Formal analysis of firewalls using tree automata techniques

Tony Bourdier

Formal analysis of firewalls using tree automata techniques

Tony Bourdier INRIA Nancy Research Center – PAREO Team
615, rue du Jardin Botanique
54600 Villers-lès-Nancy, France
Email: Tony.Bourdier@inria.fr

***Abstract*—Since the late 80s, firewalls are at the heart of network security. First designed to enable private networks to be opened up to the outside in a secure way, the growing complexity of organizations make them indispensable to control information flow within a company. The central role of firewalls in the security of the organization information make their management a critical task. That is why for years many works have focused on checking and analysing firewalls. In this paper, we propose a new approach for analysing firewalls, based on tree automata techniques: we show that tree automata provide a way to compare firewalls and to perform all usual analysis of firewalls (including the network address translation (NAT) functionality) in a unique formalism.**

***Index Terms*—firewall; network security; tree automata; formal analysis.**

I. Introduction

Since the late 1980s, firewalls are at the heart of network security. First designed to enable private networks to be opened up to the outside in a secure way, the growing complexity of organizations make them indispensable to control information flow within a company. The central role of firewalls in the security of the organization information make their management a critical task. That is why for years a lot of works have been focused on checking and analysing firewalls.

Many methods and tools have been developped since about ten years for analysing and testing firewall policies. These methods are broken down into two different categories: the active methods and the passive methods. The former consist in sending packets to the network and to make a diagnosis according to the received packets. The main advantage of these methods is that they require no abstract representation of firewalls and thus no error can be introduced between the specification and the implementation. However, such methods have the major drawback of consuming bandwidth, interfering with the traffic and being no exhaustive. That is why we focused on passive methods, that is methods which send no packet and make an offline analysis. Two main categories of passive analysis are investigated in the literature: structural analysis and query analysis. Structural analysis examine the relationships that rules have with other rules within a firewall configuration or across multiple firewalls. A misconfiguration (or conflict) occurs when several rules match the same packet or when a rule can be removed without changing the behaviour of the firewall. Query analysis provides a way to ask questions of the form "Which computers in the private network can receive packets from `www.inria.fr`?". It then consists in defining a language to describe a firewall query and a way to compute its solutions. Some interesting work [1], [2], [3], [4], [5], [6], [7], [8], [9] looked into structural analysis and others [10], [11], [12], [13] looked into query analysis. Indeed, [1], [2], [3], [4], [5], [6], [7], [8], [9] focus on defining, detecting and discussing misconfigurations. To illustrate their reasoning and their detection algorithms, some of these works abstract firewall filtering rules as one or two-dimensional ranges of IP, which does not allow to take completely advantage of the obtained results. Moreover, all of them assume that packets are not modified during their network traversal and then do not support network translation address capabilities. [10], [11], [12], [13] use structures based on decision diagrams which provide a way to represent both rule sets of firewalls and solutions of some queries over firewalls. [13] improves the structure to include the NAT functionality but decision diagrams do not offer an easy way to compare several sets of rules. As for [14], [15], they propose an automatic process generating firewall rules from an abstract specification of a security policy. However, the validity of the translations was not proved and the obtained results are not proved to be conflict free.

Tree automata techniques have already been shown suitable to perform cryptographic protocol verification [16] but have not been explored for checking properties over firewalls. In this paper, we show that tree automata techniques are very relevant to deal with problems on firewalls. Indeed, tree automata have been used to (i) perform pattern-matching [17], [18], (ii) compute solutions of some first order formulae [19], [20] due to their closure properties and (iii) represent some rewriting relations [21]. We claim that these three topics are exactly the ones concerned by firewalls: a firewall is modeled as a sequence of filtering rules whose semantics can be intensionally described by a tree automaton since it corresponds to a matching problem (topic (i)). In this context, a query analysis consists in finding solutions of a first order formula over tree automata, which exactly corresponds to the issue (ii) and finally, the network address translation functionality is a process rewriting a packet (seen as a symbolic term) into another one and then corresponds to the problem (iii). Moreover, tree automata have the advantage of being mutually comparable due to the existence and the uniqueness of a minimal representation of a set (or a relation) by a tree automaton. Thus, tree automata offer us a full support to perform usual analysis and comparison over firewalls in a

unique formalism. In SECTION II, we recall basic definitions of terms, rewrite systems and tree automata. In SECTION III, we show how to specify firewalls and give in SECTION IV a semantics of firewalls based on tree automata. We describe in SECTION V how our framework provide a way to perform usual analysis. Finally, we will conclude by giving some future possible works.

II. PRELIMINARIES

This section briefly recall basic notions used in this paper; more details can be found in [22], [23] for first order terms and rewrite systems and in [24], [25] for tree automata and tree language theory.

A. Term algebra and rewrite systems

A *signature* Σ consists of a finite set $\mathcal{S}_\Sigma$ whose elements are called *sorts*, possibly in relation by $<$ (which means "is a *subsort* of"), and an alphabet of symbols together with an application which associates to any symbol f a non empty sequence of sorts, which is denoted by $f : s_1 \times \ldots \times s_n \mapsto s$. $ar(f) = n$ is called the *arity* of f. Given a signature Σ, a sort $\kappa \in \mathcal{S}_\Sigma$ and a countable set $\mathcal{X}^s$ of *variables* for each sort s, we denote by $\mathcal{T}^\kappa_{\Sigma,\mathcal{X}}$ the set whose elements are called *terms sorted by* κ inductively defined as follows: for any $x \in \mathcal{X}^\kappa$, x is in $\mathcal{T}^\kappa_{\Sigma,\mathcal{X}}$ and for any $f : s_1 \times \ldots \times s_n \mapsto \kappa$ and $\langle t_1, \ldots, t_n \rangle \in \mathcal{T}^{s'_1}_{\Sigma,\mathcal{X}} \times \ldots \times \mathcal{T}^{s'_n}_{\Sigma,\mathcal{X}}$, with $s'_i \leq s_i$ for any i, the word $f(t_1, \ldots, t_n)$ is in $\mathcal{T}^\kappa_{\Sigma,\mathcal{X}}$. $\mathcal{T}_{\Sigma,\mathcal{X}}$ is the union of $\mathcal{T}^s_{\Sigma,\mathcal{X}}$ for every sort s. Note that for any sorts κ and κ', either $\mathcal{T}^\kappa_{\Sigma,\mathcal{X}} \cap \mathcal{T}^{\kappa'}_{\Sigma,\mathcal{X}} = \varnothing$ or one of these sorts is a subsort of the other. The set of variables occurring in $t \in \mathcal{T}_{\Sigma,\mathcal{X}}$ is denoted by $\mathcal{V}ar(t)$. If any variable of $\mathcal{V}ar(t)$ occurs only once in t, t is said to be *linear*. If $\mathcal{V}ar(t)$ is empty, t is called a ground term. $\mathcal{T}_\Sigma$ denotes the set of all ground terms. A *position* of a term t is a finite sequence of positive integers describing the path from the root of t to the root of the sub-term at that position. We write $\omega >_{pref} \omega'$ if there exists a position ω'' such that $\omega = \omega'.\omega''$, *i.e.* if ω is deeper than ω'. The empty sequence representing the root position is denoted by ε. $\mathcal{P}os(t)$ is called the set of positions of t. $t_{|\omega}$, resp. $t(\omega)$, denotes the subterm of t, resp. the symbol of t, at position ω. We denote by $t\,[s]_\omega$ the term t with the subterm at position ω replaced by s. We call *substitution* any mapping from $\mathcal{X}$ to $\mathcal{T}_{\Sigma,\mathcal{X}}$ which is the identity except over a finite set of variables $\mathcal{D}om(\sigma)$ called *domain* of σ extended to an endomorphism of $\mathcal{T}_{\Sigma,\mathcal{X}}$. σ is often denoted by $\{x \mapsto \sigma(x) \mid x \in \mathcal{D}om(\sigma)\}$. If $\mathcal{C}odom(\sigma) \subseteq \mathcal{T}_\Sigma$, σ is said to be ground. For any ground substitution σ, $\sigma(t)$ is called a *ground instantiation* of t. A *rewrite rule* (over Σ) is a pair $(lhs, rhs) \in \mathcal{T}_{\Sigma,\mathcal{X}} \times \mathcal{T}_{\Sigma,\mathcal{X}}$ such that $\mathcal{V}ar(lhs) \subseteq \mathcal{V}ar(rhs)$ and a *rewrite system* is a set of rewrite rules $\mathcal{R}$ inducing a *rewriting relation* over $\mathcal{T}_\Sigma$, denoted by $\rightarrow_\mathcal{R}$ and such that $t \rightarrow_\mathcal{R} t'$ iff there exist $(l, r) \in \mathcal{R}$, $\omega \in \mathcal{P}os(t)$ and a ground substitution σ such that $t_{|\omega} = \sigma(l)$ and $t' = t\,[\sigma(r)]_\omega$. Finally, we denote by $\xrightarrow{*}_\mathcal{R}$ the reflexive transitive closure of $\rightarrow_\mathcal{R}$.

B. Tree automata

We call *n-ary (ascending) tree automaton* any quadruple $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$ such that Σ is an alphabet of function symbols, Q is a finite set of *states*, F is a subset of Q whose elements are called *final states* and Δ is a relation over $\mathcal{T}_{\Sigma^n[Q]} \times Q$ whose elements are called *transitions* where Λ is a new symbol and $\Sigma^n[Q]$ consisting of the unique sort $conf$ and the alphabet $(\Sigma \cup \{\Lambda\})^n \setminus \{\langle \Lambda, \ldots, \Lambda \rangle\} \cup Q$ such that $\langle f_1, \ldots, f_n \rangle \in (\Sigma \cup \{\Lambda\})^n$ is of sort $conf \times \ldots \times conf \mapsto conf$ with $ar(f_1, \ldots, f_n) = \max_{i\in[1,n]}(ar(f_i) \mid f_i \neq \Lambda)$ and any $q \in Q$ is a constant of sort $conf$. Examples of tree automata are given in APPENDIX B. An element of $\mathcal{T}_{\Sigma^n[Q]}$ is called a *configuration*. A transition $lhs \rightarrow rhs$ of Δ is *normalized* iff for any $\omega \neq \varepsilon$, $lhs(\omega) \in Q$. An automaton whose transitions are normalized is said normalized. A tree automaton is said *deterministic* iff all its transitions have a different left-hand side. Without loss of generality, we can consider that all automata are normalized and deterministic. The rewriting relation induced by Δ over $\mathcal{T}_{\Sigma^n[Q]}$ is denoted by $\rightarrow_\mathbb{A}$ and the language recognized by $\mathbb{A}$ is $\mathcal{L}(\mathbb{A}) = \{\langle t_1, \ldots, t_n \rangle \in \mathcal{T}_\Sigma \mid \exists q_f \in F, t_1 \otimes \ldots \otimes t_n \xrightarrow{*}_\mathbb{A} q_f\}$ where $t = t_1 \otimes \ldots \otimes t_n$ is the configuration such that: $\forall \omega \in \bigcup_{i=1}^n \mathcal{P}os(t_i)$, $t(\omega) = \langle t_1[\omega), \ldots, t_n[\omega) \rangle$ where $u[\omega) = u(\omega)$ if $\omega \in \mathcal{P}os(t)$ and Λ otherwise. A set E of n-tuples of terms (or equivalently n-ary relation) is said *recognizable* iff there exists an n-ary tree automaton $\mathbb{A}$ such that $E = \mathcal{L}(\mathbb{A})$. The FIGURE 1 recalls usual automata and operations over tree automata together with their semantics. Note that we will often write $t \in \mathbb{A}$ instead of $t \in \mathcal{L}(\mathbb{A})$. We also recall that the membership, the emptiness, the finiteness, the equivalence and the inclusion problems are decidable for tree automata. We suppose that all considered automata are deterministic and minimal. Finally, when there is no ambiguity, q_F^{aut} and Δ_{aut} denote respectively the final state and the set of transitions of the automaton $\mathbb{A}_{aut}$.

Notation	Language recognized by the automaton
$\mathbb{A} \oplus \mathbb{A}'$	$\mathcal{L}(\mathbb{A}) \oplus \mathcal{L}(\mathbb{A}')$ where $\oplus$ is $\cap$, $\cup$, or $\times$
$\overline{\mathbb{A}}$	$(\mathcal{T}_\Sigma)^n \setminus \mathcal{L}(\mathbb{A})$
Ω^n_κ	n-tuples $\langle t_1, \ldots, t_n \rangle$ of $\mathcal{T}^\kappa_\Sigma$
Id^n_κ	n-tuples $\langle t, \ldots, t \rangle$ of $\mathcal{T}^\kappa_\Sigma$
$rec(t)$	ground instantiations of t
$\Pi_i(\mathbb{A})$	$(n+1)$-tuples $\langle t_1, \ldots, t_{i-1}, t, t_i, \ldots, t_n \rangle$ $s.t.$ $\langle t_1, \ldots, t_n \rangle \in \mathcal{L}(\mathbb{A})$
$\Pi_i(\mathbb{A})$	$(n-1)$-tuples $\langle t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n \rangle$ $s.t.$ there exists a $t \in \mathcal{T}_\Sigma$: $\langle t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n \rangle \in \mathcal{L}(\mathbb{A})$
$\Pi_{i/t}(\mathbb{A})$	$(n-1)$-tuples $\langle t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n \rangle$ $s.t.$ $\langle t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n \rangle \in \mathcal{L}(\mathbb{A})$
$\partial^k_{\langle f_1,\ldots,f_n \rangle}(\mathbb{A})$	n-tuples $\langle t_1, \ldots, t_n \rangle$ $s.t.$ there exists a tuple $\langle f_1(x^1_1, \ldots, x^{k-1}_1, t_1, x^{k+1}_1, \ldots, x^m_1), \ldots, f_n(x^1_n, \ldots, x^{k-1}_n, t_n, x^{k+1}_n, \ldots, x^m_n) \rangle \in \mathcal{L}(\mathbb{A})$

where n and i are integers, t is a term of $\mathcal{T}_{\Sigma,\mathcal{X}}$, κ is a sort and $\mathbb{A}$ and $\mathbb{A}'$ are n-ary tree automata.

Fig. 1. Main automata and operators with their semantics

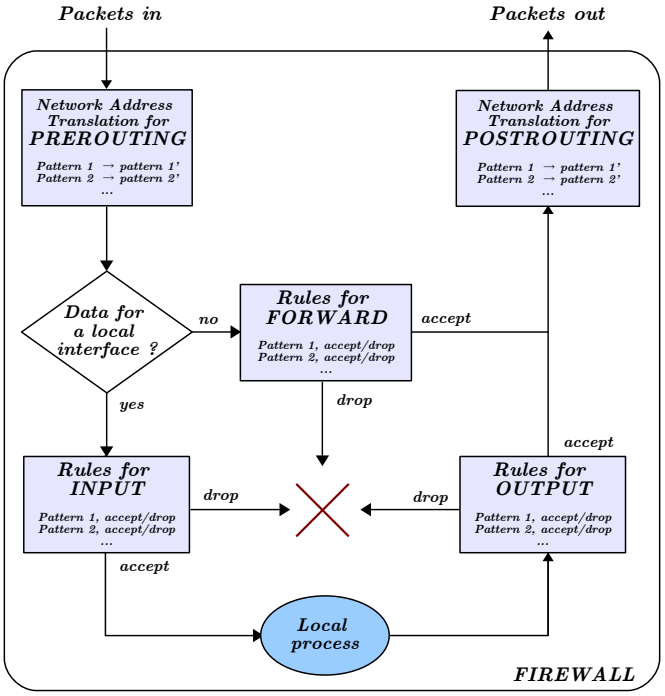


Fig. 2. Global architecture of a firewall

C. Firewalls

Based on the description of firewalls embedded in Linux 2.4 [26] and FreeBSD, NetBSD and other main operating systems [27], we define a firewall as an element of a network designed to control packets traffic between different domains by using a combination of:

- the *packet filter* technique, which consists in inspecting each packet and either allowing them to continue their traversal or dropping them;
- and the *network address translation* functionality, which consists in modifying network address information in packet headers.

The FIGURE 2 gives the global architecture of a firewall. A firewall contains three filter blocks called *chains*: INPUT, OUTPUT and FORWARD. When a packet arrives to the entry of a filter block, the corresponding chain decides if the packet must be dropped or must continue its traversal of the diagram. In order to compute this decision, each chain is made up of a list of *rules* which map the description of a set of packets to a decision. The most often used criteria that chains use when inspecting packets are the following [27], [26]: the IP source address, the IP destination address, the protocol, the source port and the destination port. Moreover, when a packet comes in or out, the firewall rewrites a part of the packet header. At the input of the firewall, we talk about *prerouting* and at the output we talk about *postrouting*. The former consists in modifying information concerning the destination of the packet and the latter information concerning the source of the packet.

III. FIREWALL SPECIFICATION

In this section, we show how to model a firewall by first order terms and tree automata.

A. Packet representation

First, we present the signature and the terms used for representing IPs, protocols, packets, ...We thus define a signature $\Sigma_{Firewall}$ containing the constants 0 and 1 of sort Bit, TCP, UDP, ICMP of sort $Protocol$, and the following non constant symbols:

$$\begin{array}{lrl} bit & : Bit & \mapsto Binary \\ bin & : Bit \times Binary & \mapsto Binary \\ ip & : Octet \times Octet \times Octet \times Octet & \mapsto IP \\ port & : Binary[16] & \mapsto Port \end{array}$$

where $Binary[n]$ is the subsort of $Binary$ containing terms built with n-1 symbols bin and $Octet$ is a shortcut for $Binary[8]$. Thus, the (real) octet $o = b_7.b_6.b_5.b_4.b_3.b_2.b_1.b_0$ is represented by the following term of sort $Octet$: $\overrightarrow{o} = bin(b_7, bin(b_6, bin(\ldots, bin(b_1, bit(b_0)))))$. Finally, $\Sigma_{Firewall}$ contains the following symbol: $packet : IP \times IP \times Protocol \times Port \times Port \mapsto Packet$. Note that given an IP ip, we denote by $\overrightarrow{ip}$ the symbolic representation (*i.e.* the first order term) of ip. We could add, without any difficulty, other attributes of packet headers but for readability reasons we propose to consider these five parameters in this paper. Note that we have two kinds of parameters: the ones which can take a small number of values (protocols in our case) and the others which can take a large range of values (ports and IPs). The values of former can be encoded by themselves (*e.g.* TCP for TCP, UDP for UDP, ...) and the values of the latter must be encoded such that we can represent a range of values regardless of the size of the range, which avoid combinatorial explosion of computations.

B. Packet filtering

As said before, a filtering chain consists of an ordered list of rules of the form $\langle\, set, decision \,\rangle$ where set is a set of packets and $decision$ is either ACCEPT or DROP. Usually, the sets of packets are represented by a tuple containing two ranges of IP (source and destination), a set protocol and two ranges of ports (source and destination). Then, encoding of a filtering rule $\langle\, set, decision \,\rangle$ is straightforward: we build an automaton $rec(set)$ which recognizes the set of packets matching the rule. It is based on the tree automaton recognizing pairs of binary numbers $\langle\, b_1, b_2 \,\rangle$ such that $b_1 \leq b_2$ whose construction is explained in APPENDIX A. From this automaton, we easily build $\mathbb{A}_{ip\_inf}$ (recognizing $\{ip_1, ip_2 \mid ip_1 \leq ip_2\}$) and another one recognizing ranges of IP $[ip_1, ip_2]$: $\mathbb{A}_{[ip_1,ip_2]} = \Pi_{1/ip_1}(\mathbb{A}_{ip\_inf}) \cap \Pi_{2/ip_2}(\mathbb{A}_{ip\_inf})$ (we do the same for ranges of ports).

Thus, we associate to any rule[1] $\langle\, set, decision \,\rangle$, an automaton recognizing exactly the representation of packets

[1] set being the tuple containing $[ip_1^{src}, ip_2^{src}]$, $[ip_1^{dest}, ip_2^{dest}]$, pr, $[port_1^{src}, port_2^{src}]$, $[port_1^{dest}, port_2^{dest}]$

matching the rule, denoted by $rec(set)$, and defined as follows: $rec(set) = \langle \Sigma_{firewall}, Q_{set}, \{q_F^{set}\}, \Delta_{set} \rangle$ where $\Delta_{set} = \Delta_{ip^{src}} \cup \Delta_{ip^{dest}} \cup \Delta_{pr} \cup \Delta_{port^{src}} \cup \Delta_{port^{dest}} \cup \{packet(q_{ip^{src}}, q_{ip^{dest}}, q_{pr}, q_{port^{src}}, q_{port^{dest}}) \to q_F^{set}\}$, and q_k is the final state of the automaton $\mathbb{A}_k$ with:

- $\mathbb{A}_{ip^{src}} = \mathbb{A}_{[ip_1^{src}, ip_2^{src}]}$
- $\mathbb{A}_{ip^{dest}} = \mathbb{A}_{[ip_1^{dest}, ip_2^{dest}]}$
- $\mathbb{A}_{pr} = (\Sigma_{firewall}, \{q_{pr}\}, \{q_{pr}\}, \Delta_{pr})$ and $\Delta_{pr} = \{protocol \to q_{pr} \mid protocol \in pr\}$
- $\mathbb{A}_{port^{src}} = \mathbb{A}_{[port_1^{src}, port_2^{src}]}$
- $\mathbb{A}_{port^{dest}} = \mathbb{A}_{[port_1^{dest}, port_2^{dest}]}$

and where Q_{set} contains all states occurring in Δ_{set}.

C. Network address translation

In typical cases, network address translation (NAT) simply maps a set of addresses to a constant address, *e.g.* $\langle 192.168.*.*, 80 \rangle \twoheadrightarrow \langle 254.128.215.12, 8080 \rangle$. In this paper, in order to cover all reasonable implementations of network address translation, we consider the hypothesis where the mapping can build the translated address from the original one, *e.g.* $\langle 192.168.x.y, 80 \rangle \twoheadrightarrow \langle 192.182.x.y, 8080 \rangle$. Thus, we assume that a NAT rule is a pair of terms of the form $\langle ip, port \rangle \twoheadrightarrow \langle ip', port' \rangle$ where ip, ip' are IP addresses possibly containing variables such that any variable occurring in ip' occurs in ip at the same position, $port$ and $port'$ are either port numbers or a variable such that $port'$ is a variable only if $port$ is the same variable.

D. Specification of firewalls

We can therefore define the specification of firewalls we use for performing formal analysis:

DEFINITION III.1: (FORMAL FIREWALL) A *formal firewall* (or simply *firewall*) is a 5-tuple:

$$\mathfrak{f} = \langle \mathcal{R}_{pre}, \mathbb{A}_{\text{INPUT}}, \mathbb{A}_{\text{FORWARD}}, \mathbb{A}_{\text{OUTPUT}}, \mathcal{R}_{post} \rangle$$

where:

- $\mathcal{R}_{pre}$ and $\mathcal{R}_{post}$ are linear ordered rewrite systems over $\Sigma_{firewall}$ whose rules $lhs \to rhs$ are such that lhs and rhs are of sort $packet$ and for any position ω such as $rhs(\omega) \in \mathcal{X}$, we have $lhs(\omega) = rhs(\omega)$,
- $\mathbb{A}_{\text{INPUT}}, \mathbb{A}_{\text{FORWARD}}$ and $\mathbb{A}_{\text{OUTPUT}}$ are tree automata over $\Sigma_{firewall}$ recognizing a subset of $\mathcal{T}_{Packet}$.

Taking advantage of the specification of packet filters and address translations described in the previous section, we will explain how to automatically translate a usual specification of a firewall into a formal one corresponding to the definition given above. Thus, for any chain CHAIN and its associated ordered set $Rules_{\text{CHAIN}} = \{\langle set_i, decision_i \rangle \mid i \in [1, n_{\text{CHAIN}}]\}$, we build $\mathbb{A}_{\text{CHAIN}}$ as follows:

$$\bigcup_{\substack{i=1 \\ decision_i = \text{ACCEPT}}}^{n_{\text{CHAIN}}} \left(rec(set_i) \setminus \left(\bigcup_{k=1}^{i-1} rec(set_k) \right) \right)$$

Contrary to the three filter chains of the firewall which are translated in the same way, NAT rules are translated differently depending on the target (pre- or postrouting). Indeed, prerouting translates destination addresses of packets while postrouting translates source ones. Thus, to any NAT rule $\langle ip_1, port_1 \rangle \twoheadrightarrow \langle ip_2, port_2 \rangle$ we associate the pair of first order terms: $packet(x_{ip^{src}}, \overrightarrow{ip_1}, x_{pr}, x_{port^{src}}, \overrightarrow{port_1}, x_{ste})$ and $packet(x_{ip^{src}}, \overrightarrow{ip_2}, x_{pr}, x_{port^{src}}, \overrightarrow{port_2}, x_{ste})$ in the prerouting case, where $\overrightarrow{ip_i}$ can be or contain variables. In the postrouting case, we build the pair $packet(\overrightarrow{ip_1}, x_{ip^{dest}} x_{pr}, \overrightarrow{port_1}, x_{port^{dest}}, x_{ste})$ and $packet(\overrightarrow{ip_2}, x_{ip^{dest}} x_{pr}, \overrightarrow{port_2}, x_{port^{dest}}, x_{ste})$. We then obtain for prerouting (resp. postrouting) an ordered list of pairs of first order terms which corresponds to $\mathcal{R}_{pre}$ (resp. $\mathcal{R}_{post}$).

EXAMPLE III.2: An example of firewall is given in Figure 3. Since there is useless to give the whole corresponding formal firewall, we give only a part of $\mathbb{A}_{\text{INPUT}}$ and $\mathcal{R}_{pre}$:

- $\mathbb{A}_{\text{INPUT}} = (\Sigma_{firewall}, Q_{\text{INPUT}}, \{q_F\}, \Delta_{\text{INPUT}})$ with

$$\Delta_{\text{INPUT}} = \left\{ \begin{array}{c} packet(q_1^1, q_2, q_3, q_4, q_5, q_6) \to q_F \\ packet(q_1^2, q_2, q_3, q_4, q_5, q_6) \to q_F \\ tcp \to q_3 \quad ; \quad \overrightarrow{80} \to q_4 \quad ; \quad new \to q_6 \\ \dots \end{array} \right.$$

  where q_1^1 is the final state of $\mathbb{A}_{[192.168.10.0, 192.168.10.100]}$, q_1^2 is the final state of $\mathbb{A}_{[192.168.10.102, 192.168.10.255]}$, q_2 the one of Ω_{IP} and q_5 the one of Ω_{Port}.
- $\mathcal{R}_{pre}$ consists of:

$$\left\{ \begin{array}{l} packet(x_1, \overrightarrow{121.130.1.x}, x_2, x_3, \overrightarrow{5222}, x_4) \\ \to packet(x_1, \overrightarrow{121.130.1.x}, x_2, x_3, \overrightarrow{777}, x_4) \\ packet(x_1, \overrightarrow{216.239.37.125}, x_2, x_3, \overrightarrow{5222}, x_4) \\ \to packet(x_1, \overrightarrow{195.20.241.122}, x_2, x_3, \overrightarrow{777}, x_4) \end{array} \right.$$

IV. SEMANTICS

We define in this section the semantics of a firewall starting from its formal representation as defined in the previous section. Let we recall that when a packet enters a firewall, three cases are conceivable:

- the packet is intended to the firewall itself, then it will pass through prerouting and the input chain;
- the destination of the packet is not the firewall, and then it will cross the prerouting, the forward chain and the postrouting;
- or the packet is sent out by the firewall, then it will go across the output chain and the prerouting.

We then study the semantics of the traversal of each of these three possible paths. We write $pkt_1 \to_{pre} pkt_2$ (resp. $pkt_1 \to_{post} pkt_2$) iff pkt_2 is obtained by applying the first applicable rule of $\mathcal{R}_{pre}$ (resp. $\mathcal{R}_{post}$) from pkt_1 or $pkt_2 = pkt_1$ if no rule is applicable. We say that a firewall $\mathfrak{f}$ *accepts* a packet pkt :

- in the INPUT chain iff $pkt \to_{pre} pkt' \in \mathcal{L}(\mathbb{A}_{\text{INPUT}})$;
- in the FORWARD chain iff $pkt \to_{pre} pkt' \in \mathcal{L}(\mathbb{A}_{\text{FORWARD}})$;
- in the OUTPUT chain iff $pkt \in \mathcal{L}(\mathbb{A}_{\text{OUTPUT}})$;

for some pkt'. If $\mathfrak{f}$ does not accept pkt in the chain CHAIN, we say that $\mathfrak{f}$ drops pkt in CHAIN. Moreover, we say that $\mathfrak{f}$ *delivers a packet* pkt *under the form* pkt' *through* :

Chain	*IP address src*	*IP address dst*	*Protocol*	*Port src*	*Port dst*	*Decision*
IN	192.168.10.101	$*.*.*.*$	*tcp*	80	*any*	DROP
IN	192.168.10.$*$	$*.*.*.*$	*tcp*	80	*any*	ACCEPT
IN	$*.*.*.*$	$*.*.*.*$	*any*	*any*	*any*	DROP
OUT	$*.*.*.*$	192.168.10.$*$	*tcp*	80	*any*	ACCEPT
OUT	$*.*.*.*$	$*.*.*.*$	*any*	*any*	*any*	DROP
FWD	192.168.20.$*$	121.130.$*.*$	*tcp*	80	*any*	ACCEPT
FWD	$*.*.*.*$	$*.*.*.*$	*tcp*	*any*	$\{5222, 443\}$	DROP
FWD	192.168.20.$*$	209.85.229.104	*tcp*	80	*any*	DROP
FWD	$*.*.*.*$	$*.*.*.*$	*tcp*	80	*any*	ACCEPT
FWD	$*.*.*.*$	$*.*.*.*$	*any*	*any*	*any*	DROP

$$\begin{aligned} \textit{Prerouting } NAT: \quad & \langle\, 121.130.1.x, 5222 \,\rangle \twoheadrightarrow \langle\, 121.130.1.x, 777 \,\rangle \\ & \langle\, 216.239.37.125, 5222 \,\rangle \twoheadrightarrow \langle\, 195.20.241.122, 777 \,\rangle \\ \textit{Postrouting } NAT: \quad & \langle\, 192.168.2.38, x \,\rangle \twoheadrightarrow \langle\, 195.20.241.122, x \,\rangle \end{aligned}$$

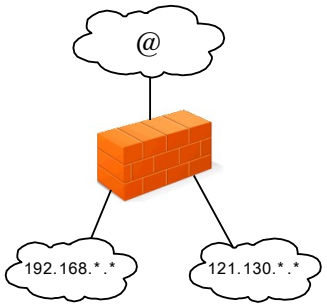


Fig. 3. Example of firewall

- INPUT iff $\mathfrak{f}$ accepts pkt in INPUT and $pkt \rightarrow_{pres} pkt'$;
- FORWARD iff $\mathfrak{f}$ accepts pkt in FORWARD and $pkt \rightarrow_{pres} pkt'' \rightarrow_{post} pkt'$;
- OUTPUT iff $\mathfrak{f}$ accepts pkt in OUTPUT and $pkt \rightarrow_{post} pkt'$.

We define for any chain CHAIN a binary relation $\overset{\text{CHAIN}}{\rightsquigarrow}_{\mathfrak{f}}$ such that for any packet pkt:

$$pkt \overset{\text{CHAIN}}{\rightsquigarrow}_{\mathfrak{f}} pkt'$$

iff $\mathfrak{f}$ deliver pkt through CHAIN under the form pkt'. When there is no pkt' such that $pkt \overset{\text{CHAIN}}{\rightsquigarrow}_{\mathfrak{f}} pkt'$, *i.e.* when the packet is dropped by CHAIN, we write $pkt \overset{\text{CHAIN}}{\rightsquigarrow}_{\mathfrak{f}} \times$.

DEFINITION IV.3: (SEMANTICS OF A FIREWALL) Given a firewall $\mathfrak{f} = \langle\, \mathcal{R}_{pre}, \mathbb{A}_{\text{INPUT}}, \mathbb{A}_{\text{FORWARD}}, \mathbb{A}_{\text{OUTPUT}}, \mathcal{R}_{post} \,\rangle$, we call *semantics of* $\mathfrak{f}$ the 3-tuple

$$[\![\mathfrak{f}]\!] = \left\langle \overset{\text{IN}}{\rightsquigarrow}_{\mathfrak{f}}, \overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}}, \overset{\text{OUT}}{\rightsquigarrow}_{\mathfrak{f}} \right\rangle$$

EXAMPLE IV.4: We still consider the firewall described in Figure 3. We have for example: $packet(\overrightarrow{192.168.10.2}, \overrightarrow{208.68.163.220}, tcp, 5222, 5222) \overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}} \times$ and $packet(\overrightarrow{121.130.1.2}, \overrightarrow{216.239.37.125}, tcp, 5222, 5222) \overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}} packet(\overrightarrow{121.130.1.2}, \overrightarrow{195.20.241.122}, tcp, 5222, 777)$

We next will show that the semantics of a firewall is a tuple of recognizable relations. Such a result will allow us to compare firewalls and to detect anomalies.

LEMMA IV.5: For any firewall $\mathfrak{f} = \langle\, \mathcal{R}_{pre}, \mathbb{A}_{\text{INPUT}}, \mathbb{A}_{\text{FORWARD}}, \mathbb{A}_{\text{OUTPUT}}, \mathcal{R}_{post} \,\rangle$, $\rightarrow_{pre}$ and $\rightarrow_{post}$ are recognizable relations, in other terms, there exist two automata $\mathbb{A}_{pre}$ and $\mathbb{A}_{post}$ such that $\langle\, pkt, pkt' \,\rangle \in \mathbb{A}_{pre}$ iff $pkt \rightarrow_{pre} pkt'$ and $\langle\, pkt, pkt' \,\rangle \in \mathbb{A}_{post}$ iff $pkt \rightarrow_{post} pkt'$.

PROOF: Let $lhs \rightarrow rhs$ be a rule such that $lhs, rhs : Packet$ and $\forall \omega \in \mathcal{P}os(rhs)$, if $rhs(\omega) \in \mathcal{X}$ then $lhs(\omega) = rhs(\omega)$. The set of pairs of ground terms pkt_1, pkt_2 such that $pkt_1 \rightarrow_{\{lhs \rightarrow rhs\}} pkt_2$ can be denoted by the following tree automaton $\mathbb{A}[lhs \rightarrow rhs] = (\Sigma_{firewall}, Q, \{q_F\}, \Delta)$ where Δ contains:

- $\langle\, packet, packet \,\rangle\, (q_1, q_2, q_3, q_4, q_5, q_6) \rightarrow q_F$;
- for any $\omega \in \mathcal{P}os(lhs)$ *s.t.* $lhs(\omega)$ is not a variable,
$$\langle\, lhs(\omega), rhs[\omega) \,\rangle\, (q_{\omega.1}, \ldots, q_{\omega.n}) \rightarrow q_\omega$$
with $n = \max\left(ar(lhs(\omega)), ar(rhs[\omega))\right)$;
- for any $\omega \in \mathcal{P}os(rhs) \setminus \mathcal{P}os(lhs)$,
$$\langle\, \Lambda, rhs(\omega) \,\rangle\, (q_{\omega.1}, \ldots, q_{\omega.n}) \rightarrow q_\omega$$
with $n = ar(rhs(\omega))$;
- for any $\omega \in \mathcal{P}os(lhs)$ *s.t.* $lhs(\omega)$ is a variable of sort s and $rhs(\omega)$ is not a variable, the set of rules of the automaton $\mathbb{A}^s_{sort} \times rec\left(\{rhs_{|\omega}\}\right)$ whose final state is renamed into q_ω;
- for any ω *s.t.* $rhs(\omega) = lhs(\omega)$ is a variable of sort s, the set of rules of the automaton Id^2_s;

and Q contains all states occurring in Δ.
For any rewrite system $\mathcal{R}$ containing an ordered set of rules of the form described above, we denote by $\mathbb{A}_{\mathcal{R}}$ the following automaton:

$$\bigcup_{i=1}^{n} \mathbb{A}[lhs_i \rightarrow rhs_i] \setminus \left(\bigcup_{k=1}^{i-1} (rec(lhs_k) \times rec(rhs_i)) \right)$$

The automaton recognizing $\rightarrow_{pres}$ (resp. $\rightarrow_{post}$) is the one denoted by $\mathbb{A}_{pre}$ (resp. $\mathbb{A}_{post}$) such that:

$$\mathbb{A}_{pre} = \mathbb{A}_{\mathcal{R}_{pre}} \cup \left(Id^2_{Packet} \setminus \left(\Pi_1(\mathbb{A}_{\mathcal{R}_{pre}}) \times \Pi_1(\mathbb{A}_{\mathcal{R}_{pre}})\right)\right)$$

resp. $\mathbb{A}_{\mathcal{R}_{post}} \cup \left(Id^2_{Packet} \setminus \left(\Pi_1(\mathbb{A}_{\mathcal{R}_{post}}) \times \Pi_1(\mathbb{A}_{\mathcal{R}_{post}})\right)\right)$ □

PROPOSITION IV.6: For any firewall $\mathfrak{f} = \langle\, \mathcal{R}_{pre}, \mathbb{A}_{\text{INPUT}}, \mathbb{A}_{\text{FORWARD}}, \mathbb{A}_{\text{OUTPUT}}, \mathcal{R}_{post} \,\rangle$, We have:

$$\begin{aligned} \overset{\text{IN}}{\rightsquigarrow}_{\mathfrak{f}} &= \mathcal{L}\left(\mathbb{A}_{pre} \cap (\Pi_1(\mathbb{A}_{\text{INPUT}}))\right) \\ \overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}} &= \mathcal{L}\left(\Pi_2\left(\Pi_3(\mathbb{A}_{pre}) \cap \Pi_{1,3}(\mathbb{A}_{\text{FORWARD}}) \cap \Pi_1(\mathbb{A}_{post})\right)\right) \\ \overset{\text{OUT}}{\rightsquigarrow}_{\mathfrak{f}} &= \mathcal{L}\left(\Pi_2(\mathbb{A}_{\text{OUTPUT}}) \cap \mathbb{A}_{post}\right) \end{aligned}$$

The previous proposition states that the semantics of any firewall is a tuple of recognizable sets, which allows us to perform analysis and comparison, as we will see in the next section.

V. Analysis of firewalls

Many recent works focus their activities on analysing firewall policies. Two main categories of analysis come out: structural analysis and query analysis. Structural analysis examine the relationships that rules have with others rules within a firewall configuration or across multiple firewall. A misconfiguration (or conflict) occurs when several rules match the same packet or when a rule can be removed without changing the behaviour of the firewall. Query analysis provides a way to ask questions of the form "Which computers in the private network can receive packets from `www.inria.fr`?". It then consists in defining a language to describe a firewall query and a way to compute its solutions. As said before, many works [1], [2], [3], [4], [5], [6], [7], [8], [9] looked into structural analysis and others [10], [11], [12], [13] query analysis. All these works presents at least one of the following drawback: not dealing with both problems in the same formalism, not supporting network address translation and using a formalism which do not provide an easy way to compare several firewalls. In this section, we show that our framework allow us to easily compare firewalls and how to perform structural and query analysis.

We first define comparison relations over firewalls.

Definition V.7: (Firewall comparators) Given two firewalls $\mathfrak{f}$ and $\mathfrak{f}'$, we say that:

- $\mathfrak{f}$ and $\mathfrak{f}'$ are *equivalent* if they have the same semantics;
- $\mathfrak{f}$ is *more permissive* than $\mathfrak{f}'$ if for any chain CHAIN, $\overset{\text{CHAIN}}{\rightsquigarrow}_{\mathfrak{f}} \supseteq \overset{\text{CHAIN}}{\rightsquigarrow}_{\mathfrak{f}'}$.

From the recognizability of the semantics of firewalls leads the decidability of equivalence and of the relation "being more permissive".

A. Structural analysis

[2], [28] present several main misconfigurations that a structural analysis must detect.

- intrapolicy conflicts: shadowing, redundancy, correlation, exception,...
- interpolicy conflicts: shadowing, redundancy, spuriousness, irrelevance,...

Such misconfigurations are properties expressed as relationships that rules have with other rules. We claim that our framework allow us to deal with these misconfigurations but it would be very tedious and not relevant to cover all these anomalies in this paper. That is why we handle here two examples of usual misconfigurations (intrapolicy shadowing and redundancy) and assure that others misconfigurations are treated in the same way. We will then show that our framework provides a simple way to analyse some other interpolicy anomalies.

1. Intrapolicy analysis

Definition V.8: (Shadowing) We say that a firewall has *shadowing* iff it contains at least one filtering rule such that all packet it accepts (resp. drops) are dropped (resp. accepted) by a prior rule. In such a case, the concerned rule is said to be *shadowed*.

For any chain CHAIN, the ith rule $\langle\, set_i, decision_i \,\rangle$ is shadowed iff the following automaton is recognize the empty language (set operators must be seen as operators over tree automata):

$$\mathbb{A}_{shadow_i} = rec(set_i) \setminus \left(\bigcup_{\substack{k=1 \\ decision_k \neq decision_i}}^{i-1} rec(set_k) \right)$$

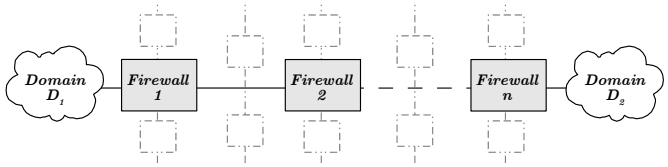
Since the emptiness of tree automata is decidable, testing shadowing consists in computing $\mathbb{A}_{shadow_i}$ for any rule and testing its emptiness.

Definition V.9: (Redundancy) We say that a firewall has *redundancy* iff it contains at least one filtering rule which is not shadowed by any other rule but which can be removed without changing the filtering result.

This definition of redundancy is relevant only if we consider only packet filtering. Indeed, addresses translation can make a rule useless whithout any redundancy. That is why we rather say *uselessness* instead of redundancy when including NAT functionality. This being so, since the equivalence of firewall is decidable, it is sufficient for checking the uselessness of a filtering rule r to compute the semantics of $\mathfrak{f}_{[r]}$ where $\mathfrak{f}_{[r]}$ is the firewall in which the rule r has been removed and to verify if $[\![\mathfrak{f}_{[r]}]\!] = [\![\mathfrak{f}]\!]$. However, we can see that this is not necessary. Indeed, a rule r is uselessness if either the union of prior rules matches all packets which can be matched by r or when r matches packet that the prerouting transform into another packet. The first case is equivalent to check for non shadowed rules $\langle\, set_i, decision_i \,\rangle$ if the set $rec(set_i) \setminus \left(\bigcup_{k=1}^{i-1} rec(set_k) \right)$ is empty. For the second case, it is sufficient to compute the set of packets which can be at the output of the prerouting process. Since it is the set of packets pkt' such that there exists another packet pkt such as $\langle\, pkt, pkt' \,\rangle \in \mathbb{A}_{pre}$, this set is recognized by $\Pi_1(\mathbb{A}_{pre})$. Thus, any non shadowed rule $\langle\, set_i, decision_i \,\rangle$ such that $rec(set_i) \setminus \left(\bigcup_{k=1}^{i-1} rec(set_k) \right) \neq \varnothing$ is *bypassed* by the prerouting process iff the automaton $\Pi_1(\mathbb{A}_{pre}) \cap rec(set_i) \setminus \left(\bigcup_{k=1}^{i-1} rec(set_k) \right)$ recognizes the empty set.

2. Interpolicy analysis

We are now interested in analysing sequences of firewalls in series. Thus, we focuse our attention on traversal flows, in other terms, we study the part of firewalls consisting in the sequence of prerouting, FORWARD chain filtering and postrouting and analyse compositions of such sequences. Let us consider the problem of analysis traffic between two domains separated by n firewalls:



As the figure let us guess, we do not care the flows of packets which do not come from D_1 and whose destination is not D_2. Thus, all following definitions have to be considered with respect to flows from D_1 to D_2. Moreover, to study interpolicy anomalies, we need to give a semantics to a sequence of firewalls:

DEFINITION V.10: (SEMANTICS OF FIREWALL SEQUENCES) Given n firewalls $\mathfrak{f}_1, \ldots, \mathfrak{f}_n$, the semantics of the sequence $\langle \mathfrak{f}_1, \ldots, \mathfrak{f}_n \rangle$ from D_1 to D_2 is the relation:

$$[\![\mathfrak{f}_1 \oplus \ldots \oplus \mathfrak{f}_n]\!]_{D_1 \to D_2} = \overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}_1,\ldots,\mathfrak{f}_n / D_1 \to D_2}$$

such that:

$$\begin{cases} \overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}_1,\ldots,\mathfrak{f}_n / D_1 \to D_2} \\ \qquad = \mathcal{L}(\mathbb{A}^{\text{FORWARD}}_{\mathfrak{f}_1,\ldots,\mathfrak{f}_n} \cap \ rec(D_1 \to D_2) \times \Omega^1_{Packet}) \\ \mathbb{A}^{\text{FORWARD}}_{\mathfrak{f}_1,\ldots,\mathfrak{f}_n} = \Pi_2 \left(\Pi_3 \left(\mathbb{A}^{\text{FORWARD}}_{\mathfrak{f}_1,\ldots,\mathfrak{f}_{n-1}} \right) \cap \Pi_1 \left(\mathbb{A}^{\text{FORWARD}}_{\mathfrak{f}_n} \right) \right) \\ \mathbb{A}^{\text{FORWARD}}_{\mathfrak{f}_i} = \Pi_2 \left(\Pi_3(\mathbb{A}_{pre}) \cap \Pi_{1,3}(\mathbb{A}_{\text{FORWARD}}) \cap \Pi_1(\mathbb{A}_{post}) \right) \end{cases}$$

where $\mathfrak{f}_i = \langle \mathcal{R}_{pre}, \mathbb{A}_{\text{INPUT}}, \mathbb{A}_{\text{FORWARD}}, \mathbb{A}_{\text{OUTPUT}}, \mathcal{R}_{post} \rangle$ and $rec(D_1 \to D_2)$ is the automaton recognizing packets whose source address belongs to D_1 and destination address belongs to D_2.

The relation $\overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}_1,\ldots,\mathfrak{f}_n / D_1 \to D_2}$ corresponds to a function which associates to any packet whose origin belongs to D_1 and whose destination belongs to D_2 the packet under which it will be delivered after the traversal of the sequence of firewalls $\mathfrak{f}_1, \ldots, \mathfrak{f}_n$. A packet has no image if it is dropped during its traversal or redirected toward an other domain. Thus, it is very easy to check (since equivalence between recognizable tree languages is decidable) if severals sequences of firewalls have the same behaviour, $i.e.$ generate the same set of allowed traffics between D_1 and D_2. One can see that we can easily check if a firewall $\mathfrak{f}_i$ is useless with respect to the sequence $\langle \mathfrak{f}_1, \ldots, \mathfrak{f}_n \rangle$ from D_1 to D_2. It suffices to compute the semantics of the sequence whithout $\mathfrak{f}_i$ and to check its equivalence with $[\![\mathfrak{f}_1 \oplus \ldots \oplus \mathfrak{f}_n]\!]_{D_1 \to D_2}$.

B. Query analysis

We show in this subsection that tree automata provide a way to perform query analysis as done in [10], [11], [12], [13]. Indeed, [19] investigated the computation of the solutions of some first order formulae when predicates denote recognizables relations. The following proposition is a consequence of results presented in [19]:

PROPOSITION V.11: Let be $\mathcal{P}$ a set of predicates, $\mathcal{F}$ a set of function symbols and $\mathcal{X}$ a set of variables. Given a first order formula φ over $\langle \mathcal{P}, \mathcal{F}, \mathcal{X} \rangle$ containing no atom $p(t_1, \ldots, t_n)$ such that there are two positions $\omega <_{pref} \omega'$ and two terms t_i and t_j with $t_i(\omega) \in \mathcal{X}$ and $t_j(\omega') \in \mathcal{X}$ ($i.e.$ no atom in which a variable occurs at a deeper position than another variable), if any predicate p of $\mathcal{P}$ denotes a recognizable set or relation, then there is an algorithm rewriting φ into a tree automaton $\mathbb{A}_\varphi$ such that $\langle t_1, \ldots, t_n \rangle \in \mathbb{A}_\varphi$ iff $\sigma(\varphi)$ is true for $\sigma = \{x_i \mapsto t_i \mid 1 \leq i \leq n\}$ where $x_1, \ldots, x_n$ are the free variables of φ.

Furthermore, we have shown in previous sections that relations representing allowed flows generated by firewalls are recognizable. Thus, any query which could be expressed as a first order formula built from

- variables, ground terms of $\Sigma_{Firewall}$ or terms whose head is the functional symbol $packet$ and whose subterms are variables or ground terms and
- predicates $\overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}}$, $\overset{\text{IN}}{\rightsquigarrow}_{\mathfrak{f}}$, $\overset{\text{OUT}}{\rightsquigarrow}_{\mathfrak{f}}$

can be rewritten into a tree automaton recognizing the set of solutions of the query, namely the values which can take free variables of the formula.

EXAMPLE V.12: Let us consider a firewall $\mathfrak{f}$ between a private network and the internet together with the following query "Who can access to `www.inria.fr`" ? First of all, we need to express this query in a formal way: Then, the query becomes: $\varphi := \overrightarrow{\forall x_1, x_2, x_3, \exists y, packet(x_{ip}, x_1, x_2, x_3, 80)} \overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}} packet(y, \overrightarrow{138.96.146.2}, tcp, y_1, 80) \wedge \exists y_1, w_1, w_2, packet(\overrightarrow{138.96.146.2}, y, tcp, 80, y_1) \overset{\text{FWD}}{\rightsquigarrow}_{\mathfrak{f}} packet(w_1, x_{ip}, tcp, w_2, 80)$ where 138.96.146.2 is the IP address of `www.inria.fr` and x_{ip} the only one free variable of the formula (the variable whose values must be computed). The algorithm described in APPENDIX.C transforms this formula into a tree automaton $\mathbb{A}_\varphi$ recognizing the values of x_{ip} satisfying φ.

VI. CONCLUSION

We have proposed in this paper an original way to specify firewalls using tree automata. We have shown that this framework allows us to perform usual analysis over firewall policies. Some advantages of our approach are the following: First, unlike most of existing works, tree automata allow us to consider the network address translation functionality. Second, we can perform with the same formalism both structural and query analysis. Third, contrary to decision diagrams, tree automata provide an easy way to compare firewalls or sequences of firewalls. We currently work on two follows-up of the results presented in this paper. The first one concerns composition of firewalls: we try to extend the specification of firewalls to take into account the routing fonctionality. It would allow us to define the semantics of more complex compositions of firewalls ($i.e.$ not in series). The second possible progress is the development of an algorithm which build a firewall configuration (such as an iptable file) from a semantics of firewall, which would allow us to address the problem of minimalizing the set of rules of a given firewall, $i.e.$ build from a firewall another equivalent one with a minimum of filtering and NAT rules, and to reduce the number of firewalls in a network.

ACKNOWLEDGMENT

We would like to thank Jean-Christophe Bach for discussion about firewalls and Horatiu Cirstea for its improvement suggestions. This work was supported by ANR SSURF and région Lorraine.

REFERENCES

[1] E. Al-shaer and H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *In 23rd IEEE International Conference on Computer Communications*. IEEE Computer Society, 2004, pp. 2605–2616.

[2] F. Cuppens, N. Cuppens-Boulahia, and J. Garcia Alfaro, "Detection of network security component misconfiguration by rewriting and correlation," in *1st joint conference on Security in network ARchitectures and Security of Information Systems*, 2006.

[3] T. Abbes, A. Bouhoula, and M. Rusinowitch, "An inference system for detecting firewall filtering rules anomalies," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2008, pp. 2122–2128.

[4] E. Al-Shaer and H. Hamed, "Firewall policy advisor for anomaly detection and rule editing," in *Proc. IEEE/IFIP 8th Int. Symp. Integrated Network Management*, ser. IFIP Conference Proceedings, vol. 246. Kluwer, 2003, pp. 17–30.

[5] A. Benelbahri and A. Bouhoula, "Tuple based approach for anomalies detection within firewall filtering rules," in *12th IEEE Symposium on Computers and Communications*. IEEE Computer Society, 2007, pp. 63–70.

[6] M. Gouda and A. Liu, "Firewall design: Consistency, completeness, and compactness," in *24th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, 2004.

[7] F. Cuppens, N. Cuppens-Boulahia, and J. Garcia-Alfaro, "Detection and removal of firewall misconfiguration," in *2nd International Conference on Communication, Network and Information Security*, M. Hamza, Ed. ACTA Press, 2005.

[8] A. Liu, "Formal verification of firewall policies," in *IEEE International Conference on Communications (ICC)*. IEEE Computer Society, 2008, pp. 1494 – 1498.

[9] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," in *IEEE Journal on Selected Areas in Communications*, vol. 23, 2005, pp. 2069 – 2084.

[10] S. Hazelhurst, "Algorithms for analysing firewall and router access lists," Department of Computer Science, University of the Witwatersrand, South Africa, Tech. Rep. TR-WITS-CS-1999-5, 2000.

[11] P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," in *6th Nordic Workshop on Secure IT Systems*, 2001, pp. 100–107.

[12] A. X. Liu and M. G. Gouda, "Firewall policy queries," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 20, no. 6, pp. 766–777, 2009.

[13] R. Marmorstein and P. Kearns, "An open source solution for testing NAT'd and nested iptables firewalls," in *LISA '05: Proceedings of the 19th conference on Large Installation System Administration Conference*. Berkeley, CA, USA: USENIX Association, 2005, pp. 103–112.

[14] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, "A formal approach to specify and deploy a network security policy," in *Formal Aspects in Security and Trust*, ser. Lecture Notes In Computer Science, vol. 173. Springer-Verlag, 2004, pp. 203–218.

[15] H. Hamdi, M. Mosbah, and A. Bouhoula, "A domain specific language for securing distributed systems," in *ICSNC '07: Proceedings of the Second International Conference on Systems and Networks Communications*. Washington, DC, USA: IEEE Computer Society, 2007, p. 76.

[16] T. Genet and F. Klay, "Rewriting for cryptographic protocol verification," in *CADE-17: Proceedings of the 17th International Conference on Automated Deduction*, ser. Lecture Notes In Computer Science, vol. 1831. London, UK: Springer-Verlag, 2000, pp. 271–290.

[17] H. Hosoya and B. Pierce, "Regular expression pattern matching for xml," *ACM SIGPLAN Notices*, vol. 36, no. 3, pp. 67–80, 2001.

[18] H. Comon and Y. Jurski, "Higher-order matching and tree automata," in *Computer Science Logic*, ser. Lecture Notes In Computer Science, vol. 1414. Springer-Verlag, 1997, pp. 157–176.

[19] T. Bourdier and H. Cirstea, "Constrained rewriting in recognizable theories," http://hal.inria.fr/inria-00456848/en/, 2010, submitted.

[20] T. Fruhwirth, E. Shapiro, M. Vardi, and E. Yardeni, "Logic programs as types for logic programs," in *IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, 1991, pp. 300–309.

[21] S. Tison, "Tree automata and term rewrite systems," in *RTA 2000: Rewriting Techniques and Applications*, ser. Lecture Notes in Computer Science, vol. 1833. Springer-Verlag, 2000, pp. 27–30.

[22] F. Baader and T. Nipkow, *Term rewriting and all that*. New York, NY, USA: Cambridge University Press, 1998.

[23] C. Kirchner and H. Kirchner, "Rewriting solving proving," Preliminary version of a book, 2006, available on: http://www.loria.fr/~hkirchne/rsp.pdf.

[24] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi, "Tree automata techniques and applications," Available on: http://www.grappa.univ-lille3.fr/tata, 2008, release November, 18th 2008.

[25] F. Jacquemard, "Automates d'arbres et réécriture de termes," Ph.D. dissertation, Université de Paris-Sud, UFR scientifique d'Orsay, 1996.

[26] R. Russell, "Linux 2.4 packet filtering howto," Available on: http://www.netfilter.org/documentation, 2002.

[27] B. Conoboy and E. Fictner, "Ip filter based firewalls howto," Available on: http://www.obfuscation.org/ipf/ipf-howto.pdf, 2002.

[28] H. Hamed and E. Al-shaer, "Taxonomy of conflicts in network security policies," in *In IEEE Communications Magazine*, 2006, pp. 134–141.

APPENDIX

A. Arithmetic on binary numbers

$\mathbb{A}_{bin\_add} = \{(\Sigma_{firewall} \cup \{\Lambda\})^3, Q, \{q_0\}, \Delta_{Add}\}$ where $Q = \{q_0, q_1, q_{000}, q_{001}, q_{100}, q_{101}, q_{010}, q_{011}, q_{110}, q_{111}\}$ and

$$\Delta_{Add} = \begin{cases} \langle 0,0,0 \rangle & \to & q_{000} \\ \langle 0,0,1 \rangle & \to & q_{001} \\ \langle 0,1,0 \rangle & \to & q_{010} \\ \langle 0,1,1 \rangle & \to & q_{011} \\ \langle 1,0,0 \rangle & \to & q_{100} \\ \langle 1,0,1 \rangle & \to & q_{101} \\ \langle 1,1,0 \rangle & \to & q_{110} \\ \langle 1,1,1 \rangle & \to & q_{111} \\ \langle bit,bit,bit \rangle (q_{000}) & \to & q_0 \\ \langle bit,bit,bit \rangle (q_{101}) & \to & q_0 \\ \langle bit,bit,bit \rangle (q_{011}) & \to & q_0 \\ \langle bit,bit,bit \rangle (q_{110}) & \to & q_1 \\ \langle bin,bin,bin \rangle (q_{000}, q_0) & \to & q_0 \\ \langle bin,bin,bin \rangle (q_{001}, q_1) & \to & q_0 \\ \langle bin,bin,bin \rangle (q_{011}, q_0) & \to & q_0 \\ \langle bin,bin,bin \rangle (q_{010}, q_1) & \to & q_1 \\ \langle bin,bin,bin \rangle (q_{101}, q_0) & \to & q_0 \\ \langle bin,bin,bin \rangle (q_{100}, q_1) & \to & q_1 \\ \langle bin,bin,bin \rangle (q_{110}, q_0) & \to & q_1 \\ \langle bin,bin,bin \rangle (q_{111}, q_1) & \to & q_1 \end{cases}$$

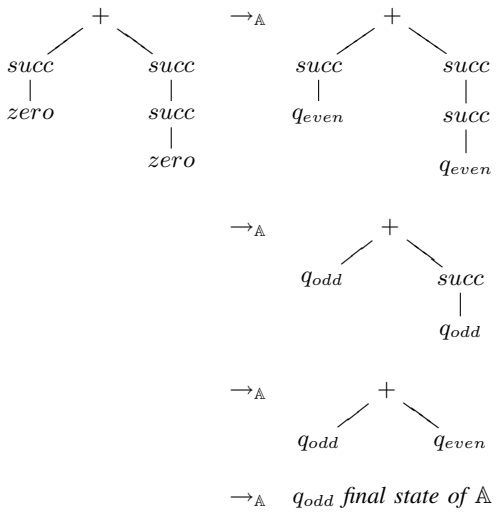
- $\mathbb{A}_{bin\_inf} = \Pi_2(\mathbb{A}_{bin\_add})$
- $\mathbb{A}_{octet\_inf} = \Pi_2(\mathbb{A}_{bin\_add} \cap \Omega^3_{octet})$

B. Examples of tree automata

Let be Σ the signature containing $+ : Nat \mapsto Nat$, $zero :\mapsto Nat$ and $s : Nat \mapsto Nat$. The following unary automaton $\mathbb{A} = (\Sigma, \{q_{odd}, q_{even}\}, \{q_{odd}\}, \Delta)$ where Δ consists of:

$$\begin{cases} zero \to q_{even} \\ succ(q_{even}) \to q_{odd} \\ succ(q_{odd}) \to q_{even} \\ +(q_{odd}, q_{odd}) \to q_{even} \\ +(q_{even}, q_{even}) \to q_{even} \\ +(q_{odd}, q_{even}) \to q_{odd} \\ +(q_{even}, q_{odd}) \to q_{odd} \end{cases}$$

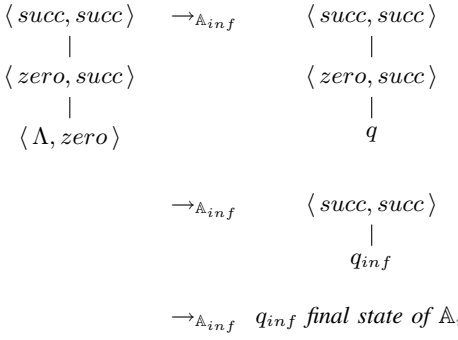
recognizes the set of odd numbers in peano representation. The term $succ(zero) + succ(succ(zero))$ is in $\mathcal{L}(\mathbb{A})$:



An example of binary automaton is given by the following which recognize the set of tuples $\langle n_1, n_2 \rangle$ such that $n_1 \leq n_2$: $\mathbb{A}_{inf} = ((\Sigma \cup \{\Lambda\})^2, \{q, q_{inf}\}, \{q_{inf}\}, \Delta)$ where Δ consists of:

$$\begin{cases} \langle succ, succ \rangle (q_{inf}) & \to & q_{inf} \\ \langle zero, succ \rangle (q) & \to & q_{inf} \\ \langle \Lambda, succ \rangle (q) & \to & q \\ \langle \Lambda, zero \rangle & \to & q \end{cases}$$

$\langle succ(zero), succ(succ(zero)) \rangle$ is recognized by $\mathbb{A}_{inf}$:



C. Transformation algorithm

$$(N_1)\quad (Q_i x_i)_{i=1}^{q}.\exists y.\ \bigwedge_{i=1}^{n}\bigvee_{j=1}^{m_i} a_{i,j} \ \rightarrow\ (Q_i x_i)_{i=1}^{q}.\exists y.\ \bigvee_{(i_1,\ldots,i_n)\in\prod_{k=1}^{n}[1,m_k]}\ \bigwedge_{j=1}^{n} a_{j,i_j}$$

$$(N_2)\quad (Q_i x_i)_{i=1}^{q}.\forall y.\ \bigvee_{i=1}^{n}\bigwedge_{j=1}^{m_i} a_{i,j} \ \rightarrow\ (Q_i x_i)_{i=1}^{q}.\forall y.\ \bigwedge_{(i_1,\ldots,i_n)\in\prod_{k=1}^{n}[1,m_k]}\ \bigvee_{j=1}^{n} a_{j,i_j}$$

$$(Q_1)\quad (Q_i x_i)_{i=1}^{q}.\exists y.\ \bigvee_{i=1}^{n}\bigwedge_{j=1}^{m_i} a_{i,j} \ \rightarrow\ (Q_i x_i)_{i=1}^{q}.\ \bigvee_{i=1}^{n}\exists y.\bigwedge_{j=1}^{m_i} a_{i,j}$$

$$(Q_2)\quad (Q_i x_i)_{i=1}^{q}.\forall y.\ \bigwedge_{i=1}^{n}\bigvee_{j=1}^{m_i} a_{i,j} \ \rightarrow\ (Q_i x_i)_{i=1}^{q}.\ \bigwedge_{i=1}^{n}\forall y.\bigvee_{j=1}^{m_i} a_{i,j}$$

$$(U_1)\quad \forall y.\bigvee_{j=1}^{m} a_j \ \rightarrow\ \neg\exists y.\bigwedge_{j=1}^{m}\neg a_j \qquad \textit{if } y \textit{ occurs in at least one } a_j$$

$$(U_2)\quad \forall y.\bigvee_{j=1}^{m} a_j \ \rightarrow\ \bigvee_{j=1}^{m} a_j \qquad \textit{if } y \textit{ occurs in no } a_j$$

$$(E)\quad \exists y.\bigwedge_{j=1}^{m} a_j \ \rightarrow\ \bigwedge_{j=1}^{m} a_j \qquad \textit{if } y \textit{ occurs in no } a_j$$

$$(P)\quad \exists y.(\exists y_k)_{k=1}^{m}.p(x_1,\ldots,x_{i-1},y,x_{i+1},\ldots,x_n)\wedge\psi \ \rightarrow\ (\exists y_k)_{k=1}^{m}.\Pi_i(p)(x_1,\ldots,x_n)\wedge\psi \quad \textit{if } y\notin\{x_1,\ldots,x_n\}\cup\mathcal{FV}ar(\psi)$$

$$(C)\quad \neg p(t_1,\ldots,t_n) \ \rightarrow\ \overline{p}(t_1,\ldots,t_n)$$

$$(M_1)\quad p(x_1,\ldots,x_n)\wedge q(x_1,\ldots,x_n) \ \rightarrow\ (p\cap q)(x_1,\ldots,x_n)$$
$$(M_2)\quad p(x_1,\ldots,x_n)\vee q(x_1,\ldots,x_n) \ \rightarrow\ (p\cup q)(x_1,\ldots,x_n)$$

$$(O_1)\quad p(x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_n) \ \rightarrow\ \mathfrak{S}_{i,i+1}(p)(x_1,\ldots,x_{i-1},x_{i+1},x_i,\ldots,x_n) \quad \textit{if } x_i > x_{i+1}$$

$$(G_1)\quad p(x_1,\ldots,x_n)\ \wedge\ q(y_1,\ldots,y_m) \ \rightarrow\ \Pi_k(p)(x_1,\ldots,x_{k-1},y,x_k,\ldots,x_n)\ \wedge\ q(y_1,\ldots,y_m)$$
$$\textit{if } \{x_1,\ldots,x_n\}\cap\{y_1,\ldots,y_m\}\neq\varnothing \textit{ and } y\in\{y_1,\ldots,y_m\}\setminus\{x_1,\ldots,x_n\} \textit{ and } x_{k-1}<y<x_k$$

$$(G_2)\quad p(x_1,\ldots,x_n)\wedge\psi\ \wedge\ q(y_1,\ldots,y_m) \ \rightarrow\ \Pi_k(p)(x_1,\ldots,x_{k-1},y,x_k,\ldots,x_n)\ \wedge\ q(y_1,\ldots,y_m)\wedge\psi$$
$$\textit{if } \{x_1,\ldots,x_n\}\cap\{y_1,\ldots,y_m\}=\varnothing \textit{ and } y\in\{y_1,\ldots,y_m\} \textit{ and } x_{k-1}<y<x_k \textit{ and } \psi \textit{ contains no equality}$$

$$(G_3)\quad p(x_1,\ldots,x_n)\ \vee\ q(y_1,\ldots,y_m) \ \rightarrow\ \Pi_k(p)(x_1,\ldots,x_{k-1},y,x_k,\ldots,x_n)\ \vee\ q(y_1,\ldots,y_m)$$
$$\textit{if } y\in\{y_1,\ldots,y_m\}\setminus\{x_1,\ldots,x_n\} \textit{ and } x_{k-1}<y<x_k$$

$$(L_1)\quad p(y_1,\ldots,y,\ldots,y,\ldots,y_n) \ \rightarrow\ \Pi_j(p\cap\Pi_{[1,n]\setminus\{i,j\}}(Id^2_\kappa))(y_1,\ldots,y,\ldots,y_n)$$
$$\textit{if } y \textit{ occurs in positions } i \textit{ and } j \textit{ in } p(\ldots,y,\ldots,y,\ldots) \textit{ and } y \textit{ is sorted by } \kappa$$

$$(T_1)\quad t=t' \ \rightarrow\ Id^2_\kappa(t,t') \qquad \textit{if } \kappa \textit{ is the sort of } t \textit{ and } t'$$

$$(L_2)\quad p(\ldots,f(\ldots,y,\ldots,y,\ldots),\ldots) \ \rightarrow\ \exists z.p(\ldots,f(\ldots,y,\ldots,z,\ldots),\ldots)\wedge Id^2_\kappa(y,z) \quad \textit{if } y \textit{ is sorted by } \kappa$$

$$(D)\quad p(f_1(t^1_1,\ldots,t^{m_1}_1),\ldots,f_n(t^1_n,\ldots,t^{m_n}_n)) \ \rightarrow\ \textstyle\bigwedge_{i=1}^{\max_k(m_k)}\partial^i_{\langle f_1,\ldots,f_n\rangle}(p)(t^1_i,\ldots,t^m_i)$$

$$(R)\quad p(t_1,\ldots,t_{i-1},t,t_{i+1},\ldots,t_n) \ \rightarrow\ \Pi_{t/i}(p)(t_1,\ldots,t_{i-1},t_{i+1},\ldots,t_n) \quad \textit{if } t\in\mathcal{T}_\Sigma$$

$$(T_2)\quad p(\ldots,t_{i-1},t,t_{i+1},\ldots) \ \rightarrow\ \Pi_i(p\cap\Pi_{[1,n]\setminus\{i\}}(rec\,(t)))(\ldots,t_{i-1},t_{i+1},\ldots)$$
$$(T_3)\quad \exists y.p(\ldots,t,\ldots)\wedge\psi \ \rightarrow\ p(\ldots,\sigma(t),\ldots)\wedge\psi \quad \textit{if } y\notin\mathcal{FV}ar(\psi),\ \sigma=\{y\mapsto\Omega\} \textit{ and } y \textit{ occurs exactly once in } t$$
$$(T_4)\quad \exists y.p(\ldots,t,\ldots)\wedge q(y)\wedge\psi \ \rightarrow\ p(\ldots,\sigma(t),\ldots)\wedge\psi \quad \textit{if } y\notin\mathcal{F}var(\psi),\ \sigma=\{y\mapsto p\} \textit{ and } y \textit{ occurs exactly once in } t$$

Fig. 4. Transformation rules