Simple Firewalls

Arshad Jhumka

University of Warwick

arshad@dcs.warwick.ac.uk

February 4, 2018

Previously,

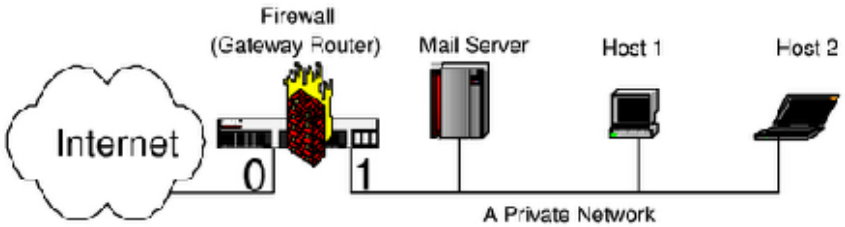
- We have studied how to develop security policies.
  - We have looked at security policies, e.g., Bell-LaPadula, Chinese Wall.
  - We have also studied threat models to drive the development of appropriate security policy.

- We have studied how to enforce security policies through execution monitoring
  - We have looked at monitors, specified in terms of security automata, to stop an execution when the policy is about to be violated.
  - We also looked at target transformers for policy enforcement.

What Now

- We are going to look at a specific target transformer: **Firewalls**.
- **Firewall**: A security guard placed at the point of entry between a private network and the Internet that monitors all incoming and outgoing packets.
  - Firewall consists of set of rules.
  - The decision on whether to forward or drop a message depends on the rule that the message satisfies.
  - However, there are many challenges when designing a firewall.
- Some challenges:
  - May have *conflicting* rules.
  - Set of rules may not be *complete*, i.e., rules for some packets are missing.
  - *Compactness*, i.e., some rules are redundant.

Firewall Example



1. Rule r_1: $(\mathbf{I} = \mathbf{0}) \wedge (\mathbf{S} = \mathbf{any}) \wedge (\mathbf{D} = \mathbf{Mail\ Server}) \wedge (\mathbf{N} = \mathbf{25}) \wedge (\mathbf{P} = \mathbf{tcp}) \rightarrow \mathbf{accept}$
   (This rule allows incoming SMTP packets to proceed to the mail server.)
2. Rule r_2: $(\mathbf{I} = \mathbf{0}) \wedge (\mathbf{S} = \mathbf{Malicious\ Hosts}) \wedge (\mathbf{D} = \mathbf{any}) \wedge (\mathbf{N} = \mathbf{any}) \wedge (\mathbf{P} = \mathbf{any}) \rightarrow \mathbf{discard}$
   (This rule discards incoming packets from previously known malicious hosts.)
3. Rule r_3: $(\mathbf{I} = \mathbf{1}) \wedge (\mathbf{S} = \mathbf{any}) \wedge (\mathbf{D} = \mathbf{any}) \wedge (\mathbf{N} = \mathbf{any}) \wedge (\mathbf{P} = \mathbf{any}) \rightarrow \mathbf{accept}$
   (This rule allows any outgoing packet to proceed.)
4. Rule r_4: $(\mathbf{I} = \mathbf{any}) \wedge (\mathbf{S} = \mathbf{any}) \wedge (\mathbf{D} = \mathbf{any}) \wedge (\mathbf{N} = \mathbf{any}) \wedge (\mathbf{P} = \mathbf{any}) \rightarrow \mathbf{accept}$
   (This rule allows any incoming or outgoing packet to proceed.)

Name	Meaning
I	Interface
S	Source IP address
D	Destination IP address
N	Destination port number
P	Protocol type

Consistency and Completeness

- Consistency issue.
- Conflicting rules; several rules can be satisfied but with different outputs.
- Example: rule r1 and r2 conflict since the SMTP packets from previously known malicious hosts to the mail server match both rules and the decisions of r1 and r2 are different

- Completeness issue.
- It is difficult to ensure that all possible packets are considered.
- to block these two types of traffic, the following two rules should be inserted immediately after rule r1 :
  - $(I = 0) \wedge (S = \text{any}) \wedge (D = \text{Mail Server}) \wedge (N = \text{any}) \wedge (P = \text{any}) \rightarrow \text{discard}$
  - $(I = 0) \wedge (S = \text{any}) \wedge (D = \text{any}) \wedge (N = 25) \wedge (P = \text{tcp}) \rightarrow \text{discard}$

Model and Notation

- Packet: n-tuple, $\langle d_1 \ldots d_n \rangle$ of data.
- Field F_i: variable with a non-negative integer domain, denoted by $D(F_i)$, e.g., source IP address $[0,2^{32}-1]$.
- Each data item d_i in packet is such that $d_i \in D(F_i)$.
- Σ : set of all packets over fields $F_1 \ldots F_n$.
- Rule: $\langle$ predicate $\rangle \rightarrow \langle$ decision $\rangle$.
- Predicate: boolean expression over $d_1 \ldots d_n$, decision $\in \{a, d\}$
- Firewall: sequence of rules, $R_1 \ldots R_m$.
- A packet *matches* a rule R_i iff the packet satisfies the predicate of R_i.
- Two rules *overlap* if there is at least one packet that can match both rules.
- Two rules *conflict* iff they overlap and have different decisions.
- When two rules $R_i, R_j, i < j$ conflict, the decision taken is that of R_i.
- The last rule is called the *default* rule; usually, a tautology.

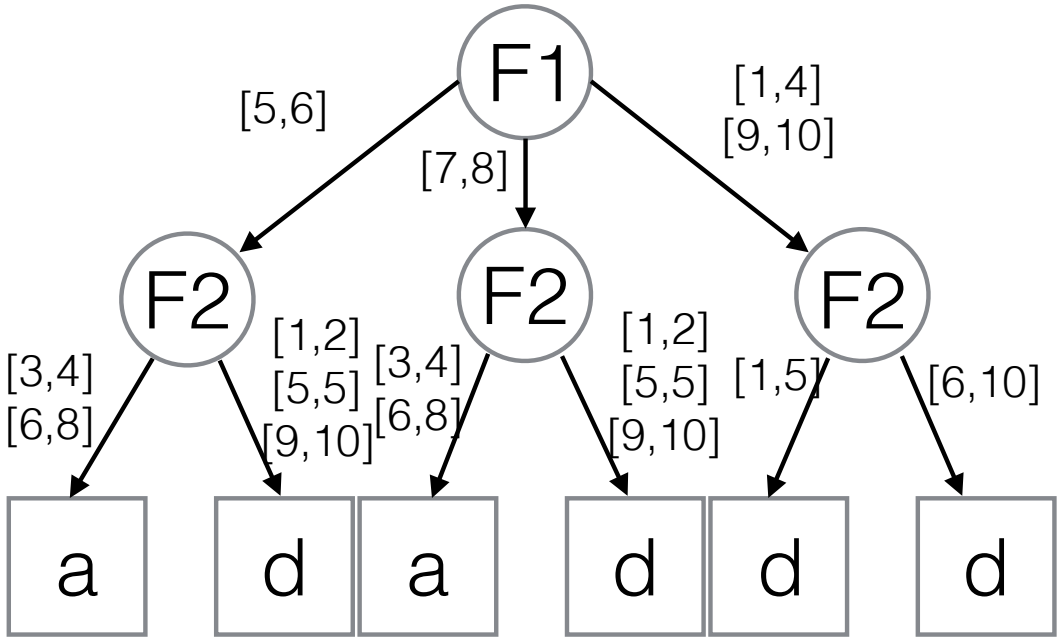
Firewall Decision Diagram (FDD)

- FDD is an *acyclic and directed* graph, defined over $F_1 \ldots F_n$, with 5 properties:
  1. Has exactly one *root*, a node with no incoming edge. Nodes with no outgoing edges are called *terminal* nodes.
  2. Each node v in FDD is labeled with a field, denoted $F(v)$, such that

$$F(v) \in \begin{cases} \{F_1, \ldots, F_n\} & \text{if } v \text{ is non-terminal} \\ \{a, d\} & \text{if } v \text{ is terminal} \end{cases}$$

  3. An edge e is labeled with a non-empty set of integer, denoted $I(e)$, such that if e is an outgoing edge of node v, then we have $I(e) \subseteq D(F(v))$.
  4. A directed path from root to a terminal nodes is called a *decision path*. No two nodes on a decision path have the same label.
  5. The set of all outgoing edges of node v, denoted $E(v)$, satisfies the following two conditions:
     1. *Consistency*: $I(e) \cap I(e') = \emptyset$, for any two distinct edges $e, e' \in E(v)$.
     2. *Completeness*: $\cup_{e \in E(v)} I(e) = D(F(v))$.

Example FDD


F1
[5,6]
[7,8]
[1,4]
[9,10]
F2
F2
F2
[3,4]
[6,8]
[1,2]
[5,5]
[9,10]
[3,4]
[6,8]
[1,2]
[5,5]
[9,10]
[1,5]
[6,10]
a
d
a
d
d
d

FDD and Decision

- FDD maps each packet to a decision by testing packet down the FDD from root to a terminal node.
- Label of the terminal node represents the decision for the packet.
- Each non-terminal node specifies a test of a packet field.
- Each outgoing edge from a non-terminal node corresponds to some values of that field.
- An edge is selected when the edge label contains the value of the packet field.
- The selection process is repeated at the new node, until a terminal node is reached.

FDD Traversal

Example (Traversing FDD for Decision)

```
Input: FDD, packet
output: decision

current := root;
while (label(current) != a or d) do {
  forall edge ((current, j),value) do {
     if (data is contained in value) then
        current := j;
     endif
     } od;
 } end while
 return(label(current));
```

FDD and Decision

- *Decision path*: Represented by $\langle v_1 e_1 \ldots v_k e_k v_{k+1} \rangle$, where v_1 is the root and v_{k+1} is a terminal node and each edge e_i is a directed arc from node v_i to node v_{i+1}.
- A decision path $\langle v_1 e_1 \ldots v_k e_k v_{k+1} \rangle$ represents the following rule: $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \langle$ decision $\rangle$, where *decision* is the label of the terminal node v_{k+1} in the path and

$$S_i = \begin{cases} I(e_j) & \text{if } \exists \text{ node } v_j \text{ in decision path with labeled field } F_i \\ D(F_i) & \text{otherwise} \end{cases}$$

- Each path (together with labels) in FDD represents a rule in the firewall.
- There is only *one* rule that any packet p will match in an FDD.

Example FDD and Firewall Rules

- Continue using existing FDD, firewall rules are:
  - R1. $F_1 \in [5,6] \wedge F_2 \in [3,4] \cup [6,8] \rightarrow a$
  - R2. $F_1 \in [5,6] \wedge F_2 \in [1,2] \cup [5,5] \cup [9,10] \rightarrow d$
  - R3. $F_1 \in [7,8] \wedge F_2 \in [3,4] \cup [6,8] \rightarrow a$
  - R4. $F_1 \in [7,8] \wedge F_2 \in [1,2] \cup [5,5] \cup [9,10] \rightarrow d$
  - R5. $F_1 \in [1,4] \cup [9,10] \wedge F_2 \in [1,5] \rightarrow d$
  - R6. $F_1 \in [1,4] \cup [9,10] \wedge F_2 \in [6,10] \rightarrow d$

Important result for FDDs

- For an FDD f, $f.accept$ is the set of all packets accepted by f.
- For an FDD f, $f.discard$ is the set of all packets discarded by f.
- These two sets precisely define the semantics of the FDD.

Theorem (Theorem of FDDs)

For any FDD f, we have

1. $f.accept \cap f.discard = \emptyset$.
2. $f.accept \cup f.discard = \Sigma$.

- Two FDDs f and f' are *equivalent* iff (i) $f.accept = f'.accept$ and (ii) $f.discard = f'.discard$.

FDD Reduction

- The greater the number of decision path in an FDD, the greater the number of rules for the firewall.
- Hence, it takes more time for the firewall to arrive at a decision, for any packet.
- Thus, number of decision paths (rules) must be reduced. How to compress the graph?
- Use the concept of *isomorphic nodes*.

Definition (Isomorphic Nodes)

Two nodes v and v' in an FDD are *isomorphic* iff

1. both v and v' are terminal nodes with identical labels.
2. Both v and v' are non-terminal nodes, and there is a 1-1 correspondence between the outgoing edges of v and those of v' such that every pair of corresponding edges have identical labels and they point to the same node.

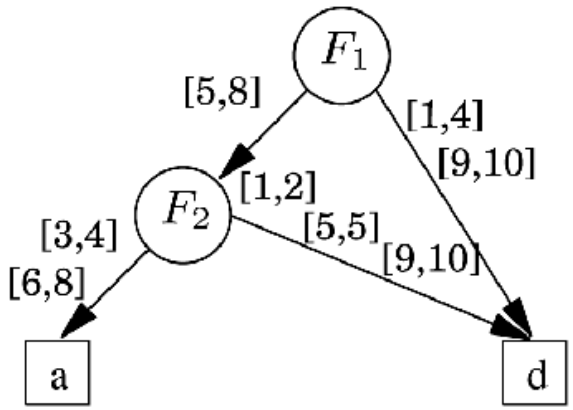
Algorithm 1 - Reduced FDD

An FDD is *reduced* iff it satisfies the following three conditions:

- No node in FDD has only one outgoing edge.
- No two nodes in FDD are isomorphic.
- No two nodes have more than one edge between them.

```
Algorithm for FDD Reduction
Input: FDD f, Output: Reduced FDD equivalent to f

Steps: (apply 1-3 repeatedly until FDD cannot be reduced any further)

1. If there is a node v with only 1 outgoing edge e, e points to v',
remove both v and e, and all edges that point to v point to v'.

2. Two isomorphic edges v and v', remove v' together with all its
outgoing edges, and let all edges that point to v' point to v.

3. Two edges e and e' that are both between the same pair of nodes,
then remove e' and change label of e from I(e) to I(e) U I(e').
```

Reduced FDD - Example (Cont'd)



FDD Marking

- Reduced FDD $\Rightarrow$ Reduced number of decision paths.
- Also, order of rules in firewall is important. How to determine the order?
- FDD Marking; mark which transitions to be considered last.
- $F_1 \in S_1 \wedge \ldots \wedge F_n \in S_n \rightarrow \langle \text{decision} \rangle$ is *simple* iff every $S_i, 1 \leq i \leq n$, is an interval of consecutive non-negative integers.
- Most firewalls require simple rules;hence, minimise number of simple rules generated from FDD.
- Number of simple rules generated from marked FDD $\leq$ Number of simple rules generated from original FDD

Marked FDD

- A *marked* version f' of an FDD f is the same as f except that exactly one outgoing edge of each non-terminal node if f' is marked *"all"*.
- FDD f and f' are equivalent since the label of edges marked "all" do not change.
- FDD f' is called a *marked* FDD.

Minimal Load Marked FDD

- The *load of a non-empty set of integers S*, denoted *load(S)*, is the *minimal number* of non-overlapping integer intervals that cover S.
- Load of a edge e in a marked FDD is defined as follows:

$$load(e) \in \begin{cases} 1 & \text{if } e \text{ is marked } all \\ load(I(e)) & \text{otherwise} \end{cases}$$

- Load of a node v in a marked FDD is defined recursively as follows:

$$load(v) \in \begin{cases} 1 & \text{if } v \text{ is terminal} \\ \sum_{i=1}^{k}(load(e_i) * load(v_i)) & \text{if } v \text{ is non-terminal; suppose } v \text{ has } k \text{ outgoing edges } e_1, \ldots, e_k \text{ which point to nodes } v_1, \ldots, v_k \text{ respectively} \end{cases}$$

Algorithm 2 - Minimal Load Marked FDD

Algorithm M-FDD:
Input: FDD f, Output: marked FDD f' with minimal load
(no the marked FDD f'' has lower load)

Steps:

1. Compute the load of each terminal node v in f as follows: load(v) := 1

2. WHILE (there is a node v whose load has not yet been computed,
suppose v has k outgoing edges e1,..., ek, and these edges point to
v1,...,vk respectively, and the loads of these k nodes have been computed)
DO {

(a) Among the k edges e1,..., ek, choose an edge ej with the largest value of (load(ej) -1) * load(v), and mark edges ej with ''all''.

(b) Compute the load of v as follows:

$load(v) := \sum_{i=1}^{k}(load(e_i) * load(v_i))$

} END WHILE

Marked FDD Remarks

- Load of a marked FDD f, denoted *load(f)*, equals the load of the root of f.
- Marked FDD with smaller load $\Rightarrow$ generation of smaller number of simple rules.

Theorem (Algorithm and Minimal Load FDD)

Given an FDD f, algorithm M-FDD returns a marked FDD f' with minimal load.

From Marked FDD to Firewall Generation

- Semantics of firewall: A packet $(p_1, \ldots, p_n)$ matches a rule $F_1 \in S_1 \wedge \ldots \wedge F_n \in S_n \rightarrow \langle \text{decision} \rangle$ iff the condition $F_1 \in S_1 \wedge \ldots \wedge F_n \in S_n$ holds.
- Firewall consists of a sequence of rules such that, for any packet, there is at least one rule that the packet matches; maps every packet to the decision of the first matching rule.

Theorem (Firewall Theorem)

For a firewall f of a sequence of rules,

1. $f.accept \cap f.discard = \emptyset$
2. $f.accept \cup f.discard = \Sigma$

- For a FDD f and firewall f', $f \equiv f'$ iff $f.accept = f'.accept \wedge f.discard = f'.discard$.

From Marked FDD to Firewall Generation (Informal)

- Start with a marked FDD f, perform a depth-first traversal of f such that for each non-terminal node v, the outgoing edge marked "all" of v is traversed after all other other outgoing edges of v have been traversed.
- Whenever a terminal node encountered, if $v_1 e_1 \ldots v_k e_k v_{k+1}$ is a decision path, output rule r as follows:
  $F_1 \in S_1 \wedge \ldots \wedge F_n \in S_n \rightarrow F(v_{k+1})$, where

$$S_i = \begin{cases} I(e_j) & \text{if the decision has a node } v_j \\ & \text{and that is labeled with field } F_i \\ & \text{and } e_j \text{ is not marked "all"} \\ D(F_i) & \text{otherwise} \end{cases}$$

- Rule represented by $\langle v_1 e_1 \ldots v_k e_k v_{k+1} \rangle$ is $F_1 \in T_1 \wedge \ldots \wedge F_n \in T_n \rightarrow F(v_{k+1})$ where

$$T_i = \begin{cases} I(e_j) & \text{if the decision has a node } v_j \\ & \text{that is labeled with field } F_i \\ D(F_i) & \text{otherwise} \end{cases}$$

- Predicate $F_1 \in S_1 \wedge \ldots \wedge F_n \in S_n$ is called the *matching predicate* of rule.
- Predicate $F_1 \in T_1 \wedge \ldots \wedge F_n \in T_n$ is called the *resolving predicate* of the rule.

Algorithm 3 - Firewall Generation

Input: Marked FDD f

Output: Firewall equivalent to f. For each rule r, r.mp and r.rp is computed.

Steps:

Depth-first traverse f s.t for each nonterminal node v, the outgoing edge marked as "all" of v is traversed after all other outgoing edges of v have been traversed. Whenever a terminal node is encountered, assuming $v_1 e_1 \ldots v_k e_k v_{k+1}$ is the decision path where each $e_i$is the most recently traversed outgoing edge of node v_i, output a rule r together with its matching predicate $r.mp$ and its resolving predicate as follows:

r is the rule $F_1 \in S_1 \wedge \ldots \wedge F_n \in S_n \rightarrow F(v_{k+1})$ where

$$S_i = \begin{cases} I(e_j) & \text{if the decision has a node } v_j \\ & \text{and that is labeled with field } F_i \\ & \text{and } e_j \text{ is not marked "all"} \\ D(F_i) & \text{otherwise} \end{cases}$$

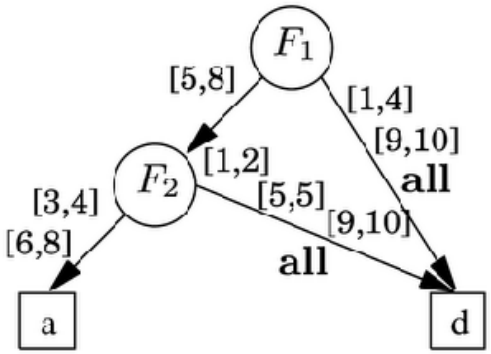
$r.mp$ is the predicate of rule r.

$r.rp$ is the predicate $F_1 \in T_1 \wedge \ldots \wedge F_n \in T_n$, where

$$T_i = \begin{cases} I(e_j) & \text{if the decision has a node } v_j \\ & \text{that is labeled with field } F_i \\ D(F_i) & \text{otherwise} \end{cases}$$

- Firewall may however have redundant rules, i.e., removing rules do not change the semantics of the firewall.
- Removing them results in more efficient firewall. How to identify redundant rules?

Firewall Generation - Example (Cont'd)



$$
\begin{aligned}
r_1 &= F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8] \rightarrow a, \\
r_1.mp &= F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8] \\
r_1.rp &= F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8]
\end{aligned}
$$

$$
\begin{aligned}
r_2 &= F_1 \in [5,8] \wedge F_2 \in [1,10] \rightarrow d, \\
r_2.mp &= F_1 \in [5,8] \wedge F_2 \in [1,10] \\
r_2.rp &= (F_1 \in [5,8] \wedge F_2 \in [1,2] \cup [5,5] \cup [9,10])
\end{aligned}
$$

$$
\begin{aligned}
r_3 &= F_1 \in [1,10] \wedge F_2 \in [1,10] \rightarrow d, \\
r_3.mp &= F_1 \in [1,10] \wedge F_2 \in [1,10] \\
r_3.rp &= F_1 \in [1,4] \cup [9,10] \wedge F_2 \in [1,10]
\end{aligned}
$$

Algorithm 4 - Firewall Compaction

- Present an efficient algorithm for discovering redundant rules.
- Redundant rules are those with same decision but one is implied by the other.

Input: A firewall $\langle r_1, \ldots, r_m \rangle$
Output: An equivalent but more compact firewall.
Steps:

1. **for** i = m to 1 **do**
   redundant[i] := 0;
2. **for** i = m to 1 **do**
   if there exist a rule r_k in the firewall, where $i < k \leq m$, such that the following 4 conditions hold:
   2.1. redundant[k] = false;
   2.2. r_i, r_k have same decisions.
   2.3. $r_i.rp$ implies $r_k.mp$
   2.4. for every rule r_j, where $i < j < k$, at least one of the following three conditions holds:
   2.4.1. redundant[j] = 1.
   2.4.2. r_i, r_j have the same decision.
   2.4.3. no packet satisfies both $r_i.rp$ and $r_j.mp$.
   then redundant[i] := 1
   else redundant[i] := 0
3. **for** i = m to 1 **do**
   if redundant[i] = 1 **then** remove r_i from the firewall.

Firewall Compaction - Example (Cont'd)

$$
\begin{aligned}
r_1 &= F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8] \rightarrow a, \\
r_1.mp &= F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8] \\
r_1.rp &= F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8]
\end{aligned}
$$

$$
\begin{aligned}
r_2 &= F_1 \in [5,8] \wedge F_2 \in [1,10] \rightarrow d, \\
r_2.mp &= F_1 \in [5,8] \wedge F_2 \in [1,10] \\
r_2.rp &= (F_1 \in [5,8] \wedge F_2 \in [1,2] \cup [5,5] \cup [9,10])
\end{aligned}
$$

$$
\begin{aligned}
r_3 &= F_1 \in [1,10] \wedge F_2 \in [1,10] \rightarrow d, \\
r_3.mp &= F_1 \in [1,10] \wedge F_2 \in [1,10] \\
r_3.rp &= F_1 \in [1,4] \cup [9,10] \wedge F_2 \in [1,10]
\end{aligned}
$$

1. $F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8] \rightarrow a,$
2. $F_1 \in [1,10] \wedge F_2 \in [1,10] \rightarrow d$

Algorithm 5 - Firewall Simplification

Input: A (compact) firewall f
Output: A simple firewall f' which is equivalent to f.
Steps:

while f has a rule of the form $F_1 \in S_1 \wedge \ldots \wedge F_i \in S_i \wedge \ldots \wedge F_n \in S_n \rightarrow \langle \text{decision} \rangle$ where some S_i is represented by $[a_1, b_1] \cup \ldots \cup [a_k, b_k]$, $k \geq 2$.
do
replace this rule by the following k non-overlapping rules:
$F_1 \in S_1 \wedge \ldots \wedge F_i \in [a_1, b_1] \wedge \ldots \wedge F_d \in S_d \rightarrow \langle \text{decision} \rangle$
$F_1 \in S_1 \wedge \ldots \wedge F_i \in [a_2, b_2] \wedge \ldots \wedge F_d \in S_d \rightarrow \langle \text{decision} \rangle$
$\vdots$
$F_1 \in S_1 \wedge \ldots \wedge F_i \in [a_k, b_k] \wedge \ldots \wedge F_d \in S_d \rightarrow \langle \text{decision} \rangle$
end

Summary

Showed how to design a simple firewall from a user-specified firewall design diagram.

- **Step 1:** Starting with a user-specified FDD f, algorithm **algorithm 1** transforms it into a reduced FDD f_1.
- **Step 2:** From f_1, algorithm **algorithm 2** transforms it into a marked FDD f_2.
- **Step 3:** From f_2, algorithm **algorithm 3** transforms it into a firewall f_3.
- **Step 4:** **Algorithm 4** transforms f_3 into a compacted firewall f_4.
- **Step 5:** **Algorithm 5** transforms f_4 into a simple firewall f_5.