls become ambiguous.

A firewall policy is conventionally defined as a sequence of order-dependent rules. A rule is composed of filter conditions and a target action. Filter conditions usually consist of fields/attributes from IP, TCP/UDP headers; with the most commonly used attributes being source/destination IP/port, and network protocol. Target actions are usually *allow* or *deny*. When a network packet matches with two or more policy rules, the policy is anomalous [2, 10].

A firewall policy may be developed as a collection of independent or related specifications that an administrator will need to replace by a policy that adequately captures the requirements of the individual specifications. A configuration may need to be updated with additional policy specifications when a new

[*]Corresponding author. E-mail: u.neville@cs.ucc.ie.

threat is identified, or when a new permissible access is required. Therefore, having a consistent means of composing these specifications is desirable. The objective of this paper is to develop a theory about composing anomaly-free firewall policies. When a policy is anomaly-free, there is no ambiguity as to whether a given network packet is allowed or denied by the firewall. Our goal is to provide an algebra where policies are anomaly-free by construction.

Example 1. When configuring the rules that define a firewall policy, the specifier must understand the relationship of each rule to every other rule in the policy. Consider, as a running example, a company that employs a team of administrators and a team of developers. There are two network security policy requirements, whereby network traffic destined to the IP range $[1 \,..\, 3]$ on ports $[1 \,..\, 3]$ is to be allowed from the administrators, and traffic destined to the IP range $[2 \,..\, 4]$ on ports $[2 \,..\, 4]$ is to be allowed from the developers. For ease of exposition, we give the IPs as natural numbers, and the IP and port ranges as intervals of $\mathbb{N}$. For simplicity, we do not consider the source IP ranges for the administrators and the developers.

Specifying The Requirements. System Administrator *Bob* manages the network access controls for the administration and development teams. He specifies the network security policy requirement for the administration team as: $\mathsf{Pol}_{\mathsf{Admin}} == \langle(\mathsf{1}, [1 \,..\, 3], [1 \,..\, 3], allow)\rangle$, whereby $\mathsf{1}$ is the position of the rule in the policy, the first instance of $[1 \,..\, 3]$ is the required destination IP range, the second instance of $[1 \,..\, 3]$ is the required destination port range, and *allow*, means that network traffic matching this pattern is permitted traversal of the firewall. Similarly, for the development team, he specifies: $\mathsf{Pol}_{\mathsf{Dev}} == \langle(\mathsf{1}, [2 \,..\, 4], [2 \,..\, 4], allow)\rangle$. Bob needs to combine $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$ into a single firewall policy, and security requirements may change. He requires a consistent means of composing firewall policies, whereby the result is anomaly-free and upholds the enforcements of each policy involved in the composition.

Sequential Composition. To specify the firewall policy for the company, he tries sequentially composing $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$ as follows: $\mathsf{Pol}_{\mathsf{Admin}} \frown \mathsf{Pol}_{\mathsf{Dev}} = \langle(\mathsf{1}, [1..3], [1..3], allow), (\mathsf{2}, [2..4], [2..4], allow)\rangle$. This approach, however, does not does yield the desired result. That is, in the above specification $(\mathsf{Pol}_{\mathsf{Admin}} \frown \mathsf{Pol}_{\mathsf{Dev}})$, the rule at position $\mathsf{1}$ allows some of the IP/port pairs allowed by the rule at position $\mathsf{2}$ and vice-versa. Therefore, the policy is anomalous. From this, we have that the naïve sequential composition of rules is not a consistent operation when specifying an anomaly-free policy.

A Flattening Approach. To specify the firewall policy, Bob considers the approach whereby for each IP/port pair allowed by the company's network security policy requirements, there is a rule to allow each IP/port pair. He specifies: $\langle(\mathsf{1}, [1 \,..\, 1], [1 \,..\, 1], allow), (\mathsf{2}, [1 \,..\, 1], [2 \,..\, 2], allow) \,..\, (\mathsf{14}, [4 \,..\, 4], [4 \,..\, 4], allow)\rangle$. This policy does provide the desired result, in that it is both anomaly-free and consistent with the two network security policy requirements involved in the composition. We observe however, that this approach is tedious, error-prone and not practical for large numbers of IPs/ports or large numbers of policy rules.

A Different Approach. Bob is required to specify the firewall policy whereby policy rules allow the IP/port-range pairs from either $\mathsf{Pol}_{\mathsf{Admin}}$ or $\mathsf{Pol}_{\mathsf{Dev}}$. To specify the policy for the company, he decides to compose the two requirements into a single requirement as follows: $\langle(\mathsf{1}, [1 \,..\, 4], [1 \,..\, 4], allow)\rangle$. This approach, however, is inconsistent with the two network security policy requirements outlined by the company. That is, the result of composition is an overly-permissive firewall policy, whereby network traffic is permitted to IP $\mathsf{1}$ on port $\mathsf{4}$, and to IP $\mathsf{4}$ on port $\mathsf{1}$. Conversely, this approach would result in an overly-restrictive policy if the rules had a target action of *deny*.

A Better Approach. Bob is required to specify the firewall policy whereby the desired result is the smallest number of anomaly-free rules that allow all the IP/port pairs from either $\mathsf{Pol}_{\mathsf{Admin}}$ or $\mathsf{Pol}_{\mathsf{Dev}}$. He specifies the policy: $\langle(1,[1\mathinner{..}4],[2\mathinner{..}3],\mathit{allow}),(2,[1\mathinner{..}3],[1\mathinner{..}1],\mathit{allow}),(3,[2\mathinner{..}4],[4\mathinner{..}4],\mathit{allow})\rangle$. In this case, the result is as desired, as it is the smallest number of anomaly-free rules that allow all the IP/port pairs from either $\mathsf{Pol}_{\mathsf{Admin}}$ or $\mathsf{Pol}_{\mathsf{Dev}}$.

Mutual Policy Enforcements. Bob receives a request to configure the policy that allows the IP/port pairs from both $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$. He specifies: $\langle(1,[2\mathinner{..}3],[2\mathinner{..}3],\mathit{allow})\rangle$. In this case, the specification provides the desired result, and defines the smallest number of anomaly-free rules that allow all the IP/port pairs from both $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$. We observe that the resulting policy upholds the restrictions of both $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$.

Conflicting Policy Decisions. Suppose, for example, the policy rules specified were: $\langle(1,[1\mathinner{..}3],[1\mathinner{..}3],\mathit{allow}),(2,[2\mathinner{..}4],[2\mathinner{..}4],\mathit{deny})\rangle$. Then this policy is anomalous, as rule 2 denies packets already allowed by rule 1. As such, then a semantically-equivalent interpretation of the policy, defining the smallest number of anomaly-free rules, may be specified by Bob as follows: $\langle(1,[1\mathinner{..}3],[1\mathinner{..}3],\mathit{allow}),(2,[2\mathinner{..}4],[4\mathinner{..}4],\mathit{deny}),(3,[4\mathinner{..}4],[2\mathinner{..}3],\mathit{deny})\rangle$. $\triangle$

For an administrator, it may be relatively straightforward to understand policy composition where only a small number of rules are involved, however, this does not scale. We argue that to reason confidently a policy or distributed policy configuration is anomaly-free and adequately mitigates the identified threats; a common framework is required, whereby knowledge related to detailed access control configurations and standards-based firewall policies can be represented and reasoned about.

In this paper, we present a firewall policy algebra $\mathcal{FW}_1$ for constructing and reasoning over anomaly-free policies. The algebra allows policies to be composed in such a way, that the result upholds the access requirements of each policy involved; and permits one to reason as to whether some policy is a safe (secure) replacement for another policy in the sense of [19, 20, 28]. In this paper, when one policy is considered a safe (secure) replacement for another, then this means that the former is no less restrictive than the latter. The proposed algebra is used to reason about *iptables* firewall policy configurations. We derive a filter condition specification for $\mathcal{FW}_1$ from a collection of attributes expressible in firewall rules from the iptables filter table. iptables is a command line utility used to define policies for the Linux kernel firewall *Netfilter* [43]. We focus on stateful and stateless firewall policies that are defined in terms of constraints on source and destination IP and port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes. Note, the notion of state in iptables is an abstraction, where given literals signify user-land 'states' that packets within tracked connections can be related to.

The primary contribution of this paper is an algebra $\mathcal{FW}_1$, that can be used to reason about firewall policies using refinement and composition operators. The effectiveness of the algebra is demonstrated by using it to characterise anomalies in firewall policies and to define standards compliance. The effectiveness of the approach in practice is evaluated through a mapping to/from iptables. The evaluation shows that the approach is practical for large policies. This paper is a revised and extended version of the work in [37]. In the sequel, we encode additional filtering specifications in $\mathcal{FW}_1$, in particular, for OSI Layer 2 packet-types and MAC addresses, OSI Layer 7 Linux UIDs and GIDs, and Application layer protocols recognisable by iptables. We also develop a definition for time-based filtering rules. We include new definitions for anomaly detection through policy composition, and describe how to incorporate an additional target action of 'log' for firewall rules in the algebra. An extended definition for firewall rule (duplet datatype) ordering, as well as implementation definitions for duplet join and difference are given.

The paper is organised as follows. In Section 2, we specify the core filter condition specification for rules in the $\mathcal{FW}_1$ algebra. The specification is derived from filter condition attributes expressible in firewall rules for the iptables filter table. Section 3 introduces the notion of adjacency, which is at the heart of reasoning about/composing firewall rules that involve IP/port ranges. In Section 4 we define datatypes for firewall rule attributes, such as IP/port ranges. Section 5 defines the firewall policy algebra $\mathcal{FW}_1$. In Section 6, we use $\mathcal{FW}_1$ to reason about firewall policies in practice. Section 7 describes a prototype policy management toolkit for iptables and presents some preliminary results. Related work is outlined in Section 8 and Section 9 concludes the paper. The Z notation [42] is used to provide a consistent syntax for structuring and presenting the definitions and examples in this paper. We use only those parts of Z that can be intuitively understood and Appendix A gives a brief overview of the notation used. Mathematical definitions have been syntax- and type-checked using the fUZZ tool.

2. Attributes of a Linux-based Firewall

In this section, the core filter condition attributes used to define firewall rules in the $\mathcal{FW}_1$ firewall algebra are formally specified. Attributes are derived from the Data Link, Network, Transport and Application Layers of the OSI model. Additional filter condition attributes are also defined. Attribute definitions are extended in Section 4 to specify range-based filter conditions used to define firewall rules in the $\mathcal{FW}_1$ policy algebra. The attributes defined in this paper are based on the expressiveness on the iptables firewall. The following example demonstrates the specification of a firewall rule using the iptables command-line syntax.

Example 2. The following iptables access-control rule specifies that inbound (`INPUT`) TCP packets (`-p tcp`) originating from the IP address 0.0.0.1 (`-s` 0.0.0.1) destined to the IP address 0.0.0.2 (`-d` 0.0.0.2) will be permitted traversal of the firewall (`-j ACCEPT`).

```
iptables -t filter -A INPUT -p tcp -s 0.0.0.1 -d 0.0.0.2 -j ACCEPT
```

Note, the filter table is the default table for iptables, therefore it is not necessary to include the (`-t filter`) option when specifying a firewall rule. △

2.1. Data Link Layer Filtering

Packet-type. iptables allows for the specification of firewall rules that filter the type of packet at the Data Link layer of the OSI model. Let *PktTpe* be the set of Data Link-layer packet types, whereby:

$$PktTpe ::= \mathsf{unicast} \mid \mathsf{broadcast} \mid \mathsf{multicast}$$

Media Access Control (MAC) Addresses. iptables allows for constructing firewall rules that filter the MAC address of a Ethernet device at the Data Link layer of the OSI model. Filtering is applied to source MAC addresses entering the FORWARD or INPUT chains of the filter table [43]. Let basic type *MAC* be the set of all MAC addresses. We define:

$$[MAC]$$

For simplicity, we do not consider how the values of *MAC* may be constructed, other than to assume that the usual human-readable notation can be used, such as 00:0F:EA:91:04:08 and 00:0F:EA:91:04:09 $\in$ *MAC*.

2.2. Network Layer Filtering

IP Addresses. iptables allows the specification of firewall rules that filter the source/destination IP address of a network packet. For simplicity, we consider only the IPv4 address range, as a firewall rule with an IPv6 address filter condition attribute must be specified using *ip6tables*; the equivalent IPv6 firewall [25]. In this paper, IP addresses as encoded using natural numbers, as there is a logical mapping from IPs to natural numbers, and a logical mapping from IP ranges and CIDR blocks to intervals of $\mathbb{N}$. This is done for ease of exposition, and to exploit the natural ordering of $\leqslant$ over $\mathbb{N}$. We define the constant maxIP, whereby:

$$\text{maxIP} == 2^{32} - 1$$

ICMP. iptables allows for filtering by ICMP Type and Code. Let *TypesCodes* be the set of all valid ICMP Type/Code pairs, as detailed in [40]. Most Type/Code pairs $i,j \in \mathbb{N}$ are given as $(i,j) \in$ *TypesCodes*, for example, (8,0) is an ICMP Echo Request, however, some ICMP Types $i \in \mathbb{N}$; for example, Type 19 (reserved for security), have no ICMP Code, and are given as $(i, -1) \in$ *TypesCodes*. Note, for ease of exposition we use syntactic sugar in the definition of *TypesCodes* as a free-type.

$$\begin{aligned} \mathit{TypesCodes} ::= {} & (0,0) \mid (3,0) \mid (3,1) \mid (3,2) \mid (3,3) \mid (3,4) \mid (3,5) \mid (3,6) \mid (3,7) \mid \\ & (3,8) \mid (3,9) \mid (3,10) \mid (3,11) \mid (3,12) \mid (3,13) \mid (3,14) \mid (3,15) \mid (4,0) \mid (5,0) \mid \\ & (5,1) \mid (5,2) \mid (5,3) \mid (6,0) \mid (8,0) \mid (9,0) \mid (9,16) \mid (10,0) \mid (11,0) \mid (11,1) \mid \\ & (12,0) \mid (12,1) \mid (12,2) \mid (13,0) \mid (14,0) \mid (15,0) \mid (16,0) \mid (17,0) \mid (18,0) \mid \\ & (19,-1) \mid (20,-1) \mid (21,-1) \mid (22,-1) \mid (23,-1) \mid (24,-1) \mid (25,-1) \mid (26,-1) \mid \\ & (27,-1) \mid (28,-1) \mid (29,-1) \mid (30,-1) \mid (31,-1) \mid (32,-1) \mid (33,-1) \mid (34,-1) \mid \\ & (35,-1) \mid (36,-1) \mid (37,-1) \mid (38,-1) \mid (39,-1) \mid (40,0) \mid (40,1) \mid (40,2) \mid \\ & (40,3) \mid (40,4) \mid (40,5) \mid (41,-1) \mid (253,-1) \mid (254,-1) \end{aligned}$$

2.3. Transport Layer Filtering

Network Ports. A network port is a communication end-point used by the Transport layer protocols (for example, TCP/UDP) of the OSI model. iptables allows for constructing firewall rules that filter by source/destination ports. A port is a 16-bit unsigned integer. We define the constant maxPrt, where:

$$\text{maxPrt} == 2^{16} - 1$$

UDP. iptables allows the specification of firewall rules that filter UDP traffic. Let *UDP* define the set of packet values for the UDP protocol; whereby 1 signifies that a packet is using UDP or 0 signifies it is not. We define:

$$\mathit{UDP} ::= 1 \mid 0$$

TCP. iptables allows for TCP firewall rules to be constructed using a pair of TCP flags specifications. The first, specifies the flags that are to be examined in a packet-header, and the second specifies the flags that are to be set in a packet-header, these are the *mask* and *comp* values for a TCP packet [25]. Let *Flags* be the set of TCP flags filterable in an iptables rule (as a mask or a comp specification), whereby:

$$Flags ::= \mathsf{syn} \mid \mathsf{ack} \mid \mathsf{fin} \mid \mathsf{psh} \mid \mathsf{rst} \mid \mathsf{urg}$$

and let $Flag_{Spec}$ be the set of all (mask, comp) pairs, we define:

$$Flag_{Spec} == \mathbb{P}\, Flags \times \mathbb{P}\, Flags$$

2.4. Application Layer Filtering

Layer 7 Protocol Filtering. iptables allows for constructing firewall rules that filter certain protocols at the Application Layer of the OSI model. Let $Proto_7^L$ be the set of all OSI Application-layer protocols recognised by iptables [33]. Note, for ease of exposition we use syntactic sugar in the definition of $Proto_7^L$ as a free-type. We define:

$Proto_7^L$ ::= 100bao | aim | aimwebcontent | applejuice | ares | armagetron |
audiogalaxy | battlefield1942 | battlefield2 | battlefield2142 | bgp | biff |
bittorrent | chikka | cimd | ciscovpn | citrix | counterstrike-source | cvs |
dayofdefeat-source | dazhihui | dhcp | directconnect | dns | doom3 | edonkey |
fasttrack | finger | freenet | ftp | gkrellm | gnucleuslan | gnutella | goboogy |
gopher | gtalk | guildwars | h323 | halflife2-deathmatch | hddtemp | hotline |
http | http-rtsp | http-dap | http-freshdownload | http-itunes | httpaudio |
httpcachehit | httpcachemiss | httpvideo | ident | imap | imesh | ipp | irc |
jabber | kugoo | live365 | liveforspeed | lpd | mohaa | msn-filetransfer |
msnmessenger | mute | napster | nbns | ncp | netbios | nntp | ntp | openft |
pcanywhere | poco | pop3 | pplive | pressplay | qq | quicktime | quake-halflife |
quake1 | radmin | rdp | replaytv-ivs | rlogin | rtp | rtsp | runesofmagic |
shoutcast | sip | skypeout | skypetoskype | smb | smtp | snmp | snmp-mon |
snmp-trap | socks | soribada | soulseek | ssdp | ssh | ssl | stun | subspace |
subversion | teamfortress2 | teamspeak | telnet | tesla | tftp | thecircle |
tonghuashun | tor | tsp | uucp | validcertssl | ventrilo | vnc | whois |
worldofwarcraft | x11 | xboxlive | xunlei | yahoo | zmaap

The Layer 7 protocol definitions specify how these protocol names correspond to regular expressions that are matched by Netfilter on the packet application layer data. For example, the following regular expression [32]:

```
^220[\x09-\x0d -~]*ftp
```

is used by the Netfilter firewall to match packets that are part of FTP traffic.

Packet-owner/creator Filtering. For locally generated packets; iptables allows filtering by various characteristics of the packet creator through the *owner* module. Filtering is applied to the OUTPUT chain of the filter table [43]. We define the constant maxUID for 32-Bit Linux UIDs, where:

$$\mathsf{maxUID} == 2^{32} - 1$$

Similarly, the constant maxGID for 32-Bit Linux GIDs is defined as:

$$\mathsf{maxGID} == 2^{32} - 1$$

2.5. Additional Filtering Specifications

State. The iptables *conntrack* module defines the *state* extension [43]. The state extension allows access to the connection tracking state for a packet. Let *State* be the set of connection tracking states for a packet/connection.

$$State ::= \mathsf{new} \mid \mathsf{established} \mid \mathsf{related} \mid \mathsf{invalid} \mid \mathsf{untracked}$$

Time-based Filtering. iptables allows for a filtering decision to be made if the packet arrival time/date is within a given range. The possible time range expressible in a rule is 1970-01-01T00:00:00 to 2038-01-19T04:17:07 [16], and is specified in ISO 8601 “T” notation. We define the constant maxTime, whereby $\mathcal{T}(\mathsf{date})$ is a function that converts a date specified in “T” notation to a Unix timestamp:

$$\mathsf{maxTime} == \mathcal{T}(\mathsf{2038\text{-}01\text{-}19T04{:}17{:}07})$$

Network Interfaces and Direction-oriented Filtering. iptables allows for a filtering decision to be made with respect to the interface the packet is arriving at/leaving through. Let basic type *IFACE* be the set of all interfaces on a machine, where for simplicity, we assume elements of *IFACE* resemble lo, eth0, wlan0, tun0, etc. We define:

$$[IFACE]$$

Network traffic can be inbound or outbound on an interface. Direction-oriented filtering is defined as *Dir*, whereby:

$$Dir ::= \mathsf{ingress} \mid \mathsf{egress}$$

iptables Chains. Let *Chain* be the set of chains for the iptables filter table, we define:

$$Chain ::= \mathsf{input} \mid \mathsf{output} \mid \mathsf{forward}$$

3. A Theory of Adjacency

Range-based filter condition attributes (for example, IPs/ports) have logical mappings to intervals of $\mathbb{N}$. For example, the port range that includes all ports from SSH upto and including HTTP can be written as the interval $[22 \mathrel{..} 80]$.

Example 3. Consider, as part of a running example, a system that is capable of enforcing firewall rules whereby the filter condition attribute for the rules is a destination port range only. A rule that allowed all ports from SSH to HTTP would be: $(i, [22 \mathrel{..} 80], \mathit{allow})$, where i is the index of the rule in the policy, $[22 \mathrel{..} 80]$ is the required port range, and *allow* means that network traffic matching this pattern is permitted. Suppose we had a second rule, that specifies *allow* everything from Quote Of The Day (QOTD) up to and including FTP Control, then $(j, [17 \mathrel{..} 21], \mathit{allow})$ specifies that for the rule at index j; the required port range $[17 \mathrel{..} 21]$ is allowed. Intuitively we can see that the port ranges for the rules at index i and index j are *adjacent*, and we may want to join rule i and rule j into a single rule that looks like $(k, [17 \mathrel{..} 80], \mathit{allow})$. This notion of adjacency becomes more complex when we consider comparing/composing firewall rules comprising two or more filter condition attributes. △

3.1. The Adjacency Specification

In this section we define the filter condition attribute relationships of *adjacency, disjointness* and *subsumption*. These can be applied to any ordered set, not just number intervals. These relationships are at the heart of adjacency, and ultimately the $\mathcal{FW}_1$ algebra.

Let $\mathcal{IV}[\mathit{min}, \mathit{max}]$ be the set of all intervals on the natural numbers, from *min* up to and including *max*. Intervals are defined by their corresponding sets.

$$\mathcal{IV}[\mathit{min}, \mathit{max}] == \{S : \mathbb{P}\,\mathbb{N} \mid \exists \bot, \top : S \bullet \forall x : S \bullet \mathit{min} \leqslant \bot \leqslant x \leqslant \top \leqslant \mathit{max}\}$$

Example 4. For ease of exposition and when no ambiguity arises, we may write an interval as a pair $[\bot \mathrel{..} \top]$, rather than by the set it defines. For example, $\mathcal{IV}[1,3] = \{[1 \mathrel{..} 1], [1 \mathrel{..} 2], [1 \mathrel{..} 3], [2 \mathrel{..} 2], [2 \mathrel{..} 3], [3 \mathrel{..} 3]\}$ △

Let $\mathit{IPv4}$ define the set of all possible IPv4 address ranges, whereby:

$$\mathit{IPv4} == \mathcal{IV}[0, \mathsf{maxIP}]$$

Similarly, let *Port* define the set of all possible network port ranges, whereby:

$$\mathit{Port} == \mathcal{IV}[0, \mathsf{maxPrt}]$$

Adjacency. The relation $(\_ \wr_{\_} \_)$ defines adjacency over any ordered set. Adjacency is a general notion that is not limited to $\mathbb{N}$. We generalize adjacency to any attribute of generic type X, whereby for $a, b \in X$, if $a \wr_X b$, then a and b are adjacent in the set X. The property of reflexivity is required as any $a \in X$ should be adjacent to itself, that is; if for any a, that $a \wr_X a$ did not hold, then an inconsistency would exist. Symmetry follows from a similar requirement, where for $a, b \in X$, if a is adjacent to b in X, then b must also be adjacent to a in X. The following schema defines a generic adjacency relation that can be instantiated for adjacency over different datatypes.

$$
\begin{array}{|l}
[X] \\
\hline
\_ \wr\_ \_ : \mathbb{P}X \nrightarrow (X \leftrightarrow X) \\
\hline
\forall a, b : X \bullet \\
\qquad a \wr_X a \wedge \\
\qquad (a \wr_X b \Rightarrow b \wr_X a) \\
\hline
\end{array}
$$

Given a set $S \in \mathbb{P}X$, then the transitive closure of the adjacency relation for elements in S is defined as follows.

$$
\begin{array}{|l}
[X] \\
\hline
\_ \overset{+}{\wr_\_} \_ : \mathbb{P}X \nrightarrow (X \leftrightarrow X) \\
\hline
\forall S : \mathbb{P}X \bullet \\
\qquad (\_ \overset{+}{\wr_S} \_) = (S \lhd (\_ \wr_X \_) \rhd S)^+ \\
\hline
\end{array}
$$

Interval Adjacency. Two intervals on the set of natural numbers are adjacent if their union defines a single interval. For a given maximum value $\mathsf{max} \in \mathbb{N}$, we define:

$$\forall I, J : \mathcal{IV}[0, \mathsf{max}] \bullet I \wr_{\mathcal{IV}[0,\mathsf{max}]} J \Leftrightarrow I \cup J \in \mathcal{IV}[0, \mathsf{max}]$$

Example 5. Interval $[1 \mathbin{..} 2]$ is adjacent to interval $[3 \mathbin{..} 3]$ since $[1 \mathbin{..} 2] \cup [3 \mathbin{..} 3] = [1 \mathbin{..} 3]$, thus $[1 \mathbin{..} 2] \wr_{IPv4} [3 \mathbin{..} 3]$. $\triangle$

Number Adjacency. Two numbers are adjacent if they are the same or if they are different by a value of one. We define:

$$\forall a, b : \mathbb{N} \bullet a \wr_{\mathbb{N}} b \Leftrightarrow (a = b \vee a + 1 = b \vee b + 1 = a)$$

Set Adjacency. For a generic type X, and sets $S, T \in \mathbb{P}X$, then S and T are adjacent, as $S \cup T \in \mathbb{P}\, X$. We define:

$$\forall S, T : \mathbb{P}X \bullet S \wr_{\mathbb{P}X} T$$

Disjointness. The relation $(\_ \mid_\_ \_)$ is used to define the notion of disjointness over any ordered set. Given $a, b \in X$, $a \mid_X b$ denotes a and b are disjoint in X. The property of irreflexivity is required, as a cannot be disjoint from itself, that is; if for any a, that $\neg\ (a \mid_X a)$ did not hold, then an inconsistency would exist. Symmetry is also required for consistency, as if a and b are disjoint in X, then b and a must be disjoint in X also. The following schema defines a generic disjointness relation that can be instantiated for disjointness over different datatypes.

$$
\begin{array}{|l}
[X] \\
\hline
\_ \mid_\_ \_ : \mathbb{P}\, X \nrightarrow (X \leftrightarrow X) \\
\hline
\forall a, b : X \bullet \\
\qquad \neg\ (a \mid_X a) \wedge \\
\qquad (a \mid_X b \Rightarrow b \mid_X a) \\
\hline
\end{array}
$$

Interval Disjointness. Two intervals are disjoint if they don't intersect. For a given maximum value $\mathsf{max} \in \mathbb{N}$, we define:

$$\forall I, J : \mathcal{IV}[0, \mathsf{max}] \bullet I \mid_{\mathcal{IV}[0,\mathsf{max}]} J \Leftrightarrow I \cap J = \emptyset$$

Example 6. The interval $[1 \mathinner{..} 2]$ and interval $[3 \mathinner{..} 3]$ are disjoint, since $[1 \mathinner{..} 2] \cap [3 \mathinner{..} 3] = \emptyset$, thus $[1 \mathinner{..} 2] \mid_{IPv4} [3 \mathinner{..} 3]$. $\triangle$

Number Disjointness. Two numbers are disjoint if they are different. We define:

$$\forall a, b : \mathbb{N} \bullet a \mid_{\mathbb{N}} b \Leftrightarrow a \neq b$$

Set Disjointness. Two sets are disjoint if they don't intersect. We define:

$$\forall S, T : \mathbb{P} X \bullet S \mid_{\mathbb{P}\, X} T \Leftrightarrow S \cap T = \emptyset$$

Subsumption. The relation $(\_ \overset{-}{\leftarrow} \_)$ is used to define subsumption over any ordered set. For $a, b \in X$, if $a \overset{X}{\leftarrow} b$, then b covers a in X. Reflexivity is required, as any a must cover itself. Transitivity follows from a similar requirement, where for $a, b, c \in X$, if a covers b and b covers c, then a must cover c. Antisymmetry follows from the assumption of irredundant elements, where if a covers b and b covers a then one of them is unnecessary [13]. The properties of reflexivity, transitivity and antisymmetry define $\overset{X}{\leftarrow}$ as a non-strict partial order over X [5]. The following schema defines a generic subsumption relation that can be instantiated for subsumption over different datatypes.

$$\begin{array}{l}
[X] \\
\hline
\_ \overset{-}{\leftarrow} \_ : \mathbb{P}\, X \mathrel{+\!\!\!\rightarrow} (X \leftrightarrow X) \\
\hline
\forall a, b, c : X \bullet \\
\quad a \overset{X}{\leftarrow} a \wedge \\
\quad (a \overset{X}{\leftarrow} b \wedge b \overset{X}{\leftarrow} c \Rightarrow a \overset{X}{\leftarrow} c) \wedge \\
\quad (a \overset{X}{\leftarrow} b \wedge b \overset{X}{\leftarrow} a \Rightarrow a = b) \\
\hline
\end{array}$$

Some subsumption orderings (for example, subset) may form a lattice with greatest lower bound (glb) $\_ \cap_X \_$ and least upper bound (lub) $\_ \cup_X \_$ operators for values in X.

Interval Subsumption. An interval I subsumes (covers) an interval J, if $J \subseteq I$. For a given maximum value $\mathsf{max} \in \mathbb{N}$, we define:

$$\forall I, J : \mathcal{IV}[0, \mathsf{max}] \bullet J \overset{\mathcal{IV}[0,\mathsf{max}]}{\leftarrow} I \Leftrightarrow J \subseteq I$$

Example 7. Interval $[1 \mathinner{..} 3]$ covers interval $[3 \mathinner{..} 3]$, since $[3 \mathinner{..} 3] \subseteq [1 \mathinner{..} 3]$, thus: $[3 \mathinner{..} 3] \overset{IPv4}{\leftarrow} [1 \mathinner{..} 3]$. $\triangle$

Number Subsumption. For $a, b \in \mathbb{N}$ then b covers a if a equals b. We define:

$$\forall a, b : \mathbb{N} \bullet a \overset{\mathbb{N}}{\leftarrow} b \Leftrightarrow a = b$$

The equality relation is both symmetric and antisymmetric, and defines both an equivalence relation and a non-strict partial order. Thus, $a \overset{\mathbb{N}}{\leftarrow} b$ denotes that b subsumes/covers a.

Set Subsumption. The definition for set subsumption is as expected, we define:

$$\forall S, T : \mathbb{P}X \bullet S \overset{\mathbb{P}X}{\leftarrow} T \Leftrightarrow S \subseteq T$$

For a generic type X, and $S \in \mathbb{P}X$, then the flattening function $\lceil S \rceil$ gives the cover-set for the elements of S, whereby the cover-set of S has no subsumption. We define:

$$\begin{array}{|l}
[X] \\
\lceil \_ \rceil : \mathbb{P}X \nrightarrow \mathbb{P}X \\
\hline
\forall S : \mathbb{P}X \bullet \\
\quad \lceil S \rceil = S \setminus \{a, a' : S \mid a \overset{X}{\leftarrow} a' \land a \neq a' \bullet a\}
\end{array}$$

Example 8. Given the set of all intervals on the natural numbers from $1 \mathrel{..} 3$, then we have:

$$\begin{aligned}
\lceil \mathcal{IV}[1,3] \rceil &= \{[1 \mathrel{..} 1], [1 \mathrel{..} 2], [1 \mathrel{..} 3], [2 \mathrel{..} 2], [2 \mathrel{..} 3], [3 \mathrel{..} 3]\} \\
&\qquad \setminus \{[1 \mathrel{..} 1], [1 \mathrel{..} 2], [2 \mathrel{..} 2], [2 \mathrel{..} 3], [3 \mathrel{..} 3]\} \\
&= \{[1 \mathrel{..} 3]\}
\end{aligned}$$

$\triangle$

We define a *difference* operator for $S, T \in \mathbb{P}X$, where $S \setminus_{\mathbb{P}X} T$ gives the relative compliment of T in S. That is, everything that is of type X that is covered in S, but not in T. We define this as:

$$\begin{array}{|l}
[X] \\
\_ \setminus_{\_} \_ : \mathbb{P}(\mathbb{P}X) \nrightarrow \mathbb{P}X \times \mathbb{P}X \to \mathbb{P}X \\
\hline
\forall S, T : \mathbb{P}X \bullet \\
\quad S \setminus_{\mathbb{P}X} T = \lceil \{a : S;\ c : X \mid c \overset{X}{\leftarrow} a \land (\forall b : T \bullet \neg\ (c \overset{X}{\leftarrow} b)) \bullet c\} \rceil
\end{array}$$

Example 9. Given the cover-set for the set of all intervals on the natural numbers from $1 \mathrel{..} 3$, and the set $\{[1 \mathrel{..} 1], [3 \mathrel{..} 3]\}$, we have $\lceil \mathcal{IV}[1,3] \rceil \setminus_{IPv4} \{[1 \mathrel{..} 1], [3 \mathrel{..} 3]\} = \{[2 \mathrel{..} 2]\}$. $\triangle$

3.2. The Adjacency Datatype

For a generic type X, the Adjacency datatype $\alpha[X]$, is the set of all closed subsets of X partitioned by adjacency.

$$\alpha[X] == \{S : \mathbb{P}X \mid (\forall a, b : S \mid a \neq b \bullet \neg\ (a \wr_X b))\}$$

Example 10. We can use this to define all ways that an interval can be partitioned into sets of non-adjacent intervals.

$$\alpha[\mathcal{IV}[1,3]] = \{\{[1\,..\,1]\},\{[1\,..\,2]\},\{[1\,..\,3]\},\{[2\,..\,2]\},\{[2\,..\,3]\},\{[3\,..\,3]\},\{[1\,..\,1],[3\,..\,3]\}\}$$

$\triangle$

Let IP_{Spec} define the set of all closed subsets for the intervals of the IPv4 address range, partitioned by adjacency, and similarly, let Prt_{Spec} define the set of all closed subsets for the intervals of the network port range, partitioned by adjacency. We define:

$$IP_{Spec} == \alpha[IPv4]$$

$$Prt_{Spec} == \alpha[Port]$$

Adjacency Datatype Ordering. An ordering can be defined over Adjacency-sets of a generic type X as follows:

$[X]$

$\bot, \top : \alpha[X]$
$\mathbf{not} : \alpha[X] \to \alpha[X]$
$\_ \leqslant \_ : \alpha[X] \leftrightarrow \alpha[X]$
$\_ \otimes \_,$
$\_ \oplus \_ : \alpha[X] \times \alpha[X] \to \alpha[X]$

$\bot = \emptyset \wedge \top = \lceil X \rceil$
$\forall S, T : \alpha[X] \bullet$
$\quad \mathbf{not}\, S = \top \setminus_{\alpha[X]} S \wedge$
$\quad S \leqslant T \Leftrightarrow (\forall a : S \bullet \exists b : T \bullet a \overset{X}{\leftarrow} b) \wedge$
$\quad S \otimes T = \lceil \bigcup \{U : \alpha[X] \mid \forall c : U \bullet \exists a : S;\ b : T \bullet c \overset{X}{\leftarrow} a \wedge c \overset{X}{\leftarrow} b\} \rceil \wedge$
$\quad S \oplus T = \bigcap \{U : \alpha[X] \mid \forall c : U \bullet \exists a : S;\ b : T \bullet a \overset{X}{\leftarrow} c \vee b \overset{X}{\leftarrow} c\}$

Lemma 3.1. *The ordering relation $\leqslant$ is a non-strict partial order over $\alpha[X]$.*

Proof For $S, T \in \alpha[X]$, then $S \leqslant T$ means that T covers S, that is, every $a \in S$ is covered by some $b \in T$. The ordering relation $\leqslant$, is defined as a subsumption ordering/an antisymmetric preorder, where the properties of reflexivity, transitivity and antisymmetry hold for $\leqslant$ over $\alpha[X]$ as $(\_ \overset{X}{\leftarrow} \_)$ is a non-strict partial order for elements of type X. We have:

$$\begin{aligned}
&\forall S, T, U : \alpha[X] \bullet\\
&\quad S \leqslant S \wedge\\
&\quad (S \leqslant T \wedge T \leqslant U \Rightarrow S \leqslant U) \wedge\\
&\quad (S \leqslant T \wedge T \leqslant S \Rightarrow S = T)
\end{aligned}$$

$$\begin{tikzcd}
& \begin{array}{c}\top\\ \lceil IP_{Spec} \rceil\end{array} & \\
& \begin{array}{c}\mathsf{ranges}_1 \oplus \mathsf{ranges}_2\\ \{[1\mathinner{..}4],[6\mathinner{..}8],[10\mathinner{..}10]\}\end{array} \arrow[u, dashed] & \\
\begin{array}{c}\mathsf{ranges}_1\\ \{[1\mathinner{..}3],[6\mathinner{..}6],[8\mathinner{..}8]\}\end{array} \arrow[ur, dashed] & & \begin{array}{c}\mathsf{ranges}_2\\ \{[2\mathinner{..}4],[7\mathinner{..}7],[10\mathinner{..}10]\}\end{array} \arrow[ul, dashed] \\
& \begin{array}{c}\mathsf{ranges}_1 \otimes \mathsf{ranges}_2\\ \{[2\mathinner{..}3]\}\end{array} \arrow[ul, dashed] \arrow[ur, dashed] & \\
& \begin{array}{c}\emptyset\\ \bot\end{array} \arrow[u, dashed] &
\end{tikzcd}$$

Fig. 1. IP_{Spec} ordering fragment

The elements $\bot, \top \in \alpha[X]$ define the least and greatest bounds, respectively, on $\alpha[X]$, where $\bot$ is the unique minimal element that is covered by all elements, and $\top$ is the unique maximal element that covers all other elements. We have:

$$\forall S : \alpha[X] \bullet \bot \leqslant S \leqslant \top$$

$\blacksquare$

Example 11. Given $\mathsf{ranges}_1, \mathsf{ranges}_2 \in IP_{Spec}$, where:

$$\mathsf{ranges}_1 == \{[1\mathinner{..}3],[6\mathinner{..}6],[8\mathinner{..}8]\}$$

$$\mathsf{ranges}_2 == \{[2\mathinner{..}4],[7\mathinner{..}7],[10\mathinner{..}10]\}$$

then Figure 1 depicts a partial Hasse diagram, for the composition of ranges_1 and ranges_2 under the relative ordering of $\leqslant$ over IP_{Spec}. $\triangle$

Adjacency Datatype Union. The *join* of $S, T \in \alpha[X]$ is defined using subsumption, as the generalized intersection of all Adjacency-sets, where each element of $(S \oplus T)$ covers an element in *either* S or T. Intuitively, this means that the values of the join are exactly a union of the elements from both S and T.

Lemma 3.2. *The operator $\oplus$ is a least upper bound operator on $\alpha[X]$.*

Proof The generalized intersection in the join operation for some $S, T \in \alpha[X]$ defines the smallest collection of $x \in X$ that cover all of the elements from both S and T by subsumption. If we take some $U \in \alpha[X]$, such that $U \leqslant (S \oplus T)$ and $S \leqslant U \wedge T \leqslant U$, then $(S \oplus T) = U$.

Thus, Adjacency join provides a lowest upper bound operator. Since $\oplus$ provides a lub operator we have $S \leqslant (S \oplus T)$ and $T \leqslant (S \oplus T)$. ∎

Adjacency Datatype Intersection. Under this ordering, the *meet*, or intersection $S \otimes T$ of $S, T \in \alpha[X]$ is defined using subsumption, as the cover-set for the generalized union of all Adjacency-sets, where each element of $(S \otimes T)$ is covered by an element in *both* S and T. Intuitively, this means that the values of the meet are all non-empty intersections of each value in S with each value in T.

Corollary 3.3. *The operator $\otimes$ is a greatest lower bound operator on $\alpha[X]$.*

Proof It follows from Lemma 3.2 by an analogous argument that Corollary 3.3 holds. ∎

Since $\otimes$ provides the glb operator, then for $S, T \in \alpha[X]$ we have $S \otimes T$ is covered by both S and T, that is $(S \otimes T) \leqslant S$ and $(S \otimes T) \leqslant T$.

Adjacency Datatype Negation. Given $S \in \alpha[X]$, then $\mathbf{not}\, S$ defines a complement operator in $\alpha[X]$, where $\mathbf{not}\, S$ is the cover-set for all elements of type X that are not covered by some member of S. We have:

$$\begin{array}{l} \forall\, S : \alpha[X] \bullet \\ \quad (S \oplus \mathbf{not}\, S) = \top \wedge (S \otimes \mathbf{not}\, S) = \bot \end{array}$$

Theorem 3.4. *The poset $(\alpha[X], \leqslant)$ forms a lattice with lowest-upper, and greatest lower bound operators, $\oplus$ and $\otimes$ respectively, and complement operator* **not***.*

Proof This follows from the definition of $\leqslant$ as a subsumption ordering/an antisymmetric preorder, the properties of **not**, the definition of the meet as the cover-set for the generalized union of all Adjacency-sets, where for $S, T \in \alpha[X]$, each element of $(S \otimes T)$ is covered by an element in *both* S and T, and the definition of the join as the generalized intersection of all Adjacency-sets, where each element of $(S \oplus T)$ covers an element in *either* S or T. ∎

3.3. The Duplet Datatype

The notion of adjacency becomes more complex when we consider comparing/composing firewall rules comprising two or more filter condition attributes. When joining adjacent firewall rules, in some cases the rules may coalesce and in other cases they may partition into a number of disjoint rules.

Example 12. Recall from Example 3 in this section, the firewall system that supports only destination port range filter conditions. Suppose we want to extend the expressiveness of the policy rules for this system to include a definition for destination IP range. Consider, two policy requirements; whereby network traffic is to be allowed to the IP range $[1 \mathinner{..} 3]$ on ports $[1 \mathinner{..} 3]$, and to the IP range $[2 \mathinner{..} 4]$ on ports $[2 \mathinner{..} 4]$. Then modelling this using adjacency-free IP/port range pairs, we have $\mathsf{p}_1, \mathsf{p}_2 \in (IP_{Spec} \times Prt_{Spec})$, whereby:

$$\begin{array}{l} \mathsf{p}_1 == (\{[1 \mathinner{..} 3]\}, \{[1 \mathinner{..} 3]\}) \\ \mathsf{p}_2 == (\{[2 \mathinner{..} 4]\}, \{[2 \mathinner{..} 4]\}) \end{array}$$

$R \backslash A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\{[1\,..\,3]\} \cup_{IP_{Spec}} \{[2\,..\,4]\}$	$\{[1\,..\,3]\} \cap_{Prt_{Spec}} \{[2\,..\,4]\}$
$\boxed{2}$	$\{[1\,..\,3]\}$	$\{[1\,..\,3]\} \setminus_{Prt_{Spec}} \{[2\,..\,4]\}$
$\boxed{3}$	$\{[2\,..\,4]\}$	$\{[2\,..\,4]\} \setminus_{Prt_{Spec}} \{[1\,..\,3]\}$

Table 1
A two-attribute rule join, 1st attribute major ordering

$R \backslash A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\{[1\,..\,3]\} \cap_{IP_{Spec}} \{[2\,..\,4]\}$	$\{[1\,..\,3]\} \cup_{Prt_{Spec}} \{[2\,..\,4]\}$
$\boxed{2}$	$\{[1\,..\,3]\} \setminus_{IP_{Spec}} \{[2\,..\,4]\}$	$\{[1\,..\,3]\}$
$\boxed{3}$	$\{[2\,..\,4]\} \setminus_{IP_{Spec}} \{[1\,..\,3]\}$	$\{[2\,..\,4]\}$

Table 2
A two-attribute rule join, 2nd attribute major ordering

If we consider the attributes separately, we observe that the IP range in p_1 is adjacent to the IP range in p_2, and the port ranges in p_1 and p_2 are also adjacent. However, in composing p_1 and p_2 under a lowest-upper-bound style operation one cannot simply take a union of the sets of intervals to be the IP/port range pair: $(\{[1\,..\,4]\}, \{[1\,..\,4]\})$, as this results in an overly permissive policy, given that network traffic is permitted to IP 1 on port 4, and to IP 4 on port 1 as a result of composition. Conversely, this would result in an overly restrictive policy if we were composing *deny* rules.

When we consider how the join of p_1 and p_2 may be defined, whereby the desired result is the smallest number of non-adjacent rules that cover both p_1 and p_2, then we can apply an adjacency-precedence to the IP ranges in p_1 and p_2, and observe that the port ranges in p_1 and p_2 are not disjoint. We refer to this as the 1st *attribute major ordering*, and the cover for p_1 and p_2 is given as:

$$\{(\{[1\,..\,4]\}, \{[2\,..\,3]\}), (\{[1\,..\,3]\}, \{[1\,..\,1]\}), (\{[2\,..\,4]\}, \{[4\,..\,4]\})\}$$

In this case, the result is a set of disjoint rules that exactly cover the IP/port-range pair constraints from p_1 and p_2. We note, however, that instead, the adjacency-precedence may be applied to the second attribute, where in this case we observe that the port ranges in p_1 and p_2 are adjacent, and the IP ranges in p_1 and p_2 are not disjoint. We refer to this as a 2nd *attribute major ordering*, and would therefore expect the set of disjoint rules that exactly cover the IP/port-range pair constraints from p_1 and p_2 to be:

$$\{(\{[2\,..\,3]\}, \{[1\,..\,4]\}), (\{[1\,..\,1]\}, \{[1\,..\,3]\}), (\{[4\,..\,4]\}, \{[2\,..\,4]\})\}$$

The resulting operations for the 1st attribute major ordering are illustrated in Table 1, whereby the label R signifies the new rule, and the label A means filter condition attribute. In Table 2, the operations for the 2nd attribute major ordering the are encoded.

For the remainder of this paper, we consider firewall rule join in terms of the 1st attribute major ordering. However, we also consider the join of rules where there is an adjacency in other than the first attribute, we refer to this type of adjacency as *forward* adjacency. $\triangle$

Duplets. A duplet is an ordered pair, where the set of all duplets for generic types X, Y, is defined as $\delta[X, Y]$, whereby:

$$\delta[X, Y] == X \times Y$$

Example 13. For $\mathcal{IV}[1, 1]$ and $\mathcal{IV}[1, 2]$, we have:

$$\delta[\mathcal{IV}[1, 1], \mathcal{IV}[1, 2]] = \{([1\,..\,1], [1\,..\,1]), ([1\,..\,1], [1\,..\,2]), ([1\,..\,1], [2\,..\,2])\}$$

and $\delta[IP_{Spec}, Prt_{Spec}]$ gives the set of all duplets for adjacency-free IP/port-range pairs. $\triangle$

Lemma 3.5. *If the ordering over X is a lattice and the ordering over Y is a lattice, then the ordering over $\delta[X, Y]$ is also a lattice.*

Proof Given the definition of $\delta[X, Y]$ as the Cartesian product of X and Y, then if the ordering over X is a lattice and the ordering over Y is a lattice; it follows that $\delta[X, Y]$ forms a lattice under the product order of X and Y. ▮

Forward Adjacency. A pair of duplets are forward adjacent to each other if the attributes in the first coordinate are equal and the attributes in the second coordinate are adjacent. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define forward adjacency, whereby:

$$(a_1 = a_2 \wedge b_1 \wr_Y b_2)$$

Example 14. Given duplets $(\{[1 \,..\, 3]\}, \{[2 \,..\, 3]\}), (\{[1 \,..\, 3]\}, \{[1 \,..\, 1]\}) \in \delta[IP_{Spec}, Prt_{Spec}]$, then these duplets are forward adjacent, as: $(\{[1 \,..\, 3]\} = \{[1 \,..\, 3]\}) \wedge (\{[2 \,..\, 3]\} \wr_{Prt_{Spec}} \{[1 \,..\, 1]\})$. $\triangle$

Duplet Adjacency. A pair of duplets $a, b \in \delta[X, Y]$ are adjacent, if the attributes in the first coordinate are adjacent, and the attributes in the second coordinate are not disjoint, or a and b are forward adjacent. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define:

$$(a_1, b_1) \wr_{\delta[X,Y]} (a_2, b_2) \Leftrightarrow ((a_1 \wr_X a_2 \wedge \neg\ (b_1 \mid_Y b_2)) \vee (a_1 = a_2 \wedge b_1 \wr_Y b_2))$$

Example 15. We have that $\mathsf{p}_1 \wr_{\delta[IP_{Spec}, Prt_{Spec}]} \mathsf{p}_2$, since the IP ranges are adjacent and the port ranges are not disjoint and we have $\{[1 \,..\, 3]\} \wr_{IP_{Spec}} \{[2 \,..\, 4]\} \wedge \neg\ (\{[1 \,..\, 3]\} \mid_{Prt_{Spec}} \{[2 \,..\, 4]\})$. $\triangle$

Duplet Disjointness. A pair of duplets are disjoint if the attributes in the first coordinate are disjoint, and/or the attributes in the second coordinate are disjoint. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define:

$$(a_1, b_1) \mid_{\delta[X,Y]} (a_2, b_2) \Leftrightarrow (a_1 \mid_X a_2 \vee b_1 \mid_Y b_2)$$

Example 16. We have $\neg\ (\mathsf{p}_1 \mid_{\delta[IP_{Spec}, Prt_{Spec}]} \mathsf{p}_2)$, since

$$\neg\ (\{[1 \,..\, 3]\} \mid_{IP_{Spec}} \{[2 \,..\, 4]\} \wedge \{[1 \,..\, 3]\} \mid_{Prt_{Spec}} \{[2 \,..\, 4]\})$$

$\triangle$

Duplet Intersection. The definition for duplet intersection is defined as the intersection of the attributes in each coordinate under their respective orderings. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define:

$$(a_1, b_1) \cap_{\delta[X,Y]} (a_2, b_2) = ((a_1 \cap_X a_2), (b_1 \cap_Y b_2))$$

Example 17. For p_1 and p_2, we have $\mathsf{p}_1 \cap_{\delta[IP_{Spec}, Prt_{Spec}]} \mathsf{p}_2 = (\{[2 \,..\, 3]\}, \{[2 \,..\, 3]\})$ $\triangle$

Duplet Merge. The definition for duplet merge is defined as the union of the attributes in each coordinate under their respective orderings. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define:

$$(a_1, b_1) \cup_{\delta[X,Y]} (a_2, b_2) = ((a_1 \cup_X a_2), (b_1 \cup_Y b_2))$$

Example 18. Given p_1 and p_2, we have $\mathsf{p}_1 \cup_{\delta[IP_{Spec}, Prt_{Spec}]} \mathsf{p}_2 = (\{[1\,..\,4]\}, \{[1\,..\,4]\})$. △

Duplet Subsumption. A duplet (a_1, b_1) covers a duplet (a_2, b_2) in $\delta[X, Y]$, if a_1 covers a_2 in X, and b_1 covers b_2 in Y. Thus, we define duplet subsumption as:

$$(a_2, b_2) \overset{\delta[X,Y]}{\leftarrow} (a_1, b_1) \Leftrightarrow (a_2 \overset{X}{\leftarrow} a_1) \wedge (b_2 \overset{Y}{\leftarrow} b_1)$$

Example 19. For p_1 and p_2, we have $(\{[2\,..\,3]\}, \{[2\,..\,3]\}) \overset{\delta[IP_{Spec}, Prt_{Spec}]}{\leftarrow} \mathsf{p}_1$ and $(\{[2\,..\,3]\}, \{[2\,..\,3]\}) \overset{\delta[IP_{Spec}, Prt_{Spec}]}{\leftarrow} \mathsf{p}_2$ △

Precedence Subsumption. A precedence subsumption is defined for duplets, whereby we explicitly define subsumption orderings separately in each coordinate. The relation $(\_ \overset{-}{\rightarrow} \_)$ defines a general format for precedence subsumption over any ordered set. For $a, b \in X$, if $a \overset{X}{\rightarrow} b$, then a covers b by precedence in X. The properties of reflexivity, transitivity and antisymmetry define $\overset{X}{\rightarrow}$ as a non-strict partial order over X [5]. The following schema defines a generic precedence subsumption relation that can be instantiated for precedence subsumption over different datatypes.

$$\begin{array}{|l}
[X] \\
\hline
\_ \overset{-}{\rightarrow} \_ : \mathbb{P}\, X \nrightarrow (X \leftrightarrow X) \\
\hline
\forall\, a, b, c : X \bullet \\
\quad a \overset{X}{\rightarrow} a\ \wedge \\
\quad (a \overset{X}{\rightarrow} b \wedge b \overset{X}{\rightarrow} c \Rightarrow a \overset{X}{\rightarrow} c)\ \wedge \\
\quad (a \overset{X}{\rightarrow} b \wedge b \overset{X}{\rightarrow} a \Rightarrow a = b) \\
\hline
\end{array}$$

A duplet (a_1, b_1) covers a duplet (a_2, b_2) by precedence in $\delta[X, Y]$, if a_1 covers a_2 in X, and b_2 covers b_1 in Y. Thus, we define precedence subsumption as:

$$(a_1, b_1) \overset{\delta[X,Y]}{\rightarrow} (a_2, b_2) \Leftrightarrow (a_2 \overset{X}{\leftarrow} a_1 \wedge (b_1 \overset{Y}{\leftarrow} b_2)$$

Example 20. For $\mathsf{p}_1, \mathsf{p}_2$, and duplets $(\{[1\,..\,4]\}, \{[2\,..\,3]\}), (\{[1\,..\,3]\}, \{[1\,..\,1]\}) \in \delta[IP_{Spec}, Prt_{Spec}]$, then we have:

$$\begin{array}{l}
(\{[1\,..\,3]\}, \{[1\,..\,1]\}) \overset{\delta[IP_{Spec}, Prt_{Spec}]}{\rightarrow} \mathsf{p}_1\ \wedge \\
(\{[1\,..\,4]\}, \{[2\,..\,3]\}) \overset{\delta[IP_{Spec}, Prt_{Spec}]}{\rightarrow} \mathsf{p}_1\ \wedge \\
(\{[1\,..\,4]\}, \{[2\,..\,3]\}) \overset{\delta[IP_{Spec}, Prt_{Spec}]}{\rightarrow} \mathsf{p}_2
\end{array}$$

△

Precedence Cover. For a duplet $a \in \delta[X,Y]$ and a set of duplets $S \in \mathbb{P}\ \delta[X,Y]$, then S covers a if the duplet merge of all elements in S that each cover a by precedence subsumption, cover a by duplet subsumption. We define:

$$
\begin{array}{|l}
[X,Y] \\
\hline
\_ \bar{\leftsquigarrow} \_ : \mathbb{P}\ \delta[X,Y] \nrightarrow (\delta[X,Y] \leftrightarrow \mathbb{P}\ \delta[X,Y]) \\
\hline
\forall\, a : \delta[X,Y];\ S : \mathbb{P}\ \delta[X,Y] \bullet \\
\qquad a \overset{\delta[X,Y]}{\leftsquigarrow} S \Leftrightarrow (\exists\, b, b' : S \mid (b \overset{\delta[X,Y]}{\rightarrow} a \wedge b' \overset{\delta[X,Y]}{\rightarrow} a) \wedge \\
\qquad\qquad a \overset{\delta[X,Y]}{\leftarrow} (b \cup_{\delta[X,Y]} b'))
\end{array}
$$

Example 21. For p_1, and duplets $(\{[1\,..\,3]\},\{[2\,..\,3]\}), (\{[1\,..\,3]\},\{[1\,..\,1]\}) \in \delta[IP_{Spec}, Prt_{Spec}]$, we have: $\mathsf{p}_1 \overset{\delta[IP_{Spec},Prt_{Spec}]}{\leftsquigarrow} \{(\{[1\,..\,3]\},\{[2\,..\,3]\}), (\{[1\,..\,3]\},\{[1\,..\,1]\})\}$ as $(\{[1\,..\,3]\},\{[1\,..\,3]\}) \overset{\delta[IP_{Spec},Prt_{Spec}]}{\leftsquigarrow} \{(\{[1\,..\,3]\},\{[1\,..\,3]\})\}$ holds. △

Intersecting Elements. For $a \in \delta[X,Y]$ and $S \in \mathbb{P}\,\delta[X,Y]$, then $a\lfloor S \rfloor$ is the set of all non-empty intersections of a with each value $b \in S$. We define:

$$
\begin{array}{|l}
[X,Y] \\
\hline
\_\lfloor\_\rfloor : \delta[X,Y] \times \mathbb{P}\ \delta[X,Y] \rightarrow \mathbb{P}\ \delta[X,Y] \\
\hline
\forall\, a : \delta[X,Y];\ S : \mathbb{P}\ \delta[X,Y] \bullet \\
\qquad a\lfloor S \rfloor = \{b : S \mid \neg\ (a \mid_{\delta[X,Y]} b) \bullet (a \cap_{\delta[X,Y]} b)\}
\end{array}
$$

Example 22. For p_1, and duplets $(\{[1\,..\,4]\},\{[2\,..\,3]\}), (\{[1\,..\,3]\},\{[1\,..\,1]\}), (\{[2\,..\,4]\},\{[4\,..\,4]\}) \in \delta[IP_{Spec}, Prt_{Spec}]$, then:

$$
\begin{array}{l}
\mathsf{p}_1 \lfloor \{(\{[1\,..\,4]\},\{[2\,..\,3]\}), (\{[1\,..\,3]\},\{[1\,..\,1]\}), (\{[2\,..\,4]\},\{[4\,..\,4]\})\} \rfloor = \\
\qquad \{(\{[1\,..\,3]\},\{[2\,..\,3]\}), (\{[1\,..\,3]\},\{[1\,..\,1]\})\}
\end{array}
$$

△

3.4. Duplet Adjacency Ordering

In this section, an ordering is defined for Adjacency-sets of duplets.

Duplet Adjacency Difference. Given $S, T \in \alpha[\delta[X,Y]]$, then $S \setminus_{\alpha[\delta[X,Y]]} T$ is the cover-set for the set of all duplets covered in S by a duplet, or by a collection of duplets, and not covered in T. Thus, we define the difference of Adjacency-sets of duplets as:

$$
S \setminus_{\alpha[\delta[X,Y]]} T = \lceil \{c : \delta[X,Y] \mid c \overset{\delta[X,Y]}{\leftsquigarrow} c\lfloor S \rfloor \wedge \neg\ (c \overset{\delta[X,Y]}{\leftsquigarrow} c\lfloor T \rfloor)\} \rceil
$$

Example 23. Given $\mathsf{Pol}^{\mathsf{p}}_1, \mathsf{Pol}^{\mathsf{p}}_2 \in \alpha[\delta[IP_{Spec}, Prt_{Spec}]]$, where:

$$\begin{array}{l} \mathsf{Pol}^{\mathsf{p}}_1 == \{(\{[1\mathinner{..}3]\}, \{[1\mathinner{..}3]\})\} \\ \mathsf{Pol}^{\mathsf{p}}_2 == \{(\{[2\mathinner{..}4]\}, \{[2\mathinner{..}4]\})\} \end{array}$$

then:

$$\mathsf{Pol}^{\mathsf{p}}_1 \setminus_{\alpha[\delta[IP_{Spec}, Prt_{Spec}]]} \mathsf{Pol}^{\mathsf{p}}_2 = \{(\{[1\mathinner{..}3]\}, \{[1\mathinner{..}1]\}), (\{[1\mathinner{..}1]\}, \{[2\mathinner{..}3]\})\}$$

The implementation definition for duplet difference is given in Section 7.2. $\triangle$

Duplet Adjacency Ordering. An ordering can be defined over Adjacency-sets of duplets as follows:

$$\begin{array}{|l} [X, Y] \\ \hline \bot, \top : \alpha[\delta[X, Y]] \\ \mathbf{not} : \alpha[\delta[X, Y]] \rightarrow \alpha[\delta[X, Y]] \\ \_ \leqslant \_ : \alpha[\delta[X, Y]] \leftrightarrow \alpha[\delta[X, Y]] \\ \_ \otimes \_, \\ \_ \oplus \_ : \alpha[\delta[X, Y]] \times \alpha[\delta[X, Y]] \rightarrow \alpha[\delta[X, Y]] \\ \hline \bot = \emptyset \wedge \top = \lceil \delta[X, Y] \rceil \\ \forall S, T : \alpha[\delta[X, Y]] \bullet \\ \qquad \mathbf{not}\, S = \top \setminus_{\alpha[\delta[X,Y]]} S \wedge \\ \qquad S \leqslant T \Leftrightarrow (\forall a : S \bullet a \overset{\delta[X,Y]}{\leftsquigarrow} a\lfloor T \rfloor) \wedge \\ \qquad S \oplus T = \lceil \{a, b : \bigcap\{U : \mathbb{P}(\delta[X, Y]) \mid (\forall c : U \bullet \exists a : S;\ b : T \bullet \\ \qquad\qquad c \overset{\delta[X,Y]}{\rightarrow} a \vee c \overset{\delta[X,Y]}{\rightarrow} b)\} \mid a \wr^{+}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b\} \rceil \wedge \\ \qquad S \otimes T = \lceil \bigcup\{U : \alpha[\delta[X, Y]] \mid \forall c : U \bullet c \overset{\delta[X,Y]}{\leftsquigarrow} c\lfloor S \rfloor \wedge c \overset{\delta[X,Y]}{\leftsquigarrow} c\lfloor T \rfloor\} \rceil \\ \hline \end{array}$$

Lemma 3.6. *The ordering relation $\leqslant$ is a non-strict partial order over $\alpha[\delta[X, Y]]$.*

Proof For $S, T \in \alpha[\delta[X, Y]]$, then $S \leqslant T$ means that T covers S, that is, every $a \in S$ is covered under duplet subsumption, either by a duplet, or by a collection of duplets in T. The ordering relation $\leqslant$, is defined as an antisymmetric preorder, where the properties of reflexivity, transitivity and antisymmetry hold for $\leqslant$ over $\alpha[\delta[X, Y]]$.

Reflexivity. For $S \in \alpha[\delta[X, Y]]$, we have for $a \in S$, then a covers itself under duplet subsumption. Since S is adjacency-free, then every duplet in S covers only itself under a duplet subsumption in S. Since duplet subsumption is reflexive, then $\leqslant$ is reflexive; that is:

$$\begin{array}{l} \forall S : \alpha[\delta[X, Y]] \bullet \\ \qquad (S, S) \in (\_ \leqslant \_) \end{array}$$

Transitivity. For $S, T \in \alpha[\delta[X, Y]]$, then $S \leqslant T$ if all $a \in S$ are covered in T under duplet subsumption. Then if we take some $U \in \alpha[\delta[X, Y]]$, such that $T \leqslant U$, then all $b \in T$ are covered in U under duplet subsumption. Therefore, all $a \in S$ are covered in U under duplet subsumption as duplet subsumption is transitive. Then $\leqslant$ is transitive; that is:

$$\begin{aligned} &\forall\, S, T, U : \alpha[\delta[X, Y]] \bullet \\ &\qquad (S, T) \in (\_ \leqslant \_) \wedge (T, U) \in (\_ \leqslant \_) \Rightarrow (S, U) \in (\_ \leqslant \_) \end{aligned}$$

Antisymmetry. For $S, T \in \alpha[\delta[X, Y]]$, then $S \leqslant T$ if all $a \in S$ are covered in T under duplet subsumption, and if $T \leqslant S$ then all $b \in T$ are covered in S under duplet subsumption. Therefore, from the definition of duplet subsumption, if $S \leqslant T$ and $T \leqslant S$ then $T = S$. Then $\leqslant$ is antisymmetric; that is:

$$\begin{aligned} &\forall\, S, T : \alpha[\delta[X, Y]] \bullet \\ &\qquad (S, T) \in (\_ \leqslant \_) \wedge (T, S) \in (\_ \leqslant \_) \Rightarrow S = T \end{aligned}$$

The elements $\bot, \top \in \alpha[\delta[X, Y]]$ define the least and greatest bounds, respectively, on $\alpha[\delta[X, Y]]$, where $\bot$ is the unique minimal element that is covered by all elements, and $\top$ is the unique maximal element that covers all other elements. We have:

$$\begin{aligned} &\forall\, S : \alpha[\delta[X, Y]] \bullet \\ &\qquad \bot \leqslant S \leqslant \top \end{aligned}$$

Then we have:

$$\begin{aligned} &\forall\, S, T, U : \alpha[\delta[X, Y]] \bullet \\ &\qquad S \leqslant S\, \wedge \\ &\qquad (S \leqslant T \wedge T \leqslant U \Rightarrow S \leqslant U) \wedge \\ &\qquad (S \leqslant T \wedge T \leqslant S \Rightarrow S = T) \end{aligned}$$

Thus, $\leqslant$ is a non-strict partial order over $\alpha[\delta[X, Y]]$. ∎

Example 24. Figure 2 depicts a partial Hasse diagram, for the composition of $\mathsf{Pol}_1^{\mathsf{p}}$ and $\mathsf{Pol}_2^{\mathsf{p}}$ from Example 23 under the relative ordering of $\leqslant$ over $\alpha[\delta[IP_{Spec}, Prt_{Spec}]]$. △

Adjacency Duplet Union. The join of $S, T \in \alpha[\delta[X, Y]]$ is defined using subsumption, as the cover-set for the duplet merge of the transitive closure of adjacent duplets, from the generalized intersection of all sets of sets of duplets, whereby each element of the generalized intersection covers an element in *either* S or T by duplet precedence subsumption. The generalized intersection defines the smallest collection of duplets that cover all of the duplets from both S and T by precedence subsumption. Given that all duplets in this set are now disjoint, the cover-set for the duplet merge of the transitive closure of adjacent duplets merges any forward-adjacent duplets from S and T. If we take some $U \in \alpha[\delta[X, Y]]$, such that $U \leqslant (S \oplus T)$ and $S \leqslant U \wedge T \leqslant U$, then $(S \oplus T) = U$. Thus, Adjacency join provides a lowest upper bound operator, we have:

$$\begin{aligned} &\forall\, S, T, U : \alpha[\delta[X, Y]] \bullet \\ &\qquad S \leqslant (S \oplus T) \wedge T \leqslant (S \oplus T) \wedge \\ &\qquad (S \leqslant U \wedge T \leqslant U \Rightarrow (S \oplus T) \leqslant U) \end{aligned}$$

$$\begin{tikzcd}
 & \begin{array}{c}\top \\ \lceil \delta[IP_{Spec}, Prt_{Spec}] \rceil\end{array} & \\
 & \{(\{[1\,..\,4]\}, \{[1\,..\,4]\})\} \arrow[u, dashed] & \\
 & \begin{array}{c}\mathsf{Pol}^{\mathsf{p}}_1 \oplus \mathsf{Pol}^{\mathsf{p}}_2 \\ \{(\{[1\,..\,4]\}, \{[2\,..\,3]\}), (\{[1\,..\,3]\}, \{[1\,..\,1]\}), (\{[2\,..\,4]\}, \{[4\,..\,4]\})\}\end{array} \arrow[u, dashed] & \\
\begin{array}{c}\mathsf{Pol}^{\mathsf{p}}_1 \\ \{(\{[1\,..\,3]\}, \{[1\,..\,3]\})\}\end{array} \arrow[ur, dashed] & & \begin{array}{c}\mathsf{Pol}^{\mathsf{p}}_2 \\ \{(\{[2\,..\,4]\}, \{[2\,..\,4]\})\}\end{array} \arrow[ul, dashed] \\
 & \begin{array}{c}\mathsf{Pol}^{\mathsf{p}}_1 \otimes \mathsf{Pol}^{\mathsf{p}}_2 \\ \{(\{[2\,..\,3]\}, \{[2\,..\,3]\})\}\end{array} \arrow[ul, dashed] \arrow[ur, dashed] & \\
 & \begin{array}{c}\emptyset \\ \bot\end{array} \arrow[u, dashed] &
\end{tikzcd}$$

Fig. 2. Duplet ordering fragment

The implementation definition for duplet adjacency-set join is given in Section 7.2.

Adjacency Duplet Intersection. Under this ordering, the meet $(S \otimes T)$ of $S, T \in \alpha[\delta[X, Y]]$ is defined using subsumption, as the cover-set for the generalized union of all Adjacency-sets, where each element of $(S \otimes T)$ is covered by a duplet, or by a collection of duplets in *both* S and T. The meet is defined as the largest set of adjacency-free duplets that is covered by both S and T. Thus, $\otimes$ provides the glb operator, then for $S, T \in \alpha[\delta[X, Y]]$ we have $S \otimes T$ is covered by both S and T, that is $(S \otimes T) \leqslant S$ and $(S \otimes T) \leqslant T$.

Adjacency Duplet Negation. Given $S \in \alpha[\delta[X, Y]]$, then $\mathbf{not}\, S$ defines a complement operator in $\alpha[\delta[X, Y]]$, where $\mathbf{not}\, S$ is the cover-set for all elements of type X that are not covered by some member of S. We have:

$$\begin{array}{l} \forall\, S : \alpha[\delta[X, Y]] \bullet \\ \qquad (S \oplus \mathbf{not}\, S) = \top \wedge (S \otimes \mathbf{not}\, S) = \bot \end{array}$$

Theorem 3.7. *The poset* $(\alpha[\delta[X, Y]], \leqslant)$ *forms a lattice with complement operator* **not**.

Proof This follows from the definition of $\leqslant$ as an antisymmetric preorder, the properties of **not**, and the definitions of $\oplus$ and $\otimes$.

Commutative Laws. We observe that changing the order of the operands/Adjacency-sets does not change the composition result.

$$\begin{aligned}&\forall\, S, T : \alpha[\delta[X, Y]] \bullet\\ &\qquad S \oplus T = T \oplus S \land\\ &\qquad S \otimes T = T \otimes S\end{aligned}$$

Associative Laws. We observe that the order in which the operations are performed does not change the outcome of the operation.

$$\begin{aligned}&\forall\, S, T, U : \alpha[\delta[X, Y]] \bullet\\ &\qquad S \oplus (T \oplus U) = (S \oplus T) \oplus U \land\\ &\qquad S \otimes (T \otimes U) = (S \otimes T) \otimes U\end{aligned}$$

Absorption Laws. The following identities link $\oplus$ and $\otimes$.

$$\begin{aligned}&\forall\, S, T : \alpha[\delta[X, Y]] \bullet\\ &\qquad S \oplus (S \otimes T) = S \land\\ &\qquad S \otimes (S \oplus T) = S\end{aligned}$$

Idempotent Laws. We observe that for all $S \in \alpha[\delta[X, Y]]$, S is idempotent with respect to $\oplus$ and $\otimes$.

$$\begin{aligned}&\forall\, S : \alpha[\delta[X, Y]] \bullet\\ &\qquad S \oplus S = S \land\\ &\qquad S \otimes S = S\end{aligned}$$

Identity Laws. We observe that $(\alpha[\delta[X, Y]], \oplus, \otimes, \bot, \top)$ is a bounded lattice/algebraic structure, such that $(\alpha[\delta[X, Y]], \oplus, \otimes)$ is a lattice, $\bot$ is the identity element for the join operation $\oplus$, and $\top$ is the identity element for the meet operation $\otimes$.

$$\begin{aligned}&\forall\, S : \alpha[\delta[X, Y]] \bullet\\ &\qquad S \oplus \bot = S \land\\ &\qquad S \otimes \top = S\end{aligned}$$

Distributivity Laws. The join operation distributes over the meet operation and vice-versa.

$$\begin{aligned}&\forall\, S, T, U : \alpha[\delta[X, Y]] \bullet\\ &\qquad S \oplus (T \otimes U) = (S \oplus T) \otimes (T \oplus U) \land\\ &\qquad S \otimes (T \oplus U) = (S \otimes T) \oplus (T \otimes U)\end{aligned}$$

Thus, $(\alpha[\delta[X, Y]], \leqslant, \oplus, \otimes, \bot, \top, \textbf{not})$ is a lattice. ∎

4. $\mathcal{FW}_1$ Filter Conditions

In this section, the filter condition attribute datatypes for the $\mathcal{FW}_1$ policy model are defined.

OSI Layer 2. Let $\mathcal{L}_2$ define the set of all additional filter condition attributes at the Data-Link Layer, given as the set of all duplets over the set of all sets of packet-types ($\mathbb{P}\,PktTpe$) and the set of all sets of MAC addresses ($\mathbb{P}\,MAC$).

$$\mathcal{L}_2 == \delta[\mathbb{P}\,PktTpe, \mathbb{P}\,MAC]$$

From Lemma 3.5, we have that $\mathcal{L}_2$ is a lattice.

Example 25. The following iptables rule specifies that inbound (`INPUT`) TCP broadcast packets (`-p tcp --pkt-type broadcast`) destined for the IP address 0.0.0.1 (`-d 0.0.0.1`) be denied (`-j DROP`).

```
iptables -A INPUT -p tcp --pkt-type broadcast \
    -d 0.0.0.1 -j DROP
```

The following iptables rule specifies that TCP packets being routed beyond the firewall (`FORWARD -p tcp`) destined for the IP address 0.0.0.1 (`-d 0.0.0.1`) with source MAC address 00:0F:EA:91:04:08 (`-m mac --mac-source 00:0F:EA:91:04:08`) be denied (`-j DROP`).

```
iptables -A FORWARD -p tcp -d 0.0.0.1 \
    -m mac --mac-source 00:0F:EA:91:04:08 -j DROP
```

△

OSI Layer 7. Let $\mathcal{L}_7$ define the set of all additional filter condition attributes at the OSI Application Layer, given as the set of all duplets over the set of all sets of Layer 7 protocols ($\mathbb{P}\,Proto_7^L$), the set of all closed subsets for the ranges of all Linux UIDs partitioned by adjacency (UID_{Spec}), and the set of all closed subsets for the ranges of all Linux GIDs (GID_{Spec}) partitioned by adjacency.

$$UID_{Spec} == \alpha[\mathcal{IV}[0, \mathsf{maxUID}]]$$

$$GID_{Spec} == \alpha[\mathcal{IV}[0, \mathsf{maxGID}]]$$

$$\mathcal{L}_7 == \delta[\mathbb{P}\,Proto_7^L, \delta[UID_{Spec}, GID_{Spec}]]$$

From Lemma 3.5, we have that $\mathcal{L}_7$ is a lattice.

Example 26. The following iptables rule specifies that outbound (`OUTPUT`) SSH packets (`-m layer7 --l7proto ssh`), originating from the Linux application UID 1001 (`-m owner --uid-owner 1001`), destined for the IP address 0.0.0.2 (`-d 0.0.0.2`) be allowed (`-j ACCEPT`).

```
iptables -A OUTPUT -m layer7 --l7proto ssh \
    -m owner --uid-owner 1001 -d 0.0.0.2 -j ACCEPT
```

△

The Stateful/Protocol Datatype. Let *Protocol* define the set of all stateful protocols, given as the set of all duplets over the TCP protocol ($\mathbb{P}\,Flag_{Spec}$), the UDP protocol (*UDP*), the ICMP protocol ($\mathbb{P}\,TypesCodes$), and the set of all sets of connection tracking states for a packet/connection ($\mathbb{P}\,State$).

$$Protocol == \delta[\mathbb{P}\,Flag_{Spec}, \delta[UDP, \delta[\mathbb{P}\,TypesCodes, \mathbb{P}\,State]]]$$

UDP operators are $(\wedge, \vee, \overset{UDP}{\leftarrow})$. From Lemma 3.5, we have that *Protocol* is a lattice.

Stateful/Protocol Disjointness. A pair of stateful protocols are disjoint if the TCP, UDP and ICMP attributes are disjoint, and/or their state is disjoint. For $\mathsf{t}_1, \mathsf{t}_2 \in \mathbb{P}\,Flag_{Spec}$, $\mathsf{u}_1, \mathsf{u}_2 \in UDP$, $\mathsf{i}_1, \mathsf{i}_2 \in \mathbb{P}\,TypesCodes$ and $\mathsf{s}_1, \mathsf{s}_2 \in \mathbb{P}\,State$, we define:

$$\begin{aligned}&(\mathsf{t}_1, \mathsf{u}_1, \mathsf{i}_1, \mathsf{s}_1) \mid_{Protocol} (\mathsf{t}_2, \mathsf{u}_2, \mathsf{i}_2, \mathsf{s}_2) \Leftrightarrow \\ &\quad ((\mathsf{t}_1 \mid_{\mathbb{P}\,Flag_{Spec}} \mathsf{t}_2 \wedge \mathsf{u}_1 \mid_{UDP} \mathsf{u}_2 \wedge \mathsf{i}_1 \mid_{\mathbb{P}\,TypesCodes} \mathsf{i}_2) \vee \mathsf{s}_1 \mid_{\mathbb{P}\,State} \mathsf{s}_2)\end{aligned}$$

Additional Filtering Specifications. Let *AdditionalFC* define the set of all additional filter condition attributes of interest, given as the set of all duplets over the set of all closed subsets for the ranges of all Unix timestamps, from 0 up to and including maxTime ($Time_{Spec}$), the set of all sets of all network interfaces on a machine ($\mathbb{P}\,IFACE$), the set of all sets of directions for direction-oriented filtering ($\mathbb{P}\,Dir$) and the set of all sets of iptables chains ($\mathbb{P}\,Chain$).

$$\begin{aligned}&Time_{Spec} == \alpha[\mathcal{IV}[0, \mathsf{maxTime}]] \\ &AdditionalFC == \delta[Time_{Spec}, \delta[\mathbb{P}\,IFACE, \delta[\mathbb{P}\,Dir, \mathbb{P}\,Chain]]]\end{aligned}$$

From Lemma 3.5, we have that *AdditionalFC* is a lattice.

Example 27. The following iptables rule specifies that for all inbound (`INPUT`) packets arriving on interface eth0 (`-i` eth0) from the IP address 0.0.0.1 (`-s` 0.0.0.1), attempting to traverse the firewall between 8 a.m. January 1st 2017 and 6 p.m. January 3rd 2017 (`-m time --datestart` 2017-01-01T08:00:00 `--datestop` 2017-01-03T18:00:00), are to be denied.

```
iptables -A INPUT -i eth0 -s 0.0.0.1 -m time --datestart \
    2017-01-01T08:00:00 --datestop 2017-01-03T18:00:00 -j DROP
```

△

Filter Conditions. A filter condition is a eight-tuple (s, $sprt$, d, $dprt$, p, l_2, l_7, a), representing network traffic originating from source IP ranges s, with source port ranges $sprt$, destined for destination IP ranges d, with destination port ranges $dprt$, using stateful-protocols p, with additional Layer 2 attributes l_2, additional Layer 7 attributes l_7 and additional filtering specifications a. Let *FC* define the set of all filter conditions, whereby:

$$FC == \delta[IP_{Spec}, \delta[Prt_{Spec}, \delta[IP_{Spec}, \delta[Prt_{Spec}, \delta[Protocol, \delta[\mathcal{L}_2, \delta[\mathcal{L}_7, AdditionalFC]]]]]]]$$

From Lemma 3.5, we have that *FC* is a lattice.

Forward Adjacency. Recall, for $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define forward adjacency, whereby: $(a_1 = a_2 \wedge b_1 \wr_Y b_2)$. A pair of filter conditions are forward adjacent if the attributes in the first coordinate are equal, and there is one adjacent attribute in the second coordinate, while all other attributes in the second coordinate are equal.

5. The $\mathcal{FW}_1$ Firewall Algebra

In this section we define an algebra $\mathcal{FW}_1$, for constructing and reasoning about anomaly-free firewall policies. We focus on stateless and stateful firewall policies that are defined in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes. A firewall policy defines the filter conditions that may be allowed or denied by a firewall. Let *Policy* define the set of all firewall policies, whereby:

$$Policy == \{A, D : \alpha[FC] \mid \forall\, a : A;\ d : D \bullet a \mid_{FC} d\}$$

A firewall policy $(A, D) \in Policy$ defines a policy as a pair of adjacency-free sets of filter conditions under the duplet adjacency ordering, whereby a filter condition $f \in A$ should be allowed by the firewall, while a filter condition $f \in D$ should be denied. Given $(A, D) \in Policy$ then A and D are disjoint: this avoids any contradiction in deciding whether a filter condition should be allowed or denied. The policy destructor functions *allow* and *deny* are analogous to functions *first* and *second* for ordered pairs:

$$\begin{array}{|l}
allow, \\
deny : Policy \rightarrow \alpha[FC] \\
\hline
\forall\, A, D : \alpha[FC] \bullet \\
\qquad allow\,(A, D) = A \wedge \\
\qquad deny\,(A, D) = D
\end{array}$$

Thus, we have for all $P \in Policy$ then $P = (allow(P), deny(P))$.

Lemma 5.1. *Policy defines the set of anomaly-free policies.*

Proof Given a policy $(A, D) \in Policy$, as A and D are adjacency-free sets, then A has no redundancy and D has no redundancy, as Adjacency-sets have no subsumption. Therefore, all packets matched in filter conditions allowed by the policy are distinct, as are all packets matched in filter conditions that are denied by the policy. Given a policy $P \in Policy$, as $allow(P)$ and $deny(P)$ are disjoint, then P has no shadowing.

$$\begin{array}{l}
\forall\, P : Policy \bullet \\
\qquad allow(P) \mid_{\alpha[FC]} deny(P)
\end{array}$$

As P has no subsumption and P has no shadowing, then P has no generalised filter conditions and P has no correlated filter conditions. Therefore, as P has no redundancy/shadowing/generalisation/correlation, then *Policy* defines the set of *anomaly-free* policies. ∎

Note that $(A, D) \in Policy$ need not partition $\lceil FC \rceil$: the allow and deny sets define the filter conditions to which the policy *explicitly* applies, and an *implicit* default decision is applied for those filter conditions in $\lceil FC \rceil \setminus_{\alpha[FC]} (A \oplus D)$. For the purposes of modelling iptables firewalls it is sufficient to assume *default deny*, though we observe that $\mathcal{FW}_1$ can also be used to reason about *default allow* firewall policies.

Policy Refinement. An ordering can be defined over firewall policies, whereby given $P, Q \in Policy$ then $P \sqsubseteq Q$ means that P is no less restrictive than Q, that is, any filter condition that is denied by Q is denied by P. Intuitively, policy P is considered to be a *safe replacement* for policy Q, in the sense of [19, 20, 28] and any firewall that enforces policy Q can be reconfigured to enforce policy P without any loss of security. The set *Policy* forms a lattice under the safe replacement ordering and is defined as follows.

$\mathcal{FW}_1$

$$\begin{aligned}
&\bot, \top : Policy \\
&\_ \sqsubseteq \_ : Policy \leftrightarrow Policy \\
&\_ \sqcap \_, \\
&\_ \sqcup \_ : Policy \times Policy \to Policy \\
\hline
&\bot = (\emptyset, \lceil FC \rceil) \wedge \top = (\lceil FC \rceil, \emptyset) \\
&\forall P, Q : Policy \bullet \\
&\quad P \sqsubseteq Q \Leftrightarrow ((allow\,P \leqslant allow\,Q) \wedge \\
&\qquad\qquad (deny\,Q \leqslant deny\,P)) \wedge \\
&\quad P \sqcap Q = (allow\,P \otimes allow\,Q, \\
&\qquad\qquad deny\,P \oplus deny\,Q) \wedge \\
&\quad P \sqcup Q = (allow\,P \oplus allow\,Q, \\
&\qquad\qquad deny\,P \otimes deny\,Q)
\end{aligned}$$

Formally, $P \sqsubseteq Q$ iff every filter condition allowed by P is allowed by Q and that any filter conditions explicitly denied by Q are also explicitly denied by P. Note that in this definition we distinguish between filter conditions *explicitly* denied in the policy versus those *implicitly* denied by default. This means that, everything else being equal, a policy that explicitly denies a filter condition is considered more restrictive than a policy that relies on the implicit default-deny for the same network traffic pattern. Safe replacement is defined as the Cartesian product of Adjacency orderings over allow and deny sets and it therefore follows that $(Policy, \sqsubseteq)$ is a poset. $\bot$ and $\top$ define the most restrictive and least restrictive policies, that is, for any $P \in Policy$ we have $\bot \sqsubseteq P \sqsubseteq \top$. Thus, for example, any firewall enforcing a policy P can be safely reconfigured to enforce the (not very useful) firewall policy $\bot$.

Theorem 5.2. *The set of all policies Policy forms a lattice under safe replacement, with greatest lower bound ($\sqcap$) and lowest upper bound ($\sqcup$) operators in $\mathcal{FW}_1$.*

Proof The ordering of adjacency-free filter condition/duplets is a lattice under subsumption, the Cartesian product is a lattice under the definitions of glb and lub, therefore, $\mathcal{FW}_1$ is a lattice. ▮

Policy Intersection. Under this ordering, the meet $P \sqcap Q$, of two firewall policies P and Q is defined as the policy that denies any filter condition that is explicitly denied by *either* P or Q, but allows filter conditions that are allowed by *both* P and Q. Intuitively, this means that if a firewall is required to enforce both policies P and Q, it can be configured to enforce the policy $(P \sqcap Q)$ since $P \sqcap Q$ is a safe replacement for both P and Q, that is; $(P \sqcap Q) \sqsubseteq P$ and $(P \sqcap Q) \sqsubseteq Q$. Given the definition of safe replacement as a product of two Adjacency lattices, it follows that the policy meet provides the glb operator. Thus, $P \sqcap Q$ provides the 'best'/least restrictive safe replacement (under $\sqsubseteq$) for both P and Q.

Policy Union. The join of two firewall policies P and Q is defined as the policy that allows any filter condition allowed by *either* P or Q, but denies filter conditions that are explicitly denied by *both* P and Q. Intuitively, this means that a firewall that is required to enforce either policy P or Q can be safely configured to enforce the policy $(P \sqcup Q)$. Since $\sqcup$ provides a lub operator we have $P \sqsubseteq (P \sqcup Q)$ and $Q \sqsubseteq (P \sqcup Q)$.

5.1. Constructing Firewall Policies

The lattice of policies $\mathcal{FW}_1$ provides us with an algebra for constructing and interpreting firewall polices. The following constructor functions are used to build primitive policies. Given a set of adjacency-free filter conditions A, then $(\mathsf{Allow}\, A)$ is a policy that allows filter conditions in A, and $(\mathsf{Deny}\, D)$ is a policy that explicitly denies filter conditions in D.

$$\begin{array}{|l}
\mathsf{Allow},\\
\mathsf{Deny} : \alpha[FC] \to Policy\\
\hline
\forall S : \alpha[FC] \bullet\\
\quad \mathsf{Allow}\, S = (S, \emptyset) \wedge\\
\quad \mathsf{Deny}\, S = (\emptyset, S)
\end{array}$$

This provides what we refer to as a *weak* interpretation of allow and deny. Network traffic patterns that are not explicitly mentioned in parameter S are default-deny and therefore are not specified in the deny set of the policy. The following provides us with a *strong* interpretation for these constructors:

$$\begin{array}{|l}
\mathsf{Allow}^+,\\
\mathsf{Deny}^+ : \alpha[FC] \to Policy\\
\hline
\forall S : \alpha[FC] \bullet\\
\quad \mathsf{Allow}^+\, S = (S, \mathbf{not}\, S) \wedge\\
\quad \mathsf{Deny}^+\, S = (\mathbf{not}\, S, S)
\end{array}$$

In this case $(\mathsf{Allow}^+\, A)$ allows filter conditions specified in A, while explicitly denying all other filter conditions, and $(\mathsf{Deny}^+\, D)$ denies filter conditions specified in D while allowing all other filter conditions.

A firewall policy $P \in Policy$ can be decomposed into its corresponding allow and deny sets, and re-constructed using the algebra; for any $(A, D) \in Policy$, since A and D are disjoint then:

Lemma 5.3. *Given $A, D \in \alpha[FC]$, then:*

$$(\mathsf{Allow}^+\, A) \sqcup (\mathsf{Deny}\, D) = (A, \lceil FC \rceil \setminus_{\alpha[FC]} A) \sqcup (\emptyset, D)$$

Proof Follows since $(\mathsf{Allow}^+\, A) \sqcup (\mathsf{Deny}\, D) = (\mathsf{Allow}\, A) \sqcap (\mathsf{Deny}^+\, D) = (A, D)$ ▮

Corollary 5.4. *Given $A, D \in \alpha[FC]$, then it follows that*

$$(\mathsf{Deny}^{+}\, D) \sqcap (\mathsf{Allow}\, A) = (A, \emptyset) \sqcap (\lceil FC \rceil \setminus_{\alpha[FC]} D, D)$$

6. Reasoning About Policies in Practice

Sequential Composition. A firewall policy is conventionally constructed as a sequence of rules, whereby for a given network packet, the decision to allow or deny the packet is checked against each policy rule, starting from the first, in sequence, and the first rule that matches gives the result that is returned. The algebra $\mathcal{FW}_1$ can be extended to include a similar form of sequential composition of policies. The policy constructions in Section 5.1 can be regarded as representing the individual rules of a conventional firewall policy.

Let $(\mathsf{Allow}\, A) \fatsemi Q$ denote a sequential composition of an allow rule $(\mathsf{Allow}\, A)$ with policy Q with the interpretation that a given network packet matched in A is allowed; if it does not match in A then policy Q is enforced. The resulting policy either: allows filter conditions in A (and denies all other filter conditions), or allows/denies filter conditions in accordance with policy Q. We define:

$$\begin{aligned}(\mathsf{Allow}\, A) \fatsemi Q &= (\mathsf{Allow}^{+}\, A) \sqcup Q \\ &= ((A \oplus allow(Q)), ((\lceil FC \rceil \setminus_{\alpha[FC]} A) \otimes deny(Q))) \\ &= ((A \oplus allow(Q)), (deny(Q) \setminus_{\alpha[FC]} A))\end{aligned}$$

which is as expected. A similar definition can be provided for the sequential composition $(\mathsf{Deny}\, D) \fatsemi Q$, whereby a given network packet that is matched in D is denied; if it does not match in D then policy Q is enforced. We define:

$$\begin{aligned}(\mathsf{Deny}\, D) \fatsemi Q &= (\mathsf{Deny}^{+}\, D) \sqcap Q \\ &= (((\lceil FC \rceil \setminus_{\alpha[FC]} D) \otimes allow(Q)), deny(Q) \oplus D) \\ &= (allow(Q) \setminus_{\alpha[FC]} D, deny(Q) \oplus D)\end{aligned}$$

While in practice its usual to write a firewall policy in terms of many constructions of allow and deny rules, in principle, any firewall policy $P \in Policy$ can be defined in terms of one allow policy $(\mathsf{Allow}\, allow(P))$ and one deny policy $(\mathsf{Deny}\, deny(P))$ and since the allow and deny sets of P are disjoint we have $P \fatsemi Q = (\mathsf{Deny}\, deny(P)) \fatsemi (\mathsf{Allow}\, allow(P)) \fatsemi Q$. We define this as:

$$\begin{array}{|l} \_ \fatsemi \_ : Policy \times Policy \to Policy \\ \hline \forall \mathcal{FW}_1;\ P, Q : Policy \bullet \\ \quad P \fatsemi Q = (\mathsf{Deny}^{+}\ (deny(P))) \sqcap \\ \qquad (Q \sqcup (\mathsf{Allow}^{+}\ (allow(P)))) \end{array}$$

Let *Rule* define the set of all firewall rules, whereby:

$$Rule ::= \mathsf{allow}\ \langle\!\langle FC \rangle\!\rangle \mid \mathsf{deny}\ \langle\!\langle FC \rangle\!\rangle$$

We define a rule interpretation function as:

$$
\begin{array}{|l}
\mathcal{I} : Rule \rightarrow Policy \\
\hline
\forall f : FC \bullet \\
\quad \mathcal{I}(\mathsf{allow} f) = \mathsf{Allow}(\{f\}) \wedge \\
\quad \mathcal{I}(\mathsf{deny} f) = \mathsf{Deny}(\{f\})
\end{array}
$$

A firewall policy is defined as a sequence of rules $\langle r_1, r_2, .., r_n \rangle$, for $r_i \in Rule$, and is encoded in the policy algebra as $\mathcal{I}(r_1) \mathbin{⨟} \mathcal{I}(r_2) \mathbin{⨟} .. \mathbin{⨟} \mathcal{I}(r_n)$. In practice, firewall policies are often anomalous [44], where a policy's *deny* rules are not disjoint from it's *allow* rules, and a "first match" principle is applied to filtered packets. Mapping a sequence of potentially anomalous firewall rules $\langle r_1, r_2, .., r_n \rangle$ into the $\mathcal{FW}_1$ algebra gives a semantically-equivalent anomaly-free $P \in Policy$.

Policy Negation. The policy negation of $P \in Policy$ allows filter conditions explicitly denied by P and explicitly denies filter conditions allowed by P. We define:

$$
\begin{array}{|l}
\textbf{not} : Policy \rightarrow Policy \\
\hline
\forall \mathcal{FW}_1;\ P : Policy \bullet \\
\quad \textbf{not}\, P = (\mathsf{Allow}^{+}\ (deny\,(P))) \sqcup \\
\quad\quad (\mathsf{Deny}\ (allow\,(P)))
\end{array}
$$

From this definition it follows that $(\textbf{not}\ P)$ is simply $(deny\,(P), allow\,(P))$ and thus $\textbf{not}\,(\mathsf{Deny}\, D) = (\mathsf{Allow}\, D)$ and $\textbf{not}\,(\mathsf{Allow}\, A) = (\mathsf{Deny}\, A)$. Note however, that in general policy negation does not define a complement operator in the algebra $\mathcal{FW}_1$, that is, it not necessarily the case that $(P \sqcup \textbf{not}\ P) = \top$ and $(P \sqcap \textbf{not}\ P) = \bot$. However, the sub-lattice of policies with allow and deny sets that exactly partition the same set $S \leqslant \lceil FC \rceil$ has policy negation as complement ($allow\,(P) \oplus deny\,(P) = S$ for all P in the sub-lattice).

A Target Action of Log in $\mathcal{FW}_1$. In this paper, the focus is on firewall rules with a target action of either *allow* or *deny*. However, from a compliance perspective, it is considered best practice to log traffic for auditing purposes [41]. Future work should extend the $\mathcal{FW}_1$ algebra to include a target action of *log* for firewall rules. An approach may be taken, whereby we extend $(A, D) \in Policy$ to $(A, D, L) \in Policy$, where $L \in \alpha[FC]$ and a filter condition $f \in L$ should be logged by the firewall. We give the destructor function *log* for firewall policies; whereby $log\,(A, D, L) = L$. For policy composition, then for $P, Q \in Policy$, we have $P \sqcap Q$ signifies the operation $(log\, P \oplus log\, Q)$ for logged filter conditions. Similarly, for $P \sqcup Q$ we have the logged filter conditions $(log\, P \otimes log\, Q)$. From this, we have that the ordering for logged filter conditions is defined similarly to the ordering for denied filter conditions. In practice, a logged filter condition may be shadowed by a filter condition with a target action of *allow* or *deny*. However, it is not the case that a filter condition with a target action of *log* can shadow a filter condition with a target action of *allow* or *deny*. Therefore, for sequential composition, then we have $P \mathbin{⨟} Q$ defines the logged filter conditions for the resulting policy as: $(log\, P \oplus (log\, Q \backslash_{\alpha[fc]} (allow\, P \oplus deny\, P)))$. Encoding a definition for Network Address Translation in $\mathcal{FW}_1$ is also a topic for future research.

6.1. Detecting Anomalies in Firewall Policies

A firewall policy is conventionally constructed as a sequence of order-dependent rules, and when a network packet matches with two or more policy rules, the policy is anomalous [2, 3, 10]. By definition, the adjacency-free allow and deny sets of some $P \in \mathit{Policy}$ are disjoint, therefore P is anomaly-free by construction. We can however define anomalies using the algebra; by considering how a policy changes when composed with other policies.

Redundancy. A policy P is redundant given policy Q if their composition results in no difference between the resulting policy and Q, in particular, if:

$$P \mathbin{⨾} Q = Q$$

Further definitions may be given for redundancy. For example, there are redundant packets with a target action of allow in filter conditions between policy P and policy Q, if:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{allow}\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{allow}\,(Q)) = (\mathit{allow}\,(P) \otimes \mathit{allow}\,(Q), \emptyset)$$

A similar interpretation follows for redundant packets with a target action of deny between filter conditions in a policy P and filter conditions in a policy Q. In particular, we have redundant denies if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{deny}\,(Q)) = (\emptyset, \mathit{deny}\,(P) \otimes \mathit{deny}\,(Q))$$

Shadowing. Some part of policy Q is shadowed by the entire policy P in the composition $P \mathbin{⨾} Q$ if the by filter condition constraints that are specified by P contradict the constraints that are specified by Q, in particular, if:

$$(\mathbf{not}\ P) \mathbin{⨾} Q = Q$$

This is a very general definition for shadowing. Perhaps a more familiar interpretation of this definition is one where the policy P is a specific allow/deny rule that shadows a part or all of the policy with which it is composed. Recall that $(\mathbf{not}(\mathsf{Allow}\,A)) = (\mathsf{Deny}\,A)$ and, for example, in $(\mathsf{Allow}\,A) \mathbin{⨾} Q$ all or part of policy Q is shadowed by the rule/primitive policy $(\mathsf{Allow}\,A)$ if Q denies the filter conditions specified in A, that is, $(\mathsf{Deny}\,A) \mathbin{⨾} Q = Q$. Similarly, in $(\mathsf{Deny}\,D) \mathbin{⨾} Q$ part or all of policy Q is shadowed by the rule/primitive policy $(\mathsf{Deny}\,D)$ if $(\mathbf{not}\,(\mathsf{Deny}\,D)) \mathbin{⨾} Q = Q$.

Further definitions may also be given for shadowing. For example, we have that some of the packets denied by filter conditions in a policy P shadow some of the packets allowed by filter conditions in a policy Q if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q)) = (\emptyset, \mathit{deny}\,(P) \otimes \mathit{allow}\,(Q))$$

Similarly, some of the packets allowed by filter conditions in a policy *P* shadow some of the packets denied by filter conditions in a policy *Q* if:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q)) = (\mathit{allow}\,(P) \otimes \mathit{deny}\,(Q), \emptyset)$$

Generalisation. A generalisation anomaly exists between *P* and *Q* if some of the packets allowed by filter conditions in *P* shadow some of the packets denied by filter conditions in *Q*, in particular, if:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

and, those packets shadowed by filter conditions in *Q* are subsumed by *Q*'s denies:

$$\mathbf{not}\,(\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q))) \neq \mathsf{Deny}(\mathit{deny}\,(Q))$$

whereby:

$$\mathbf{not}\,(\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q))) = (\emptyset, \mathit{allow}\,(P) \otimes \mathit{deny}\,(Q))$$

Similarly, a generalisation anomaly exists between *P* and *Q* if some of the packets denied by filter conditions in *P* shadow some of the packets allowed by filter conditions in *Q*, in particular, if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q)) \neq (\emptyset, \emptyset)$$

and, those packets shadowed by filter conditions in *Q* are subsumed by *Q*'s allows:

$$\mathbf{not}\,(\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q))) \neq \mathsf{Allow}(\mathit{allow}\,(Q))$$

as:

$$\mathbf{not}\,(\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q))) = (\mathit{deny}\,(P) \otimes \mathit{allow}\,(Q), \emptyset)$$

Inter-policy Anomalies. We can also use the $\mathcal{FW}_1$ algebra to reason about anomalies between the different policies of distributed firewall configurations. In the following, assume that *P* is a policy on an *upstream* firewall and *Q* is a policy on a *downstream* firewall.

Redundancy. An inter-redundancy anomaly exists between policies P and Q if some part of Q is redundant to some part of P, whereby the target action of the redundant filter conditions is *deny*. Given some set of filter conditions A denied by P, and some set of filter conditions B denied by Q, then there exists an inter-redundancy between P and Q, if:

$$(\mathsf{Deny}\,A) \mathbin{;} (\mathsf{Deny}\,B) = (\mathsf{Deny}\,A)$$

Further definitions may be given for inter-redundancy. For example, there are redundant packets with a target action of deny in filter conditions between upstream policy P and downstream policy Q, if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

Shadowing. An inter-shadowing anomaly exists between policies P and Q if some part of Q's allows are shadowed by some part of P's denies. Given some set of filter conditions A denied by P, and some set of filter conditions B allowed by Q, then there is an inter-shadowing anomaly between P and Q, if:

$$(\mathsf{Deny}\,A) \mathbin{;} (\mathsf{Allow}\,B) = (\mathsf{Deny}\,A)$$

Further definitions may also be given for inter-shadowing. For example, we have that some of the packets denied by filter conditions in a policy P shadow some of the packets allowed by filter conditions in a policy Q if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q)) \neq (\emptyset, \emptyset)$$

Spuriousness. An inter-spuriousness anomaly exists between policies P and Q if some part of Q's denies are shadowed by some part of P's allows. Given some set of filter conditions A allowed by P, and some set of filter conditions B denied by Q, then there exists an inter-spuriousness anomaly between P and Q, if:

$$(\mathsf{Allow}\,A) \mathbin{;} (\mathsf{Deny}\,B) = (\mathsf{Allow}\,A)$$

Spuriousness may also be defined as follows, whereby some of the packets allowed by filter conditions in a policy P shadow some of the packets denied by filter conditions in a policy Q. We have an inter-spuriousness anomaly from an upstream policy P to a downstream policy Q, if:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

To apply the definitions presented in this section to the detection of anomalies in firewall policies in practice, an approach may be taken, whereby an algorithm is constructed, that incorporates the anomaly definitions into the sequential composition of policies/rules when mapping a potentially anomalous, already deployed firewall policy into $\mathcal{FW}_1$. That is, given a sequence of firewall rules $\langle r_1, r_2, .., r_n \rangle$, then before each sequential composition, an anomaly check can be made using the definitions in this section, the result of which can be used to alert an administrator to the presence of anomalies in the policies/rules under question.

6.2. Standards Compliance

RFC 5735 [9], details fifteen IPv4 address blocks that have been assigned by IANA for specialized/global purposes. Some of these address spaces may appear on the Internet, and may be used legitimately outside a single administrative domain, however, while the assigned values of the address blocks do not directly raise security issues; unexpected use may indicate an attack [9]. For example, packets with a source IP address from the private address space 172.16.0.0/12, arriving on the Wide Area Network interface of a network router, may be considered spoofed, and may be part of a Denial of Service (DoS), or Distributed DoS attack.

RFC 5735 Compliance. Best practice recommendations are implemented for each of the fifteen specialized IP address block ranges in [9], resulting in one hundred and twenty iptables *deny* rules. In [18], we defined this *deny* ruleset for a firewall management tool. We define IP spoof-mitigation policies for each iptables chain separately. For the INPUT chain, a compliance policy $\mathsf{RFC5735}^{\mathsf{I}}$ is defined, whereby for each of the IP address block ranges, the following iptables rules are enforced.

```
iptables -A INPUT -i $in\ -m iprange -src-range $min:$max -j DROP
iptables -A INPUT -i $in\ -m iprange -dst-range $min:$max -j DROP
```

Similarly, for the OUTPUT chain, an IP spoof-mitigation compliance policy $\mathsf{RFC5735}^{\mathsf{O}}$ is defined, whereby for each of the specialized IP address block ranges we have:

```
iptables -A OUTPUT -o $out\ -m iprange -src-range $min:$max -j DROP
iptables -A OUTPUT -o $out\ -m iprange -dst-range $min:$max -j DROP
```

For the FORWARD chain, then $\mathsf{RFC5735}^{\mathsf{F}}$ enforces the following iptables rules for each of the IP address block ranges.

```
iptables -A FORWARD -i $in\ -m iprange -src-range $min:$max -j DROP
iptables -A FORWARD -i $in\ -m iprange -dst-range $min:$max -j DROP
iptables -A FORWARD -o $out\ -m iprange -src-range $min:$max -j DROP
iptables -A FORWARD -o $out\ -m iprange -dst-range $min:$max -j DROP
```

Each policy, $\mathsf{RFC5735}^{\mathsf{I}}$, $\mathsf{RFC5735}^{\mathsf{O}}$, $\mathsf{RFC5735}^{\mathsf{F}}$, terminates with a final iptables rule that specifies all other traffic be permitted on the given iptables chain.

A Redefined Firewall Policy. We model these iptables rules in the algebra by redefining some policy-model attributes, and provide more formal definitions of $\mathsf{RFC5735}^{\mathsf{I}}$, $\mathsf{RFC5735}^{\mathsf{O}}$ and $\mathsf{RFC5735}^{\mathsf{F}}$. Let $AdditionalFC_I$ be the set of all duplets for additional filter condition attributes of interest, where:

$$AdditionalFC_I == \delta[\mathbb{P}\,Chain, \delta[\mathbb{P}\,Dir, \mathbb{P}\,IFACE]]$$

A revised definition for the set of all filter conditions $FC_{\mathcal{I}}$ is given as:

$$FC_{\mathcal{I}} == \delta[IP_{Spec}, \delta[Prt_{Spec}, \delta[IP_{Spec}, \delta[Prt_{Spec}, \delta[Protocol, AdditionalFC_I]]]]]$$

A revised definition for the set of all policies $Policy_{\mathcal{I}}$ is given as:

$$Policy_{\mathcal{I}} == \{A, D : \alpha[FC_{\mathcal{I}}] \mid \forall\, a : A;\ d : D \bullet a \mid_{FC_{\mathcal{I}}} d\}$$

The compliance policies $\mathsf{RFC5735}^{\mathsf{I}}, \mathsf{RFC5735}^{\mathsf{O}}, \mathsf{RFC5735}^{\mathsf{F}} \in Policy_{\mathcal{I}}$, define the minimum requirements for what it means for some perimeter network firewall policy to mitigate the threat of IP spoofing for all traffic, in accordance with RFC 5735. Thus, we have for all $P \in Policy_{\mathcal{I}}$ if:

$$P \sqsubseteq (\mathsf{RFC5735}^{\mathsf{I}} \sqcap \mathsf{RFC5735}^{\mathsf{O}} \sqcap \mathsf{RFC5735}^{\mathsf{F}})$$

then P complies with the best practice recommendations outlined in [9] for IP address spoof-mitigation.

7. Encoding and Evaluating iptables Policies

A prototype policy management toolkit has been implemented in Python for iptables. The prototype allows for parsing the system's currently enforced iptables ruleset $\langle r_1, r_2 .. r_n \rangle$ by chain, using the Python-iptables package [36], and then normalizing each rule to a primitive/singleton policy. The overall policy for the chain being evaluated as $\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2) \fatsemi .. \fatsemi \mathcal{I}(r_n)$. Once the sequential composition of the system's currently enforced iptables policy is computed, the prototype generates a semantically-equivalent adjacency-free set of iptables rules and re-writes this new ruleset to the system.

We reason over $Policy_{\mathcal{I}}$ policies using the $\fatsemi, \sqcup$, and $\sqsubseteq$ policy operators. The test-bed for the experiments was a 64-Bit Ubuntu 14.04 LTS OS, running on a Dell Latitude E6430, with a quad-core Intel i5-3320M processor and 4GB of RAM. Every experiment was conducted three times; the median result chosen for inclusion in this paper. Overall, the results are promising. In this section, the firewall datatypes for the prototype are described.

Firewall Rules. An iptables rule is modelled as a list of generic filter conditions. The current implementation defines firewall rules with filter condition attributes for source/destination IP/port ranges, the ICMP, UDP and TCP protocols, and additional filter condition attributes. The relationships of adjacency, disjointness and subsumption have been encoded, as have composition operators for rule intersection and rule join/combination.

Range-based Attributes. Filter condition attributes defined as ranges in the $\mathcal{FW}_1$ framework, for example, source/destination IP/port ranges, are implemented as interval sets, using the Pyinter Python package [39]. The package was modified to include definitions for relative compliment operators and adjacency over intervals and interval sets.

ICMP and UDP. The ICMP protocol is modelled as the set of all valid ICMP Type/Code pairs, given in Section 4. UDP has been defined as a binary attribute. Boolean operators apply for the UDP filter condition attributes.

TCP. The TCP protocol is encoded as a 2^{12} bit array, whereby the position of each bit is mapped to an index value in a table. This table is the implementation of the $Flag_{Spec}$ object defined in Section 4, and is encoded as the list of TCP (mask, comp) pairs, as pairs of six-bit arrays. A value of `1` at a given position in the 2^{12} bit array indicates that this particular arrangement of TCP flags are matched in the packets specified by the firewall rule. Table 3 gives an overview of the *FlagSpec* lookup.

Additional Attributes. Attributes for direction-oriented filtering, network interface and iptables chains have been encoded as sets for firewall rules.

Index	`Mask`	`Comp`
1	‘000000’	‘000000’
2	‘000000’	‘000001’
..	..	..
8	‘000000’	‘000111’
..	..	..
4096	‘111111’	‘111111’

Table 3
Partial TCP *FlagSpec* Lookup Table

Firewall Policies. A policy is implemented as a disjoint pair of adjacency-free sets of firewall rules. The adjacency-free sets of rules have been modelled following the approach taken to model the interval-sets in the Pyinter package [39].

Transitive Closure of Adjacent Rules. The transitive closure for the adjacency relation over rules in firewall policies has been implemented recursively, following the approach used in the Pyinter package to implement the transitive closure over adjacent intervals [39]. A set of firewall rules is adjacency-free by construction. When a new rule is added to the ruleset, a check is made firstly to determine if there are any rules in the set that are adjacent to the new rule. If there are none, the new rule is added. Otherwise, the adjacent rules are removed from the ruleset, and rules resulting from the combination of the new rule with the adjacent rules are added to the ruleset, starting again with a check to determine if there are any rules in the set that are adjacent to the new rule.

7.1. Evaluating Policy Operators

Evaluating Sequential Policy Composition. The implementation parses the system's currently enforced iptables ruleset $\langle r_1, r_2 \,..\, r_n \rangle$ by chain, and then normalizes each rule to a primitive/singleton policy $\langle \mathcal{I}(r_1), \mathcal{I}(r_2) \,..\, \mathcal{I}(r_n) \rangle$. The overall policy for the chain is evaluated as $\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2) \fatsemi .. \fatsemi \mathcal{I}(r_n)$. Two datasets were generated for experimentation. Each dataset consists of iptables policies of size $2^4 \,..\, 2^{11}$. One dataset contains policies where no rule is adjacent to any other rule (other than itself), and the other dataset consists of policies where every new rule is adjacent to the previous rule; to ensure the maximum number of possible rules are generated as a result of composition. The rules all have a target action of *allow*. Table 4 details the results for the non-adjacent dataset, while the results for the adjacent dataset are illustrated in Table 5.

We observe that as the number of rules increase, the cost of computing the sequential composition of non-adjacent rules is relatively cheap, and we see that for the largest ruleset, 2^{11},the evaluation time is approximately one minute. For the adjacent dataset, the cost of computing the sequential composition of adjacent rules is expensive, but is also proportional to the number of rules used in the experiment. However, the cost is by orders of magnitude more expensive than the cost for evaluating the sequential composition of non-adjacent policies of the same size. For 2^9 rules, the time taken for the evaluation of sequential composition is around three minutes, and the time taken for 2^{11} rules is approximately forty six minutes.

Evaluating Policy Union. Experiments were conducted to test policy lub, whereby each policy in the adjacent dataset used in the sequential composition experiments was split into two policies, whereby the first policy contains the odd (index) rules from the original policy, and the second policy contains

Num rules	**Time**	**Ratio**
2^4	0.07	-
2^5	0.13	1.85
2^6	0.29	2.23
2^7	0.67	2.31
2^8	1.73	2.58
2^9	4.98	2.87
2^{10}	16.09	3.23
2^{11}	57.81	3.59

Table 4

Sequential composition with no adjacent rules (in seconds)

Num rules	**Time**	**Ratio**
2^4	0.80	-
2^5	2.02	2.53
2^6	5.13	2.98
2^7	15.32	3.34
2^8	51.18	3.58
2^9	183.42	3.85
2^{10}	707.15	3.85
2^{11}	2792.81	3.94

Table 5

Sequential composition with adjacent rules (in seconds)

P \ Q	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
2^3	$\boxed{0.65}$	0.79	0.81	0.99	1.40	2.51	5.73	16.93
2^4	0.79	$\boxed{1.86}$	2.09	2.32	2.91	4.50	8.83	22.19
2^5	0.81	2.09	$\boxed{4.97}$	5.45	6.78	9.17	15.50	32.89
2^6	0.99	2.32	5.45	$\boxed{14.70}$	17.01	21.93	32.29	57.47
2^7	1.40	2.91	6.78	17.01	$\boxed{48.85}$	58.44	76.94	119.28
2^8	2.51	4.50	9.17	21.93	58.44	$\boxed{179.87}$	217.34	294.56
2^9	5.73	8.83	15.50	32.29	76.94	217.34	$\boxed{699.11}$	839.49
2^{10}	16.93	22.19	32.89	57.47	119.28	294.56	839.49	$\boxed{2722.63}$

Table 6

Time taken to compute $P \sqcup Q$ (in seconds)

the even (index) rules from the original policy. The odd (index) policies are adjacency-free, as are the even (index) policies. Constructing the dataset from the odd-index and even-index policies allows us to evaluate the cost, in terms of time, of composing policies of different sizes, whereby for the policy union experiments, the maximum number of rules are generated as a result of composition. For each $P, Q \in Policy_{\mathcal{I}}$ in this split dataset, the time taken for the operation $P \sqcup Q$ is given by Table 6.

A benefit of conducting the policy join experiments in this way, is that in practice, we may want to update a policy P, comprising a large number of rules, with a smaller policy Q that permits some new accesses. The time taken for composition of policies of equal size is approximately the same as (slightly less than) the time necessary to sequentially compose the rules from both policies. That is; for example, the time taken for the sequential composition of 2^9 rules is around three minutes, as is the join of the two policies of size 2^8. This is highlighted through the diagonal in the matrix, and is as expected; given that we used all *allow* rules, and the sequential composition of the rules used in these experiments results in the eventual join of the rules.

Evaluating Policy Compliance. A dataset consisting of iptables policies of size $2^5, 2^6 .. 2^{11}$ is generated to test policy compliance. Each policy in this dataset is RFC 5735 compliant by construction, for TCP traffic arriving on the iptables INPUT chain to/from each of the fifteen special-use IPv4 addresses [9]. Recall, the compliance policy $\mathsf{RFC5735}^{\mathsf{I}}$ defined in Section 6.2 for $Policy_{\mathcal{I}}$ policies. Rules from the previously defined non-adjacent dataset have been re-written with a target action of *deny*, and are used

Num rules	Time	Ratio
2^5	0.07	-
2^6	0.09	1.28
2^7	0.15	1.66
2^8	0.32	2.13
2^9	0.50	1.56
2^{10}	0.87	1.74
2^{11}	1.64	1.88

Table 7
Time taken to compute $\mathcal{P} \sqsubseteq \mathsf{RFC5735}^{\mathsf{I}}$ (in seconds)

to construct the remaining rules for each policy in the compliance dataset. For each $\mathcal{P}$ in the compliance dataset, the time taken in seconds for the evaluation of $\mathcal{P} \sqsubseteq \mathsf{RFC5735}^{\mathsf{I}}$ is given in Table 7. We observe that the compliance test has a cheap cost in terms time, and all evaluation times for $\mathcal{P} \sqsubseteq \mathsf{RFC5735}^{\mathsf{I}}$ are negligible.

7.2. Implementing Duplet Join and Difference

In this section, the Python prototype implementations of duplet/rule join and duplet/rule difference are defined. We demonstrate how Algorithm 1 (duplet join) and Algorithm 2 (duplet difference) relate to firewall rule composition in $\mathcal{FW}_1$. For ease of exposition, we use colours in tables when illustrating rule/duplet composition. The colours used are of no particular significance. The Python implementation is expected to shortly be uploaded to an open-source repository.

Duplet Combination. For $f, g \in \delta[X, Y]$, then the combination operation $(f \uplus_{\delta[X,Y]} g)$ defines a set of adjacency-free duplets that exactly cover f and g.

```
[X]
  $\_ \uplus_{\_}\ \_ : \mathbb{P}X \nrightarrow (X \times X) \rightarrow \mathbb{P}X$
  ────────
  $\forall a, b : X \bullet$
      $\forall c : (a \uplus_X b) \bullet$
          $a \overset{X}{\leftarrow} c \vee b \overset{X}{\leftarrow} c \wedge$
              $\nexists d : (a \uplus_X b) \bullet$
                  $c \wr_X d \mid c \neq d$
```

The operation is described using three recursive functions; *center* $\mathcal{C}(f, g)$, *left* $\mathcal{L}(f, g)$ and *right* $\mathcal{R}(f, g)$, and is defined as the set union of the duplets resulting from $\mathcal{C}(f, g)$, $\mathcal{L}(f, g)$ and $\mathcal{R}(f, g)$. For ease of exposition, a duplet is given as a sequence of filter condition attributes. We assume f and g always have the same number of attributes. The functions are defined as follows.

Center. For $f, g \in \delta[X, Y]$, then $\mathcal{C}(f, g)$ defines the join of the adjacent and common attributes in f and g. For duplets comprising two attributes, we define:

$$\mathcal{C}(\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle) = \langle a_1 \cup_X a_2, b_1 \cap_Y b_2 \rangle$$

D A	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1 \cup_X a_1}$	$\boxed{b_1 \cap_Y b_2}$

Table 8
Center function for two-attribute duplets

D A	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$
$\boxed{1}$	$\boxed{a_1 \cup_X a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{c_1 \cap_Y c_2}$

Table 9
Center function for three-attribute duplets

D A	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$

Table 10
Left function for two-attribute duplets

D A	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$
$\boxed{1}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$	$\boxed{c_1}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{c_1 \setminus_Y c_2}$

Table 11
Left function for three-attribute duplets

Table 8 specifies the operations and duplet resulting from $\mathcal{C}(f,g)$ for two-attribute duplets. The label D signifies duplet, while A means the attribute. For f and g of length greater than two, we define for each additional attribute:

$$\mathcal{C}(f\,\hat{}\,\langle c_1\rangle, g\,\hat{}\,\langle c_2\rangle) = \mathcal{C}(f,g) \frown \langle c_1 \cap_Y c_2\rangle$$

Table 9 specifies the operations and duplet resulting from $\mathcal{C}(f,g)$ for duplets with three attributes.

Left. For $f, g \in \delta[X,Y]$, then $\mathcal{L}(f,g)$ defines the remaining attribute constraints covered in f, that are not covered in $\mathcal{C}(f,g)$. For duplets comprising two attributes, we define:

$$\mathcal{L}(\langle a_1,b_1\rangle, \langle a_2,b_2\rangle) = \{\langle a_1, b_1 \setminus_Y b_2\rangle\}$$

Table 10 specifies the operations and duplet resulting from $\mathcal{L}(f,g)$ for two-attribute duplets. For f and g of length greater than two, we define for each additional attribute:

$$\begin{aligned}\mathcal{L}(f\,\hat{}\,\langle c_1\rangle, g\,\hat{}\,\langle c_2\rangle) = {} & \{\langle \mathit{head}\, f\rangle \frown \mathit{tail}\,(\mathcal{C}(f,g)) \frown \langle c_1 \setminus_Y c_2\rangle\} \cup \\ & \{t : \mathcal{L}(f,g) \bullet t \frown \langle c_1\rangle\}\end{aligned}$$

Table 11 specifies the operations and duplets resulting from $\mathcal{L}(f,g)$ for duplets comprising three attributes.

Right. For $f, g \in \delta[X,Y]$, then $\mathcal{R}(f,g)$ defines the remaining attribute constraints covered in g, that are not covered in $\mathcal{C}(f,g)$. For duplets comprising two attributes, we define:

$$\mathcal{R}(\langle a_1,b_1\rangle, \langle a_2,b_2\rangle) = \{\langle a_2, b_2 \setminus_Y b_1\rangle\}$$

Table 12 specifies the operations and duplet resulting from $\mathcal{R}(f,g)$ for two-attribute duplets. For f and g of length greater than two, we define for each additional attribute:

$$\begin{aligned}\mathcal{R}(f\,\hat{}\,\langle c_1\rangle, g\,\hat{}\,\langle c_2\rangle) = {} & \{\langle \mathit{head}\, g\rangle \frown \mathit{tail}\,(\mathcal{C}(f,g)) \frown \langle c_2 \setminus_Y c_1\rangle\} \cup \\ & \{t : \mathcal{R}(f,g) \bullet t \frown \langle c_2\rangle\}\end{aligned}$$

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_2}$	$\boxed{b_2 \setminus_Y b_1}$

Table 12

Right function for two-attribute duplets

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$
$\boxed{1}$	$\boxed{a_2}$	$\boxed{b_2 \setminus_Y b_1}$	$\boxed{c_2}$
$\boxed{2}$	$\boxed{a_2}$	$\boxed{b_2 \cap_Y b_1}$	$\boxed{c_2 \setminus_Y c_1}$

Table 13

Right function for three-attribute duplets

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1 \cup_X a_1}$	$\boxed{b_1 \cap_Y b_2}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$
$\boxed{3}$	$\boxed{a_2}$	$\boxed{b_2 \setminus_Y b_1}$

Table 14

A two-attribute duplet join

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$	$\boxed{\cdot\cdot}$	$\boxed{k}$
$\boxed{1}$	$\boxed{a_1 \cup_X a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1 \cap_Y x_2}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1}$
$\boxed{3}$	$\boxed{a_2}$	$\boxed{b_2 \setminus_Y b_1}$	$\boxed{\cdot\cdot}$	$\boxed{x_2}$
$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{x_1}$
$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{x_2}$
$\boxed{(k+k-2)}$	$\boxed{a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1 \setminus_Y x_2}$
$\boxed{(k+k-1)}$	$\boxed{a_2}$	$\boxed{b_2 \cap_Y b_1}$	$\boxed{\cdot\cdot}$	$\boxed{x_2 \setminus_Y x_1}$

Table 15

A k-attribute duplet join

Table 13 specifies the operations and duplets resulting from $\mathcal{R}(f, g)$ for duplets comprising three attributes.
Thus, we define the combination operation for f and g as:

$$f \uplus_{\delta[X,Y]} g = \{\mathcal{C}(f, g)\} \cup \mathcal{L}(f, g) \cup \mathcal{R}(f, g)$$

Theorem 7.1. *The duplet combination operation defines the adjacency-free combination for all* $f, g \in \delta[X, Y]$; $n \in \mathbb{N}$, *where* $(n = \#f = \#g) \geqslant 2$.

Proof We will show that for $f, g \in \delta[X, Y]$, then $f \uplus_{\delta[X,Y]} g$ defines the adjacency-free combination for all $n \in \mathbb{N}$, where $(n = \#f = \#g) \geqslant 2$, using induction on n.

Base Case. For $n = 2$, then for $f \uplus_{\delta[X,Y]} g$, the resulting operations and duplets for f and g as two-attribute duplets are given in Table 14. From these results, we have that Theorem 7.1 holds when $n = 2$.

Inductive Hypothesis. Suppose Theorem 7.1 holds for $k \in \mathbb{N}$, where $k > n$, and $k = \#f = \#g$. Then for $f \uplus_{\delta[X,Y]} g$, the resulting operations and duplets for f and g as k-attribute duplets are given in Table 15:

Inductive Step. Let $n = k + 1$. Then by the recursive definitions of $\mathcal{C}(f, g)$, $\mathcal{L}(f, g)$ and $\mathcal{R}(f, g)$, the resulting operations and duplets for f and g as $(k + 1)$-attribute duplets in $(f \uplus_{\delta[X,Y]} g)$ are given in Table 16. Therefore, from these results, we have that Theorem 7.1 holds for $n = k + 1$. By the principal of mathematical induction, the theorem holds for all $n \in \mathbb{N}$, where $n \geqslant 2$. ▮

Algorithm 1 summarises the duplet combination operation, whereby the input is a pair of duplets (f, g), the number of attributes (*len*), and an empty list (*ruleSet*) to hold the result. The output is the list (*ruleSet*), whereby *ruleSet* $= \langle \mathcal{C}(f, g), \mathcal{L}(f, g), \mathcal{R}(f, g) \rangle$.

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$	$\boxed{..}$	$\boxed{k}$	$\boxed{k+1}$
$\boxed{1}$	$\boxed{a_1 \cup_X a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{..}$	$\boxed{x_1 \cap_Y x_2}$	$\boxed{y_1 \cap_Y y_2}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$	$\boxed{..}$	$\boxed{x_1}$	$\boxed{y_1}$
$\boxed{3}$	$\boxed{a_2}$	$\boxed{b_2 \setminus_Y b_1}$	$\boxed{..}$	$\boxed{x_2}$	$\boxed{y_2}$
$\boxed{..}$	$\boxed{..}$	$\boxed{..}$	$\boxed{..}$	$\boxed{x_1}$	$\boxed{y_1}$
$\boxed{..}$	$\boxed{..}$	$\boxed{..}$	$\boxed{..}$	$\boxed{x_2}$	$\boxed{y_2}$
$\boxed{(k+k-2)}$	$\boxed{a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{..}$	$\boxed{x_1 \setminus_Y x_2}$	$\boxed{y_1}$
$\boxed{(k+k-1)}$	$\boxed{a_2}$	$\boxed{b_2 \cap_Y b_1}$	$\boxed{..}$	$\boxed{x_2 \setminus_Y x_1}$	$\boxed{y_2}$
$\boxed{(k+k)}$	$\boxed{a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{..}$	$\boxed{x_1 \cap_Y x_2}$	$\boxed{y_1 \setminus_Y y_2}$
$\boxed{(k+k+1)}$	$\boxed{a_2}$	$\boxed{b_2 \cap_Y b_1}$	$\boxed{..}$	$\boxed{x_2 \cap_Y x_1}$	$\boxed{y_2 \setminus_Y y_1}$

Table 16
A $(k + 1)$-attribute duplet join

A Bottom-up Approach to Duplet Join. Recall, the definition for the join operation of $S, T \in \alpha[\delta[X, Y]]$ given in Section 3.3 is constructed following a top-down approach with respect to the ordering relation $\leqslant$, whereby:

$$S \oplus T = \lceil \{a, b : \bigcap \{U : \mathbb{P}(\delta[X, Y]) \mid (\forall c : U \bullet \exists a : S;\ b : T \bullet c \overset{\delta[X,Y]}{\to} a \vee c \overset{\delta[X,Y]}{\to} b)\} \mid a \wr^{+}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b\} \rceil$$

For all $S, T \in \alpha[\delta[X, Y]]$, we define the implementation definition for sets of adjacency-free duplets as:

$$S \oplus T = \lceil \{a, b : \lceil \bigcup \{a, b : (S \cup T) \mid a \wr^{+}_{(S \cup T)} b \bullet (a \uplus_{\delta[X,Y]} b)\} \rceil \mid a \wr^{+}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b\} \rceil$$

Adjacency Duplet Union Implementation. The implementation definition for the join of $S, T \in \alpha[\delta[X, Y]]$ is defined as the cover-set for the duplet merge of the transitive closure of adjacent duplets, from the coverset for the generalized union of sets from the duplet combination operation $(\_ \uplus \_\,\_)$, for all transitively adjacent duplets in S and T. The coverset for the the generalized union defines the smallest collection of duplets that cover all of the duplets from both S and T by precedence subsumption. Given that all duplets in this set are now disjoint, the cover-set for the duplet merge of the transitive closure of adjacent duplets merges any forward-adjacent duplets from S and T. If we take some $U \in \alpha[\delta[X, Y]]$, such that $U \leqslant (S \oplus T)$ and $S \leqslant U \wedge T \leqslant U$, then $(S \oplus T) = U$. Thus, the implementation definition for duplet Adjacency-set join provides a lowest upper bound operator.

Algorithm 1 Duplet combination operation

```
1:  function Combine(f, g, len, ruleSet)
2:      if ($len == 2$) then
3:          $ruleSet(0) \leftarrow \langle a_1 \cup_X a_2, b_1 \cap_Y b_2 \rangle$
4:          $ruleSet(1) \leftarrow \{\langle a_1, b_1 \setminus_Y b_2 \rangle\}$
5:          $ruleSet(2) \leftarrow \{\langle a_2, b_2 \setminus_Y b_1 \rangle\}$
6:          return ruleSet
7:      else
8:          $len \leftarrow len - 1$
9:          $i \leftarrow \#ruleSet(0)$
10:         $l \leftarrow {}^\frown/\langle\langle head\, f\rangle, tail\,(ruleSet(0)), \langle f(i) \setminus_Y g(i)\rangle\rangle$
11:         $r \leftarrow {}^\frown/\langle\langle head\, g\rangle, tail\,(ruleSet(0)), \langle g(i) \setminus_Y f(i)\rangle\rangle$
12:         $ruleSet(0) \leftarrow ruleSet(0) \frown \langle f(i) \cap_Y g(i)\rangle$
13:         $ruleSet(1) \leftarrow \{t : ruleSet(1) \bullet t \frown \langle f(i)\rangle\} \cup \{l\}$
14:         $ruleSet(2) \leftarrow \{t : ruleSet(2) \bullet t \frown \langle g(i)\rangle\} \cup \{r\}$
15:         return Combine(f, g, len, ruleSet)
16: end function
```

Theorem 7.2. *The two given definitions for joining sets of adjacency-free duplets are equivalent.*

$$\begin{aligned}
&\forall S, T : \alpha[\delta[X,Y]] \bullet \\
&\quad S \oplus T = \lceil \{a, b : \textstyle\bigcap \{U : \mathbb{P}(\delta[X,Y]) \mid (\forall c : U \bullet \exists a : S;\ b : T \bullet \\
&\qquad\qquad c \overset{\delta[X,Y]}{\rightarrow} a \vee c \overset{\delta[X,Y]}{\rightarrow} b)\} \mid a \overset{+}{\wr}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b\} \rceil \\
&\qquad\quad = \lceil \{a, b : \lceil \textstyle\bigcup \{a, b : (S \cup T) \mid a \overset{+}{\wr}_{(S \cup T)} b \bullet (a \uplus_{\delta[X,Y]} b)\} \rceil \mid \\
&\qquad\qquad a \overset{+}{\wr}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b\} \rceil
\end{aligned}$$

Proof Given that both definitions define the cover-set for the duplet merge of the transitive closure of forward adjacent duplets from the smallest collection of disjoint adjacency-free duplets that cover all of the duplets from both S and T by precedence subsumption, then Theorem 7.2 holds. ∎

Duplet Difference. For $f, g \in \delta[X, Y]$, the operation $(f \setminus_{\delta[X,Y]} g)$ defines a set of adjacency-free duplets that are covered by f but not by g. The operation is described using two recursive functions; *center* $\mathcal{C}^{diff}(f, g)$, and the *left* $\mathcal{L}(f, g)$ function given previously. The function $(f \setminus_{\delta[X,Y]} g)$ is defined as the set union of the duplets resulting from $\mathcal{C}^{diff}(f, g)$ and $\mathcal{L}(f, g)$.

Center. For $f, g \in \delta[X, Y]$, then for duplets comprising two attributes; $\mathcal{C}^{diff}(f, g)$ is defined as follows:

$$\mathcal{C}^{diff}(\langle a_1, b_1\rangle, \langle a_2, b_2\rangle) = \langle a_1 \setminus_X a_2, b_1 \cap_Y b_2\rangle$$

The operations and duplet resulting from $\mathcal{C}^{diff}(f, g)$ for two-attribute duplets are given in Table 17. For f and g of length greater than two, we define for each additional attribute:

$$\mathcal{C}^{diff}(f \hat{}\langle c_1\rangle, g \hat{}\langle c_2\rangle) = \mathcal{C}^{diff}(f, g) \frown \langle c_1 \cap_Z c_2\rangle$$

$D \setminus A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1 \setminus_X a_1}$	$\boxed{b_1 \cap_Y b_2}$

Table 17

Center difference function for two-attribute duplets

$D \setminus A$	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$
$\boxed{1}$	$\boxed{a_1 \setminus_X a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{c_1 \cap_Y c_2}$

Table 18

Center difference function for three-attribute duplets

$D \setminus A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1 \setminus_X a_1}$	$\boxed{b_1 \cap_Y b_2}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$

Table 19

A two-attribute duplet difference

$D \setminus A$	$\boxed{1}$	$\boxed{2}$	$\boxed{\cdot\cdot}$	$\boxed{k}$
$\boxed{1}$	$\boxed{a_1 \setminus_X a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1 \cap_Y x_2}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1}$
$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{x_1}$
$\boxed{k}$	$\boxed{a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1 \setminus_Y x_2}$

Table 20

A k-attribute duplet difference

$D \setminus A$	$\boxed{1}$	$\boxed{2}$	$\boxed{\cdot\cdot}$	$\boxed{k}$	$\boxed{k+1}$
$\boxed{1}$	$\boxed{a_1 \setminus_X a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1 \cap_Y x_2}$	$\boxed{y_1 \cap_Y y_2}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1}$	$\boxed{y_1}$
$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{\cdot\cdot}$	$\boxed{x_1}$	$\boxed{y_1}$
$\boxed{(k)}$	$\boxed{a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1 \setminus_Y x_2}$	$\boxed{y_1}$
$\boxed{(k+1)}$	$\boxed{a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{\cdot\cdot}$	$\boxed{x_1 \cap_Y x_2}$	$\boxed{y_1 \setminus_Y y_2}$

Table 21

A $(k + 1)$-attribute duplet difference

The operations and duplet resulting from $\mathcal{C}^{diff}(f, g)$ for duplets with three attributes are given in Table 18. Thus, we define the difference operation for f and g as:

$$f \setminus_{\delta[X,Y]} g = \{\mathcal{C}^{diff}(f, g)\} \cup \mathcal{L}(f, g)$$

Theorem 7.3. *The duplet difference operation defines the adjacency-free duplet-difference for all* $f, g \in \delta[X, Y]$; $n \in \mathbb{N}$, *where* $(n = \#f = \#g) \geqslant 2$.

Proof We will show that for $f, g \in \delta[X, Y]$, then $f \setminus_{\delta[X,Y]} g$ defines the adjacency-free duplet difference for all $n \in \mathbb{N}$, where $(n = \#f = \#g) \geqslant 2$, using induction on n.

Base Case. For $n = 2$, then for $f \setminus_{\delta[X,Y]} g$, the resulting operations and duplets for f and g as two-attribute duplets are given in Table 19. From these results, we have that Theorem 7.3 holds when $n = 2$.

Inductive Hypothesis. Suppose Theorem 7.3 holds for $k \in \mathbb{N}$, where $k > n$, and $k = \#f = \#g$. Then for $f \setminus_{\delta[X,Y]} g$, the resulting operations and duplets for f and g as k-attribute duplets are given in Table 20.

Algorithm 2 Duplet difference operation

```
 1: function $Difference(f, g, len, ruleSet)$
 2:     if $(len == 2)$ then
 3:         $ruleSet \leftarrow \langle\langle a_1 \setminus_X a_2, b_1 \cap_Y b_2\rangle, \{\langle a_1, b_1 \setminus_Y b_2\rangle\}\rangle$
 4:         return $ruleSet$
 5:     else
 6:         $len \leftarrow len - 1$
 7:         $i \leftarrow \#ruleSet(0)$
 8:         $l \leftarrow {}^\frown/\langle\langle head\, f\rangle, tail\,(ruleSet(0)), \langle f(i) \setminus_Y g(i)\rangle\rangle$
 9:         $ruleSet(0) \leftarrow ruleSet(0) {}^\frown \langle f(i) \cap_Y g(i)\rangle$
10:         $ruleSet(1) \leftarrow \{t : ruleSet(1) \bullet t {}^\frown \langle f(i)\rangle\} \cup \{l\}$
11:         return $Difference(f, g, len, ruleSet)$
12: end function
```

Inductive Step. Let $n = k + 1$. Then by the recursive definitions of $\mathcal{C}^{diff}(f, g)$ and $\mathcal{L}(f, g)$, the resulting operations and duplets for f and g as $(k + 1)$-attribute duplets in $(f \setminus_{\delta[X,Y]} g)$ are given in Table 21. Therefore, from the results we observe that Theorem 7.3 holds for $n = k + 1$. By the principal of mathematical induction, the theorem holds for all $n \in \mathbb{N}$, where $n \geqslant 2$. ∎

Algorithm 2 summarises the duplet difference operation, whereby the input is a pair of duplets (f, g), the number of attributes (*len*), and an empty list (*ruleSet*) to hold the result. The output is the list (*ruleSet*), whereby $ruleSet = \langle\mathcal{C}^{diff}(f, g), \mathcal{L}(f, g)\rangle$.

8. Related Work

There is a rich literature of work on managing firewall policy configurations. For example, The goal of [2, 3, 8, 10, 23, 45] is to provide an administrator with the means to detect/resolve anomalies, and work such as [15, 17, 34, 35] allows for querying a policy configuration with regard to the filtering of specific network traffic. High-level specification languages such as [1, 4, 12, 26, 31] allow an administrator to abstractly specify what would otherwise be low-level rules. However, in general, the literature for policy management is focused on the conventional five-tuple firewall rule with a binary target action of *allow* or *deny*, and few have considered stateful firewall configurations.

Hari et al. [27] report some of the earliest research on conflict detection and resolution in policies for packet-filters, and model rule relations within a policy in a directed graph. Al-Shaer et al. [2, 3] provide definitions for firewall policy anomalies. The authors use a form of Binary Decision Diagram (BDD) to represent a firewall policy, and define relationships between pairwise rules. The *Firewall Policy Advisor* [2] tool implements algorithms used to identify firewall rule anomalies using set theory. In this paper, we use a subset of the classifications from [2, 3] when reasoning about firewall policy anomalies. Cuppens et al. [10] and García-Alfaro et al. [24], present MIRAGE (MIsconfiguRAtion manaGEr), and provide definitions alternative to [2, 3] for intra- and inter-firewall policy anomalies. Given a firewall configuration, MIRAGE will automatically detect and remove intra-redundant [10, 24] and intra-shadowed [10, 24] rules, and generate a semantically-equivalent order-independent set of disjoint rules that are anomaly-free. In contrast to [27], the approach incorporates the automatic re-writing of anomalous rules. Yuan et al. [45] present the FIREMAN (FIREwall Modelling ANalysis) tool. Policy configurations are analysed for inconsistencies that consider intra-shadowing, intra-generalisation, intra-correlation and inter-shadowing anomalies [2, 3]. Firewall inefficiency in packet classification and memory consumption is also considered as a result of intra-redundant rules, and *'verbosities'*, whereby a set of rules may be summarized into a smaller number of rules without changing the filtering semantics of the policy. Chomsiri and Pornavalai [8] propose a method of firewall policy analysis using relational algebra. The definitions provided for intra-redundant and intra-shadowed rules are analogous to [10], and upon detection, such rules are removed in order to reduce the size of the policy. The definitions given for intra-generalization and intra-correlation are analogous to [2]. Similar to the notion of verbosities in [45], Chomsiri and Pornavalai propose combining rules that may be 'summarized' without changing the filtering semantics of the policy. Buttyán et al. [7] propose a tool based on FIREMAN [45] for managing anomalies in stateful firewall policies. The authors argue that verifying a stateful firewall for inconsistencies can be reduced to the problem of verifying a stateless firewall for inconsistencies. A limitation of the approach is that their model does not distinguish rules with different state information, that is, for example, there is no differentiation between the establishment and termination phase of a given stateful protocol, and as a consequence, they do not consider more complex anomalies that may occur specifically in the stateful case. Cuppens et al. [11] and García-Alfaro et al. [23] propose an algorithmic approach to detect and resolve anomalies in a stateful firewall policy. A connection-oriented protocol is modelled using general automata, whereby the permitted protocol states and transitions are encoded. Intra-redundant and intra-shadowed rules [10] are considered for the stateful firewall policy. Further definitions are proposed, whereby an *intra-state* anomaly occurs in a stateful firewall policy if there are policy rules that partially match (complete) the paths of the protocol automata. In the case of missing rules, then covering-rules are suggested to the administrator as a means of completing the path of the protocol automata. Their work also considers invalid protocol states, and *inter-state* anomalies that may occur in a firewall policy that filters packets against both stateful and stateless rules. The work in [11, 23] also extends the MIRAGE [24] tool. To consider the types of stateful anomalies from [23] in the proposed model $\mathcal{FW}_1$, then it would be necessary to apply additional constraints when constructing and composing anomaly-free firewall policies. For example, a policy that specifies a firewall rule that enables the establishment of a TCP connection from host X to host Y, should also include rules that allow for the various other permissible transitions of the TCP protocol.

Firewall query analysis allows an administrator to pose questions of a policy configuration, such as, for example, *"does the policy permit SSH traffic from system X to system Y?"*. Mayer et al. [35], present Fang (Firewall ANalysis enGine). Based on graph algorithms and a rule-base simulator, Fang parses vendor-specific firewall policy and configuration files, and constructs a model of the network topology and a global firewall policy for the network. Fang interacts with the administrator through a query-and-answer session, and queries are constructed as triples, consisting of source IPs, destination IPs and endpoint services/ports. Eronen and Zitting [15] propose an expert system for query analysis, implemented in constraint programming logic. A query is constructed as a six-tuple, consisting of source and destination IP and port, network protocol, and flags. The flags attribute is for TCP connections, however, only the SYN and ACK flags are considered. Eronen and Zitting argue that in comparison to Fang [35], the expert system is a more natural solution for query analysis. Liu et al. [34] present the Structured Firewall Query Language (SFQL), an SQL-like query language. Liu et al. state that constructing an expert system such as [15] *"just for analysing a firewall is overwrought and impractical"* [34], however, they do not give their reasoning for this assertion. SFQL queries can be constructed over *allow* and *deny* rules for an arbitrary number of filter fields. While query analysis provides an administrator with a means of asking questions of a firewall policy, what can actually be queried is restricted by the collection of filter condition attributes and target actions expressible in the query language. Effective query analysis may be further hampered by the complexity of the query language, or through an administrators inability to construct useful queries. A consequence of the $\mathcal{FW}_1$ algebra proposed in this paper, is that it enables an administrator to perform effective query analysis of a firewall policy configuration. While we do not construct individual high-level queries, we do however demonstrate how policies in the algebra may be tested/queried for compliance with best practice standards and recommendations.

High-level specification languages provide an administrator with the means to reduce the complexity of constructing a firewall policy configuration. Guttman [26] reported some of the earliest research in this area. Bartal et al. [4] present the Firmato toolkit. The proposed specification language allows an administrator to specify the network security policy and the topology for the network in terms of an entity-relationship (ER) model. Subsequently, a policy configuration is synthesised from the ER model. A limitation of the work is that it only applies to packet-filter policy configurations. The High Level Firewall Language (HLFL) [31] translates high-level firewall rules into usable rulesets for iptables, Cisco ACLs, IPFW and others. However, the generated rulesets are order-dependant and may contain anomalies, and the approach does not provide support for incorporating knowledge about a network topology when specifying the high-level rules. Fitzgerald and Foley [17] propose using ontologies to represent knowledge about firewall policy configurations. Policies are specified using Description Logic and SWRL. Semantic Threat Graphs [21] are used to encode catalogues of best practice firewall rules, and an automated synthesis of standards-compliant rules for a policy configuration is considered. However, the administrator must manually construct the rulesets for the catalogues then populate the Semantic Threat Graphs, and this process is error-prone. The proposed model in [17] can also be used for firewall policy query analysis. Adão et al. [1] propose a declarative policy specification language, and present Mignis, a tool that translates high-level access control specifications into low-level policy configurations for Netfilter. An abstract model of the Netfilter firewall is proposed, and definitions for Network Address Translation and stateful filtering are encoded. The synthesised policies consist of order-independent iptables firewall rules. However, the proposed approach is tightly coupled with Netfilter. Brucker et al. [6] present a formal model of both stateless and stateful firewalls, including Network Address Translation. The authors follow a theorem-proving approach to reason about firewall policies, and provide formal and machine-verifiable proof of correctness using the Isabelle theorem prover. Diekmann et al. [14] present a fully verified firewall ruleset analysis framework for iptables, that similar to Brucker et al. [6], also provides proof of correctness using Isabelle. In this paper, we use the Z notation to provide a consistent syntax for systematically presenting the proposed model $\mathcal{FW}_1$. Mathematical definitions have been syntax- and type-checked, however the proofs in this paper are of the conventional pen-and-paper variety.

We model a firewall policy as an ordered pair of disjoint adjacency-free sets, where the set of policies *Policy* forms a lattice under $\sqsubseteq$, and each $P \in Policy$ is anomaly-free by construction. In [2, 3, 10, 27, 45] an algorithmic approach is taken to detect/resolve anomalies. In contrast, we follow an algebraic approach towards modelling anomalies in a single policy, and across a distributed policy configuration through policy composition. In [46], a firewall policy algebra is proposed. However, the authors note that an anomaly-free composition is not guaranteed as a result of using their algebraic operators. Our work differs, in that policy composition under the $\sqcap$, $\sqcup$ and $\fatsemi$ operators defined in this paper all result in anomaly-free policies. In earlier work [22], we developed the algebra $\mathcal{FW}_0$, and used it to reason over host-based and network access controls in OpenStack. In the $\mathcal{FW}_0$ algebra, we focused on stateless firewall policies that are defined in terms of constraints on individual IPs, ports and protocols. In this paper, the algebra $\mathcal{FW}_1$ is defined over stateful firewall policies constructed in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes. We argue that $\mathcal{FW}_1$ gives a more expressive means for reasoning over OpenStack security group and perimeter firewall configurations. In [30], *cloud calculus* is used to capture the topology of cloud computing systems and the global firewall policy for a given configuration. This paper could extend the work in [30], given that $\mathcal{FW}_1$ may be used in conjunction with cloud calculus to guarantee anomaly-free dynamic firewall policy reconfiguration, whereby the ordering relation $\sqsubseteq$ may give a viable alternative for the given equivalence relation defined over 'cloud' terms for the formal verification of firewall policy preservation after a live migration. The proposed algebra $\mathcal{FW}_1$ is used to reason about and compose anomaly-free policies and therefore we do not have to worry about dealing with conflicts that may arise. Anomaly conflicts are dealt with in composition by computing anomaly-free policies, rather than using techniques such as [29] to resolve conflicts in policy decisions.

9. Conclusion

A policy algebra $\mathcal{FW}_1$ is defined in which firewall policies can be specified and reasoned about. At the heart of this algebra is the notion of safe replacement, that is, whether it is secure to replace one firewall policy by another. The set of policies form a lattice under safe replacement and this enables consistent operators for safe composition to be defined. Policies in this lattice are anomaly-free by construction, and thus, composition under glb and lub operators preserves anomaly-freedom. A policy sequential composition operator is also proposed that can be used to interpret firewall policies defined more conventionally as sequences of rules. The algebra can be used to characterize anomalies, such as shadowing and redundancy, that arise from sequential composition. Best practice policy compliance may be defined using $\sqsubseteq$. The algebra $\mathcal{FW}_1$ provides a formal interpretation of the network access controls for a partial mapping to the iptables filter table.

$\mathcal{FW}_1$ is a generic algebra and can also be used to model other firewall systems. The results in this paper are described in terms of the algebra $\mathcal{FW}_1$, for stateful firewall policies that are defined in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable feedback. This work was supported, in part, by Science Foundation Ireland under grant SFI 10/CE/I1853 and Irish Research Council/Chist-ERA. This manuscript is a pre-print version, the final publication [38] is available at IOS Press through http://dx.doi.org/10.3233/JCS-17971.

References

[1] P. Adão, C. Bozzato, G. Dei Rossi, R. Focardi, and F.L. Luccio. Mignis: A semantic based tool for firewall configuration. In *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*, pages 351–365. IEEE, 2014.
[2] E. Al-Shaer and H. Hamed. Firewall Policy Advisor for Anomaly Discovery and Rule Editing. *8th IFIP/IEEE International Symposium on Integrated Network Management, Colorado Springs, USA*, March 2003.
[3] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict Classification and Analysis of Distributed Firewall Policies. *IEEE Journal on Selected Areas in Communications, Issue: 10, Volume: 23, Pages: 2069 - 2084*, October 2005.

[4] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A Novel Firewall Management Toolkit. *20th IEEE Symposium on Security and Privacy, Oakland, CA, USA*, May 1999.

[5] G. Birkhoff. *Lattice Theory. Volume XXV of American Mathematical Society Colloquium Publications*. American Mathematical Society, 3rd edition, 1967.

[6] A. D. Brucker, L. Brügger, and B. Wolff. Formal firewall conformance testing: An application of test and proof techniques. *Softw. Test. Verif. Reliab.*, 25(1):34–71, January 2015.

[7] L. Buttyán, G. Pék, and T. Vinh Thong. Consistency verification of stateful firewalls is not harder than the stateless case. *Infocommunications Journal*, 64(1):2–8, 2009.

[8] T. Chomsiri and C. Pornavalai. Firewall Rules Analysis. *International Conference on Security and Management (SAM), Las Vegas, Nevada, USA*, June 2006.

[9] M. Cotton and L. Vegoda. Special Use IPv4 Addresses. RFC 5735, January 2010.

[10] F. Cuppens, N. Cuppens-Boulahia, and J. García-Alfaro. Detection and Removal of Firewall Misconfiguration. *IASTED International Conference on Communication, Network and Information Security (CNIS)*, November 2005.

[11] F. Cuppens, N. Cuppens-Boulahia, J. García-Alfaro, T. Moataz, and X. Rimasson. Handling stateful firewall anomalies. In *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings*, pages 174–186, 2012.

[12] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège. A Formal Approach to Specify and Deploy a Network Security Policy. *2nd Workshop on Formal Aspects in Security and Trust (FAST)*, August 2004.

[13] D.E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, May 1976.

[14] C. Diekmann, J. Michaelis, M.P.L. Haslbeck, and G. Carle. Verified iptables firewall analysis. In *2016 IFIP Networking Conference, Networking 2016 and Workshops, Vienna, Austria, May 17-19, 2016*, pages 252–260. IEEE, 2016.

[15] P. Eronen and J. Zitting. An Expert System for Analyzing Firewall Rules. *6th Nordic Workshop on Secure IT Systems (NordSec), Copenhagen, Denmark*, pages 100–107, November 2001.

[16] M. Fabrice. iptables Extensions - Time module. http://ipset.netfilter.org/iptables-extensions.man.html. [Online; accessed February-2017].

[17] W.M. Fitzgerald and S.N. Foley. Management of heterogeneous security access control configuration using an ontology engineering approach. In *3rd ACM Workshop on Assurable and Usable Security Configuration, SafeConfig 2010, Chicago, IL, USA, October 4, 2010*, pages 27–36, 2010.

[18] W.M. Fitzgerald, U. Neville, and S.N. Foley. MASON: Mobile Autonomic Security for Network Access Controls. *Journal of Information Security and Applications (JISA)*, 18(1):14–29, 2013.

[19] S.N. Foley. Reasoning about confidentiality requirements. In *Seventh IEEE Computer Security Foundations Workshop - CSFW'94, Franconia, New Hampshire, USA, June 14-16, 1994, Proceedings*, pages 150–160, 1994.

[20] S.N. Foley. The specification and implementation of commercial security requirements including dynamic segregation of duties. In *ACM Conference on Computer and Communications Security*, pages 125–134, 1997.

[21] S.N. Foley and W.M. Fitzgerald. An approach to security policy configuration using semantic threat graphs. In *23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec)*, pages 33–48. Springer LNCS, 2009.

[22] S.N. Foley and U. Neville. A firewall algebra for openstack. In *2015 IEEE Conference on Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015*, pages 541–549, 2015.

[23] J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martínez Perez, and J. Cabot. Management of stateful firewall misconfiguration. *Computers & Security*, 39:64–85, 2013.

[24] J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda. MIRAGE: A management tool for the analysis and deployment of network security policies. In *Data Privacy Management and Autonomous Spontaneous Security - 5th International Workshop, DPM 2010 and 3rd International Workshop, SETOP 2010, Athens, Greece, September 23, 2010, Revised Selected Papers*, pages 203–215, 2010.

[25] L. Gheorghe. *Designing and Implementing Linux Firewalls with QoS using netfilter, iproute2, NAT and l7-filter*. PACKT Publishing, October 2006.

[26] J.D. Guttman. Filtering Postures: Local Enforcement for Global Policies. *IEEE Symposium on Security and Privacy*, pages 120–129, May 1997.

[27] A. Hari, S. Suri, and G. Parulkar. Detecting and Resolving Packet Filter Conflicts. *19th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM*, 3:1203–1212, March 2000.

[28] J.L. Jacob. The varieties of refinement. In J.M. Morris and R.C. Shaw, editors, *Proceedings of the 4th Refinement Workshop*, pages 441–455. Springer, Heidelberg, 1991.

[29] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.

[30] Y. Jarraya, A. Eghtesadi, M. Debbabi, Y. Zhang, and M. Pourzandi. Cloud calculus: Security verification in elastic cloud computing platform. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 447–454. IEEE, 2012.

[31] A. Launay. High Level Firewall Language. https://www.cusae.com/hlfl/, 2003. [Online; accessed August-2016].
[32] J. Levandoski et al. iptables L7-filter Pattern Writing. http://l7-filter.sourceforge.net/Pattern-HOWTO, April 2008. [Online; accessed February-2017].
[33] J. Levandoski et al. iptables L7-filter Supported Protocols. http://l7-filter.sourceforge.net/protocols, August 2008. [Online; accessed February-2017].
[34] A. Liu, M. Gouda, H. Ma, and A. Hgu. Firewall Queries. *8th International Conference, On Principles of Distributed Systems (OPODIS), Grenoble, France*, pages 197–212, December 2004.
[35] A. Mayer, A. Wool, and E. Ziskind. Fang: A Firewall Analysis Engine. *21st IEEE Symposium on Security and Privacy, Oakland, CA, USA*, May 2000.
[36] V. Nebehaj. python-iptables - Python bindings for iptables. https://pypi.python.org/pypi/python-iptables. [Online; accessed August-2016].
[37] U. Neville and S.N. Foley. Reasoning About Firewall Policies Through Refinement and Composition. In *Data and Applications Security and Privacy XXX - 30th Annual IFIP WG 11.3 Conference, DBSec 2016, Trento, Italy, July 18-20, 2016. Proceedings*, pages 268–284, 2016.
[38] U. Neville and S.N. Foley. Reasoning about firewall policies through refinement and composition. *Journal of Computer Security*, 26(2):207–254, 2018.
[39] I. Ocean. Pyinter - a small and simple library written in Python for performing interval and discontinous range arithmetic. https://pypi.python.org/pypi/pyinter/, August 2015. [Online; accessed August-2016].
[40] J. Postel, D. Johnson, T. Markson, B. Simpson, and Z. Su. Internet Control Message Protocol (ICMP) Parameters. https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml, April 2013. [Online; accessed February-2017].
[41] K. Scarfone and P. Hoffman. Guidelines on Firewalls and Firewall Policy: Recommendations of the National Institute of Standards and Technology. *NIST Special Publication 800-41, Revision 1*, September 2009.
[42] J.M. Spivey. *The Z Notation: A Reference Manual.* Series in Computer Science. Prentice Hall International, second edition, 1992.
[43] Netfilter Core Team. Linux iptables - CLI for configuring the Linux kernel firewall, Netfilter. http://www.netfilter.org/projects/iptables/index.html. [Online; accessed February-2017].
[44] A. Wool. Trends in firewall configuration errors: Measuring the holes in swiss cheese. *IEEE Internet Computing*, 14(4):58–65, 2010.
[45] L. Yuan, H. Chen, J. Mai, C. Chuah, Z. Su, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.
[46] H. Zhao and S.M. Bellovin. Policy algebras for hybrid firewalls. Number CUCS-017-07, March 2007.

Appendix A. The Z Notation

A set may be defined in Z using set specification in comprehension. This is of the form $\{ D \mid P \bullet E \}$, where D represents declarations, P is a predicate and E an expression. The components of $\{ D \mid P \bullet E \}$ are the values taken by expression E when the variables introduced by D take all possible values that make the predicate P true. For example, the set of squares of all even natural numbers is defined as $\{ n : \mathbb{N} \mid (n \bmod 2) = 0 \bullet n^2 \}$. When there is only one variable in the declaration and the expression consists of just that variable, then the expression may be dropped if desired. For example, the set of all even numbers may be written as $\{ n : \mathbb{N} \mid (n \bmod 2) = 0 \}$. Sets may also be defined in display form such as $\{1, 2\}$.

In Z, relations and functions are represented as sets of pairs. A (binary) relation R, declared as having type $A \leftrightarrow B$, is a component of $\mathbb{P}(A \times B)$, where $\mathbb{P}\, X$ is the powerset of X. For $a \in A$ and $b \in B$, then the pair (a, b) is written as $a \mapsto b$, and $a \mapsto b \in R$ means that a is related to b under relation R. Functions are treated as special forms of relations. The schema notation is used to structure specifications. A schema such as $\mathcal{FW}_1$ defines a collection of variables (limited to the scope of the schema) and specifies how they are related. The variables can be introduced via schema inclusion, as done, for example, in the definition of sequential composition.