Title	Representation of Classification Functions by Head-Tail Expressions
Author(s)	Infall, Syafalni
Issue Date	2014-03
URL	http://hdl.handle.net/10228/5316
Rights	

Representation of Classification Functions by Head-Tail Expressions

Infall Syafalni

Abstract

Packet classification is used in various network applications such as firewalls, access control lists, and network address translators. This technology uses ternary content addressable memories (TCAMs) to perform high speed packet forwarding. However, TCAMs dissipate high power and their cost are high. Thus, reduction of TCAMs is crucial.

First, this thesis derives the prefix sum-of-products expression (PreSOP) and the number of products in a PreSOP for an interval function. Second, it derives $\Psi(n,\tau_p)$, the number of n-variable interval functions that can be represented with τ_p products. Finally, it shows that more than 99.9% of the n-variable interval functions can be represented with $\lceil \frac{3}{2}n - 1 \rceil$ products when n is sufficiently large. These results are useful for fast PreSOP generator and for estimating the size of Ternary Content Addressable Memories (TCAMs) for packet classification.

Second, this thesis shows a method to represent interval functions by using head-tail expressions. The head-tail expressions represent *greater-than* $GT(n : A)$ functions, *less-than* $LT(n : B)$ functions, and interval functions $IN_0(n : A, B)$ more efficiently than sum-of-products expressions, where n denotes the number of bits to represent the largest value in the interval (A, B). This paper proves that a head-tail expression (HT) represents an interval function with at most n words in a ternary content addressable memory (TCAM) realization. It also shows the average numbers of factors to represent interval functions by HTs for up to $n = 16$, which were obtained by a computer simulation. It also conjectures that, for sufficiently large n, the average number of factors to represent n-variable interval functions by HTs is at most $\frac{2}{3}n - \frac{5}{9}$. Experimental results also show that, for $n \geq 10$, to represent interval functions, HTs require at least 20% fewer factors than MSOPs, on the average.

Third, this thesis presents a method to generate head-tail expressions for single-field classification functions. First, it introduces a fast prefix sum-of-product (PreSOP) generator (FP) which generates products using the bit patterns of the endpoints. Next, it shows a direct head-tail expression generator (DHT). Experimental results show that

DHT generates much smaller TCAM than FP. The proposed algorithm is useful for simplified TCAM generator for packet classification.

Finally, this thesis shows methods to simplify rules in TCAMs for packet classification. First method, it partitions the rules into groups so that each group has the same source address, destination address and protocol. After that, it simplifies rules in each group by removing redundant rules. A computer program was developed to simplify rules among groups. Experimental results show that this method reduces the size of rules up to 57% of the original specification for ACL5 rules, 73% for ACL3 rules, and 87% for overall rules. This algorithm is useful to reduce TCAMs for packet classification. In the second method, we reduce the number of words in TCAM for multi-field classification functions by using head-tail expressions. It presents MFHT, an $O(r^2)$-algorithm to generate simplified TCAMs for two-field classification functions, where r is the number of rules. Experimental results show that MFHT achieves a 58% reduction of words for random rules and a 52% reduction of words for ACL and FW rules. Moreover, MFHT is fast. The methods are useful for simplifying TCAM for packet classification.

Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

ACL	Access Control List
DA	Destination Address
DHT	Direct Head-Tail Expression
DP	Destination Port
FP	Fast Prefix Sum-of-Products
FPGA	Field Programmable Gate Array
FW	Firewall
HT	Head-Tail Expression
IPv4	Internet Protocol Version 4
LSI	Large Scale Integrated Circuit
MFHT	Multi-Field Head-Tail Expression
MPreSOP	Minimum Prefix Sum-of-Products Expression
MSOP	Minimum Sum-of-Products Expression
PI	Prime Implicant
PO	Protocol
PreSOP	Prefix Sum-of-Products Expression
RAM	Random Access Memory
SA	Source Address
SFHT	Single-Field Head-Tail Expression
SOP	Sum-of-Products Expression
SP	Source Port
TCAM	Ternary Content Addressable Memory

Symbols

f	single-field logic function
F	multi-field logic function
n	the number of variables
m	the number of bits
r	the number of fields
x	variable
$\vec{a}$	vector $(a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$
(A, B)	an open interval
$GT(n : A)$	greater-than function
$LT(n : B)$	less-than function
$IN_0(n : A, B)$	interval function
τ_p	the number of products in PreSOP
τ_m	the number of products in MSOP
τ_h	the number of products in HT
μ	average
σ^2	variance
$\Psi(n,\tau_p)$	the number of n-variable interval functions that require τ_p products in their MPreSOPs
ζ	the minimum number of factors to represent a function f by an HT
S_i	the logic functions represented by products of PreSOPs that show segments of an interval
O	order notation showing the computational complexity

Chapter 1

Introduction

1.1 Background and Purposes of Research

The demand for fast network has been increased with the growth of data transmission and memory technology. One of the core technology in fast network connection is packet classification. Packet classification [6, 11, 23, 45, 47, 57, 62, 63] is a function to map or filter each incoming packet for a decision in an ordered list of rules. Packet classifications have been used in various networking applications such as routers, firewalls (FW) and access control lists (ACL).

A packet classifier is specified by a set of rules. And a rule is specified by fields. A field can be represented by a product or an interval. In internet IPv4, fields are source address (32-bit word), destination address (32-bit word), source port (16-bit interval), destination port (16-bit interval), and protocol (8-bit word). Figure 1.1 illustrates of packet classifiers in various network applications.

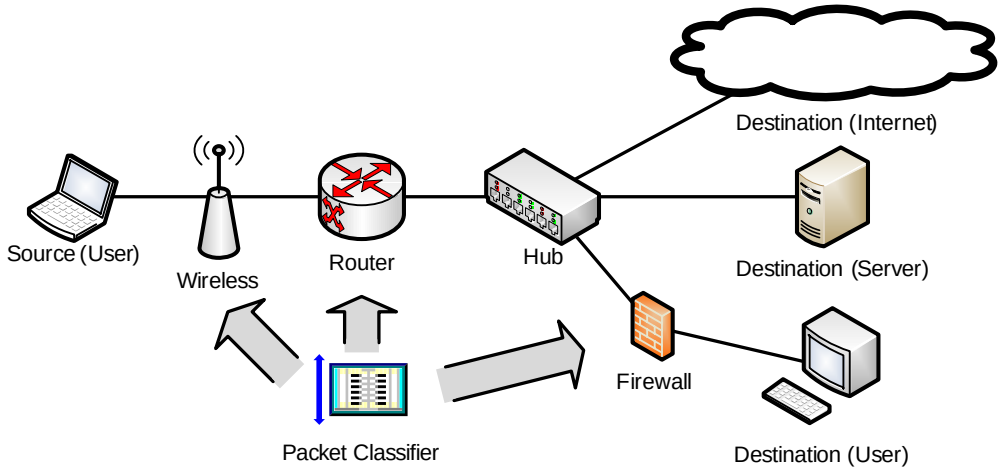


FIGURE 1.1: Illustration of packet classifier in network applications

In packet classification, Ternary Content Addressable Memories (TCAMs) [3, 5, 17, 29, 32, 34, 48] are used as the hardware to perform high-speed packet forwarding. TCAMs have become *de facto* standard in network applications. TCAM has three values, *i.e.,* 0, 1, and * (don't care).

Since a TCAM works simultaneously, it only takes $O(1)$ time to find a matched rules in the packet classifier. Moreover, a TCAM has priority encoder to return the first matched rule action. Suppose that a packet header comes, then the TCAM checks the rules simultaneously by comparing the header with all entries. After that, if there are multiple matches, then the TCAM will return the index of the upmost matched rule.

Inspite of the fact that TCAMs are fast, TCAMs have drawbacks:

- **Disipate high power**: Since TCAMs are used in network processor in packet classification and routing look up which require high performance, the power issue is also the main consideration in TCAMs [2]. It is reported in [30] that a conventional 4.5 Mb static TCAM dissipates approximately 7 Watt assuming a suply voltage of 1.5 Volt and the operating frequency of 143 Mhz. The issue will cause problems if we consider a very high-speed router or a packet classifier which requires high amount of TCAMs and works in a GHz speed.
- **Expensive**: The TCAM cost is expensive. In [28], a 1 Mb TCAM chip costs about 200-250 U.S. dollars and the TCAM cost is a significant fraction of router or packet classifier costs.

Moreover, packet classifiers have their own problems which are directly related to the TCAMs:

- **Rule expansion**: To represent a port field, we use an interval function. An interval function can be represented as a sum of prefixes. The expression represented by a sum of prefixes is called a prefix sum-of-products expression (**PreSOP**). Note that each prefix corresponds to a word in a TCAM. To represent any interval function by a PreSOP, in the worst case, $2n - 2$ products are necessary [60]. For instance, when $n = 16$, the worst case is $[1, 65534]$ where requires $2n-2 = 30$ words in single field. Thus, if we have two fields (*i.e,* source and destination ports), it requires $30 \times 30 = 900$ words in a TCAM. Moreover, in IPv4 the total bits in a word is atleast 104 bits. This can cause a significant expansion in the TCAM size.
- **Rapid increasing of internet**: The exponential growth of the internet has stressed its routing system. While the data rates of links have kept pace with

the increasing traffic, it has been difficult for paket processing capacity of routers to keep up with these increased data rates [36]. This issue will impact the increasing of the need of router's capacity and speed which is directly related to the need of a huge TCAM size in a router or a packet classifier.

To overcome these problems, minimization of TCAM size is necessary. Logic minimization in packet classification is quite different from that of large-scale integrated circuit (LSI) design. When a TCAM is used instead of a two-level AND-OR circuit, the optimization is more complicated than that of sum-of-products expressions (SOPs) [26]. Also, the logic optimization must be done much faster than conventional ones. For some cases, the data must be updated in every second. So, we cannot use conventional time-consuming algorithms [9, 40].

In this thesis, we presents some methods related to logic minimization in TCAMs. First, we show that the minimum PreSOP (MPreSOP) and the number of products in a MPreSOP to represent an open interval (A, B) can be easily generated. We also the statistical properties on the numbers of products in MPreSOP for interval functions. Second, we introduce an efficient method to represent interval functions called head-tail expressions (HTs). We prove that any interval function can be represented by an HT with at most n factors. Next, we use HTs to reduce TCAM size for single-field classification functions. We evaluate it by using benchmark function of packet classification called ClassBench. Finally, we propose methods to simplify TCAM size in multi-field classification functions. In the first method, we remove redundant rules by partitioning the rules into groups having the same source address, destination address and protocol. Then, we check the relation among rules within a group and remove the redundant rules. In the second method, we use HTs to reduce TCAM size for multi-field classification functions. In this case, we propose a fast method to generate a simplified TCAM directly.

1.2 Organization of Thesis

This thesis consists of seven chapters. Each chapter is organized as follows.

Chapter 1 is an indroduction. It shows applications to the internet.

Chapter 2 defines basic terminology on logic functions, prefix sum-of-products expressions (PreSOPs), interval functions, and packet classifications.

Chapter 3 formulates PreSOPs for interval functions, shows a method to generate PreSOPs from the binary representations of endpoints of an interval. It also derives statistical properties of PreSOPs, which are useful to estimate TCAM sizes.

Chapter 4 shows a new representation for interval functions called head-tail expressions (HTs). It shows a special property of interval functions to represent the interval functions with fewer words than PreSOPs. Finally, it shows that any interval function can be represented by an HT with at most n factors.

Chapter 5 shows a method to represent single-field classification functions by HTs. It explains the algorithm in detail and evaluates its performance using benchmark functions generated by ClassBench. The direct HT algorithm (DHT) is 6×10^5 times faster and produces smaller TCAMs than ESPRESSO-EXACT.

Chapter 6 proposes two methods to reduce TCAMs for multi-field classification functions. The first one is a fast redundancy removal for packet classification. And the second one is a TCAMs generator for multi-field classification functions.

Chapter 7 summarize the thesis.

Chapter 2

Preliminary

This chapter defines terminology and theoretical background used in this thesis.

2.1 Logic Function

Definition 2.1. A **logic function**, denoted by $f(x_{n-1}, x_{n-2}, \ldots, x_0)$ or simply f, is a mapping:

$$f(x_{n-1}, x_{n-2}, \ldots, x_0) : \{0, 1, \ldots, r-1\}^n \to \{0, 1, \ldots, r-1\},$$

where each x_i is called a variable. When $r = 2$, a logic function is a **two-valued logic function** that is a mapping:

$$f(x_{n-1}, x_{n-2}, \ldots, x_0) : \{0, 1\}^n \to \{0, 1\},$$

where each x_i is called a **binary variable**. When $r > 2$, a logic function is a **multi-valued logic function**, and each x_i is called a **multi-valued variable**.

Definition 2.2. A **multiple-output logic function** $F = (f_0, f_1, \ldots, f_{m-1})$ is a mapping:

$$F : \{0, 1, \ldots, r-1\}^n \to \{0, 1, \ldots, r-1\}^m.$$

Specially, when $m = 1$, it is called **single-output logic function**.

Definition 2.3. Let $S \subseteq \{0, 1, \ldots, r-1\}$. Then, x^S is a **literal** of a variable x.

Definition 2.4. Shannon expansion of a logic function f with respect to a variable x_i is:

$$f(x_{n-1}, x_{n-2}, \ldots, x_0) = \bigvee_{r-1}^{j=0} x_i^j \cdot f(x_{n-1}, x_{n-2}, \ldots, x_{i-1}, j, x_{i+1}, \ldots, x_0),$$

and each $f(x_{n-1}, x_{n-2}, \ldots, x_{i-1}, j, x_{i+1}, \ldots, x_0)$ is called a **cofactor** of f with respect to x_i

In this thesis, we assume that a given logic funtion is completely specified and has no redundant variables.

2.2 Interval Function

Definition 2.5. Let A and B be integers such that $A < B$. An **open interval** (A, B) denotes the set of integers X such that $A < X < B$. Note that endpoints are not included. The **size of an open interval** (A, B) is $C = B - A - 1$.

In this paper, only open intervals are considered. Thus from here, an open interval is simply denoted by an interval.

Definition 2.6. An n-input **interval function** is:

$$IN_0(n : A, B) = \begin{cases} 1, & \text{if } A < X < B \\ 0, & \text{otherwise,} \end{cases}$$

where $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$, A and B are integers.

An interval function can be represented by a product of a **greater-than** (GT) function and a **less-than** (LT) function.

Definition 2.7. An n-input GT function is:

$$GT(n : A) = \begin{cases} 1, & \text{if } X > A \\ 0, & \text{otherwise.} \end{cases}$$

An n-input LT function is:

$$LT(n : B) = \begin{cases} 1, & \text{if } X < B \\ 0, & \text{otherwise,} \end{cases}$$

where $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$, A and B are integers.

2.3 Prefix Sum-of-Products Expression

Definition 2.8. A **binary literal** has a form x^a, where x is a binary variable and $a \in \{0, 1\}$, and

$$x^a = \begin{cases} 1, & \text{if } x = a \\ 0, & \text{if } x \neq a. \end{cases}$$

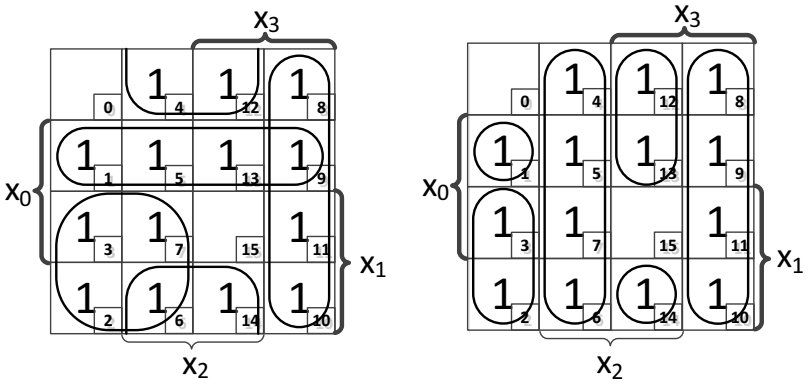
Definition 2.9. $x = x^1$ and $\bar{x} = x^0$ are **literals** of a variable x. An AND of literals is a **product**. An OR of products is a **sum-of-products expression** (SOP).

Definition 2.10. Let $\mathcal{F}$ be an SOP. $\tau(\mathcal{F})$ denotes the number of products in $\mathcal{F}$.

Definition 2.11. A prefix SOP (PreSOP) is an SOP consisting of products having the form $x^*_{n-1}x^*_{n-2}\ldots x^*_{m+1}x^*_m$, where x^*_i is x_i or $\bar{x}_i$ and $n - 1 \geq m$.

Definition 2.12. An SOP representing a given function f with the fewest products is a minimum sum-of-products expression (MSOP). A PreSOP representing a given function f with the fewest products is a minimum PreSOP (MPreSOP). An MSOP and an MPreSOP for f are denoted by MSOP(f) and MPreSOP(f), respectively.

Lemma 2.1. *Let $\tau_m(f) = \tau(MSOP(f))$ and $\tau_p(f) = \tau(MPreSOP(f))$. Since an MPreSOP is a restricted case of an MSOP, we have $\tau_m(f) \leq \tau_p(f)$.*



(a) MSOP (b) MPreSOP

FIGURE 2.1: Maps for $f = IN_0(4 : 0, 15)$

Example 2.1. *Fig. 2.1(a) shows an MSOP: $\bar{x}_0x_2 \vee x_0\bar{x}_1 \vee x_1\bar{x}_3 \vee \bar{x}_2x_3$ for the interval function $f = IN_0(4 : 0, 15)$. Fig. 2.1(b) shows the MPreSOP: $\bar{x}_3\bar{x}_2\bar{x}_1x_0 \vee \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3x_2 \vee x_3\bar{x}_2 \vee x_3x_2\bar{x}_1 \vee x_3x_2x_1\bar{x}_0$. Thus, $\tau(f) = 4$ and $\tau(f) = 6$.* ■

Lemma 2.2. *Any product in a PreSOP can be represented by an interval function:*

$$x_{n-1}^{a_{n-1}} x_{n-2}^{a_{n-2}} \cdots x_m^{a_m} = IN_0(n : k2^m - 1, (k+1)2^m),$$

where $k = \sum_{i=0}^{n-m-1} a_{m+i} \cdot 2^i$*, and* m *denotes the number of missing variables. Note that* $k = 0, 1, \ldots, 2^{n-m} - 1$ *and* $m = 0, 1, 2, \ldots, n$*.*

Thus, the products in the PreSOP for $IN_0(4 : 0, 15)$ are represented as follows:

$$\begin{aligned}
&\bar{x}_3\bar{x}_2\bar{x}_1x_0 : (0, 2) && (k = 1, m = 0)\\
&\bar{x}_3\bar{x}_2x_1 : (1, 4) && (k = 1, m = 1)\\
&\bar{x}_3x_2 : (3, 8) && (k = 1, m = 2)\\
&x_3\bar{x}_2 : (7, 12) && (k = 2, m = 2)\\
&x_3x_2\bar{x}_1 : (11, 14) && (k = 6, m = 1)\\
&x_3x_2x_1\bar{x}_0 : (13, 15) && (k = 14, m = 0)
\end{aligned}$$

The numbers in the smaller boxes in Fig. 2.1 show the values $X = 8x_3 + 4x_2 + 2x_1 + x_0$.

A PreSOP is a special case of an SOP. Thus, for a given function f, a PreSOP may require more products than an SOP. However, PreSOPs are often used in the internet applications since they can be generated quickly from the tree or a decision diagram (DD) representing the function. Also, the PreSOPs generated from trees or DDs are disjoint [60]. This means that we cannot apply the absorption law to simplify the expression.

On the other hand, to simplify an SOP, we have to apply the absorption law. The time complexity for the absorption law to an SOP is $O(np^2)$, where n denotes the number of bits to represent the maximum value of the interval, and p denotes the number of products. Thus, the SOP minimizer tends to be slow. This is the reason why PreSOPs are used instead of SOPs in internet applications.

2.4 Ternary Content Addressable Memory

A content addressable memory (CAM) simultaneously compares the inputs vector with the entire list of registered vectors [33]. TCAM is a *de facto* standard in routers and devices for packet classification. Fig. 2.2 shows an example of a TCAM circuit [43]. The search data is compared with the stored words. When there is a match, the match line sends the signal to the priority encoder to produce the match address.

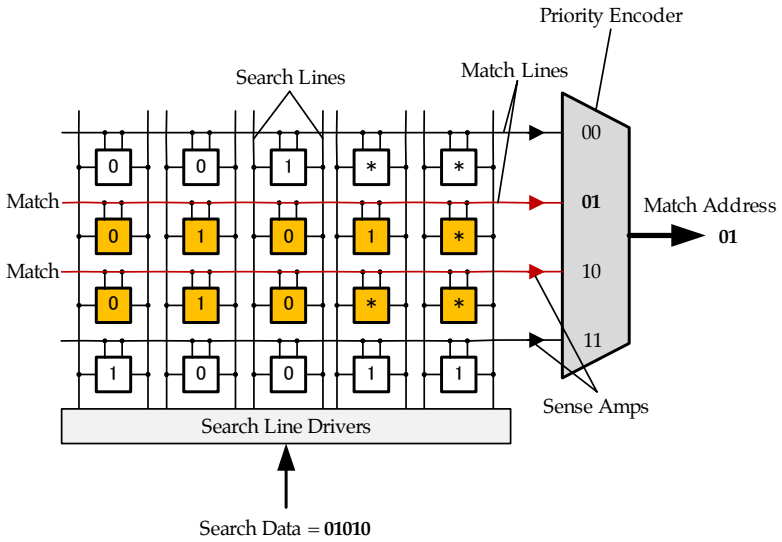


FIGURE 2.2: TCAM Circuit

To show the concept of a packet classification using a TCAM, for simplicity, assume the packets are accepted when

$$(1 \leq X \leq 14),$$

where $X = 8x_3 + 4x_2 + 2x_1 + x_0$. When multiple matches occur, the priority encoder detects the match line with the smallest index. In this example, the packet classification uses only one field specified by four bits. The packet classifier can be implemented as shown in Table 2.1(a). Note that the condition for *Accept* can be represented as

$$f = \bar{x}_3\bar{x}_2\bar{x}_1x_0 \vee \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3x_2 \vee x_3\bar{x}_2 \vee x_3x_2\bar{x}_1 \vee x_3x_2x_1\bar{x}_0.$$

The bottom word of the TCAM in Table 2.1(a) consists of all *don't cares*. Thus, the TCAM requires 7 words.

TABLE 2.1: Packet Classifier Based on Accept or Discard Condition

(a) Accept

TCAM	SRAM
0001	Accept
001*	Accept
01**	Accept
10**	Accept
110*	Accept
1110	Accept
****	Discard

(b) Discard

TCAM	SRAM
0000	Discard
1111	Discard
****	Accept

Since, f can be simplified as

$$f = \bar{x}_3 x_2 \vee \bar{x}_2 x_1 \vee \bar{x}_1 x_0 \vee x_3 \bar{x}_0,$$

the number of TCAM words can be reduced to five. However, the complement of the function can be represented as

$$\bar{f} = \bar{x}_3 \bar{x}_2 \bar{x}_1 \bar{x}_0 \vee x_3 x_2 x_1 x_0.$$

Thus, the circuit can be simplified by implementing the *Discard* condition instead of the *Accept* conditions as shown in Table 2.1(b), which uses only three words. In this case, the entries for the static random access memory (SRAM) also must be modified.

2.5 Packet Classification

Definition 2.13. [41] A **classification function** with k **fields** is a mapping $F : D_1 \times D_2 \times \cdots \times D_k \to \{0, 1, 2, \cdots, r\}$, where $D_i = \{0, 1, \cdots, 2^{t_i} - 1\}$ $(i = 1, 2, \cdots, k)$. F is specified by a set of r rules. A **rule** consists of k fields, and each field D_i is specified by an interval of t_i bits.

In the internet, packets are classified by the source address (SA), the destination address (DA), the source port (SP), the destination port (DP), and the protocol type (PO). Table 2.2 shows an example of a packet classifier. In this case, the classifier consists of three **rules**, and each rule consists of five **fields**. In IPV4, an internet address is specified by a 32-bit number, while the ports are specified by intervals of 16-bit numbers. The protocol type is specified by an 8-bit number. The fields often have *, which denotes *don't care*. If the result of the classification is *Accept*, then the corresponding packet is sent to the next destination. Otherwise, the packet is discarded.

TABLE 2.2: Example of rules in packet classifier

SA	DA	SP	DP	PO	Action
66.219.40.∗	176.31.166.∗	[0, 65535]	6790	TCP	Accept
∗	15.238.61.128	∗	[1024, 65535]	∗	Accept
∗	∗	∗	∗	∗	Discard

In packet classification [7], rules are applied from the top to the bottom. Thus, in Table 2.2, if the source address is 66.219.40.11, the destination address is 176.31.166.23, the source port is 1025, the destination port is 6790, and the protocol type is TCP, then the first rule is satisfied, and the packet is sent to the next address. If the first rule is not satisfied, then the second rule is checked. If the second rule is not satisfied, then

the last rule is checked. Since the last rule has * in all the fields, the last rule is always satisfied. In this case, the packet is discarded. Thus, the packet classification in Table 2.2 can be considered as a five-field classification function.

Note that all the fields can be represented by intervals [41]. For example, an 8-bit address

$$1001****$$

can be represented by the interval

$$[9 \times 16, 9 \times 16 + 15] = [144, 159].$$

Also, a single value *i.e* 7 can be represented by the interval $[7, 7]$.

Chapter 3

Generating Prefix Sum-of-Products Expressions for Interval Functions

This chapter is related to prefix sum-of-products expressions for interval functions. We derive the PreSOPs and show the optimality of PreSOPs in interval functions. We also analyze the statistical properties. The results are useful to estimate the size of TCAMs for packet classification.

3.1 Introduction

Packet classification [7, 15–17, 31, 61] is a core function in computer network components, such as routers [19], firewalls [14, 20, 21], network address translators, and access control lists (ACL) [22]. A ternary content addressable memory (TCAM) [33, 43] implements packet classification functions. Although a TCAM is fast, it is expensive and dissipates high power [1].

TABLE 3.1: Example of rules in packet classifier

Rule	Source IP	Destination IP	Source Port	Destination Port	Protocol	Action
1	66.219.40.∗	176.31.166.∗	$(-1, 65536)$	6790	TCP	Permit
2	∗	15.238.61.128	∗	$(1023, 65536)$	∗	Permit
3	∗	∗	∗	∗	∗	Deny

Table 3.1 shows an example of a packet classifier consisting of five fields: the source IP, the destination IP, the source port, the destination port, and the protocol. In this example, both the source IP and the destination IP are represented by 32 bits, and they

TABLE 3.2: Various methods to represent classification functions

Method [Ref.]	Representation	Bound
Binary Tree [60]	n-variable binary prefixes	$2n-2$
2-valued MSOP [41]	n-variable binary non-prefixes	$2n-4$
Gray encoding [8]	n-variable binary prefixes	$2n-4$
Output encoding [38]	$(n+1)$-variable binary prefixes	n
4-valued MSOP [42]	$(\frac{n}{2})$-variable 4-valued non-prefixes	$n-1$
TCAM with comparator circuit [46]	A direct interval	1

are specified by prefixes. Both the source port and the destination port are represented by 16 bits, and they are specified by intervals. The protocol is represented by a value of 8 bits or $*$ (*don't care*). The action has two values: *permit* and *deny*. However, it can have more values such as *deny and log* or *permit and log*. When each of the port fields is specified by either $*$ (*don't care*) or a single value, each rule corresponds to one word in a TCAM. However, when a port field is specified by an open interval such as $(0, 7)$, **rule expansion** occurs, *i.e.,* each rule corresponds to many words in a TCAM [12]. If the packet header matches all the fields of a rule, then the rule is considered to be matched. If several rules match at the same time, then the rule with the smallest number is applied.

To represent a port field, we use an interval function. An interval function can be represented as a sum of prefixes. The expression represented by a sum of prefixes is called a prefix sum-of-products expression (**PreSOP**). Note that each prefix corresponds to a word in a TCAM. To represent any interval function by a PreSOP, in the worst case, $2n-2$ products are necessary [60].

A sum-of-product expression (**SOP**) requires no more products than a PreSOP to represent the same function, and in many cases, the SOP requires fewer products than the PreSOP. For example to represent the interval $(0, 2^n-1)$, a PreSOP requires $2(n-1)$ products, while an SOP requires only n products [39]. Although we can obtain an exact minimum SOP, it is often time consuming. In the design of integrated circuits for mass production, SOPs are routinely used [9, 40]. Their minimization cost can be amortized by many integrated circuits. However, in the case of packet classification, PreSOPs are used instead of SOPs.

To reduce the number of words in a TCAM for ACL, minimization algorithms for SOPs, that are suitable for embedded microprocessors, have been developed [4, 24]. It uses a ternary trie as a basic data structure. Such an algorithm is fast, but produces solutions with many more products than the exact minimum solutions for some classes of functions [39].

Table 3.2 lists various methods to represent classification functions and their TCAM sizes. In [41, 44], the number of products needed to represent an interval function of n variables by a standard encoding in SOPs is analyzed. They show that any interval function can be represented with at most $2n-4$ products in an SOP. Especially, the paper [41] shows that only two interval functions require $2n-4$ products in SOPs. In the paper [42], a 4-valued TCAM is proposed, where inputs are encoded with a 1-out-of-4 code. In this method, any interval function can be represented with at most $n-1$ products by a 4-valued SOP. To use this method in packet classification, a special CAM and a 4-valued logic minimizer are necessary. In the paper [46], a special TCAM that performs interval matching directly by a special circuit is proposed. In this case, each rule corresponds to just one word in a TCAM, but we need a special TCAM which would be expensive. In the paper [8], Gray encoding is used to reduce the number of products. In the Gray encoding, any interval $(A-1, A+2)$, where A is a non-negative integer, can be represented with a single product. The paper [8] showed that any interval function can be represented with at most $2n-4$ products in a PreSOP.

In a TCAM, a priority encoder is included to produce a unique address among the matched data [33]. By changing the order of rules stored in a TCAM, we can often reduce the number of products needed to represent the function [12, 28]. In [26], the authors presented a method to minimize the number of TCAM words. Although this method can find an exact minimum solution, it requires computation time that is impractically large. In [38], an output encoding is used to reduce the number of products. To use this method in a real packet classification, a TCAM and an external memory are necessary.

In this chapter, we show that 1) the minimum PreSOP (MPreSOP) and the number of products in a MPreSOP to represent an open interval (A, B) can be quickly generated and calculated, and 2) the average number of products in MPreSOPs for interval functions is $\mu(n) \approx n-2$, and the variance is $\sigma^2(n) \approx \frac{n}{2}+1$[1]. Also, by numerical calculation, we show that 99.9% of the functions can be represented by PreSOPs with at most $\lceil \frac{3}{2}n-1 \rceil$ products when $n > 12$.

3.2 Definitions and Basic Properties

Lemma 3.1. *Let $-1 \leq A < B \leq 2^n$. The number of distinct open interval functions in (A, B), is $N(n) = 2^{n-1}(2^n+1)$.*

[1] $A(n) \approx B(n)$ i ff$|A(n) - B(n)| \to 0$ as $n \to \infty$.

Proof: Let the size of an interval (A, B) be $C = B-A-1$. For $C = 1, C = 2, \ldots, C = 2^n$, the numbers of distinct interval functions are $2^n, 2^n-1, 2^n-2, \ldots, 1$, respectively. Thus, we have $N(n) = 2^n + (2^n - 1) + (2^n - 2) + \ldots + 1 = 2^{n-1}(2^n + 1)$. □

3.3 Number of Products in an MPreSOP

In this section, we show that, given an interval (A, B), the PreSOP and the number of products in the PreSOP can be generated quickly and calculated. Previously, PreSOPs are generated by using binary trees [60]. However, the PreSOP of an interval (A, B) can be generated directly from its binary representations of the endpoints (A and B). Thus, the PreSOP for a given interval can be generated quickly.

Definition 3.1. A **vector literal** has the form $X^{\vec{a}}$, where $X = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ and $\vec{a} = (a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$.

$$X^{\vec{a}} = \begin{cases} 1, & \text{if } X = \vec{a} \\ 0, & \text{if } X \neq \vec{a}. \end{cases}$$

It is equivalent to $x_{n-1}^{a_{n-1}} x_{n-2}^{a_{n-2}} \cdots x_0^{a_0}$.

Lemma 3.2. $x^a = \bar{x} \cdot \bar{a} \vee x \cdot a$.

Lemma 3.3. *A GT function has the following MPreSOP:*

$$GT(n : A) = (x_{n-1}\bar{a}_{n-1}) \vee \bigvee_{i=n-2}^{0} \left(\bigwedge_{j=n-1}^{i+1} x_j^{a_j} \right) x_i \bar{a}_i,$$

where $\vec{a} = (a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$ *is the binary representation of* A. $\tau_p(GT(n : A)) = \sum_{i=0}^{n-1} \bar{a}_i$.

Proof: See 3.6.1. □

Similarly to Lemma 3.3, we have:

Lemma 3.4. *An LT function has the following MPreSOP:*

$$LT(n : B) = (\bar{x}_{n-1} b_{n-1}) \vee \bigvee_{i=n-2}^{0} \left(\bigwedge_{j=n-1}^{i+1} x_j^{b_j} \right) \bar{x}_i b_i,$$

where $\vec{b} = (b_{n-1}, b_{n-2}, \ldots, b_1, b_0)$ *is the binary representation of* B. $\tau_p(LT(n : B)) = \sum_{i=0}^{n-1} b_i$.

Similarly to $GT(n:A)$ and $LT(n:B)$, an interval function $IN_0(n:A,B)$ is represented as an MPreSOP.

Theorem 3.1. *Let* $\vec{a}=(a_{n-1},a_{n-2},\ldots,a_1,a_0)$ *and* $\vec{b}=(b_{n-1},b_{n-2},\ldots,b_1,b_0)$ *be the binary representations of* A *and* B*, respectively. Let* s *be the largest index such that* $a_s\neq b_s$*, then* $IN_0(n:A,B)$ *can be represented by:*

$$\bigvee_{i=s-1}^{0}\left[\left(\bigwedge_{j=n-1}^{i+1}x_j^{a_j}\right)x_i\bar{a}_i\vee\left(\bigwedge_{j=n-1}^{i+1}x_j^{b_j}\right)\bar{x}_ib_i\right]. \tag{3.1}$$

Proof: Since both $GT(n:A)$ and $LT(n:B)$ have at most n products, the AND operation between GT and LT functions produces at most n^2 products.

Let a_i and b_i be the components of $\vec{a}$ and $\vec{b}$, respectively, where $s+1\leq i\leq(n-1)$. For every pair of products in GT and LT functions, there exist a pair of components $(x_i\bar{a}_i)$ and $(\bar{x}_ib_i)$. These components cancel each other when $s+1\leq i\leq(n-1)$. The cancellation occurs for three cases: the first case occurs when $(x_i\bar{a}_i)\cdot(\bar{x}_ib_i)$; the second case occurs when $(x_i\bar{a}_i)\cdot x_i^{b_i}$; and the last case occurs when $(\bar{x}_ib_i)\cdot x_i^{a_i}$. In the first case, $x_i\cdot\bar{x}_i$ yields 0. In the second and the third cases, by Lemma 3.2, we have $x_i\bar{a}_i(\bar{x}_i\bar{b}_i\vee x_ib_i)=x_i\bar{a}_ib_i=0$ and $\bar{x}_ib_i(\bar{x}_i\bar{a}_i\vee x_ia_i)=x_i\bar{a}_ib_i=0$, respectively. Thus, IN_0 can be simplified to:

$$\begin{aligned}IN_0(n:A,B)=&\left(x_{n-1}^{a_{n-1}}x_{n-2}^{a_{n-2}}\cdots x_{s+1}^{a_{s+1}}x_s\bar{a}_s\vee\cdots\vee x_{n-1}^{a_{n-1}}x_{n-2}^{a_{n-1}}\cdots x_1^{a_1}x_0\bar{a}_0\right)\\&\cdot\left(x_{n-1}^{b_{n-1}}x_{n-2}^{b_{n-2}}\cdots x_{s+1}^{b_{s+1}}\bar{x}_sb_s\vee\cdots\vee x_{n-1}^{b_{n-1}}x_{n-2}^{b_{n-2}}\cdots x_1^{b_1}\bar{x}_0b_0\right).\end{aligned}$$

Thus, the interval function IN_0 has at most $(s+1)^2$ products. Since $a_s\neq b_s$, there are only two cases that produce non-zero products: the first case occurs for the AND operation between a GT product $x_s\bar{a}_s$, and an LT product without a $\bar{x}_s$ literal. The second case occurs for the AND operation between an LT product $\bar{x}_sb_s$, and a GT product without a x_s literal.

By performing all AND operations for those cases, the interval function IN_0 is represented as a disjunction of the products for GT and LT functions without

$$x_{n-1}^{a_{n-1}}x_{n-2}^{a_{n-2}}\cdots x_{s+1}^{a_{s+1}}x_s\bar{a}_s\text{ and }x_{n-1}^{b_{n-1}}x_{n-2}^{b_{n-2}}\cdots x_{s+1}^{b_{s+1}}\bar{x}_sb_s.$$

Thus, we have the theorem. □

Lemma 3.5. *Let* $f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0) = \bar{x}_{n-1} f_0 \vee x_{n-1} f_0$, *where* $f_0 = f(0, x_{n-2}, \ldots, x_1, x_0)$ *and* $f_1 = f(1, x_{n-2}, \ldots, x_1, x_0)$. *Then, MPreSOP*$(f)$ *has the form*

$$\bar{x}_{n-1} MPreSOP(f_0) \vee x_{n-1} MPreSOP(f_1).$$

Proof: See 3.6.2. □

Theorem 3.2. *Let* $\vec{a} = (a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$ *and* $\vec{b} = (b_{n-1}, b_{n-2}, \ldots, b_1, b_0)$ *be the binary representations of* A *and* B, *respectively. Let* s *be the largest index such that* $a_s \neq b_s$. *Then, the expression* (3.1) *in Theorem 3.1 is the MPreSOP, and*

$$\tau_p(IN_0(n : A, B)) = \sum_{i=0}^{s-1} (\bar{a}_i + b_i).$$

Proof: It is clear from Eq. (3.1) and Lemma 3.5. □

Example 3.1. *Derive* $\tau_p(IN_0(6 : 2, 14))$ *and the MPreSOP for* $IN_0(6 : 2, 14)$. *The binary representations of* $A = 2$ *and* $B = 14$ *are* $\vec{a} = (0, 0, 0, 0, 1, 0)$ *and* $\vec{b} = (0, 0, 1, 1, 1, 0)$, *respectively. Since* $(a_5, a_4) = (b_5, b_4) = (0, 0)$ *and* $a_3 \neq b_3$, *we have* $s = 3$. *Thus,*

$$\tau_p(IN_0(6 : 2, 14)) = \sum_{i=0}^{2} (\bar{a}_i + b_i) = 4.$$

In Figure 3.1, the top row shows products of $GT(n : 2)$ *and* $LT(n : 14)$. *The middle row shows their maps. And the bottom row shows the MPreSOP for* $IN_0(n : 2, 14)$. *Note that the largest products in* $GT(n : 2)$ *and* $LT(n : 14)$ *are removed, where the grey side indicates the* $GT(n : 2)$ *part and the white side indicates the* $LT(n : 14)$ *part. Thus, the MPreSOP is* $IN_0(6 : 2, 14) = \bar{x}_5\bar{x}_4\bar{x}_3\bar{x}_2 x_1 x_0 \vee \bar{x}_5\bar{x}_4\bar{x}_3 x_2 \vee \bar{x}_5\bar{x}_4 x_3\bar{x}_2 \vee \bar{x}_5\bar{x}_4 x_3 x_2\bar{x}_1$. ■

Theorem 3.1 can be used to generate MPreSOPs from binary representations of endpoints (A, B).

Lemma 3.3, Lemma 3.4, and Theorem 3.1 do not cover the cases when endpoints are $A = -1$ and $B \leq 2^n - 1$, or, $A \geq 0$ and $B = 2^n$, or, $A = -1$ and $B = 2^n$, because they require $(n+1)$ bits to represent the interval functions. These points are called **extremal endpoints**.

Lemma 3.6. *In the extremal endpoints, we have* $GT(n : -1) = LT(n : 2^n) = IN_0(n : -1, 2^n) = 1$, $IN_0(n : -1, B) = LT(n : B)$, *and* $IN_0(n : A, 2^n) = GT(n : A)$.

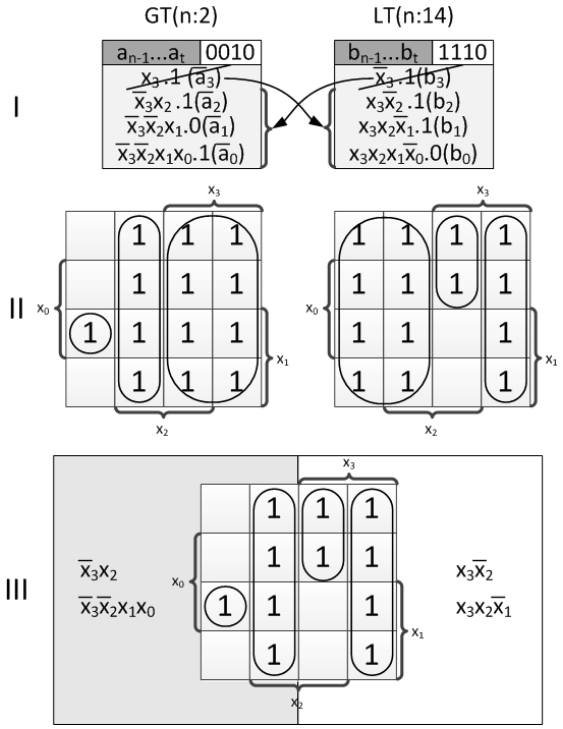


FIGURE 3.1: Derivation of MPreSOP for $IN_0(6:2,14)$

3.4 Number of Interval Functions Requiring τ_p Products

In this section, we derive the formula for the number of interval functions requiring τ_p products in their MPreSOPs. With this, we can estimate the TCAM sizes for packet classification [17].

Definition 3.2. Let $\Psi(n,\tau_p)$ be the number of n-variable interval functions that require τ_p products in their MPreSOPs.

To derive $\Psi(n,\tau_p)$, we have to consider the *extremal endpoints* that appeared before Lemma 3.6. Thus, the integers must be represented by $(n+1)$ bits for enumeration.

Definition 3.3. Let A and B be integers such that $-1 \leq A < B \leq 2^n$. Let $s(A,B)$ be the largest integer such that $a_s \neq b_s$. $s(A,B)$ is called **separation index**, where $0 \leq s(A,B) \leq n$. The binary representations of A and B are $\vec{a} = (a_n, a_{n-1}, \ldots, a_1, a_0)$ and $\vec{b} = (b_n, b_{n-1}, \ldots, b_1, b_0)$, respectively. $A = -1$ and $B = 2^n$ correspond to *extremal endpoints* that require $(n+1)$ bits.

In deriving$\Psi(\ n,\tau_p)$, the separation index s is used as a parameter to enumerate the number of interval functions. Table 3.3 shows all $IN_0(4:A,B)$ functions of $n=4$ variables that require $\tau_p = 3$ products, where the upper integer denotes A, while the lower integer denotes B. The integers at the left side of the bits are the decimal representations

of A and B. Also, A and B are represented by $n+1 = 5$ bit numbers, the least significant s bits are separated by | to show the patterns of the combinations. For example, when $s = 2$, two cases satisfy the requirements. In the first case, the lower endpoint (A) contributes $\binom{2}{1} = 2$ products, while the upper endpoint (B) contributes $\binom{2}{2} = 1$ product. In the second case, A contributes $\binom{2}{2} = 1$ product, while B contributes $\binom{2}{1} = 2$ products. Thus, we have $\binom{2}{2}\binom{2}{1} + \binom{2}{1}\binom{2}{2} = 4$.

The top four pairs in Table 3.3 in the column headed by $s = 2$ denote all possible combinations of the least significant s bits that produce $\tau_p = 3$ products. The bottom four pairs correspond to the repetition of the top four pairs. Note that, in the top four pairs, the 4th bits in A (*i.e.,* a_3) and B (*i.e.,* b_3) are 0s, while, in the bottom four pairs, the 4th bits are 1s.

By Theorem 3.2, we enumerate τ_p with respect to the least significant s bits in the binary representations of the endpoints. Thus, we define:

Definition 3.4. Let $\eta(s,\tau_p)$ be the number of interval functions with the separation index s that require τ_p products in their PreSOPs.

Lemma 3.7. $\eta(s,\tau_p) = \binom{2s}{\tau_p}$.

Proof: By Theorem 3.2, $\eta(s,\tau_p)$ is equal to the number of combinations such that

$$\sum_{j=0}^{s-1}(\bar{a}_j + b_j) = \tau_p.$$

This number is equal to the number of ways to distribute τ_p elements to $2s$ bins. Thus, we have $\binom{2s}{\tau_p}$. □

Lemma 3.8. *Let $r(n,s)$ be the number of repetitions that multiplies $\eta(s,\tau_p)$. Then, we have $r(n,s) = 2^{n-s-1}$.*

From here, we will consider the *extremal endpoints* that require $(n+1)$ bits to represent the numbers. These cases occur when one of endpoints is -1 or 2^n. Note that the binary representation of -1 is $\vec{a} = (a_n, a_{n-1}, \ldots, a_1, a_0) = (1, 1, \cdots, 1, 1)$ and the binary representation of 2^n is $\vec{b} = (b_n, b_{n-1}, \ldots, b_1, b_0) = (1, 0, \cdots, 0, 0)$.

Lemma 3.9. *For the interval functions with $s(A,B) = n$ and $\tau_p \leq n$, we have $\eta(n,\tau_p) = 2\binom{n}{\tau_p}$.*

Proof: When $s(A,B) = n$, $a_n \neq b_n$. This occurs when $A = -1$ and $B \leq 2^n - 1$, or, $A \geq 0$ and $B = 2^n$. In these cases, one of endpoints *i.e.,* $A = -1$ or $B = 2^n$, contributes no product. Because of that, the other endpoints must contribute τ_p products. The number

TABLE 3.3: List of interval functions for $n = 4$ and $\tau_p = 3$ for different s

$s = 2$	$s = 3$		$s = 4$
0: 000 | 00	0: 00 | 000	3: 00 | 011	1: 0 | 0001
5: 001 | 01	8: 01 | 000	11: 01 | 011	16: 1 | 0000
0: 000 | 00	1: 00 | 001	3: 00 | 011	2: 0 | 0010
6: 001 | 10	9: 01 | 001	13: 01 | 101	16: 1 | 0000
1: 000 | 01	1: 00 | 001	3: 00 | 011	4: 0 | 0100
7: 001 | 11	10: 01 | 010	14: 01 | 110	16: 1 | 0000
2: 000 | 10	1: 00 | 001	5: 00 | 101	8: 0 | 1000
7: 001 | 11	12: 01 | 100	11: 01 | 011	16: 1 | 0000
8: 010 | 00	2: 00 | 010	5: 00 | 101	-1: 1 | 1111
13: 011 | 01	9: 01 | 001	13: 01 | 101	7: 0 | 0111
8: 010 | 00	2: 00 | 010	5: 00 | 101	-1: 1 | 1111
14: 011 | 10	10: 01 | 010	14: 01 | 110	11: 0 | 1011
9: 010 | 01	2: 00 | 010	6: 00 | 110	-1: 1 | 1111
15: 011 | 11	12: 01 | 100	11: 01 | 011	13: 0 | 1101
10: 010 | 10	4: 00 | 100	6: 00 | 110	-1: 1 | 1111
15: 011 | 11	9: 01 | 001	13: 01 | 101	14: 0 | 1110
	4: 00 | 100	6: 00 | 110	
	10: 01 | 010	14: 01 | 110	
	4: 00 | 100	7: 00 | 111	
	12: 01 | 100	15: 01 | 111	

of ways to produce τ_p products is $\binom{n}{\tau_p}$. Thus, we have $\eta(n,\tau_p) = \binom{n}{\tau_p}\binom{n}{0}+\binom{n}{0}\binom{n}{\tau_p} = 2\binom{n}{\tau_p}$. □

Example 3.2. *In Table 3.3, when $s = n = 4$ and $\tau_p = 3$, the* extremal endpoints *occur, and we have $\eta(4, 3) = 2\binom{4}{3} = 8$.* ■

Lemma 3.10. *For $s(A, B) \geq n$ and $\tau_p > n$, $\eta(s,\tau_p) = 0$.*

Proof: Note that no interval function satisfies the condition. For $s = n$ and $\tau_p > n$, the only allowable endpoints are 2^n+1 and 2^n. These endpoints contribute no products, and the remaining space from another endpoint can be represented with only n products. □

Theorem 3.3.

$$\Psi(n,\tau_p) = \left(\sum_{s=1}^{n-1} 2^{n-s-1}\binom{2s}{\tau_p}\right) + 2\binom{n}{\tau_p}.$$

Proof: $\Psi(n,\tau_p)$ is given as the sum of the number of open interval functions $\eta(s,\tau_p)$ times the repetition factor $r(n, s)$ for every possible s:

$$\Psi(n,\tau_p) = \left(\sum_{s=\lceil \frac{1}{2}\tau_p \rceil}^{n-1} r(n, s)\eta(s,\tau_p)\right) + \eta(n,\tau_p). \tag{3.2}$$

For the separation index $s(A,B)$, we have $\lceil \frac{1}{2}\tau_p \rceil \leq s(A,B) \leq n$. $s(A,B)$ takes its maximum and minimum values in the extremal endpoints. Thus, the sum operation in Eq. (3.2) is bounded from $\lceil \frac{1}{2}\tau_p \rceil$ to $n-1$. Moreover, since $\binom{2s}{\tau_p} = 0$ when $2s < \tau_p$, the lower bound can be simply 1. From Lemmas 3.7 to 3.10 and Eq. (3.2), we have the theorem. □

3.5 Statistical Properties

In this section, we show some statistical properties of the number of products in PreSOPs. We assume that each function is equally likely. Therefore, the probability distribution function of the number of interval functions with τ_p products in PreSOPs is given by $\frac{\Psi(n,\tau_p)}{N(n)}$, where $N(n)$ is the number of distinct open interval functions in (A,B).

3.5.1 Average Number of Products

The average number of products in PreSOPs (**mean**) for n-variable interval functions is given by

$$\mu(n) = \frac{1}{N(n)} \sum_{\tau_p=1}^{2(n-1)} \tau_p \cdot \Psi(n,\tau_p),$$

where $N(n)$ is the total number of distinct interval functions of n variables, and $\Psi(n,\tau_p)$ is the number of interval functions that require τ_p products.

Theorem 3.4.

$$\mu(n) \approx n - 2.$$

Proof: See 3.6.3. □

3.5.2 Variance of the Numbers of Products

The **variance** [10] of the numbers of products in PreSOPs for interval functions is given by

$$\sigma^2(n) = \sum_{\tau_p=1}^{2(n-1)} \tau_p^2 \frac{\Psi(n,\tau_p)}{N(n)} - \mu(n)^2.$$

Theorem 3.5.

$$\sigma^2(n) \approx \frac{n}{2} + 1.$$

Proof: See 3.6.4. □

Lemma 3.11. ***(Chebyshev's inequality [10])*** *Let X be a random variable with a finite expected value μ and a finite non-zero variance σ^2. Then, for any real number $k > 0$, we have*

$$Pr(|X - \mu|) \geq k\sigma) \leq \frac{1}{k^2}.$$

For $k = 2$, we have,

$$Pr(|X - \mu|) \geq 2\sigma) \leq \frac{1}{4}.$$

Corollary 3.1. *For large n, at least 75% of n-variable interval functions can be represented by PreSOPs with $(n-2) + \sqrt{2(n+2)}$ products.*

3.6 Proof of Optimality and Statistical Properties

3.6.1 Proof of Lemma 3.3

Since we consider PreSOPs, we have the following:

When $a_{n-1} = 0$:

$$GT(n : A) = x_{n-1} \vee \bar{x}_{n-1}\text{MPreSOP}(GT(n-1 : \hat{A})),$$

where $\hat{A}$ is the integer represented by $(a_{n-2}, a_{n-3}, \ldots, a_1, a_0)$.

When $a_{n-1} = 1$:

$$GT(n : A) = x_{n-1}\text{MPreSOP}(GT(n-1 : \hat{A})).$$

For $n = 1$ and $n = 2$, it is clear that the lemma holds. By mathematical induction, we have the lemma. □

3.6.2 Proof of Lemma 3.5

In the case of SOPs, an MSOP can be found by the following approach [40]:

1) Generate the set of all the prime implicants (PIs).

2) Among the set of PIs, select a minimum set of PIs that covers the minterms.

Let PI(f) be the set of all the PIs for f. Then, we have the following relation [35]

$$\mathrm{PI}(f) = \bar{x}_{n-1}\mathrm{PI}(f_0) \cup x_{n-1}\mathrm{PI}(f_1) \cup \mathrm{PI}(f_0 \cdot f_1).$$

In the case of PreSOPs, an MPreSOP can be found by a similar approach, but the first step should be modified as follows:

1') Generate PrePI(f), the set of all the PIs for f having the form $x^*_{n-1}x^*_{n-2}\cdots x^*_m$. Note that PrePI($f$) can be written as

$$\mathrm{PrePI}(f) = \bar{x}_{n-1}\mathrm{PrePI}(f_0) \cup x_{n-1}\mathrm{PrePI}(f_1).$$

Since functions are represented by PreSOPs, all the products have the form $x^*_{n-1}x^*_{n-2}\cdots x^*_m$. Thus, we do not have to generate PrePI($f_0 \cdot f_1$). This implies that to obtain the MPreSOP(f), we can minimize the subfunctions independently as follows: MPreSOP(f) = $\bar{x}_{n-1}$MPreSOP(f_0) $\vee$ x_{n-1}MPreSOP(f_1). □

Corollary 3.2. *Let* $f(x_{n-1}, x_{n-2}, \ldots, x_0) = \bar{x}_{n-1}f_0 \vee x_{n-1}f_1$, *where* $f_0 = f(0, x_{n-2}, \ldots, x_0)$ *and* $f_1 = f(1, x_{n-2}, \ldots, x_0)$. *Then,*

$$\tau_p(f) = \tau_p(f_0) + \tau_p(f_1).$$

Lemma 3.12.

$$\sum_{i=1}^{n} i\binom{n}{i} = n2^{n-1}.$$
$$\sum_{i=1}^{n} i^2\binom{n}{i} = n(n+1)2^{n-2}.$$

Lemma 3.13.

$$\sum_{i=1}^{n} i2^i = 2^{n+1}(n-1) + 2.$$
$$\sum_{i=1}^{n} i^2 2^i = 2^{n+1}(n^2 - 2n + 3) - 6.$$

3.6.3 Proof of Theorem 3.4

The average number of products in PreSOPs is

$$\begin{aligned}\mu(n) &= \frac{1}{N(n)} \sum_{k=1}^{2(n-1)} k\Psi(n,k) \\ &= \frac{1}{N(n)} \sum_{k=1}^{2(n-1)} k \sum_{s=1}^{n-1} 2^{n-s-1} \binom{2s}{k} + \frac{2}{N(n)} \sum_{k=1}^{2(n-1)} k \binom{n}{k} \\ &= \frac{1}{N(n)} \sum_{k=1}^{2(n-1)} k \sum_{s=1}^{n-1} 2^{n-s-1} \binom{2s}{k} + \frac{n2^n}{(2^n+1)2^{n-1}}.\end{aligned}$$

The second term is negligibly smaller than the first term. For the first term, by Lemmas 3.12 and 3.13, we have

$$\begin{aligned}&\sum_{k=1}^{2(n-1)} k \sum_{s=1}^{n-1} 2^{n-s-1} \binom{2s}{k} \\ &\quad= \sum_{s=1}^{n-1} 2^{n-s-1} \sum_{k=1}^{2(n-1)} k \binom{2s}{k} = \sum_{s=1}^{n-1} 2^{n-s-1} s 2^{2s} \\ &\quad= \sum_{s=1}^{n-1} s 2^{n+s-1} = 2^{n-1} \sum_{s=1}^{n-1} s 2^s \\ &\quad= 2^{n-1}(2^n(n-2)+2) = 2^{2n-1}(n-2) + 2^n.\end{aligned}$$

Thus,

$$\mu(n) \approx \frac{2^{2n-1}(n-2)+2^n}{N(n)} \approx \frac{2^{2n-1}(n-2)}{2^{2n-1}} = n-2.$$

Hence, we have the theorem. □

3.6.4 Proof of Theorem 3.5

The variance of the numbers of products in PreSOPs is

$$\begin{aligned}\sigma^2(n) &= \frac{1}{N(n)} \sum_{k=1}^{2(n-1)} k^2\Psi(n,k) - \mu^2(n) \\ &= \frac{G}{N(n)} - \mu^2(n).\end{aligned}$$

Note that by Theorem 3.3, we have

$$\begin{aligned} G &= \sum_{k=1}^{2(n-1)} k^2 \Psi(n,k) \\ &= \sum_{k=1}^{2(n-1)} k^2 \sum_{s=1}^{n-1} 2^{n-s-1} \binom{2s}{k} + 2 \sum_{k=1}^{2(n-1)} k^2 \binom{n}{k}. \end{aligned}$$

The first term in G is equal to

$$\begin{aligned} &\sum_{s=1}^{n-1} 2^{n-s-1} \sum_{k=1}^{2(n-1)} k^2 \binom{2s}{k} \\ &\quad = \sum_{s=1}^{n-1} 2^{n-s-1} 2s(2s+1) 2^{2s-2} \\ &\quad = \sum_{s=1}^{n-1} 2^{n+s-1} (s^2 + \frac{s}{2}) \\ &\quad = 2^{n-1} \sum_{s=1}^{n-1} s^2 2^s + 2^{n-2} \sum_{s=1}^{n-1} s 2^s. \end{aligned}$$

The second term in G is negligibly smaller than the first term, so we can ignore it. Thus, from Lemma 3.13, G is approximated by

$$2^{n-1}(2^n((n-1)^2 - 2(n-1) + 3) - 6) + 2^{n-2}((n-2)2^n)$$

or,

$$2^{2n-1}((n-1)^2 - 2(n-1) + 3) + 2^{2n-2}(n-2).$$

Thus, we have

$$\begin{aligned} \frac{G}{N(n)} &\approx \frac{2^{2n-1}((n-1)^2 - 2(n-1) + 3)}{2^{2n-1}} + \frac{n-2}{2} \\ &\approx (n-1)^2 - 2(n-1) + 3 + \frac{n-2}{2}. \end{aligned}$$

Hence,

$$\begin{aligned} \sigma^2(n) &\approx \frac{G}{N(n)} - (n-2)^2 \\ &\approx (n-1)^2 - 2(n-1) + 3 - (n-2)^2 + \frac{n-2}{2} \\ &\approx \frac{n+2}{2}. \end{aligned}$$

Thus, we have the theorem. □

3.7 Experimental Results

Although we have the formula for the number of products in MPreSOPs, we do not have one in MSOPs. To compare the numbers of products of PreSOPs with MSOPs [9], we generated all the interval functions for $n = 1$ to $n = 12$. To obtain exact minimum SOPs, we used ESPRESSO-EXACT [9]. In MSOPs, the maximums occur for $IN_0(n : 2^{n-3}, 7 \cdot 2^{n-3} - 1)$ and $IN_0(n : 2^{n-2}, 3 \cdot 2^{n-2} - 1)$ [41]. On the other hand, as shown in Section 3, in PreSOPs, the maximum occurs for $IN_0(n : 0, 2^n - 1)$.

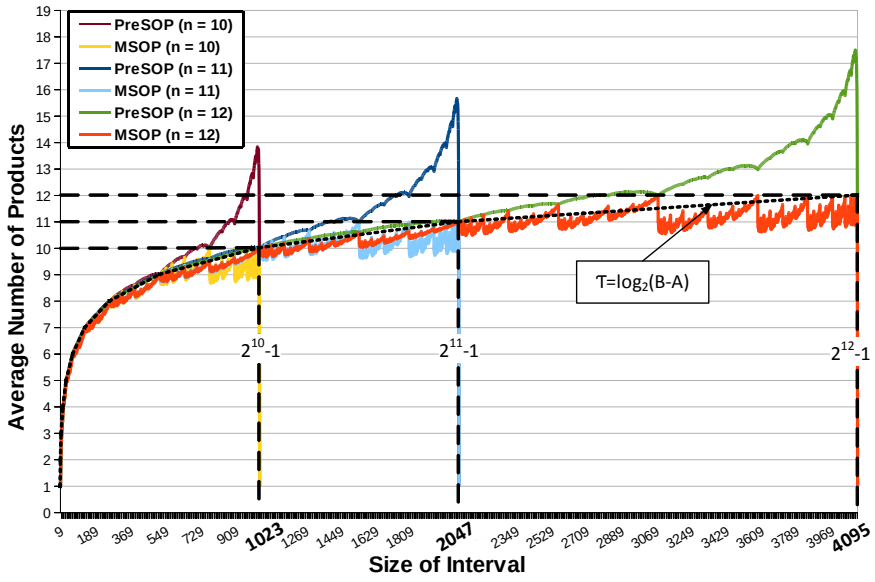


FIGURE 3.2: Average numbers of products in MSOPs and PreSOPs for different sizes of intervals

Fig. 3.2 compares average numbers of products in MSOPs and PreSOPs for different sizes of intervals. The curves for MSOPs tangent to the envelope $\tau = \log_2(B - A)$ and τ is the number of products in an MSOP. In Fig. 3.2 the envelope is shown by the black dotted curve. The peaks of the curves show that PreSOPs require more products (including the maximum, $\tau_p = 2(n - 1)$) when the size of interval approaches to $2^n - 2$, and the curves drop sharply when the sizes of intervals are $2^n - 1$ ($\tau_p = n$) and 2^n ($\tau_p = 1$). Table 3.4 shows the average numbers of products needed to represent interval functions. The ratio of the numbers of products in PreSOPs to that of MSOPs for $n > 5$ is about 1.05.

We calculated the probabilities of the interval functions that can be represented with at most τ_p products for $n = 8$ to $n = 16$. Table 3.5 shows that, for $n > 12$, more than 98.75% of the interval functions can be represented with $\lceil n + \sqrt{2(n+2)} - 2 \rceil$ products, and more than 99.9% of the interval functions can be represented with $\lceil \frac{3}{2}n - 1 \rceil$ products.

TABLE 3.4: Average numbers of products needed to represent interval functions

n	$N(n)$	MSOP	PreSOP	Difference	Ratio
4	136	2.39706	2.47794	0.08088	1.03374
5	528	3.12879	3.27462	0.14583	1.04661
6	2080	3.94327	4.15433	0.21106	1.05352
7	8256	4.81541	5.08539	0.26999	1.05607
8	32896	5.72671	6.04672	0.32001	1.05588
9	131328	6.66450	7.02535	0.36084	1.05414
10	524800	7.62032	8.01366	0.39334	1.05162
11	2098176	8.58858	9.00732	0.41874	1.04876
12	8390656	9.56540	10.00391	0.43850	1.04584

TABLE 3.5: Probabilities of interval functions that can be represented with at most τ_p products

n	$\tau_p = \lceil \mu(n) + c \rceil$		
	$c = \sigma(n)$	$c = 2\sigma(n)$	$c = \sigma^2(n)$
8	95.03587%	99.67169%	99.67169%
9	94.00204%	99.44642%	99.89416%
10	93.01715%	99.17569%	99.80526%
11	92.08265%	99.68811%	99.93380%
12	91.19742%	99.54448%	99.88636%
13	90.35921%	99.37685%	99.95988%
14	89.56528%	99.18796%	99.93412%
15	88.81274%	98.98061%	99.97608%
16	88.09875%	98.75744%	99.96191%

3.8 Conclusions and Comments

As key contributions of this chapter, we

1. showed that the MPreSOP and the number of products in the MPreSOP (τ_p) to represent an interval (A, B) can be directly obtained.

2. derived the formula $\Psi(n,\tau_p)$ for the number of interval functions that require τ_p products in PreSOPs.

3. showed by experiments that the average number of products in MSOPs is approximated by $\tau = \log_2(B - A)$.

4. showed that the average number of products needed to represent interval functions of an n-bit field by PreSOPs is $\mu(n) \approx n - 2$, with a variance $\sigma^2(n) \approx \frac{n}{2} + 1$.

5. showed by numerical computations that more than 99.9% of the interval functions of an n-bit field can be represented by PreSOPs with at most $\lceil \frac{3}{2}n - 1 \rceil$ products, when $n > 12$.

To represent an interval of an n-bit field, a PreSOP requires up to $2(n-1)$ products. When both the source and the destination ports have $n = 16$ bits fields, the number of products needed to represent one rule by a PreSOP can be up to $2(n-1) \times 2(n-1) = 30 \times 30 = 900$. Theorem 3.4 shows that the average number of products needed to represent an interval is about $n-2$. Thus, when the rule have two interval fields, the average numbers of products needed to represent one rule by PreSOP would be $(n-2)^2$, which is $14^2 = 196$ when $n = 16$. In the real applications, the rule expansion is not so large. For example, the paper [60] reported that the average number of TCAM entries for 12 real packet classifiers is 6.2 times of their number of rules. This is because, in the real applications, the distributions of the interval functions are not uniform.

Chapter 4

Derivation of Head-Tail Expressions for Interval Functions

This chapter shows a new method to represent interval functions called head-tail expressions (HTs). The head-tail expressions represent *greater-than* $GT(X : A)$ functions, *less-than* $LT(X : B)$ functions, and interval functions $IN_0(X : A, B)$ more efficiently than sum-of-products expressions, where n denotes the number of bits to represent the largest value in the interval (A, B). This paper proves that a head-tail expression (HT) represents an interval function with at most n words in a ternary content addressable memory (TCAM). It also shows the average numbers of factors to represent interval functions by HTs for up to $n = 16$, which were obtained by a computer simulation. It also conjectures that, for sufficiently large n, the average number of factors to represent n-variable interval functions by HTs is at most $\frac{2}{3}n - \frac{5}{9}$. Experimental results also show that, for $n \geq 10$, to represent interval functions, HTs require at least 20% fewer factors than MSOPs, on the average.

4.1 Introduction

Table 4.1 shows an example of a classification function. This function has two fields that correspond to the source and the destination ports represented by intervals. In Table 4.1, values are tested in a sequential manner from the top to the bottom. In a TCAM, the operation is equivalent to testing rows in a sequential order [26]. When each port is specified by either * (*don't care*) or a single value, each rule corresponds to one word in a TCAM. However, when a port is specified by an interval such as $(0, 65536)$, the interval must be represented by multiple words in a TCAM [12]. For example, the interval $(0, 65536)$ requires 16 words. Suppose that the header of incoming packets with source

port 1080 wants to access a destination with destination port 2080. As in Table 4.1, the header does not match to the first rule, but matches to the second rule, thus the action is *Accept* and the packet is sent to the destination.

TABLE 4.1: Example of classification function

Rule	Source Port	Destination Port	Action
1	$(0, 65536)$	6790	Accept
2	$(999, 2001)$	$(0, 5590)$	Accept
3	$*$	$*$	Deny

Table 4.2 compares our work with previous works, where n denotes the number of bits to represent the largest value in the interval. The first method [46] uses a special circuit to represent an interval directly. Thus, any interval can be represented by a single word. However, this method is the most expensive because it uses non-standard TCAMs.[1] The second method [41] uses an exact minimum sum-of-products expression (MSOPs) to represent an interval. This method uses standard TCAM, and any interval function can be represented with at most $2(n-2)$ products. Since we have to minimize TCAM words, this method is quite time consuming. The third method [37] uses output encoding. This method also uses a special circuit in addition to the TCAM, while it requires at most n words to represent an interval. The method proposed in this paper uses a head-tail expression (HT) [13] to represent an interval. This method requires a RAM in addition to the TCAM. Since HTs can be generated from the binary representations of endpoints of the intervals, time to generate HTs is quite short. The third method and our methods require the same number of TCAM words to represent a field. However, our method uses only standard components such as TCAM and RAM. On the other hand the method of [37] requires special hardware, which would be very expensive.

TABLE 4.2: Comparison with previous works

Parameter	Ref. [46]	Ref. [41]	Ref. [37]	This paper
Method	Comparator	MSOP	Output encoding	Head-tail expr.
Hardware	Special circuit	TCAM	TCAM + Special circuit	TCAM + RAM
Representation	Direct interval	n-bit non-prefix	$n+1$-bit prefix	n-bit prefix
Max. # of words to represent a field	1	$2(n-2)$	n	n
Cost	High	Low	High	Low

In this paper, we show a method to represent an interval function using a head-tail expression (HT). The head-tail expressions efficiently represent *greater-than* $GT(n : A)$ functions, *less-than* $LT(n : B)$ functions, and interval functions $IN_0(n : A, B)$. We prove that any interval function can be represented by an HT with at most n factors. We also show the average numbers of factors to represent interval functions by HTs for up to $n = 16$, which were obtained by a computer simulation. And, we conjecture that,

[1] Using non-standard LSIs is very expensive, since the development cost of such LSIs is very high, and the size of the market is not large enough to amortize the development cost.

for sufficiently large n, the average number of factors in HTs to represent n-variable interval functions is $\frac{2}{3}n - \frac{5}{9}$. By computer simulation, we also show that, for $n \geq 10$, to represent interval functions, HTs require at least 20% fewer factors than MSOPs, on the average.

This chapter is organized as follows: In Section 4.2, important words are defined and the basic properties of interval function are explained. In Section 4.3, a head-tail expression (HT) is introduced to represent GT, LT and IN_0 functions. In Section 4.4, experimental results are shown. Finally, in Section 4.5, the paper is concluded. A preliminary version of this chapter was presented in [50].

4.2 Definition and Basic Properties

In this section, we present definitions and basic properties before we step into the main contribution of this chapter *i.e.,* **head-tail expression**.

4.2.1 Prefix Sum-of-Products Expression

Definition 4.1. Let $\mathcal{F}$ be an SOP. $\tau(\mathcal{F})$ denotes the number of products in $\mathcal{F}$. $\tau_p(f)$ denotes the number of products in MPreSOP(f).

In general, an SOP requires fewer products than a PreSOP to represent the same function [41]. However, in the internet communication area, PreSOPs are used instead of SOPs, since PreSOPs can be quickly generated from the binary decision trees of the functions [60].

Lemma 4.1. *The minimum PreSOPs (MPreSOPs) of GT and LT functions can be represented as follows:*

$$GT(n:A) = (x_{n-1}\bar{a}_{n-1}) \vee \bigvee_{i=n-2}^{0} \left(\bigwedge_{j=n-1}^{i+1} x_j^{a_j} \right) x_i \bar{a}_i,$$

$$LT(n:B) = (\bar{x}_{n-1}b_{n-1}) \vee \bigvee_{i=n-2}^{0} \left(\bigwedge_{j=n-1}^{i+1} x_j^{b_j} \right) \bar{x}_i b_i,$$

where $\vec{a} = (a_{n-1}, \cdots, a_0)$ *and* $\vec{b} = (b_{n-1}, \cdots, b_0)$ *are the binary representations of* A *and* B*, repectively. GT and LT have* $\sum_{i=0}^{n-1} \bar{a}_i$ *and* $\sum_{i=0}^{n-1} b_i$ *disjoint products, respectively.*

Example 4.1. *When* $n = 4$ *and* $A = 0$*, we have* $\vec{a} = (0,0,0,0)$*. Thus,* $GT(4:0) = x_3 \vee x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_0\bar{x}_1\bar{x}_2\bar{x}_3$. ■

Example 4.2. *When* $n = 4$ *and* $B = 15$*, we have* $\vec{b} = (1, 1, 1, 1)$*. Thus,* $LT(n : 15) = \bar{x}_3 \vee \bar{x}_2 x_3 \vee \bar{x}_1 x_2 x_3 \vee \bar{x}_0 x_1 x_2 x_3$. ■

Theorem 4.1. *Let* $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ *and* $\vec{b} = (b_{n-1}, b_{n-2}, \cdots, b_1, b_0)$ *be the binary representations of* A *and* B*, respectively, and* $A < B$*. Let* s *be the largest index such that* $a_s \neq b_s$*. Then,* $IN_0(n : A, B)$ *can be represented by*

$$\bigvee_{i=s-1}^{0} \left[\left(\bigwedge_{j=n-1}^{i+1} x_j^{a_j} \right) x_i \bar{a}_i \vee \left(\bigwedge_{j=n-1}^{i+1} x_j^{b_j} \right) \bar{x}_i b_i \right].$$

The number of products is

$$\tau_p(IN_0(n : A, B)) = \sum_{i=0}^{s-1} (\bar{a}_i + b_i).$$

When $A = B - 1$ or $A + 1 = B$, the interval (A, B) has empty set, thus $IN_0(n : A, B)$ has no product (including the case when $s = 0$). When $A = -1$ and/or $B = 2^n$, these endpoints are called by **extremal endpoints**.

Lemma 4.2. *In the extremal endpoints, we have* $GT(n : -1) = LT(n : 2^n) = IN_0(n : -1, 2^n) = 1$*,* $IN_0(n : -1, B) = LT(n : B)$*, and* $IN_0(n : A, 2^n) = GT(n : A)$*.*

The optimality of $GT(n : A)$, $LT(n : B)$, and $IN_0(n : A, B)$ functions represented by PreSOPs has been discussed in Chapter 3.

Example 4.3. *Let* $A = 0$*,* $B = 31$ *and* $n = 5$*. In this case,* $\vec{a} = (0, 0, 0, 0, 0)$ *and* $\vec{b} = (1, 1, 1, 1, 1)$*. By Theorem 4.1, the PreSOP for* $IN_0(n : 0, 31)$ *is*

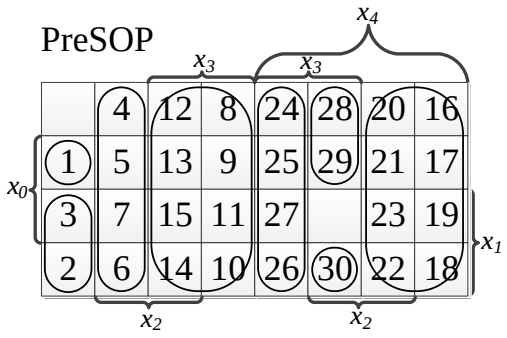
$$\bar{x}_4 \bar{x}_3 \bar{x}_2 \bar{x}_1 x_0 \vee x_4 x_3 x_2 x_1 \bar{x}_0 \vee \bar{x}_4 \bar{x}_3 \bar{x}_2 x_1 \vee x_4 x_3 x_2 \bar{x}_1$$
$$\vee \bar{x}_4 \bar{x}_3 x_2 \vee x_4 x_3 \bar{x}_2 \vee \bar{x}_4 x_3 \vee x_4 \bar{x}_3.$$

Fig. 4.1(a) shows its map. The integers in the map denote decimal representations of minterms, where $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$*. The PreSOP requires* $\tau_p(IN_0(n : 0, 31)) = 4 + 4 = 8$ *products.* ■

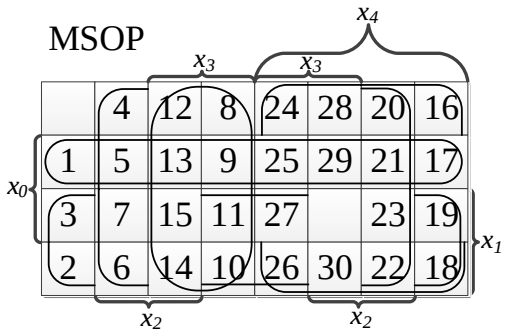
Note that an MSOP for $IN_0(n : 0, 31)$ is $\bar{x}_4 x_3 \vee \bar{x}_3 x_2 \vee \bar{x}_2 x_1 \vee \bar{x}_1 x_0 \vee \bar{x}_0 x_4$. Fig. 4.1(b) shows its map.

4.3 Head-Tail Expressions for Interval Functions

In this section, we use head-tail expressions (HTs) to represent interval functions. We use HTs [13] that were introduced to design NAND networks. Lemma 4.1 shows that



(a)



(b)

FIGURE 4.1: Maps for $IN_0(n : 0, 31)$

when the binary representation of A has t 0's, a PreSOP for $GT(n : A)$ requires t products. Especially when $a_{n-1} = a_{n-2} = \cdots = a_0 = 0$, the PreSOP requires n products. Similarly, it also shows that when the binary representation of B has t 1's, a PreSOP for $LT(n : B)$ requires t products, and n products when $b_{n-1} = b_{n-2} = \cdots = b_0 = 1$. Theorem 4.1 shows that when $a_{n-1} = a_{n-2} = \cdots = a_0 = 0$ and $b_{n-1} = b_{n-2} = \cdots = b_0 = 1$, the PreSOP for $IN_0(n : A, B)$ requires $2(n-1)$ products. Thus, if the PreSOP is used in a TCAM, we need up to $2(n-1)$ words.

However, the number of TCAM words can be reduced if we use the properties of a TCAM. We will show such a method in this section.

4.3.1 Derivation of Head-Tail Expressions for Interval Functions

Definition 4.2. A **head-tail expression** (HT) has a form

$$f = \bigvee_{i=t}^{0} \left[\bigwedge_{j=s}^{0} (\bar{h}_{ij}) \right] \left[\bigwedge_{k=v}^{0} (g_{ik}) \right], \tag{4.1}$$

where for $(i = 0, 1, \cdots, t)$, $(\bar{h}_{ij})$ is the **head factor** and (g_{ik}) is the **tail factor** and h_{ij} and g_{ik} are represented by products. In this thesis, (product) and $\overline{\text{(product)}}$ are called

factors. Products are used for PreSOPs and MSOPs, while factors are used for HTs. Both products and factors are realized in the form of **words** in TCAMs. Note that an SOP is considered as a special case of an HT.

Example 4.4. $\overline{(\bar{x}_6\bar{x}_5\bar{x}_4)} \cdot \overline{(\bar{x}_6\bar{x}_5x_4)} \cdot (x_3x_2) \vee \overline{(\bar{x}_6\bar{x}_5\bar{x}_4)} \cdot \overline{(\bar{x}_6\bar{x}_5x_4)} \cdot (\bar{x}_3\bar{x}_2)$ *is an HT.* ■

Lemma 4.3. *The circuit in Fig. 4.2 consisting of a TCAM and a RAM implements an HT.*

In Fig. 4.2, the circuit realizes the function $f = (\bar{h}_0)g_0 \vee (\bar{h}_1)g_1 \vee \cdots \vee (\bar{h}_t)g_t$. In this case, we assume that the functions represented by HTs are disjoint each other. Note that TCAM has a priority encoder in the output part [33, 43]. A factor corresponds to a word in a TCAM. Since the upper words have higher priority than the lower words, the TCAM will produce an action for the upmost matched word. Thus, in Fig. 4.2, if the input pattern mismatches h_0 and matches g_0, then the output is 1. However, if the input pattern matches both h_0 and g_0, then the output is 0. Thus, unlike Programmable Logic Arrays (PLAs) [40], the order of words in the TCAM is very important.

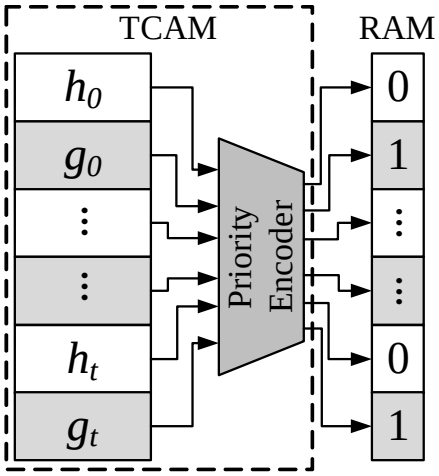


FIGURE 4.2: Circuit for a head-tail expressions

Lemma 4.4. *When $a_{m-1} = a_{m-2} = \cdots = a_{m-d} = 0$, and other bits are 1's, the GT function can be represented by the head-tail expression with two factors*

$$GT(n:A) = \overline{\left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \bigwedge_{i=m-1}^{m-d} \bar{x}_i\right)} \cdot \left(\bigwedge_{j=n-1}^{m} x_j^{a_j}\right),$$

where $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ is the binary representation of A, and d ($d \geq 1$) is the number of consecutive 0's in $\vec{a}$. When $n-1 < m$, the product $\bigwedge_{j=n-1}^{m} x_j^{a_j}$ is represented by the constant function 1.

Proof: In Lemma 4.1, let $a_{m-1} = a_{m-2} = \cdots = a_{m-d} = 0$ and let other bits be 1's. In this case, we have

$$\begin{aligned}
GT(n:A) &= \bigvee_{i=m-1}^{m-d} \left(\bigwedge_{j=n-1}^{i+1} x_j^{a_j} \right) x_i \\
&= \bigvee_{i=m-1}^{m-d} \left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \right) \left(\bigwedge_{k=m-1}^{i+1} x_k^{a_k} \right) x_i \\
&= \bigvee_{i=m-1}^{m-d} \left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \right) x_i \\
&= \left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \right) \cdot \left(\bigvee_{i=m-1}^{m-d} x_i \right) \\
&= \left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \right) \cdot \overline{\left(\bigwedge_{i=m-1}^{m-d} \bar{x}_i \right)} \\
&= \left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \right) \cdot \left(\overline{\left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \right)} \vee \overline{\left(\bigwedge_{i=m-1}^{m-d} \bar{x}_i \right)} \right) \\
&= \overline{\left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \bigwedge_{i=m-1}^{m-d} \bar{x}_i \right)} \cdot \left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \right).
\end{aligned}$$

Thus, we have the lemma. In this case: $(\bar{h}_1) = \overline{\left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \bigwedge_{i=m-1}^{m-d} \bar{x}_i \right)}$ is the head factor, and $(g_1) = \left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \right)$ is the tail factor. Note that when $n-1 < m$, the product $\bigwedge_{j=n-1}^{m} x_j^{a_j}$ is represented by the constant function 1. □

Example 4.5. *Let $A = 0$. The binary representation of A is $\vec{a} = (0,0,0,0)$. By Lemma 4.4, we have a group of consecutive 0's, where $m = 4$ and $d = 4$. Thus,*

$$\begin{aligned}
GT(4:0) &= \overline{\left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \bigwedge_{i=m-1}^{m-d} \bar{x}_i \right)} \cdot \left(\bigwedge_{j=n-1}^{m} x_j^{a_j} \right) \\
&= \overline{\left(\bar{x}_3 \bar{x}_2 \bar{x}_1 \bar{x}_0 \right)} \cdot (1).
\end{aligned}$$

It requires two factors. ■

Lemma 4.5. *When $b_{m-1} = b_{m-2} = \cdots = b_{m-d} = 1$, and other bits are 0's, the LT function can be represented by the head-tail expression with two factors*

$$LT(n:B) = \overline{\left(\bigwedge_{j=n-1}^{m} x_j^{b_j} \bigwedge_{i=m-1}^{m-d} x_i \right)} \cdot \left(\bigwedge_{j=n-1}^{m} x_j^{b_j} \right),$$

where $\vec{b} = (b_{n-1}, b_{n-2}, \cdots, b_1, b_0)$ *is the binary representation of* B*, and* d $(d \geq 1)$ *is the number of consecutive 1's in* $\vec{b}$*. When* $n-1 < m$*, the product* $\bigwedge_{j=n-1}^{m} x_j^{b_j}$ *is represented by the constant function* 1*.*

Proof: The proof is similar to that of Lemma 4.4. □

Lemmas 4.4 and 4.5 are just for interval functions with a special property. The generalization of GT and LT functions represented by head-tail expressions are:

Lemma 4.6. *A greater-than (GT) or a less-than (LT) functions can be represented by the head-tail expression:*

$$\bigvee_{i=p-1}^{0} \left(\bigwedge_{j=q-1}^{0} \bar{h}_{ij} \right) (g_i),$$

where p*,* q *are integers and* $p \leq n$*.*

As explained before, a PreSOP is a special case of HTs, thus an interval function can be represented by an HT. Note that an interval function can be *segmented* into smaller interval functions which are represented by HTs. Each HT consists of head factors and a tail factor.

Lemmas 4.4 and 4.5 are used to represent only one group of consecutive 0's or 1's. When multiple groups of consecutive 0's or 1's exist in $\vec{a}$ or $\vec{b}$, Theorems 4.2 and 4.3 are used.

Definition 4.3. In two logic function h and g, if $g(x) = 1$ for all x such that $h(x) = 1$, then g **includes** h, denoted by $h \subseteq g$.

Lemma 4.7. *Let* $\subseteq$ *denote the inclusion relation. If* $h_0 \subseteq g_0 \subseteq h_1 \subseteq g_1 \subseteq \cdots \subseteq h_{p-2} \subseteq g_{p-2} \subseteq h_{p-1} \subseteq g_{p-1}$*, then* $Z = g_0\bar{h}_0 \vee g_1\bar{h}_1 \vee \cdots \vee g_{p-2}\bar{h}_{p-2} \vee g_{p-1}\bar{h}_{p-1}$ *is represented by:*

$$Z = \bar{h}_0(\bar{h}_1 \vee g_0)(\bar{h}_2 \vee g_1) \cdots (\bar{h}_{p-2} \vee g_{p-3})(\bar{h}_{p-1} \vee g_{p-2})g_{p-1}$$

Proof: The grey area in the map of Fig. 4.3 indicates the covering of Z. Thus, we have the lemma. □

Theorem 4.2. *Let* $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ *be the binary representation of an integer* A*. Let* $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$ *be the starting indexes of consecutive* 0*'s groups in* $\vec{a}$*, where* $c_{p-1} > c_{p-2} > \cdots > c_1 > c_0$*. Let the isolated* 1*'s be* $a_{c_{p-2}+1} = a_{c_{p-3}+1} = \cdots = a_{c_1+1} = a_{c_0+1} = 1$*, where* $c_k + 1$ *is the index of isolated* 1*'s among groups of consecutive*

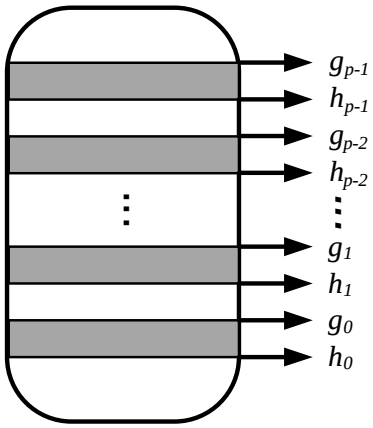


FIGURE 4.3: Map for Lemma 4.7

0*'s in* $\vec{a}$*. Then, the* $GT(n:A)$ *can be represented by* $p+1$ *factors:*

$$\overline{\left(\bigwedge_{j=n-1}^{c_0+1} x_j^{a_j} \bigwedge_{i=c_0}^{c_0+1-d_0} \bar{x}_i\right)} \cdot \overline{\left(\bigwedge_{j=n-1}^{c_1+1} x_j^{a_j} \bigwedge_{i=c_1}^{c_1-d_1} \bar{x}_i\right)} \cdots \cdot \overline{\left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{a_j} \bigwedge_{i=c_{p-1}}^{c_{p-1}-d_{p-1}} \bar{x}_i\right)} \cdot \left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{a_j}\right),$$

where $d_{p-1}, d_{p-2}, \cdots, d_1, d_0$ *(for* $i = 0, 1, \cdots, p-1$, $d_i \geq 1$*) are numbers of consecutive* 0*'s in the groups which start from the indexes* $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$*, respectively. Note that, in* $\vec{a}$*, except for the group of consecutive* 0*'s, remaining bits are* 1*'s.*

Proof: Lemma 4.4 and Lemma 4.6 are used to represent p terms of head-tail expressions. Each group of consecutive 0's in $\vec{a}$ can be represented by:

$$\bar{h}_0 g_0 = \overline{\left(\bigwedge_{j=n-1}^{c_0+1} x_j^{a_j} \bigwedge_{i=c_0}^{c_0+1-d_0} \bar{x}_i\right)} \cdot \left(\bigwedge_{j=n-1}^{c_0+1} x_j^{a_j}\right)$$

$$\cdots$$

$$\bar{h}_{p-1} g_{p-1} = \overline{\left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{a_j} \bigwedge_{i=c_{p-1}}^{c_{p-1}+1-d_{p-1}} \bar{x}_i\right)} \cdot \left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{a_j}\right),$$

where $h_0 \subseteq g_0 \subseteq \cdots \subseteq h_{p-1} \subseteq g_{p-1}$. The number of required factors is $2p$. Since the starting index of a group of consecutive 0's is c_k, the relation between m in Lemma 4.4 and c_k is $m = c_k + 1$ and $m - d_k = c_k + 1 - d_k$. Moreover, the index of isolated 1 satisfies the relation $c_k - d_k = c_{k-1} + 1$. Thus, we have $x_{c_k-d_k}^{a_{c_k-d_k}} = x_{c_{k-1}+1}^{a_{c_{k-1}+1}} = x_{c_k-d_k} = x_{c_{k-1}+1}$,

where the binary representation of A is:

$$\vec{a} = (\cdots, \overset{a_{c_k}}{\underset{\downarrow}{0}}, 0, \cdots, \overset{a_{c_k+1-d_k}}{\underset{\downarrow}{0}}, \overset{a_{c_k-d_k}=a_{c_{k-1}+1}}{\underset{\downarrow}{1}}, \overset{a_{c_{k-1}}}{\underset{\downarrow}{0}}, 0, \cdots)$$

Therefore, $\bar{h}_k \vee g_{k-1}$ can be combined to a factor:

$$\begin{aligned}
\bar{h}_k \vee g_{k-1} &= \overline{\left(\bigwedge_{j=n-1}^{c_k+1} x_j^{a_j} \bigwedge_{i=c_k}^{c_k+1-d_k} \bar{x}_i\right)} \vee \left(\bigwedge_{j=n-1}^{c_{k-1}+1} x_j^{a_j}\right) \\
&= \overline{\left(\bigwedge_{j=n-1}^{c_k+1} x_j^{a_j} \bigwedge_{i=c_k}^{c_k+1-d_{k-1}} \bar{x}_i\right)} \\
&\quad \vee \left(\bigwedge_{j=n-1}^{c_k+1} x_j^{a_j} \bigwedge_{i=c_k}^{c_k+1-d_k} \bar{x}_i\right) \cdot x_{c_{k-1}+1} \\
&= \overline{\left(\bigwedge_{j=n-1}^{c_k+1} x_j^{a_j} \bigwedge_{i=c_k}^{c_k+1-d_{k-1}} \bar{x}_i\right)} \vee x_{c_{k-1}+1} \\
&= \overline{\left(\bigwedge_{j=n-1}^{c_k+1} x_j^{a_j} \bigwedge_{i=c_k}^{c_k+1-d_k} \bar{x}_i\right)} \vee x_{c_k-d_k} \\
&= \overline{\left(\bigwedge_{j=n-1}^{c_k+1} x_j^{a_j} \bigwedge_{i=c_k}^{c_k-d_k} \bar{x}_i\right)}.
\end{aligned}$$

In this case, $2p$ factors can be reduced to only $p+1$ factors. Thus, we have the theorem. □

Theorem 4.3. *Let $\vec{b} = (b_{n-1}, b_{n-2}, \cdots, b_1, b_0)$ be the binary representation of an integer B. Let $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$ be the starting indexes of consecutive 1's groups in $\vec{b}$, where $c_{p-1} > c_{p-2} > \cdots > c_1 > c_0$. Let the isolated 0's be $b_{c_{p-2}+1} = b_{c_{p-3}+1} = \cdots = b_{c_1+1} = b_{c_0+1} = 0$, where $c_k + 1$ is the index of isolated 0's among groups of consecutive 1's in $\vec{b}$. In this case, $LT(n:B)$ can be represented by $p+1$ factors:*

$$\begin{aligned}
&\overline{\left(\bigwedge_{j=n-1}^{c_0+1} x_j^{b_j} \bigwedge_{i=c_0}^{c_0+1-d_0} x_i\right)} \cdot \overline{\left(\bigwedge_{j=n-1}^{c_1+1} x_j^{b_j} \bigwedge_{i=c_1}^{c_1-d_1} x_i\right)} \cdots \\
&\qquad \cdot \overline{\left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{b_j} \bigwedge_{i=c_{p-1}}^{c_{p-1}-d_{p-1}} x_i\right)} \cdot \left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{b_j}\right),
\end{aligned}$$

where $d_{p-1}, d_{p-2}, \cdots, d_1, d_0$ (for $i = 0, 1, \cdots, p-1$, $d_i \geq 1$) are numbers of consecutive 1's in the groups which start from the indexes $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$, respectively. Note that, in $\vec{b}$, except for the group of consecutive 1's, remaining bits are 0's.

Proof: The proof is similar to that of Theorem 4.2. □

Theorems 4.2 and 4.3 are useful when there are multiple groups consecutive 0's or consecutive 1's, and each group of consecutive 0's or 1's is separated by a single 1 in a GT, or a single 0 in an LT functions.

Lemma 4.8. *Let* $-1 \leq A < B \leq 2^n$, *an interval function* $IN_0(n : A, B)$ *can be represented by*

$$\begin{aligned} IN_0(n : A, B) &= GT(n : A) \cdot \overline{GT(n : B-1)} \\ &= \overline{LT(n : A+1)} \cdot LT(n : B). \end{aligned}$$

Proof: An interval function IN_0 can be represented by a AND of GT and LT functions

$$IN_0(n : A, B) = GT(n : A) \cdot LT(n : B).$$

Since $GT(n : A) = \overline{LT(n : A+1)}$ and $LT(n : B) = \overline{GT(n : B-1)}$, we have the lemma. □

Example 4.6. *Represent* $IN_0(n : -1, 4)$ *and* $IN_0(n : 3, 8)$ *by GT and/or LT functions.*

$$\begin{aligned} IN_0(n : -1, 4) &= GT(n : -1) \cdot \overline{GT(n : 3)} \\ &= \overline{LT(n : 0)} \cdot LT(n : 4) \\ IN_0(n : 3, 8) &= GT(n : 3) \cdot \overline{GT(n : 7)} \\ &= \overline{LT(n : 4)} \cdot LT(n : 8) \end{aligned}$$

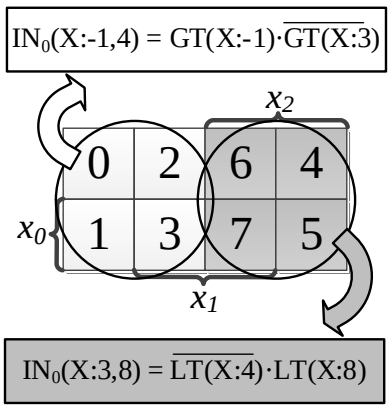


FIGURE 4.4: Example of Lemma 4.8

Fig. 4.4 illustrates Lemma 4.8. The white part corresponds to an expression for $IN_0(n : -1, 4)$, *while the grey part corresponds to an expression for* $IN_0(n : 3, 8)$. ■

4.3.2 Examples of Head-Tail Expressions for Interval Functions

Example 4.7. *Represent* $IN_0(n : 0, 15)$ *using a head-tail expression. Since the largest value is 15, we have* $n = 4$. *Binary representations of* $A = 0$ *and* $B = 15$ *are* $\vec{a} = (0, 0, 0, 0)$ *and* $\vec{b} = (1, 1, 1, 1)$, *respectively. The largest index such that* $a_s \neq b_s$ *is* $s = 3$. *So, in Theorem 4.1, the products are produced from the index* $s - 1$ *which restricts the GT for only in* $x_{n-1}^{a_{n-1}} x_{n-2}^{a_{n-2}} \cdots x_s^{a_s}$ *area, and the LT for only in* $x_{n-1}^{b_{n-1}} x_{n-2}^{b_{n-2}} \cdots x_s^{b_s}$ *area. We have* $n = 4$ *and there are three consecutive* 0*'s in* $\vec{a}$. *Thus, the interval function can be represented by*

$$\bar{x}_3 \cdot GT(n : 0) \vee x_3 \cdot LT(n : 15).$$

By applying Lemma 4.4 and the restriction $\bar{x}_3$ *with* $m = 3$ *and* $d = 3$, *we have*

$$\begin{aligned} \bar{x}_3 \cdot GT(n : 0) &= \overline{\left(\bigwedge_{j=3}^{3} x_j^{a_j} \bigwedge_{i=2}^{0} \bar{x}_i \right)} \cdot \left(\bigwedge_{j=3}^{3} x_j^{a_j} \right) \\ &= \overline{(\bar{x}_3 \bar{x}_2 \bar{x}_1 \bar{x}_0)} \cdot \bar{x}_3. \end{aligned}$$

We have three consecutive 1*'s in* $\vec{b}$ *and let* $m = 3$ *and* $d = 3$. *By applying Lemma 4.5 and the restriction* x_3, *we have*

$$\begin{aligned} x_3 \cdot LT(n : 15) &= \overline{\left(\bigwedge_{j=3}^{3} x_j^{b_j} \bigwedge_{i=2}^{0} x_i \right)} \cdot \left(\bigwedge_{j=3}^{3} x_j^{b_j} \right) \\ &= \overline{(x_3 x_2 x_1 x_0)} \cdot x_3. \end{aligned}$$

The maps for $IN_0(n : 0, 15)$ *are shown in Fig. 4.5. The top row shows the PreSOP, which requires 6 products. The middle row shows an HT. The three products produced by three consecutive* 0*'s in* $\vec{a}$ *in a PreSOP are converted into a head-tail term in the left. Likewise, the three products produced by three consecutive* 1*'s in* $\vec{b}$ *in a PreSOP are converted into a head-tail term in the right. Each HT uses two words and the total number of words is four. However, when an HT whose term shares literals of a variable (*$\bar{x}_3$ *and* x_3*), the HT can be reduced as shown in the bottom row of Fig. 4.5. This expression still needs a product to represent the universe, which is indicated by the constant 1 in the bottom row of Fig. 4.5. Table 4.3 shows realizations of the function, where the TCAM stores the words and the RAM stores the actions. Table 4.3(a) corresponds to the top row of Fig. 4.5, Table 4.3(b) corresponds to the middle row of Fig. 4.5, and Table 4.3(c) corresponds to the bottom row of Fig. 4.5.* ■

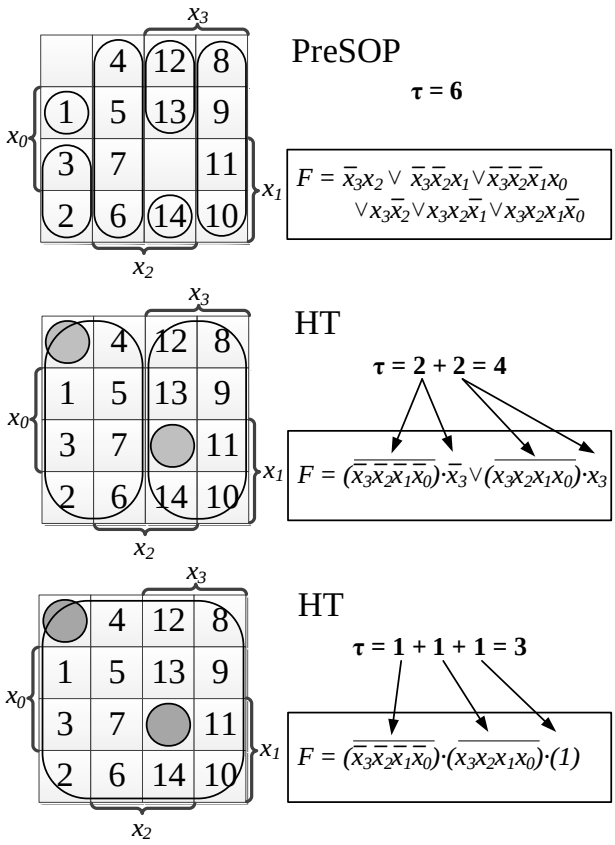


FIGURE 4.5: Derivation of a head-tail expression for $IN_0(n : 0, 15)$

TABLE 4.3: Realization of $IN_0(n : 0, 15)$ by TCAM and RAM

(a)

TCAM	RAM
0001	1
001*	1
01**	1
10**	1
110*	1
1110	1
****	0

(b)

TCAM	RAM
0000	0
0***	1
1111	0
1***	1

(c)

TCAM	RAM
0000	0
1111	0
****	1

Example 4.8. *Represent* $IN_0(n : 0, 27)$ *by an HT. Binary representations of* $A = 0$ *and* $B = 27$ *are* $\vec{a} = (0, 0, 0, 0, 0)$ *and* $\vec{b} = (1, 1, 0, 1, 1)$*, respectively. To represent the function, we use Lemma 4.8:*

$$IN_0(n : 0, 27) = \overline{LT(n : 1)} \cdot LT(n : 27).$$

By Lemma 4.1, we know that $\overline{LT(n : 1)} = \overline{(\bar{x}_4\bar{x}_3\bar{x}_2\bar{x}_1\bar{x}_0)}$*. The next step is to derive* $LT(n : 27)$*.* $\vec{b}$ *has two consecutive groups of* 1*'s and a single isolated* 0 *between them.*

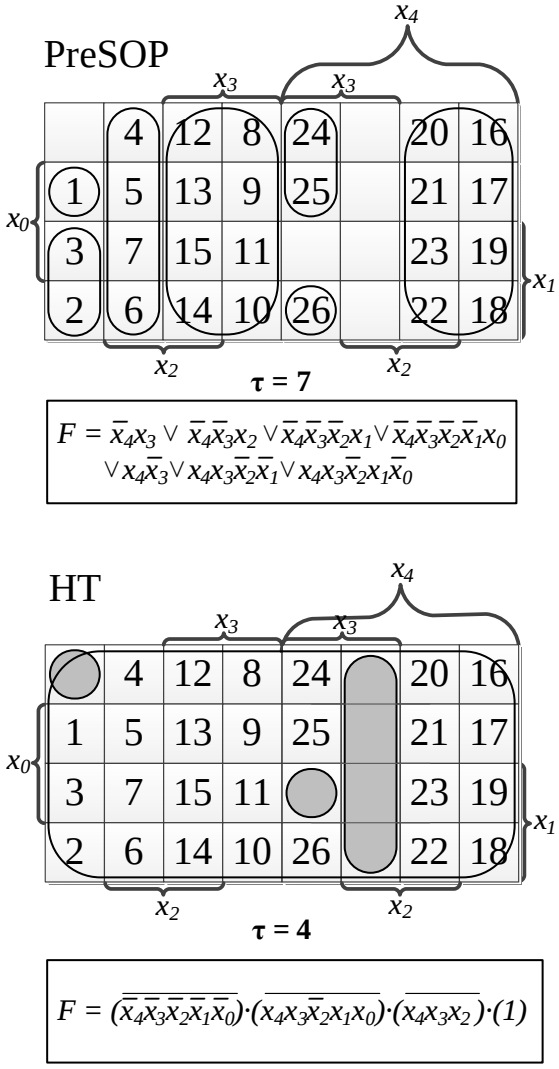


FIGURE 4.6: Maps for PreSOP and head-tail expression representing $IN_0(n : 0, 27)$

The first consecutive group of 1*'s starts from the index* $c_1 = 4$ *and the second group starts from* $c_0 = 1$*, and* $d_0 = d_1 = 2$*. By Theorem 4.3 we can represent it by an HT:*

$$LT(n : 27) = \overline{\left(\bigwedge_{j=4}^{2} x_j^{b_j} \bigwedge_{i=1}^{0} x_i\right)} \cdot \overline{\left(\bigwedge_{i=4}^{2} x_i\right)} \cdot (1)$$
$$= \overline{(x_4 x_3 \bar{x}_2 x_1 x_0)} \cdot \overline{(x_4 x_3 x_2)} \cdot (1).$$

The reduced HT for the interval function is

$$IN_0(n : 0, 27) = \overline{(\bar{x}_4 \bar{x}_3 \bar{x}_2 \bar{x}_1 \bar{x}_0)} \cdot \overline{(x_4 x_3 \bar{x}_2 x_1 x_0)}$$
$$\cdot \overline{(x_4 x_3 x_2)} \cdot (1).$$

Fig. 4.6 shows the maps for $IN_0(n : 0, 27)$*. The top row shows the PreSOP which requires*

7 products. The bottom row shows the HT which requires four factors. Table 4.4 shows the realization of the function by using a TCAM and a RAM with four words. ■

TABLE 4.4: Realization of $IN_0(n:0,27)$ in TCAM and RAM

TCAM	RAM
00000	0
11011	0
111**	0
*****	1

4.3.3 The Number of Factors to Represent an Interval Function by a Head-Tail Expression

Definition 4.4. Let $\zeta(f)$ be the minimum number of factors to represent a function f by an HT.

Lemma 4.9.

$$\zeta(GT(n:A)) \leq \lceil \frac{n+1}{2} \rceil, \textit{and}$$
$$\zeta(LT(n:B)) \leq \lceil \frac{n+1}{2} \rceil.$$

Proof: Consider a binary representation $\vec{a}$ that makes the number of factors in an HT for a GT function maximum. If there is three or more consecutive 0's in $\vec{a}$, then we can reduce the number of factors in the HT, by Theorem 4.2. Note that when more than one groups of consecutive 0's exist in arbitrary location in GT, we can use Lemma 4.6 to segment each group into an HT by Theorem 4.2.

In Theorem 4.2, regardless the number of 0's in each group, if there are p groups, the number of factors is $p+1$. For instance, if we have a group with two or more consecutive 0's, the number of factors are the same $p+1=2$. Note that when only a group with two consecutive 0's exists, the number of factors are not reduced by Theorem 4.2. So, to avoid such reduction of the factors and to get the maximum number of factors in an HT, one possibility is by alternating 0 and 1 in the binary representation. Another possibility is by alternating two consecutive 0's and two consecutive 1's in the binary representation[3]. In these cases, we have at least $\lceil \frac{n}{2} \rceil$ zeros, and their numbers of factors in HTs are the same as the numbers of products in PreSOPs which are $\lceil \frac{n}{2} \rceil$. Thus, the

[3]The numbers of factors in HTs for GT functions become maximum for various cases. Other cases occur when the binary representations are the combination of alternating 0, 1, two consecutive 0's, and two consecutive 1's in which the numbers of 0's are equal to or greater than $\lceil \frac{n}{2} \rceil$ and Theorem 4.2 cannot be used to reduce the numbers of factors.

maximum number of factors in an HT for a GT function occurs when the number of 0's is equal to or greater than $\lceil \frac{n}{2} \rceil$, and Theorem 4.2 cannot be used to reduce the number of factors. The argument for LT functions is similar.

When n is odd: Suppose that the number of the factors to represent a GT function takes its maximum when $\vec{a} = (0, 1, 0, 1, \cdots, 1, 0)$. The number of 0's is $\frac{n+1}{2}$, the number of 1's is $\frac{n-1}{2}$, and no consecutive 0's exist.

By Lemma 4.1, the number of products for GT is bounded above by $\sum \bar{a}_i$. So, when the number of 0's is equal to or less than $\frac{n+1}{2}$, the lemma holds. When the number of 0's is greater than $\frac{n+1}{2}$, there exist consecutive 0's in the sequence of a_i. In this case, we can apply Theorem 4.2 iteratively to reduce the number of factors and apply Lemma 4.6 to construct GT. In both cases, the number of factors does not exceed $\frac{n+1}{2}$. Thus, we have

$$\zeta(GT(n : A)) \leq \frac{n+1}{2}.$$

The argument for the number of the factors for an LT function is similar. When $\vec{b} = (1, 0, 1, 0, \cdots, 0, 1)$, the number of 1's is $\frac{n+1}{2}$, the number of 0's is $\frac{n-1}{2}$, and no consecutive 1's exist. When the number of 1's is equal to or greater than $\frac{n+1}{2}$, Theorem 4.3 are used iteratively to reduce the number of factors, we have

$$\zeta(LT(n : B)) \leq \frac{n+1}{2}.$$

When n is even: When $\vec{a} = (0, 1, 0, 1, \cdots, 0, 1)$, the numbers of 0's and 1's in GT are the same which are $\frac{n}{2}$. However, this does not make the number of factors in an HT maximum which is $\frac{n}{2} + 1$. The number of factors becomes its maximum when $\vec{a} = (0, 1, 0, 1, \cdots, 0, 1, 0, 0)$. The last two components are $a_1 = a_0 = 0$ $(d = 2)$ that produce two factors as explained in Lemma 4.4. Thus, we have

$$\zeta(GT(n : A)) \leq \frac{n}{2} + 1.$$

Likewise, for LT, the number of factors becomes its maximum when $\vec{b} = (1, 0, 1, 0, \cdots, 1, 0, 1, 1)$. Thus, we have

$$\zeta(LT(n : B)) \leq \frac{n}{2} + 1.$$

Combining these two cases, we have the lemma. □

Lemma 4.9 can be extended to an interval function:

Theorem 4.4.

$$\zeta(IN_0(n:A,B)) \leq n$$

Proof: Let $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ and $\vec{b} = (b_{n-1}, b_{n-2}, \cdots, b_1, b_0)$ be binary representations of A and B, respectively, and $A < B$. According to Theorem 4.1, if the most significant bits (MSBs) are the same, then we can ignore the MSBs and consider the function with fewer variables. Assume that the MSB is different, *i.e.,* $a_{n-1} = 0$ and $b_{n-1} = 1$. The function can be expanded into

$$IN_0(n:A,B) = \bar{x}_{n-1} GT(n-1:A) \vee x_{n-1} LT(n-1:B)$$

When n is odd: Let $a_{n-1} = 0$ and $b_{n-1} = 1$. Consider the case when $\zeta(GT(n-1:A))$ and $\zeta(LT(n-1:B))$ take their maximum values. Note that, according to Theorem 4.1, we only consider bits after the s'th of the binary representation such that $a_s \neq b_s$. By Lemma 4.9 (the *even* part), the vectors are $\vec{a} = (0,0,1,0,1,\cdots,0,1,0,0)$ and $\vec{b} = (1,1,0,1,0,\cdots,1,0,1,1)$, where $s = n-1$. In this case, we can apply Theorem 4.2 and Theorem 4.3 iteratively to obtain GT and LT functions. Note that there are literals of the same variable in both HTs. They are $g_1 = \bar{x}_{n-1}$ and $g_2 = x_{n-1}$. In this case, we can combine the literals of both tail factors to form one factor as follows:

$$\begin{aligned}(\bar{h}_{1_p}\bar{h}_{1_{p-1}}\cdots\bar{h}_{1_1})\bar{x}_{n-1} \vee (\bar{h}_{2_q}\bar{h}_{2_{q-1}}\cdots\bar{h}_{2_1})x_{n-1} \\ = (\bar{h}_{1_p}\bar{h}_{1_{p-1}}\cdots\bar{h}_{1_1}) \cdot (\bar{h}_{2_q}\bar{h}_{2_{q-1}}\cdots\bar{h}_{2_1}) \cdot (1).\end{aligned} \tag{4.2}$$

Thus, by Lemma 4.9, we have

$$\begin{aligned}\zeta(IN_0(n:A,B)) &= \zeta(GT(n-1:A)) + \zeta(LT(n-1:B)) \\ &\leq \lceil \frac{n}{2} \rceil + \lceil \frac{n}{2} \rceil = n+1.\end{aligned}$$

Moreover, by Eq. (4.2), we have

$$\zeta(IN_0(n:A,B)) \leq n+1-1 = n.$$

When n is even: The HTs for both GT and LT functions contribute and we have

$$\begin{aligned}\zeta(IN_0(n:A,B)) &= \zeta(GT(n-1:A)) + \zeta(LT(n-1:B)) \\ &\leq \lceil \frac{(n-1)+1}{2} \rceil + \lceil \frac{(n-1)+1}{2} \rceil = n.\end{aligned}$$

Thus, we have the theorem. □

4.4 Experimental Results

We developed a heuristic algorithm [51] to generate HTs for interval functions that uses the properties of Lemma 4.4, Lemma 4.5, Theorem 4.2, and Theorem 4.3. By the computer program, we represented all the interval functions for $n = 1$ to $n = 16$ by HTs. There are $N(n) = (2^n+1)(2^{n-1})$ distinct interval functions of n variables. When $n = 16$, the total number of the distinct interval functions is approximately $2^{31} \approx 2.147 \times 10^9$.

TABLE 4.5: Numbers of $GT(n : A)$ or $LT(n : B)$ functions requiring τ factors in HTs for $n = 1$ to $n = 16$ produced by a heuristic algorithm

n	# Factors (τ)								
	1	2	3	4	5	6	7	8	9
1	2								
2	3	1							
3	4	4							
4	5	9	2						
5	6	16	10						
6	7	25	28	4					
7	8	36	60	24					
8	9	49	110	80	8				
9	10	64	182	200	56				
10	11	81	280	420	216	16			
11	12	100	408	784	616	128			
12	13	121	570	1344	1456	560	32		
13	14	144	770	2160	3024	1792	288		
14	15	169	1012	3300	5712	4704	1408	64	
15	16	196	1300	4840	10032	10752	4992	640	
16	17	225	1638	6864	16632	22176	14400	3456	128

Table 4.5 shows the distribution of GT or LT functions that require τ factors in HTs for up to $n = 16$ produced by the heuristic algorithm [51]. As shown in the table, to represent a GT or an LT function, $\frac{n+1}{2}$ words are necessary when n is odd, and $\frac{n}{2}+1$ words are necessary when n is even. For GT and LT functions, the heuristic program generates exact minimum HTs.

Table 4.6 shows the distribution of interval functions that require τ factors in HTs for up to $n = 16$ produced by the heuristic algorithm. It shows that with an HT, any interval functions can be represented with at most n factors.

Let $\mu_h(n)$ be the average number of factors to represent n-variable interval functions by HTs produced by the heuristic algorithm. Table 4.7 shows $\mu_h(n)$ for $n = 1$ to $n = 16$. We represented all the interval functions by HTs generated by the heuristic algorithm [51]. Thus, they may not be minimum. Since $(\frac{2}{3}n - \frac{5}{9})/\mu_h(n)$ approaches to 1.00 with the increase of n, we have the following:

TABLE 4.6: Numbers of n-variable interval functions requiring τ factors in HTs for $n = 1$ to $n = 16$ produced by a heuristic algorithm

n	# Factors (τ)							
	1	2	3	4	5	6	7	8
1	3							
2	7	3						
3	15	16	5					
4	31	51	42	12				
5	63	132	181	124	28			
6	127	307	574	644	364	64		
7	255	672	1537	2384	2240	1024	144	
8	511	1419	3714	7220	9504	7424	2784	320
9	1023	2932	8405	19212	32204	35968	23520	7360
10	2047	5979	18222	46844	93996	136016	129408	71744
11	4095	12096	38401	107504	247200	435200	544816	445504
12	8191	24355	79426	236444	603152	1236272	1910016	2080768
13	16383	48900	162277	504700	1393148	3217408	5866784	7980416
14	32767	98019	328926	1054932	3090572	7840416	16323584	26503616
15	65535	196288	663329	2173104	6655392	18175744	42109520	78930432
16	131071	392859	1333410	4431844	14023392	40562080	102439456	216008512
n	# Factors (τ)							
	9	10	11	12	13	14	15	16
1								
2								
3								
4								
5								
6								
7								
8								
9	704							
10	19008	1536						
11	211904	48128	3328					
12	1476288	608768	119808	7168				
13	7620096	4731904	1707264	293888	15360			
14	31901824	26889216	14729728	4687872	711680	32768		
15	114450048	122574848	91806208	44679168	12634112	1703936	69632	
16	364880640	474360832	454498304	304347136	132431872	33488896	4038656	147456

Conjecture 4.1. *For sufficiently large n, the average number of factors to represent n-variable interval functions is at most $\frac{2}{3}n - \frac{5}{9}$.*

We also obtained $\mu_s(n)$, the average numbers of products to represent n-variable interval functions by exact MSOPs, using exact algorithm for $n = 1$ to $n = 14$. The fourth column of Table 4.7 shows values of $\mu_s(n)$. The first experiment, for $n = 1$ to $n = 13$, we used Intel Dual2Duo 3.0 GHz microprocessor with 8 GB memory. We generated all the interval functions and minimized them using ESPRESSO-EXACT [9] which obtains exact minimum SOPs. For $n = 13$, to obtain $\mu_s(13)$, it took one month. The second one, for $n = 14$, we used Intel Xeon 8-core 2.27 GHz microprocessors with 24 GB memory and paralleled the program into 8 parts, and the computation took nearly a month. By using the same method, for $n = 16$, it would take a few years to obtain $\mu_s(16)$. The rightmost column of Table 4.7 shows the ratio $\rho(n) = \frac{\mu_h(n)}{\mu_s(n)}$. It shows that $\rho(n)$

TABLE 4.7: Average numbers of factors to represent n-variable interval functions by HTs (near minimum) and exact MSOPs for $n = 1$ to $n = 16$

n	Head-Tail Expression		MSOP	Ratio ($\rho(n)$)
	$\mu_h(n)$	$\frac{(\frac{2}{3}n-\frac{5}{9})}{\mu_h(n)}$	$\mu_s(n)$	$\frac{\mu_h(n)}{\mu_s(n)}$
1	1	0.1111	1	1
2	1.3	0.5983	1.3	1
3	1.7222	0.8387	1.7778	0.97
4	2.2574	0.9352	2.3971	0.94
5	2.8523	0.9739	3.1288	0.91
6	3.4822	0.9892	3.9433	0.88
7	4.1301	0.9954	4.8154	0.86
8	4.7873	0.9980	5.7267	0.84
9	5.4492	0.9991	6.6645	0.82
10	6.1135	0.9996	7.6203	0.80
11	6.7790	0.9998	8.5886	0.79
12	7.4450	0.9999	9.5654	0.78
13	8.1114	0.9999	10.5487	0.77
14	8.7779	0.9999	11.5362	0.76
15	9.4445	0.9999	-	-
16	10.1111	0.9999	-	-

decreases with the increment of n. The experimental results also show that, for $n \geq 10$, HTs require at least 20% fewer factors than MSOPs, on the average.

Moreover, we can observe interesting sequences in Table 4.5. Let $C_\tau(n)$ be the value of the τ-th column in Table 4.5. For $\tau = 1$ to $\tau = 6$, we have:

$$
\begin{aligned}
C_1(n) &= n + 1 \\
C_2(n) &= (n-1)^2 \\
C_3(n) &= \frac{(n-3)(n-2)(2n-5)}{3} \\
C_4(n) &= \frac{(n-4)^2[(n-4)^2-1]}{3} \\
C_5(n) &= \frac{(n-7)(n-6)(n-5)(n-4)(2n-11)}{15}, \text{and} \\
C_6(n) &= \frac{4}{90}(n-7)^2[(n-7)^2-1][(n-7)^2-4].
\end{aligned}
$$

The derivation of these formulas are future work.

4.5 Conclusion

In this chapter, we introduced head-tail expressions (HTs) to represent interval functions. We showed that HTs efficiently represent GT, LT and interval functions. We also showed

that a pair of a TCAM and a RAM directly implements an HT. Finally, we prove that an HT requires at most n factors to represent any interval function $IN_0(n : A, B)$. By a heuristic algorithm, we obtained average numbers of factors to represent interval functions in HTs for up to $n = 16$. And, we conjecture that, for sufficiently large n, the average number of factors by HTs to represent n-variable interval functions is $\frac{2}{3}n - \frac{5}{9}$. We also show that, for $n \geq 10$, HTs generated by our heuristic program require at least 20% fewer factors than MSOPs, on the average.

Chapter 5

Head-Tail Expressions for Single-Field Classification Functions

In this chapter, we present a method to generate head-tail expressions for single-field classification funcions. First, we introduce a fast prefix sum-of-products expression (PreSOP) generator (FP) which generates products using the bit patterns of the endpoints. Next, we propose a direct head-tail expression generator (DHT). Experimental results show that DHT generates much smaller TCAM than FP. The proposed algorithm is useful for simplified TCAM generator for packet classification.

5.1 Introduction

Table 5.1 shows an example of a classification function with two fields that correspond to the source and the destination ports represented by intervals. The representation in TCAM is described in Table 5.2. When each port is specified by either * (*don't care*) or a single value, each rule corresponds to one word in a TCAM. However, when a port is specified by an open interval such as $(0, 16)$ or $(-1, 15)$, the interval requires multiple words in a TCAM [12]. For example, in Table 5.2, both intervals $(0, 16)$ and $(-1, 15)$ require 4 words. This problem (*i.e.*, rule expansion) is the main subject of this chapter.

In this chapter, we present a fast reduction of TCAM using a head-tail expression (HT). First, we introduce a fast PreSOP generator (FP). Then, we propose a direct head-tail expression generator (DHT). Finally, by experimental results, we show that DHT is faster and produces better solutions than other algorithms.

TABLE 5.1: Example of classification function

Rule	Source Port	Destination Port	Action
1	(0,16)	2	Accept
2	3	(-1,15)	Accept
3	*	*	Deny

TABLE 5.2: Implementation on TCAM

Rule	Source Port				Destination Port				Action
	x_3	x_2	x_1	x_0	x_3	x_2	x_1	x_0	
1	0	0	0	1	0	0	1	0	Accept
1	0	0	1	*	0	0	1	0	Accept
1	0	1	*	*	0	0	1	0	Accept
1	1	*	*	*	0	0	1	0	Accept
2	0	0	1	1	0	*	*	*	Accept
2	0	0	1	1	1	0	*	*	Accept
2	0	0	1	1	1	1	0	*	Accept
2	0	0	1	1	1	1	1	0	Accept
3	*	*	*	*	*	*	*	*	Deny

This chapter is organized as follows: Section 5.2 shows definitions, and basic properties of PreSOPs and interval functions. Section 3.5 shows a head-tail expression for an interval function. Section 4.5 presents a fast PreSOP generator (FP). Section 5.5 presents a head-tail expression generator using greedy approach (HT), and a direct head-tail expression generator (DHT). Section 5.6 shows experimental results. And finally, Section 5.7 concludes the paper.

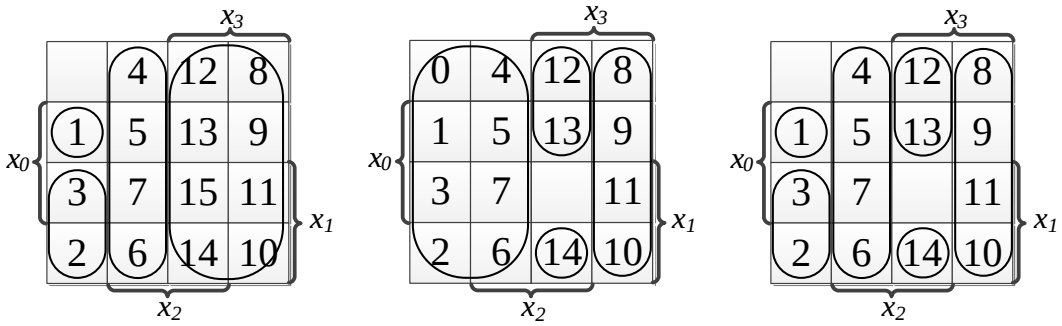
5.2 Definitions and Basic Properties

Lemma 5.1. *A GT function can be represented by the PreSOP:*

$$GT(n:A) = \bigvee_{i=0}^{n-2} \left(\bigwedge_{j=n-1}^{i+1} x_j^{a_j} \right) x_i \bar{a}_i \vee x_{n-1}\bar{a}_{n-1},$$

where $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ *is the binary representation of* A*. It has* $\sum_{i=0}^{n-1} \bar{a}_i$ *disjoint products.*

Example 5.1. *Consider the PreSOP for* $GT(n:A)$*, where* $n = 4$ *and* $A = 0$*. The binary representation of* A *is* $\vec{a} = (0,0,0,0)$*. The PreSOP is* $\bar{x}_3\bar{x}_2\bar{x}_1x_0 \vee \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3x_2 \vee x_3$*.* ■



(a) $GT(4:0)$ (b) $LT(4:15)$ (c) $IN_0(4:0,15)$

FIGURE 5.1: Maps for Example 5.2

Lemma 5.2. *An LT function can be represented by the PreSOP:*

$$LT(n:B) = \bigvee_{i=0}^{n-2} \left(\bigwedge_{j=n-1}^{i+1} x_j^{b_j} \right) \bar{x}_i b_i \vee \bar{x}_{n-1} b_{n-1},$$

where $\vec{b} = (b_{n-1}, b_{n-2}, \cdots, b_1, b_0)$ *is the binary representation of* B*. It has* $\sum_{i=0}^{n-1} b_i$ *disjoint products.*

Theorem 5.1. *Let* $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ *and* $\vec{b} = (b_{n-1}, b_{n-2}, \cdots, b_1, b_0)$ *be the binary representations of* A *and* B*, respectively, and* $A < B$*. Let* t *be the largest index such that* $a_{t-1} \neq b_{t-1}$*. Then,* $IN_0(n:A,B)$ *can be represented by:*

$$\bigvee_{i=t-2}^{0} \left[\left(\bigwedge_{j=n-1}^{i+1} x_j^{a_j} \right) x_i \bar{a}_i \vee \left(\bigwedge_{j=n-1}^{i+1} x_j^{b_j} \right) \bar{x}_i b_i \right].$$

The number of products is $\sum_{i=0}^{t-2} (\bar{a}_i + b_i)$*.*

Example 5.2. *Let* $A = 0$*,* $B = 15$ *and* $n = 4$*. Note that* $\vec{a} = (0,0,0,0)$ *and* $\vec{b} = (1,1,1,1)$*. By Lemma 5.1, the PreSOP for* $GT(4:0)$ *is* $\bar{x}_3\bar{x}_2\bar{x}_1x_0 \vee \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3x_2 \vee x_3$*. The number of products is* $\sum_{i=0}^{3} \bar{a}_i = 4$*. By Lemma 5.2, the PreSOP for* $LT(4:15)$ *is* $x_3x_2x_1\bar{x}_0 \vee x_3x_2\bar{x}_1 \vee x_3\bar{x}_2 \vee \bar{x}_3$*. The number of products is* $\sum_{i=0}^{3} b_i = 4$*. And, Theorem 5.1 shows that* $IN_0(4:0,15)$ *requires* $3+3=6$ *products. The PreSOP for* $IN_0(4:0,15)$ *is* $\bar{x}_3\bar{x}_2\bar{x}_1x_0 \vee \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3x_2 \vee x_3\bar{x}_2 \vee x_3x_2\bar{x}_1 \vee x_3x_2x_1\bar{x}_0$*. Fig. 5.1 shows their maps, where the integers in the maps denote* $X = 8x_3 + 4x_2 + 2x_1 + x_0$*. Note that a minimum SOP for* $IN_0(4:0,15)$ *is* $x_0\bar{x}_1 \vee x_1\bar{x}_2 \vee x_2\bar{x}_3 \vee x_3\bar{x}_0$*.* ■

5.3 Realization of Interval Functions on TCAM

A ternary content addressable memory (TCAM) shown in Fig. 5.2 compares the input vector with the entire list of registered vectors, simultaneously. When multiple matches occur, the priority encoder selects the match line with the smallest index. The RAM stores the corresponding *Action* for the TCAM words. A straightforward method to design TCAM is to use a PreSOP.

Example 5.3. *Design the TCAM that represents* $GT(4:0)$. *The PreSOP for* $GT(4:0)$ *is*

$$f(x_3, x_2, x_1, x_0) = \bar{x}_3\bar{x}_2\bar{x}_1x_0 \vee \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3x_2 \vee x_3.$$

Table 5.3 shows the corresponding TCAM realization.

TABLE 5.3: Realization based on PreSOP.

TCAM				RAM
x_3	x_2	x_1	x_0	
0	0	0	1	1
0	0	1	*	1
0	1	*	*	1
1	*	*	*	1
*	*	*	*	0

TABLE 5.4: Realization based on HT.

TCAM				RAM
x_3	x_2	x_1	x_0	
0	0	0	0	0
*	*	*	*	1

Note that the first product in the PreSOP corresponds to the first TCAM word, and the second product in the PreSOP corresponds to the second TCAM word, etc. In the TCAM, we append the all don't care product at to the bottom. This word represents the default value for the rest of the combinations. Thus, the number of TCAM words is $\tau(PreSOP)+1$, *where* $\tau(PreSOP)$ *denotes the number of products in the PreSOP. In the RAM, the first* $\tau(PreSOP)$ *entries are 1, while the* $\tau(PreSOP)+1$*th entry is 0.*

However, in the circuit shown in Fig. 5.2, the interval function can often be implemented more efficiently. Since $GT(4:0)$ *can be represented as*

$$f(x_3, x_2, x_1, x_0) = \overline{(\bar{x}_3\bar{x}_2\bar{x}_1\bar{x}_0)} \cdot (1),$$

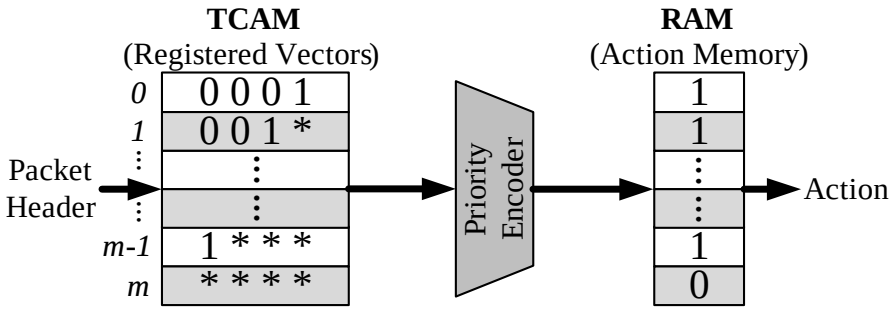


FIGURE 5.2: Realization using TCAM and RAM.

f is implemented by the TCAM shown in Table 5.4. In this case, the combination that makes $f = 0$ is first detected, and other combinations for the default value $f = 1$ is detected by the bottom word in the TCAM. Thus, we need only two TCAM words. ■

To find more efficient realizations for TCAMs, we need a new method to represent a function. In the next section, we introduce such a method.

5.4 Head-Tail Expressions for Interval Functions

In this section, we introduce head-tail expressions [13] that efficiently represent interval functions. As shown in Section II, the number of products in a PreSOP for an interval function is $\sum_{i=0}^{t-2}(\bar{a}_i + b_i)$. This value increases with the number of 0's and 1's in binary representations of A and B, respectively. However, this problem can be resolved by using a head-tail expression.

Definition 5.1. A **head-tail expression** (HT) has the form

$$f = \bigvee_{i=t}^{0} \left[\bigwedge_{j=1}^{s} (\bar{h}_{ij}) \right] \left[\bigwedge_{k=1}^{v} (g_{ik}) \right], \tag{5.1}$$

where for $(i = 0, 1, \cdots, t)$, $(\bar{h}_{ij})$ is the **head factor** and (g_{ik}) is the **tail factor**, and h_{ij} and g_{ik} denote products. In this paper, (product) and $\overline{\text{(product)}}$ are called **factors**. When there are no head factors, the HT is an SOP.

Example 5.4. $\overline{(x_1x_2)} \cdot \overline{(x_3x_4)} \cdot (x_5x_6) \vee \overline{(x_1x_4)} \cdot \overline{(x_2x_3)} \cdot (\bar{x}_5\bar{x}_6)$ *is a head-tail expression.* ■

HTs are a generalization of SOPs, and often require fewer factors to represent the same function.

Lemma 5.3. *An arbitrary logic function f can be represented by a* ***head-tail expression*** *(Eq. (5.1)).*

The next two theorems show that when the binary representations of endpoints have special property, HTs can be directly generated from the binary representations of endpoints.

Theorem 5.2. *Let $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ be the binary representation of an integer A. Let $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$ be the starting indexes of consecutive 0's groups in $\vec{a}$, where $c_{p-1} > c_{p-2} > \cdots > c_1 > c_0$. Let the isolated 1's be $a_{c_{p-2}+1} = a_{c_{p-3}+1} = \cdots = a_{c_1+1} = a_{c_0+1} = 1$, where $c_k + 1$ is the index of isolated 1's among groups of consecutive 0's in $\vec{a}$. Then, the $GT(n:A)$ function can be represented by an HT with $p+1$ factors:*

$$\overline{\left(\bigwedge_{j=n-1}^{c_0+1} x_j^{a_j} \bigwedge_{i=c_0}^{c_0+1-d_0} \bar{x}_i\right)} \cdot \overline{\left(\bigwedge_{j=n-1}^{c_1+1} x_j^{a_j} \bigwedge_{i=c_1}^{c_1-d_1} \bar{x}_i\right)} \cdots \cdot \overline{\left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{a_j} \bigwedge_{i=c_{p-1}}^{c_{p-1}-d_{p-1}} \bar{x}_i\right)} \cdot \left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{a_j}\right),$$

where $d_{p-1}, d_{p-2}, \cdots, d_1, d_0$ (for $i = 0, 1, \cdots, p-1$, $d_i > 0$) are numbers of consecutive 0's in the groups which start from the indexes $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$, respectively. Note that, in $\vec{a}$, except for the group of consecutive 0's, remaining bits are 1's.

Example 5.5. *Let $A = 0$. The binary representation of A is $\vec{a} = (0,0,0,0)$. By Theorem 5.2, we have a group of consecutive 0's, where $n = 4$, $p = 1$, $c_{p-1} = c_0 = 3$ and $d_0 = 4$. Thus,*

$$\begin{aligned} GT(4:0) &= \overline{\left(\bigwedge_{j=n-1}^{c_0+1} x_j^{a_j} \bigwedge_{i=c_0}^{c_0+1-d_0} \bar{x}_i\right)} \cdot \left(\bigwedge_{j=n-1}^{c_0+1} x_j^{a_j}\right) \\ &= \overline{(\bar{x}_3 \bar{x}_2 \bar{x}_1 \bar{x}_0)} \cdot (1). \end{aligned}$$

Theorem 5.3. *Let $\vec{b} = (b_{n-1}, b_{n-2}, \cdots, b_1, b_0)$ be the binary representation of an integer B. Let $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$ be the starting indexes of consecutive 1's groups in $\vec{b}$, where $c_{p-1} > c_{p-2} > \cdots > c_1 > c_0$. Let the isolated 0's be $b_{c_{p-2}+1} = b_{c_{p-3}+1} = \cdots = b_{c_1+1} = b_{c_0+1} = 0$, where $c_k + 1$ is the index of isolated 0's among groups of consecutive*

1*'s in* $\vec{b}$*. In this case,* $LT(n:B)$ *can be represented by an HT with* $p+1$ *factors:*

$$\overline{\left(\bigwedge_{j=n-1}^{c_0+1} x_j^{b_j} \bigwedge_{i=c_0}^{c_0+1-d_0} x_i\right)} \cdot \overline{\left(\bigwedge_{j=n-1}^{c_1+1} x_j^{b_j} \bigwedge_{i=c_1}^{c_1-d_1} x_i\right)} \cdots \overline{\left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{b_j} \bigwedge_{i=c_{p-1}}^{c_{p-1}-d_{p-1}} x_i\right)} \cdot \left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{b_j}\right),$$

where $d_{p-1}, d_{p-2}, \cdots, d_1, d_0$ *(for* $i = 0, 1, \cdots, p-1$*,* $d_i > 0$*) are numbers of consecutive* 1*'s in the groups which start from the indexes* $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$*, respectively. Note that, in* $\vec{b}$*, except for the group of consecutive* 1*'s, remaining bits are* 0*'s.*

Example 5.6. *Represent* $LT(n:B)$ *by a PreSOP and a head-tail expression, where* $n = 8$ *and* $B = 247$*.* $\vec{b} = (1,1,1,1,0,1,1,1)$ *is the binary representation of* B*. By Lemma 5.1, we have the PreSOP of* LT*:*

$$LT(n:B) = x_7x_6x_5x_4\bar{x}_3x_2x_1\bar{x}_0 \vee x_7x_6x_5x_4\bar{x}_3x_2\bar{x}_1$$
$$\vee x_7x_6x_5x_4\bar{x}_3\bar{x}_2 \vee x_7x_6x_5\bar{x}_4 \vee x_7x_6\bar{x}_5 \vee x_7\bar{x}_6 \vee \bar{x}_7.$$

The number of products is $\sum_{i=0}^{n-1} b_i = 7$*. However, the head-tail expression for* $LT(n:B)$ *requires only three factors* $(p+1 = 2+1)$*. By Theorem 5.3, we have:*

$$LT(n:B) = \overline{(x_7x_6x_5x_4\bar{x}_3x_2x_1x_0)} \cdot \overline{(x_7x_6x_5x_4x_3)} \cdot (1)$$

The binary representation of $B = 247$ *is:*

$$\vec{b} = (\underbrace{\overset{b_{c_1}=7}{\downarrow}1, \overset{b_6}{\downarrow}1, \overset{b_5}{\downarrow}1, \overset{b_4}{\downarrow}1}_{d_1}, \overset{b_3}{\downarrow}0, \underbrace{\overset{b_{c_0}=2}{\downarrow}1, \overset{b_1}{\downarrow}1, \overset{b_0}{\downarrow}1}_{d_0})$$

There are $p = 2$ *groups of consecutive 1's, which start from indexes* $c_0 = 2$ *and* $c_1 = 7$*, and the numbers of consecutive* 1*'s are* $d_0 = 3$ *and* $d_1 = 4$*, respectively.*

Table 5.5(a) shows the PreSOP realization for the interval $(-1, 247)$*. Seven TCAM words realize the interval* $(-1, 247)$*, and the RAM works as the OR function. On the other hand, Table 5.5(b) shows HT-realization for the same function: two TCAM words realize the interval* $(246, 2^8)$*, and the RAM works as the NOR function. Since the RAM can be programmed freely, the NOR function instead of the OR function can be implemented. In this way, we can generate a smaller TCAM than the conventional approach.*

■

TABLE 5.5: Realization of $LT(8:247)$ by TCAM and RAM

(a)

TCAM								RAM
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	
1	1	1	1	0	1	1	0	1
1	1	1	1	0	1	0	*	1
1	1	1	1	0	0	*	*	1
1	1	1	0	*	*	*	*	1
1	1	0	*	*	*	*	*	1
1	0	*	*	*	*	*	*	1
0	*	*	*	*	*	*	*	1
*	*	*	*	*	*	*	*	0

(b)

TCAM								RAM
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	
1	1	1	1	0	1	1	1	0
1	1	1	1	1	*	*	*	0
*	*	*	*	*	*	*	*	1

5.5 Fast Prefix SOP Generator

FP(A Fast PreSOP Generator):
/* Input: The binary representations of A and B which are stored in $Vector(\vec{a})$ and $Vector(\vec{b})$, respectively. */
/* Output: TCAM words for PreSOP. */
1: Find the largest index such that $a_s \neq b_s$,
$s \leftarrow \lfloor \log_2 (A \oplus B) \rfloor$.
2: For both vectors ($Vector(\vec{a})$ and $Vector(\vec{b})$), perform below.
3: **for** $i = 0$; $i < s$; $i++$ **do**
4: **if** $Vector[s-1-i] = B\_flg$ **then**
5: $Output[n-1,\ldots,s-i] \leftarrow Vector[n-1,\ldots,s-i]$
6: $Output[s-1-i] \leftarrow A\_flg$
7: **end if**
8: **end for**
9: Terminate.

FIGURE 5.3: Pseudocode for FP

In this section, we present a fast PreSOP generator (FP). Various PreSOP generators exist [60], [27], [12]. Fig. 5.3 shows the pseudocode of FP. The inputs are $Vector(\vec{a})$ and $Vector(\vec{b})$ that are binary representations of A and B, respectively. First, to apply Theorem 5.1, the largest index where $a_s \neq b_s$ is found by $s = \lfloor \log_2 (A \oplus B) \rfloor$. After that, each vector of the endpoints (A,B) represented by $Vector[n-1,\ldots,0]$ is checked. In this case, A_flg is *true* iff it is checking $Vector(\vec{a})$, while B_flg is *true* iff it is checking $Vector(\vec{b})$. Note that $A\_flg = \overline{B\_flg}$. If the checked vector value is *true*, then it produces $Output[n-1,\ldots,0]$ as the binary representation of the product. Because at most one product is produced for each variable, and only s bits are checked, the time complexity for n-bit FP is $O(s) \cdot n \approx O(n^2)$. Moreover, the space complexity for FP is $O(n^2)$,

because FP uses n bits to represent a vector and at most $O(n)$ vectors are necessary to represent the function.

5.6 Direct Head-Tail Expression Generators

In this section, we present a direct head-tail expression generator (DHT) to represent intervals (ports). DHT generates the TCAM words from the lower and the upper endpoints of the interval (A, B). Fig. 5.4 shows a DHT. Similar to FP, DHT finds the largest index such that $a_s \neq b_s$. After that, it checks every bit in $Vector[n-1, \ldots, 0]$ and returns *Mode*. A_flg is *true* iff it is checking $Vector(\vec{a})$, and B_flg is *true* iff it is checking $Vector(\vec{b})$. Note that $A\_flg = \overline{B\_flg}$. The detail of each *Mode* is as follows:

- *Mode 0*: Produces no output.
- *Mode 1*: Theorem 5.1 is used to produce the word.
- *Mode 2*: Theorem 5.2 or Theorem 5.3 is used to produce the word.

Fig. 5.4 shows that the time complexity for n-bit DHT is $O(s) \cdot n \approx O(n^2)$. In every case (*Mode*), the index i is incremented after it checks $Vector[n-1, \ldots, 0]$ in DHT. Thus, the algorithm iterates s times, where $s = t-1$ denotes the largest index such that $a_s \neq b_s$. Furthermore, the space complexity for DHT is $O(n^2)$, because similar to FP, DHT uses only n bits to represent a vector and at most $O(n)$ vectors are necessary to represent the function.

Example 5.7. *Let* $A = 383$, $B = 441$ *and* $n = 9$. *The binary representations of* A *and* B *are* $\vec{a} = (1,0,1,1,1,1,1,1,1)$ *and* $\vec{b} = (1,1,0,1,1,1,0,0,1)$, *respectively. To find the TCAM words for* $IN_0(9 : 383, 441)$, *the algorithm in Fig. 5.4 is used. First,* s *is computed using* $s = \lfloor \log_2 (A \oplus B) \rfloor = 7$. *Since* $a_i = 1$ *for* $i = 0$ *to* $i = s-1$, *no factor is produced from* $Vector(\vec{a})$. *Next from* $Vector(\vec{b})$, *at* $i = 0$, *1 is detected, goes to* Mode *1. At* $i = 1$, *0 is detected, goes to* Mode *0 and generates a word using Theorem 5.1: 110111000* $\rightarrow 1$, *or the factor* $(x_8 x_7 \bar{x}_6 x_5 x_4 x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0)$ *(Fig. 5.5,* ***Mode 1a****). At* $i = 2$, Mode *is 0. At* $i = 3$, *1 is detected, goes to* Mode *1. At* $i = 4$, *1 is detected, goes to* Mode 2 *and generates a word using Theorem 5.3: 110111**** $\rightarrow 0$, *or the factor* $\overline{(x_8 x_7 \bar{x}_6 x_5 x_4 x_3)}$ *(Fig. 5.5,* ***Mode 2a****, the first output). At* $i = 5$, *1 is detected, stays at* Mode 2. *At* $i = 6$, *0 is detected, goes to* Mode *2. Because the iteration finishes at* $s - 1 = 6$, *and there is a group of consecutive 1's in* $\vec{b}$, *the algorithm generates the word by Theorem 5.3: 110******* $\rightarrow 1$, *or the factor* $(x_8 x_7 \bar{x}_6)$ *(Fig. 5.5,* ***Mode 2a****, the second output) and terminates. Note that, the corresponding*

DHT(A Direct HT Generator):

/∗ Input: The binary representations of A and B which are stored in $Vector(\vec{a})$ and $Vector(\vec{b})$, respectively. ∗/

/∗ Output: TCAM words of the head-tail expression. ∗/

1: Find the largest index such that $a_s \neq b_s$,
$s \leftarrow \lfloor \log_2 (A \oplus B) \rfloor$.

2: For $i = 0$ to $i = s - 1$, iterate the below for $Vector(\vec{a})$ and $Vector(\vec{b})$. Checks $Vector(\vec{a})$ i ff$A_flg = 1$, and checks $Vector(\vec{b})$ i ff$B_flg = 1$.

3: **for** $i = 0$; $i < s$; $i++$ **do**
4: **switch** Mode
5: **case** 0:
6: **if** $Vector[i] = B\_flg$ **then**
7: Mode ← 1: Generate no output.
8: **else**
9: Mode ← 0
10: **end if**
11: **case** 1:
12: **if** $Vector[i] = B\_flg$ **then**
13: Mode ← 2: A group of consecutive 0's or 1's is detected. By Theorem 5.2 or 5.3, generate the head factor $(\bar{h}_i)$.
14: **else**
15: Mode ← 0: Only a single 0 or 1 is detected. Use Theorem 5.1 to generate a product.
16: **end if**
17: **case** 2:
18: **if** $Vector[i] = B\_flg$ **then**
19: Mode ← 2
20: **else**
21: **if** $Vector[i+1] = B\_flg$ **then**
22: Mode ← 2: Groups of consecutive 0's or 1's are detected. By Theorem 5.2 or 5.3, generate the factor $(\bar{h}_{i-1} \vee g_i)$.
23: **else**
24: Mode ← 0: If groups of consecutives 0's or 1's are detected, generate the tail factor (g_i) using Theorem 5.2 or 5.3.
25: **end if**
26: **end if**
27: **end switch**
28: **end for**
29: Terminate.

FIGURE 5.4: Pseudocode for DHT

HT is $(x_8x_7\bar{x}_6x_5x_4x_3\bar{x}_2\bar{x}_1\bar{x}_0) \vee \overline{(x_8x_7\bar{x}_6x_5x_4x_3)}(x_8x_7\bar{x}_6)$. *Table 5.6 shows the produced TCAM pattern.* ■

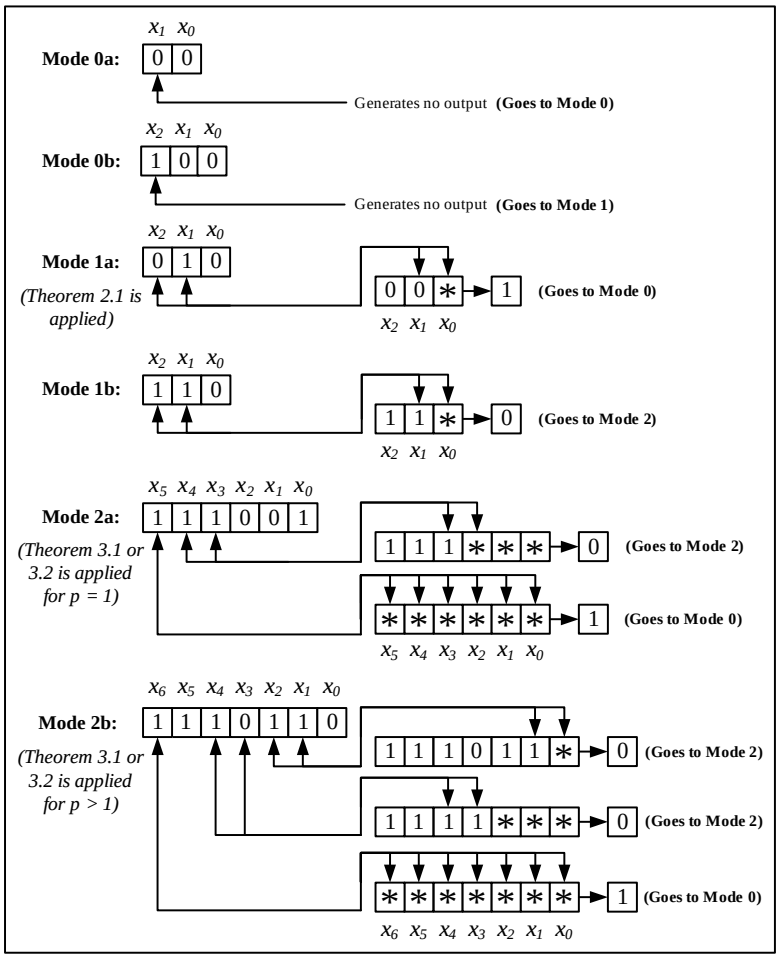


FIGURE 5.5: Steps of DHT in Example 6.1

TABLE 5.6: Realization of Example 6.1 in TCAM and RAM

TCAM									RAM
x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	
1	1	0	1	1	1	0	0	0	1
1	1	0	1	1	1	*	*	*	0
1	1	0	*	*	*	*	*	*	1
*	*	*	*	*	*	*	*	*	0

Example 5.8. *Obtain the HT for* $IN_0(9 : 383, 441)$ *by an algebraic approach. First, obtain PreSOPs for GT and LT functions:*

$$GT(9:383) = x_8x_7$$

$$LT(9:441) = \bar{x}_8 \vee x_8\bar{x}_7 \vee x_8x_7\bar{x}_6\bar{x}_5 \vee x_8x_7\bar{x}_6x_5\bar{x}_4 \vee x_8x_7\bar{x}_6x_5x_4\bar{x}_3 \vee x_8x_7\bar{x}_6x_5x_4x_3\bar{x}_2\bar{x}_1\bar{x}_0$$

TABLE 5.7: Comparison of performance

Data	Number of Rules	PreSOP		SOP		HT		
		Products	Time(μs) FP	Products	Time(ms) ESPRESSO-EXACT	Factors	Reduction(%)	Time(μs) DHT
ACL1	9760	13675	1.06	13667	786.702	12672	7.335	1.18
ACL2	9827	19449	1.11	19449	873.288	11221	42.306	1.16
ACL3	9323	16794	1.06	16699	740.238	11712	30.261	1.18
ACL4	9670	17179	1.05	17171	731.940	12022	30.019	1.20
ACL5	6457	8713	0.99	8713	723.748	7301	16.206	1.19
FW1	9753	34188	1.03	34188	758.240	12180	64.373	1.16
FW2	9865	19100	1.06	19100	824.233	11712	38.681	1.17
FW3	9583	25923	1.02	25923	705.497	11251	56.598	1.18
FW4	9517	62406	1.25	62406	670.166	17066	72.653	1.10
FW5	9513	22553	1.05	22553	723.124	11139	50.610	1.24
IPC1	9590	12969	1.03	12955	806.414	10439	19.508	1.13
IPC2	10000	10000	0.96	10000	775.081	10000	0.000	1.06

Next, obtain the PreSOP for the interval function:

$$GT(9:383)\cdot LT(9:441) = x_8x_7\bar{x}_6\bar{x}_5 \vee x_8x_7\bar{x}_6x_5\bar{x}_4 \vee x_8x_7\bar{x}_6x_5x_4\bar{x}_3 \vee x_8x_7\bar{x}_6x_5x_4x_3\bar{x}_2\bar{x}_1\bar{x}_0.$$

Finally, obtain the HT for $IN_0(9:383,441)$*:*

$$\begin{aligned} &x_8x_7\bar{x}_6(\bar{x}_5 \vee x_5\bar{x}_4 \vee x_5x_4\bar{x}_3) \vee (x_8x_7\bar{x}_6x_5x_4x_3\bar{x}_2\bar{x}_1\bar{x}_0) \\ &= (x_8x_7\bar{x}_6)\overline{(x_5x_4x_3)} \vee (x_8x_7\bar{x}_6x_5x_4x_3\bar{x}_2\bar{x}_1\bar{x}_0) \\ &= (x_8x_7\bar{x}_6)\overline{(x_8x_7\bar{x}_6x_5x_4x_3)} \vee (x_8x_7\bar{x}_6x_5x_4x_3\bar{x}_2\bar{x}_1\bar{x}_0) \end{aligned}$$

Note that DHT directly generates the TCAM patterns from the endpoints. ■

Fig. 5.5 illustrates the steps of DHT in Example 6.1. It compares each bit of the lower and the upper endpoints; decides which mode be enter; and generates the reduced TCAM patterns.

5.7 Experimental Results

Since no benchmark data for packet classifications is available, ClassBench [59] was used to generate classification functions. First, we generated PreSOPs for various functions. Then, we reduced the number of products in PreSOPs by ESPRESSO-EXACT [9]. Note that ESPRESSO-EXACT produces SOPs, which often require fewer products than PreSOPs. Table 5.7 compares PreSOPs and SOPs. However, ESPRESSO-EXACT did not significantly reduce the sizes of PreSOPs inspite of its long computation time.

Second, we applied the head-tail expression generator DHT to the same classification functions. Since the DHT in Fig. 5.4 is only for a single field function, we applied DHT

twice to obtain the HT. As shown in Table 5.7, the maximum reduction occurred in $FW4$, where the reduction ratio is more than 70%. In this case, many intervals that require many products in the PreSOP are reduced by the HT. On the other hand, no reduction occurred in $IPC2$, where each interval is represented by a single product and cannot be reduced by an HT. In terms of the speed, DHT is about 6×10^5 times faster than ESPRESSO-EXACT.

In these experiments, we generated simplified expression for each rule independently, but we did not check for the redundancy among rules. Thus, we may still be able to reduce the number of factors by spending extra time.

5.8 Conclusion

In this chapter, we used head-tail expressions to represent interval functions. We introduced a fast prefix SOP generator (FP) which generates products using the bit patterns of the endpoints. We proposed DHT to produce head-tail expressions directly from the endpoints (A,B). Experimental results showed that DHT is 6×10^5 times faster and produces smaller TCAMs than ESPRESSO-EXACT.

Chapter 6

Head-Tail Expressions for Multi-Field Classification Functions

This chapter discusses methods to reduce the TCAM size for multi-field classification functions. In the first part, we will show a redundancy removal for packet classification. By partitioning rules into groups and checking the relations among the rules in each group, we can remove some redundant rules [55]. The second one, we implement head-tail expressions for multi-field classification functions. We present an $O(r^2)$-algorithm, called MFHT, to generate simplified TCAMs for two-field classification functions, where r is the number of rules. Experimental results show that MFHT achieves a 58% reduction of words for random rules and a 52% reduction of words for ACL and FW rules. Moreover, MFHT is fast and useful for simplifying TCAM for packet classification.

6.1 Introduction

TABLE 6.1: Simplified example of a packet classifier

SA	DA	SP	DP	PO	Action
5	11	$[2, 2]$	$[5, 5]$	2	Accept
5	11	$[0, 2]$	$[6, 7]$	2	Accept
5	11	$[0, 3]$	$[5, 5]$	2	Accept
11	4	$[4, 7]$	*	2	Accept
11	4	$[3, 6]$	*	2	Accept
*	*	*	*	*	Discard

In a packet classification [7], rules are prioritized from the top to the bottom. In Table 6.1, if the source address is 11, the destination address is 4, the source port is 5, the destination port is 5, and the protocol type is 2, then the first to the fourth rules are not satisfied, but the fifth rule is satisfied. Thus, the packet is sent to the next destination. If the higher rule is not satisfied, then the lower rule is checked. Since the last rule has * in all the fields, the last rule is always satisfied. In this case, the packet is discarded. Thus, the packet classification in Table 6.1 can be considered as a five-field classification function. Note that all the fields can be represented by intervals [41]. A single value *e.g.,* 2 can be represented by the interval $[2, 2]$. Fig. 6.1 shows an example of a TCAM circuit [43] for the packet classifier in Table 6.1. The search data is compared with the stored words. When there is a match, the match line sends the signal to the priority encoder to produce the match address. In this case, the search data matches the rule with the address 101 or five.

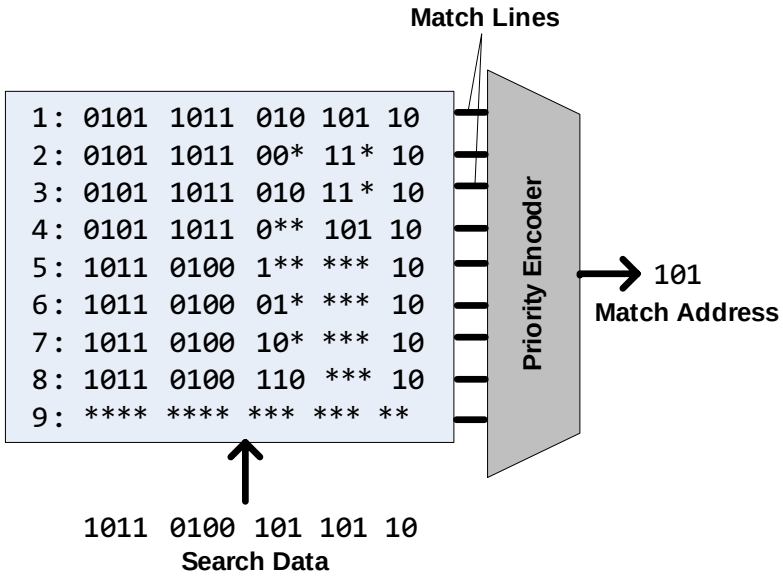


FIGURE 6.1: TCAM Circuit

The first problem is a **rule expansion**. In Table 6.1, there are six rules. However, to represent these rules by a TCAM, the source port and the destination port which are represented by intervals often require multiple words. For example, the third rule has source port $[0, 2]$ which require two words *i.e.,* 00* and 010. This phenomenon is called a **rule expansion**. A rule can be represented by a prefix SOP (**PreSOP**) which can be generated directly from an interval [53]. This is a small example of a rule expansion. When there are many rules, the rule expansion will become a burden.

In this chapter, first, we show a method to simplify rules in TCAMs for packet classification. We partition the rules into groups so that each group has the same source address, destination address and protocol. After that, we simplify rules of a group by removing

rules that are already covered by other rules. We developed a computer program to simplify rules. Experimental results show that this method reduces the size of rules up to 57% of the original specification for ACL5 rules, 73% for ACL3 rules, and 87% for overall rules. This algorithm is useful for reducing TCAMs for packet classification.

Second, we show a method to reduce the number of words in a TCAM to represent k-field classification functions. We use a head-tail expression to represent a multi-field classification function. We derive the number of factors in multi-field classification rules by head-tail expressions. Furthermore, we present an $O(r^k)$-algorithm called MFHT to generate simplified head-tail expressions for k-field classification functions, where r is the number of rules. Experimental results show that MFHT achieves a 58% reduction of words for random rules, and a 52% reduction of words for ACL and FW rules. Moreover, MFHT is fast and useful for simplifying TCAM for packet classification.

6.2 Definition and Basic Properties

Theorem 6.1. *Let* $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ *and* $\vec{b} = (b_{n-1}, b_{n-2}, \cdots, b_1, b_0)$ *be the binary representations of* A *and* B*, respectively, and* $A < B$*. Let* s *be the largest index such that* $a_s \neq b_s$*. Then,* $IN_0(n : A, B)$ *can be represented by*

$$\bigvee_{i=s-1}^{0} \left[\left(\bigwedge_{j=n-1}^{i+1} x_j^{a_j} \right) x_i \bar{a}_i \vee \left(\bigwedge_{j=n-1}^{i+1} x_j^{b_j} \right) \bar{x}_i b_i \right].$$

The number of products is

$$\tau_p(IN_0(n : A, B)) = \sum_{i=0}^{s-1} (\bar{a}_i + b_i).$$

Lemma 6.1. *Any logic function* $f(x_{n-1}, x_{n-2}, \cdots, x_0)$ *can be represented as*

$$f(x_{n-1}, x_{n-2}, \cdots, x_0) = \bigvee \xi_i(X_1) I_i(X_2),$$

where $X_1 = (x_{n-1}, x_{n-2}, \cdots, x_m)$ *and* $X_2 = (x_{m-1}, x_{m-2}, \cdots, x_0)$*.* $\xi_i(X_1)$ *are mutually disjoint products having the form* $x_{n-1}^* x_{n-2}^* \cdots x_m^*$*,* x^* *denotes either* x *or* $\bar{x}$*,* $\bigvee_i \xi_i(X_1) = 1$ *and* $I_i(X_2)$ *denotes an interval function of* m *variables.*

Theorem 6.2. *Let* $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ *be the binary representation of an integer* A*. Let* $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$ *be the starting indexes of consecutive* 0*'s groups in* $\vec{a}$*, where* $c_{p-1} > c_{p-2} > \cdots > c_1 > c_0$*. Let the isolated* 1*'s be* $a_{c_{p-2}+1} = a_{c_{p-3}+1} = \cdots = a_{c_1+1} = a_{c_0+1} = 1$*, where* $c_k + 1$ *is the index of isolated* 1*'s among groups of consecutive*

0*'s in* $\vec{a}$*. Then, the* $GT(n:A)$ *can be represented by* $p+1$ *factors:*

$$\overline{\left(\bigwedge_{j=n-1}^{c_0+1} x_j^{a_j} \bigwedge_{i=c_0}^{c_0+1-d_0} \bar{x}_i\right)} \cdot \overline{\left(\bigwedge_{j=n-1}^{c_1+1} x_j^{a_j} \bigwedge_{i=c_1}^{c_1-d_1} \bar{x}_i\right)} \cdots \cdot \overline{\left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{a_j} \bigwedge_{i=c_{p-1}}^{c_{p-1}-d_{p-1}} \bar{x}_i\right)} \cdot \left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{a_j}\right),$$

where $d_{p-1}, d_{p-2}, \cdots, d_1, d_0$ *(for* $i = 0, 1, \cdots, p-1$, $d_i \geq 1$*) are numbers of consecutive* 0*'s in the groups which start from the indexes* $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$*, respectively. Note that, in* $\vec{a}$*, except for the group of consecutive* 0*'s, remaining bits are* 1*'s.*

Theorem 6.3. *Let* $\vec{b} = (b_{n-1}, b_{n-2}, \cdots, b_1, b_0)$ *be the binary representation of an integer* B*. Let* $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$ *be the starting indexes of consecutive* 1*'s groups in* $\vec{b}$*, where* $c_{p-1} > c_{p-2} > \cdots > c_1 > c_0$*. Let the isolated* 0*'s be* $b_{c_{p-2}+1} = b_{c_{p-3}+1} = \cdots = b_{c_1+1} = b_{c_0+1} = 0$*, where* $c_k + 1$ *is the index of isolated* 0*'s among groups of consecutive* 1*'s in* $\vec{b}$*. In this case,* $LT(n:B)$ *can be represented by* $p+1$ *factors:*

$$\overline{\left(\bigwedge_{j=n-1}^{c_0+1} x_j^{b_j} \bigwedge_{i=c_0}^{c_0+1-d_0} x_i\right)} \cdot \overline{\left(\bigwedge_{j=n-1}^{c_1+1} x_j^{b_j} \bigwedge_{i=c_1}^{c_1-d_1} x_i\right)} \cdots \cdot \overline{\left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{b_j} \bigwedge_{i=c_{p-1}}^{c_{p-1}-d_{p-1}} x_i\right)} \cdot \left(\bigwedge_{j=n-1}^{c_{p-1}+1} x_j^{b_j}\right),$$

where $d_{p-1}, d_{p-2}, \cdots, d_1, d_0$ *(for* $i = 0, 1, \cdots, p-1$, $d_i \geq 1$*) are numbers of consecutive* 1*'s in the groups which start from the indexes* $c_{p-1}, c_{p-2}, \cdots, c_1, c_0$*, respectively. Note that, in* $\vec{b}$*, except for the group of consecutive* 1*'s, remaining bits are* 0*'s.*

6.2.1 Classification Functions

Definition 6.1. [41] A **classification function** with k **fields** is a mapping $F : D_1 \times D_2 \times \cdots \times D_k \to \{0, 1, 2, \cdots, r\}$, where $D_i = \{0, 1, \cdots, 2^{t_i} - 1\}$ $(i = 1, 2, \cdots, k)$. F is specified by a set of r rules. A **rule** consists of k fields, and each field D_i is specified by an interval of t_i bits.

Example 6.1. *Let* $F : SA \times DA \times SP \times DP \times PO \to \{0, 1, 2, \cdots, r\}$ *be a classification function. SA, DA, SP, DP and PO show source address, destination address, source port, destination port and protocol, respectively. The number of bits in the fields are* $t_{SA} = t_{DA} = 4$*,* $t_{SP} = t_{DP} = 3$*, and* $t_{PO} = 2$*. The classification function has* $r = 6$ *rules described in Table 6.1 and Fig. 6.1 shows its TCAM realization.* ∎

6.3 Simplification of rules

This section describes the method to simplify rules in packet classification by partitioning the rules into groups, checking the relation among the rules, and removing rules that are covered by other rules.

6.3.1 Partitioning rules into groups

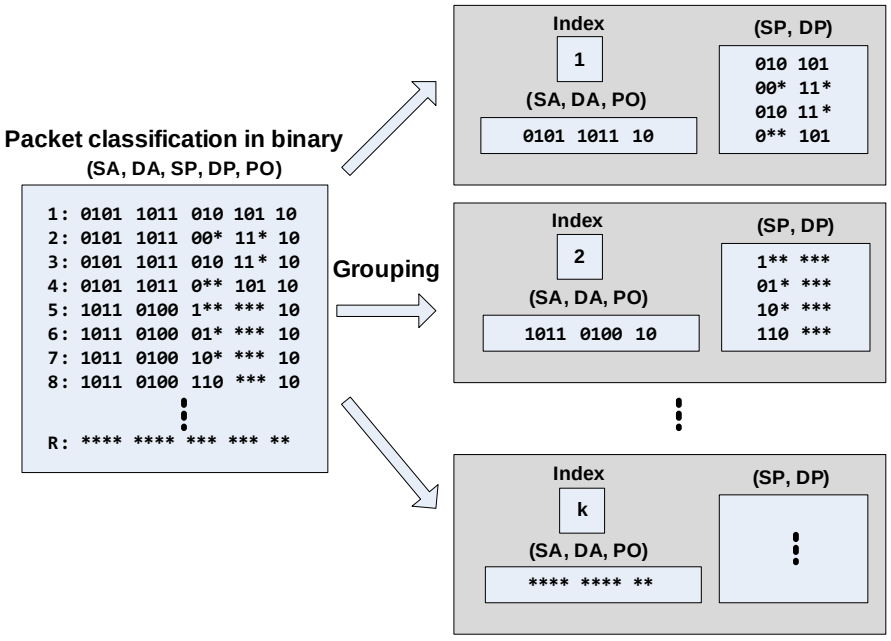


FIGURE 6.2: Illustration of grouping

In this subsection, we show an algorithm to partition rules into group so that each group have the same SA, DA and PO. Let F_{SA}, F_{DA}, F_{SP}, F_{DP} and F_{PO} be fields of source address, destination address, source port, destination port, and protocol, respectively. Each field F_i is represented by a specific number of bits. In this case, SA and DA are represented by 32 bits, SP and DP are represented by 16 bits, and PO is represented by 8 bits.

Fig. 6.2 illustrates the process. First, every rule is represented by a binary number, sorted and stored in the TCAM. After that, SA, DA, and PO is checked and compared with the existing grouped rules. If there is no match, new group is created which contains a new index of the group, a single SA, DA, and PO fields, and several SP and PO fields.

The process begins by representing the interval fields *i.e.,* the source port and the destination port are represented by PreSOPs. This can be done by using Theorem 6.1. The resulting words have τ_{SP} and τ_{DP} products and they are stored in $PreSOP_{SP}$ and $PreSOP_{DP}$, for SP and DP, respectively.

```
Grouping(F_SA, F_DA, F_SP, F_DP, F_PO):
/* Input: Classification rules, consist of 32 bits source and destination addresses (F_SA
   and F_DA), 16 bits intervals of source and destination ports (F_SP, F_DP), and 8 bits
   protocol F_PO. */
/* Output: Structured groups of classification rules. */
1:  for i = 0; i < r; i++ do
      /* PreSOPs are used to represent intervals of source and destination ports. We
      use Theorem 6.1 to represent F_SP and F_DP, where the PreSOPs products are
      stored in array PreSOP_SP and PreSOP_DP, respectively . */
2:    PreSOP_SP[τ_SP].
3:    PreSOP_DP[τ_DP].
4:    if i ≠ 0 then
5:      while k < j do
6:        if (Group[k] → F_SA == F_SA) · (Group[k] → F_DA == F_DA) · (Group[k] →
          F_PO == F_PO) then
7:          Break.
8:        else
9:          k++.
10:       end if
11:     end while
12:   else
13:     if i == 0 then
14:       j = 0;
15:     else
16:       j = k.
17:     end if
        /* No match, build a new group and store F_SA, F_DA and F_PO in the group. */
18:     Group[j] → F_SA = F_SA.
19:     Group[j] → F_DA = F_DA.
20:     Group[j] → F_PO = F_PO.
21:   end if
22:   for k = 0; k < τ_SP; k++ do
23:     for l = 0; l < τ_DP; l++ do
          /* Storing F_SP and F_DP in the form of PreSOPs */
24:       Group[j] → F_SP[m] = PreSOP_SP[k].
25:       Group[j] → F_DP[m] = PreSOP_DP[l].
26:       m++
27:     end for
28:   end for
29: end for
30: return Group.
31: Terminate.
```

FIGURE 6.3: Pseudocode for grouping the rules

After that, the current SA, DA, and PO are compared with the existing groups. If there is a match to a group, it will break the while operation (Fig. 6.3, line 7), and directly

store $PreSOP_{SP}$ and $PreSOP_{DP}$ to the group as described in lines 24 and 25. Symbol $\rightarrow$ shows a pointer to a member of structure *i.e., Group*$[j]$.

6.3.2 Elimination of redundant rules

In this subsection, we show algorithms to detect relation between rules. By knowing the relation, we can eliminate redundant rules.

Theorem 6.4. *Let S_1 and S_2 be logic functions represented by products of PreSOPs that show segments of an interval. Then, only the following three cases occur:*

$$\textit{Case 0: } S_1 \cdot S_2 = 0.$$
$$\textit{Case 1: } S_1 \subseteq S_2.$$
$$\textit{Case 2: } S_1 \supseteq S_2.$$

Proof: Suppose that S_1 and S_2 have the following form:

$$S_1 = x^*_{n-1}x^*_{n-2}\dots x^*_{m+1}x^*_m,$$
$$S_2 = x^\dagger_{n-1}x^\dagger_{n-2}\dots x^\dagger_{m+1}x^\dagger_m,$$

where x^*_i and $x^\dagger_i$ denote literals of x_i or a missing variable.

There exist two variables in S_1 and S_2 which are, x^*_i and x^*_j, and $x^\dagger_i$ and $x^\dagger_j$, respectively. Suppose that $S_1 \cdot S_2 \neq 0$. This case occurs when

1. x^*_i and $x^\dagger_i$ are the same literal or one or both of them are missing.
2. x^*_i is a literal x_i or $\bar{x}_i$, and x^*_j is missing in S_1, and $x^\dagger_j$ is a literal x_j or $\bar{x}_j$, and $x^\dagger_i$ is missing in S_2, where $i \neq j$.

When 2) is true, either S_1 or S_2 is not a product of PreSOP which contradicts the hypothesis. When 1) is true, the products of PreSOPs are still possible to occur such that

$$S_1 = x^*_{n-1}x^*_{n-2}\dots x^*_{m+1}x^*_m,$$
$$S_2 = x^\dagger_{n-1}x^\dagger_{n-2}\dots x^\dagger_{k+1}x^\dagger_k,$$

when $x^*_i = x^\dagger_i$ for $i = n-1, n-2, \cdots, k$ and $k \leq m$. However, this case is covered in Case 1 or 2 ($S_1 \subseteq S_2$ or $S_1 \supseteq S_2$). Thus, the cases when $S_1 \cdot S_2 \neq 0$, $S_1 \not\leq S_2$ and $S_1 \not\geq S_2$ never occur. $\square$

In Theorem 6.4, we assume that S_1 and S_2 are represented by products of PreSOPs. Note that if the segments of filed are represented by SOPs, then Theorem 6.4 does not hold.

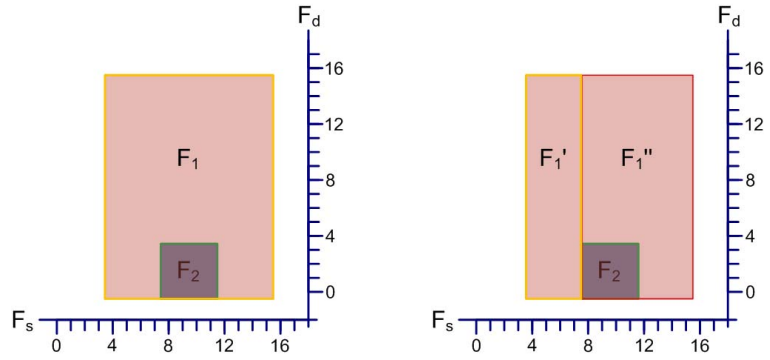
Example 6.2. *Consider the case*

$$S_1 = x_4\bar{x}_2, S_2 = \bar{x}_3x_1$$

In this case $S_1 \cdot S_2 \neq 0$, $S_1 \nsubseteq S_2$, *and* $S_1 \not\supseteq S_2$. *Note that* $\bar{x}_3x_1$ *is not a product of a PreSOP, since literals for* x_4 *and* x_2 *are missing.* ■

Lemma 6.2. *Let* F_1 *and* F_2 *be logic functions. If* $F_1 \supseteq F_2$, *then*

$$\bar{F}_1 \cdot F_2 \vee F_1 \cdot \bar{F}_2 \vee F_1 \cdot F_2 = F_1.$$



(a) F_1 in a PreSOP requires two products

(b) F_1' and F_1'' are represented by products in a PreSOPs

FIGURE 6.4: Illustration of intersection of rules represented by PreSOPs

Example 6.3. $F_{s1} := [4, 15]$[1], $F_{d1} := [0, 15]$, $F_{s2} := [8, 11]$ *and* $F_{d2} := [0, 3]$. *Assume that* $F_1 = F_{s1} \cdot F_{d1}$ *and* $F_2 = F_{s2} \cdot F_{d2}$. *In Fig. 6.4,* F_s *is the source port field that is represented by the the horizontal axis and* F_d *is destination port field that is represented by the vertical axis. By Theorem 6.1, we have*

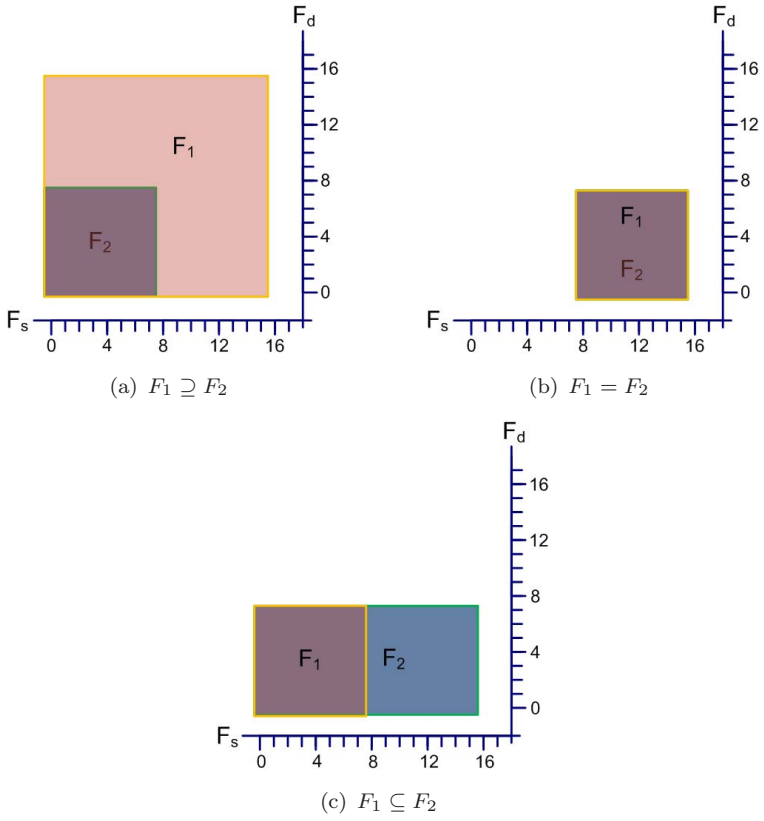
$$F_{s1} = \bar{x}_3x_2 \vee x_3,$$
$$F_{d1} = 1,$$
$$F_{s2} = x_3\bar{x}_2,$$
$$F_{d2} = \bar{x}_3\bar{x}_2.$$

Table 6.2 represents TCAM for F_1 *and* F_2. *Note that* F_1 *consists of two segments (which are* F_1' *and* F_1'' *in Fig. 6.4(b)). In fact, the interval* $[4, 15]$ *in Fig. 6.4(a) requires two words in a TCAM* i.e., *01** and 1***.* ■

[1] F_{s1} is a logic function, and $[4, 5]$ is an interval. Since any interval can be represented by a logic function, we use this notation (:=).

TABLE 6.2: TCAM representation of Example 6.3

$F_1 = F_{s1} \cdot F_{d1}$	Interval
01**-****	$[4,7] \times [0,15]$
1***-****	$[8,15] \times [0,15]$
$F_2 = F_{s2} \cdot F_{d2}$	Interval
10**-00**	$[8,11] \times [0,3]$



(a) $F_1 \supseteq F_2$ (b) $F_1 = F_2$

(c) $F_1 \subseteq F_2$

FIGURE 6.5: Relations between rules

Lemma 6.3. *Consider two rules: $F_1 = F_{s1} \cdot F_{d1}$ and $F_2 = F_{s2} \cdot F_{d2}$. Assume that F_1 has a higher priority than F_2. When, $F_{s1} \supseteq F_{s2}$ and $F_{d1} \supseteq F_{d2}$, the second rule F_2 can be removed without changing the operation of the classification function.*

Example 6.4. *Check the relation between F_1 and F_2 in Fig. 6.5(a). We have $F_1 = F_{s1} \cdot F_{d1}$ and $F_2 = F_{s2} \cdot F_{d2}$, where F_1 has a higher priority than F_2. Note that $F_{s1} := [0,15]$, $F_{d1} := [0,15]$, $F_{s2} := [0,7]$ and $F_{d2} := [0,15]$. Note that, $F_{s1} \supseteq F_{s2}$ and $F_{d1} \supseteq F_{d2}$, and*

$$F_1 = F_{s1} \cdot F_{d1} = 1,$$
$$F_2 = F_{s2} \cdot F_{d2} = \bar{x}_7 \bar{x}_3.$$

And, $F_1 \supseteq F_2$. Thus, F_2 can be removed. ∎

Lemma 6.4. *Let $F_1 = F_{s1} \cdot F_{d1}$ and $F_2 = F_{s2} \cdot F_{d2}$ be the first rule and the second rule, respectively. Assume that F_1 has a higher priority than F_2, and F_1 and F_2 have the*

same actions. When, $F_{s1} \subseteq F_{s2}$, $F_{d1} \subseteq F_{d2}$, and no rule avoids F_1 from F_2, then, the first rule F_1 can be removed without changing the operation of classification function.

Example 6.5. *Let $F_{s1} := [0,7]$, $F_{d1} := [0,7]$, $F_{s2} := [0,15]$ and $F_{d2} := [0,7]$. Assume that $F_1 = F_{s1} \cdot F_{d1}$ and $F_2 = F_{s2} \cdot F_{d2}$ and F_1 has a higher priority than F_2. Also, assume that F_1 and F_2 have the same action. Note that*

$$F_1 = F_{s1} \cdot F_{d1} = \bar{x}_7 \bar{x}_3,$$

and

$$F_2 = F_{s1} \cdot F_{d1} = \bar{x}_3.$$

In this case, $F_{s1} \subseteq F_{s2}$ and $F_{d1} \subseteq F_{d2}$. Fig. 6.5(c) shows the relation. When the packet classifier consists of only these two rules, the first rule can be removed without changing the operation of the classification function. ■

Note that Lemmas 6.3 and 6.4 also hold for logic functions. However, Theorem 6.4 holds only for PreSOPs because it requires the special property of PreSOPs.

Subset1D(F_1, F_2):

/* Input: F_1 and F_2 store the words of a same field (*e.g.*, source port) in the first rule and the second rule, respectively. */
/* Output: Integer *inc* that represent the relation between F_1 and F_2: $inc = 1$ means $F_2 \subseteq F_1$, $inc = 2$ means $F_1 \subseteq F_2$, and $inc = 0$ means $F_1 \cdot F_2 = 0$. */
1: **for** $i = 0$; $i < n$; $i++$ **do**
2: **if** $F_1[i] == *$ **then**
/* $F_2 \subseteq F_1$. */
3: $inc = 1$.
4: **else if** $F_2[i] == *$ **then**
/* $F_1 \subseteq F_2$. */
5: $inc = 2$.
6: **else if** $F_1[i] \neq F_2[i]$ **then**
/* $F_1 \cdot F_2 = 0$. */
7: $inc = 0$.
8: **end if**
9: **end for**
10: **if** $i == n$ **then**
11: $inc = 1$.
12: **end if**
13: **return** *inc*.
14: Terminate.

FIGURE 6.6: Pseudocode for checking the relation between fields

We try to simplify the rules by Theorem 6.4, Lemmas 6.2, 6.3 or 6.4. Subset1D(F_1, F_2) is an algorithm to find the relation between fields described in Fig. 6.6. The inputs are F_1 and F_2 having fields with n-bit words. The output inc shows the relation:

- $inc = 0$: $F_1 \cdot F_2 = 0$.
- $inc = 1$: $F_2 \subseteq F_1$.
- $inc = 2$: $F_1 \subseteq F_2$.

As shown in Fig. 6.6 line 10 and 11, when $F_1 = F_2$, the output is $inc = 1$.

The algorithm runs in $O(n)$ time. This algorithm is used to find the relation between rules with multiple fields.

Subset2D(F_{s1}, F_{d1}, F_{s2}, F_{d2}):
/* Input: F_{s1}, F_{d1}, F_{s2}, and F_{d2} that store the words of source port (F_s) and destination port (F_d) fields in the first rule and the second rule, respectively. */
/* Output: Integer inc that represent the relation between F_1 and F_2: $inc = 1$ means $F_2 \subseteq F_1$, $inc = 2$ means $F_1 \subseteq F_2$, and $inc = 0$ means $F_1 \cdot F_2 = 0$ where $F_1 = F_{s1} \cdot F_{d1}$ and $F_2 = F_{s2} \cdot F_{d2}$. */
1: **if** (Subset(F_{s1}, F_{s2}) == 1)·(Subset(F_{d1}, F_{d2}) == 1) **then**
   /* $F_2 \subseteq F_1$. */
2: $inc = 1$.
3: **else**
4: **if** (Subset(F_{s1}, F_{s2}) == 2)∨(Subset(F_{s1}, F_{s2}) == 1) **then**
5: **if** (Subset(F_{d1}, F_{d2}) == 1)∨(Subset(F_{d1}, F_{d2}) == 2) **then**
   /* $F_1 \subseteq F_2$. */
6: $inc = 2$.
7: **end if**
8: **else**
   /* $F_1 \cdot F_2 = 0$. */
9: $inc = 0$.
10: **end if**
11: **end if**
12: **return** inc.
13: Terminate.

FIGURE 6.7: Pseudocode for checking the relation between rules in multiple fields

By using Subset1D(F_1, F_2), we can check the relation between rules. In Fig. 6.8, Subset2D(F_{s1}, F_{d1}, F_{s2}, F_{d2}) compares F_1 and F_2 to find the relation between them where $F_1 = F_{s1} \cdot F_{d1}$ and $F_1 = F_{s2} \cdot F_{d2}$. The return value of this algorithm is inc that has the same meaning as Subset1D(F_1, F_2).

The main algorithm is Simplifying($Group[c]$) which simplify the rules in a group. In the group, NUM represents the number of products in PreSOPs of F_{SP} and F_{DP}. Symbol

Simplifying($Group[c]$):
/* Input: A group of classification rules, consists of a source address, a destination address, a protocol, and several source and destination ports. */
/* Output: A simplified group of rules. */
1: NUM is the number of source and destination ports in the group in the form of PreSOPs.
/* Checking the relation between rules by Lemma 6.3. */
2: **for** $i = 0; i < NUM - 1; i++$ **do**
3: **for** $j = i + 1; j < NUM; j++$ **do**
4: **if** Subset2D($Group[c] \rightarrow F_{SP}[i]$, $Group[c] \rightarrow F_{DP}[i]$, $Group[c] \rightarrow F_{SP}[j]$, $Group[c] \rightarrow F_{DP}[j]$) $== 1$ **then**
5: Remove $Group[c] \rightarrow F_{SP}[j]$ and $Group[c] \rightarrow F_{DP}[j]$.
6: **end if**
7: **end for**
8: **end for**
9: **for** $i = NUM - 1; i \geq 1; i--$ **do**
10: **for** $j = i - 1; j \geq 0; j--$ **do**
/* Checking the relation between rules by Lemma 6.4 */
11: $SubCase$ = Subset2D($Group[c] \rightarrow F_{SP}[i]$, $Group[c] \rightarrow F_{DP}[i]$, $Group[c] \rightarrow F_{SP}[j]$, $Group[c] \rightarrow F_{DP}[j]$).
12: **if** $SubCase == 1$ **then**
13: Remove $Group[c] \rightarrow F_{SP}[j]$ and $Group[c] \rightarrow F_{DP}[j]$.
14: **else if** $SubCase == 2$ **then**
15: Remove $Group[c] \rightarrow F_{SP}[i]$ and $Group[c] \rightarrow F_{DP}[i]$.
16: **end if**
17: **end for**
18: **end for**
19: **return** $Group[c]$.
20: Terminate.

FIGURE 6.8: Pseudocode for simplifying rules within a group

$\rightarrow$ represents a pointer to the structure $Group[c]$ which contains a word of SA, DA and PO, and several words of SP and DP. This algorithm contains two iterations. The first one checks the relations among the rules using Lemma 6.3 (Fig. 6.8 line 2). In this case, the rules are checked from the top to the bottom and if the upper rule covers the lower one, we can remove the lower rules as in Lemma 6.3. The second one uses Lemma 6.4 to check the relations among the rules (Fig. 6.8 line 9). In this case, the rules are checked from the bottom to the top. If the lower rule covers the upper one and there is no rule (with different action) that avoids between them, we can remove the upper rule using Lemma 6.4. The removing processes are described in Fig. 6.8 lines 5, 13, and 15.

TABLE 6.3: Performance of the simplification algorithm

Data	#Rules	PreSOP		A Fast Simplification					Relative
		#Products (τ_p)	Time(ms)	#Groups	#Products ($\hat{\tau}_p$)	Grouping Time(ms)	Simplifying Time(ms)	Max #Intervals in a Group	Size(%) ($\frac{\hat{\tau}_p}{\tau_p}$)
ACL1	9760	13675	20.576	7250	10953	345.935	6.627	21	80.095
ACL2	9827	19449	22.477	9614	19002	637.304	22.718	60	97.701
ACL3	9323	16794	20.064	4938	12286	271.569	11.179	35	73.157
ACL4	9670	17179	20.756	6315	14276	364.669	9.623	42	83.101
ACL5	6457	8713	12.935	3323	5008	116.605	3.539	45	57.477
FW1	9753	34188	20.459	9586	33429	892.382	72.224	99	97.779
FW2	9865	19100	22.440	9850	19058	565.168	4.039	13	99.921
FW3	9583	25923	19.473	9337	25048	887.412	51.386	143	96.624
FW4	9517	62406	24.045	9071	59778	826.639	205.120	273	95.788
FW5	9513	22553	18.975	9258	21379	814.964	45.480	147	94.794
IPC1	9590	12969	20.107	8090	11663	503.980	3.188	13	89.929
IPC2	10000	10000	20.510	10000	10000	785.166	0.215	1	100.000

6.4 Experimental Results

We applied our program to simplify packet classification rules. Since the real packet classification data is confidential, we used ClassBench to generate benchmark data for evaluation [59]. First, we generated the PreSOPs of the classification rules. In Table 6.3, in the column headed by PreSOP shows the number of products and the total CPU time to generate PreSOPs. To generate PreSOPs for a classification function with nearly 10000 rules, it took about 20 ms. We can directly generate PreSOPs from the binary representations of intervals [53].

Next, we simplified the classification rules by using algorithms in Section III. First, we applied Grouping(F_{SA}, F_{DA}, F_{SP}, F_{DP}, F_{PO}). It took longer time: to check every rule requires $O(r^2)$ time, where r is the number of rules in the original PreSOPs. Moreover, the number of rules that have the same F_{SA}, F_{DA} and F_{PO} are limited (only 15% of the total rules). On the other hand, Simplifying($Group[c]$) takes shorter time. This is because the number of intervals in a group is relatively small. Thus, it is faster although Simplifying($Group[c]$) has the complexity $O(nr^2)$, where n is the number of bits in F_{SP} or F_{DP} and r is the number of rules in the original PreSOPs. In the case of FW4 rules, it has the maximum number of intervals in a group with the largest number. Thus, the execution time for Simplifying($Group[c]$) is also the longest (205.12 ms). However, if we run all the programs (*i.e.,* PreSOP, Grouping, and Simplifying), for a classification function with nearly 10000 rules, we can simplify all the groups with total time 0.7 second on the average. In these experiments, we used a PC utilizing an 2 GHz Intel Core i7 with 8 GB memory and 64 bits Windows-OS.

Finally, we counted the number of products in PreSOPs after simplification. As we can see in Table 6.3, the largest reduction occurred in ACL5 and ACL3 rules: they are

reduced to 57% and 73% of the original specification, respectively. This is because many rules were removed by Simplifying algorithm.

We can generate an exact minimum PreSOP for a given interval. However, the benchmark functions generated by ClassBench have redundancy. Thus, the PreSOPs generated for a benchmark function also have redundancy. Thus, we can reduce the number of rules.

6.5 Simplification of TCAM for Multi-Field Classification Functions

In this subsection, we present a method to reduce TCAM words for multi-field classification functions. We show that a multi-field classification function is represented by head-tail expressions, and they can be reduced by the **absorption law**.

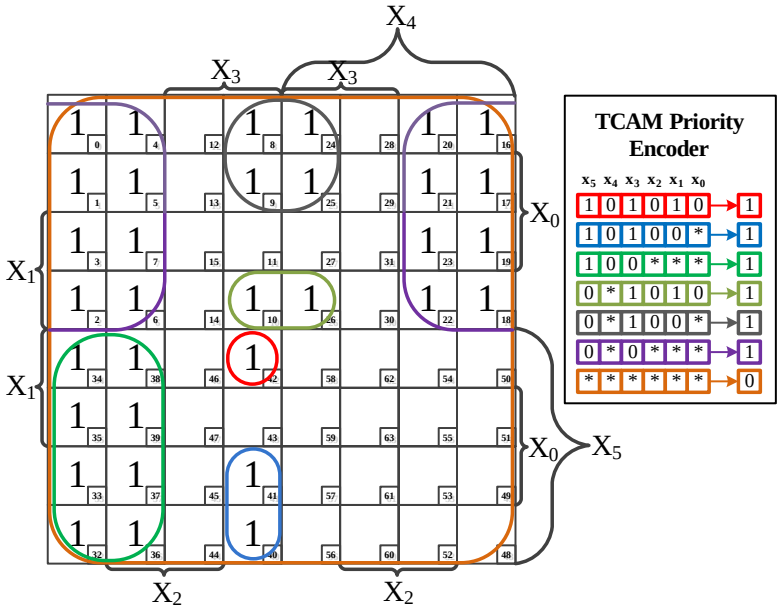
Property 6.1. *A PreSOP for a multi-field classification function cannot be reduced by the absorption law.*

On the other hand, we can generate a simplified head-tail expression for some multi-field classification functions directly as follows:

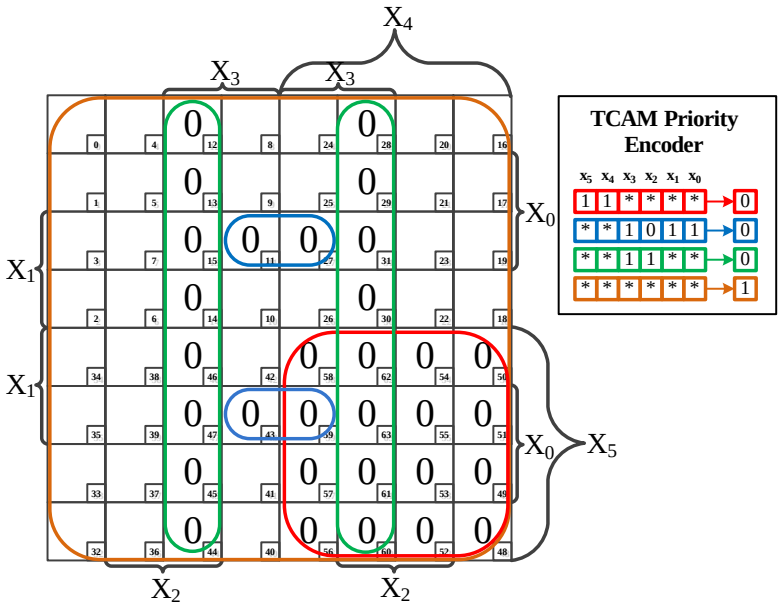
TABLE 6.4: TCAM Words for Example 6.6

Field f_1	Field f_2	Results
PreSOP $[0,2]$	PreSOP $[0,10]$	Products
$10 \to 1$ $0^* \to 1$	$1010 \to 1$ $100^* \to 1$ $0^{***} \to 1$	$101010 \to 1$ $10100^* \to 1$ $100^{***} \to 1$ $0^*1010 \to 1$ $0^*100^* \to 1$ $0^*0^{***} \to 1$ $^{******} \to 0$
Head-Tail Expr. $[0,2]$	Head-Tail Expr. $[0,10]$	Factors
$11 \to 0$ $^{**} \to 1$	$1011 \to 0$ $11^{**} \to 0$ $^{****} \to 1$	~~$111011 \to 0$~~ ~~$1111^{**} \to 0$~~ $11^{****} \to 0$ $^{**}1011 \to 0$ $^{**}11^{**} \to 0$ $^{******} \to 1$

Example 6.6. *Consider the rule of a two-field classification function $F = f_1 \cdot f_2$, where f_1 and f_2 represent intervals $[0,2] = (-1,3)$ and $[0,10] = (-1,11)$, respectively. In this*



(a) PreSOP



(b) Head-Tail Expression

FIGURE 6.9: Maps for two-field classification rule

case, by Theorem 6.1, their PreSOPs are

$$
\begin{aligned}
f_1 &= IN_0(2:-1,3) = LT(2:3) \\
&= x_1\bar{x}_0 \vee \bar{x}_1 \\
f_2 &= IN_0(4:-1,11) = LT(4:11) \\
&= x_3\bar{x}_2x_1\bar{x}_0 \vee x_3\bar{x}_2\bar{x}_1 \vee \bar{x}_3.
\end{aligned}
$$

The intersection of these PreSOPs produce a TCAM with 7 words (a 6-product PreSOP and a universal product that makes the rest values 0*'s), which are shown in the upper group of rows in Table 6.4. In this case, the action value 1 corresponds to* Accept*, while the action value 0 corresponds to* Discard*. Fig. 6.9(a) shows the map.*

However, if we represent these functions by head-tail expressions (Theorem 6.3), we have

$$f_1 = IN_0(2:-1,3) = LT(2:3) \tag{6.1}$$
$$= (\overline{x_1 x_0}) \cdot (1)$$
$$f_2 = IN_0(4:-1,11) = LT(4:11) \tag{6.2}$$
$$= (\overline{x_3 \bar{x}_2 x_1 x_0}) \cdot (\overline{x_3 x_2}) \cdot (1)$$

If we obtain the intersection of f_1 and f_2 in Table 6.4, then we have 6 words, which are shown by the lower group of rows in Table 6.4.

Furthermore, we can generate the simplified expression directly from Eq. (6.1) *and Eq.* (6.2)

$$F = f_1 \cdot f_2 = (\overline{x_5 x_4}) \cdot (\overline{x_3 \bar{x}_2 x_1 x_0}) \cdot (\overline{x_3 x_2}) \cdot (1).$$

Note that to perform the intersection *operation between f_1 and f_2, we have to increase the indexes of variable of factors in f_1 by four (the number of variable in f_2). The final TCAM requires only $p_1 + p_2 + 1 = 1 + 2 + 1 = 4$ words as shown in Fig. 6.9(b), where $p_1 + 1$ and $p_2 + 1$ are the number of words for of the head-tail expressions in f_1 and f_2, respectively.* ■

Since the application of the absorption law is time-consuming, we generate simplified expressions directly.

6.6 Number of Factors to Represent a Multi-Field Classification Rule

In this section, we derive the number of factors to represent a multi-field classification rule. This formula is useful to estimate the TCAM size for the multi-field classification functions [52].

Lemma 6.5. *Assume that a field function f_i is represented with a sum of u_i distinct head-tail expressions, and m_i (disjoint) products, and each head-tail expression has $p_{ij}+1$*

factors. Then, the total number of factors for the field function is

$$\tau_i = m_i + \sum_{j=1}^{u_i}(p_{ij}+1).$$

Proof: By Lemma 6.1, any logic function can be represented as a sum of interval functions. The number of factors for f_i is the sum of the number of products in the PreSOP and the number of factors of the head-tail expressions. □

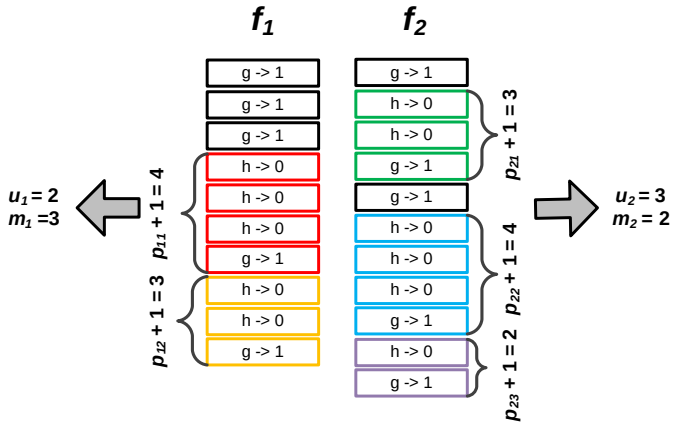


FIGURE 6.10: Description of two-field classification rule

Example 6.7. *Fig. 6.10 shows an example of a two-field classification rule: $F = f_1 \cdot f_2$, where f_1 is represented by a PreSOP with three products, and two head-tail expressions, while f_2 is represented by a PreSOP with two products, and three head-tail expressions. At the bottom rows of head-tail expressions, there are tail factors representing constants 1's. However, they are not counted in m_i.* ■

Lemma 6.6. *Let $F = \bigwedge_{i=1}^{k} f_i$ be a rule. Then, the number of factors to represent a k-field classification rule is at most*

$$\prod_{i=1}^{k}(m_i + s_i),$$

where m_i is the number of the products in a PreSOP for f_i, u_i is the number of the head-tail expressions for f_i, and $s_i = \sum_{j=1}^{u_i}(p_{ij}+1)$ for $i = 1, 2, \cdots, k$, and each head-tail expression has $p_{ij}+1$ factors.

6.6.1 Number of Products in a PreSOP to Represent a Multi-Field Classification Rule

Corollary 6.1. *When each field is represented by a PreSOP, the number of products to represent a rule is at most*

$$\prod_{i=1}^{k} m_i,$$

where m_i is the number of products in the PreSOP to represent the i-th field.

6.6.2 Number of Factors in a Head-Tail Expression to Represent a Multi-Field Classification Rule

Lemma 6.7. *When all the fields are represented by head-tail expressions, the rule can be represented by a head-tail expression with*

$$\left(1 + \sum_{i=1}^{k} \frac{(\sum_{j=1}^{u_i} p_{ij})}{u_i}\right) \cdot U$$

factors, where $U = \prod_{i=1}^{k} u_i$, and u_i is the number of the head-tail expression for f_i.

Proof: We can directly generate simplified head-tail expression for a multi-field classification rule. Consider the case of $k = 2$ and $F = f_1 \cdot f_2$. Instead of representing the function by $(p_{11} + 1)(p_{21} + 1)$ factors, it can be represented by $p_{11} + p_{21} + 1$ factors as shown in Example 6.6. When the first field has u_1 head-tail expressions and the second field has u_2 head-tail expressions, the total number of factors is

$$\sum_{i=1}^{u_2} \sum_{j=1}^{u_1} (p_{1j} + p_{2i} + 1) = u_2 \sum_{j=1}^{u_1} p_{1j} + u_1 \sum_{j=1}^{u_2} p_{2j} + u_1 u_2,$$

where $U = u_1 u_2$ is the number of the new head-tail expressions produced by the intersection operations. Extension for larger k is straightforward. □

6.6.3 Number of Factors in a PreSOP and Head-Tail Expression to Represent a Multi-Field Classification Rule

Lemma 6.8. *When some fields of a k-field classification rule are represented by PreSOPs and other fields are represented by head-tail expressions, the number of factors*

is

$$\left(\prod_{i=1}^{t-1} m_i\right)\left(1+\sum_{i=t}^{k}\frac{(\sum_{j=1}^{u_i} p_{ij})}{u_i}\right)\cdot U,$$

where m_i is the number of the products in the PreSOP ($i = 1, 2, \cdots, k$), u_i is the number of head-tail expressions for f_i, and $U = \prod_{i=t}^{k} u_i$.

Proof: It is clear from Lemma 6.6 and Lemma 6.7. □

Example 6.8. *Derive numbers of products or factors:*

- *The maximum number of products in a PreSOP for a k-field classification rule. As shown in [60],[41], an interval function can be represented with at most $2(n-1)$ products by a PreSOP. Thus, by Lemma 6.6, we have*

$$\prod_{i=1}^{k} 2(n-1) = (2n-2)^k.$$

- *The maximum number of factors in a head-tail expression for a k-field classification rule when no simplification rule is applied. The number of factors in a head-tail expression for an interval function is at most n [51]. Thus, by Lemma 6.6, we have*

$$\prod_{i=1}^{k} n = n^k.$$

- *The number of factors in a head-tail expression for a k-field classification rule, when the simplified expression is generated directly, and each field is represented by a single head-tail expression. In this case, by Lemma 6.7, we have*

$$1+\sum_{i=1}^{k} p_{i1},$$

since $U = \prod_{i=1}^{k} u_i = 1$. ■

Example 6.9. *Consider the two-field classification function $F = f_1 \cdot f_2$, where both f_1 and f_2 represent the interval $(0, 2^{16}-1)$. This function is often used to illustrate the* ***rule expansion*** *in TCAM realizations [17],[27],[60].*

Note that f_i corresponds to $IN_0(n : A, B)$ in Theorem 6.1, where $n = 16, A = 0$, and $B = 2^{16}-1$. The number of products in the PreSOP for f_i is $2(n-1) = 30$, and the PreSOP for F requires $30 \times 30 = 900$ products.

The head-tail expressions are

$$\begin{aligned} f_1 &= (\overline{\bar{x}_{n-1}\bar{x}_{n-2}\dots\bar{x}_0})\cdot(\overline{x_{n-1}x_{n-2}\dots x_0})\cdot(1). \\ f_2 &= (\overline{\bar{y}_{n-1}\bar{y}_{n-2}\dots\bar{y}_0})\cdot(\overline{y_{n-1}y_{n-2}\dots y_0})\cdot(1). \end{aligned}$$

Thus, both f_1 and f_2 require only three factors, and the head-tail expression for F requires $3\times 3 = 9$ factors if no simplification is applied. When the simplified expression is generated directly, we have $F = (\overline{\bar{x}_{n-1}\bar{x}_{n-2}\dots\bar{x}_0})\cdot(\overline{x_{n-1}x_{n-2}\dots x_0})\cdot(\overline{\bar{y}_{n-1}\bar{y}_{n-2}\dots\bar{y}_0})\cdot(\overline{y_{n-1}y_{n-2}\dots y_0})\cdot(1)$, which requires only five factors. ■

Example 6.10. *Find the number of factors for a two-field rule $F = f_1\cdot f_2$, where each function is represented by a PreSOP and head-tail expressions. First, by Lemma 6.6, we have*

$$\begin{aligned} &(m_1+s_1)(m_2+s_2) \\ &\quad = m_1m_2 + m_1s_2 + m_2s_1 + s_1s_2, \end{aligned}$$

where $s_i = \sum_{j=1}^{u_i}(p_{ij}+1)$. By Lemma 6.7 and 6.8, the number of factors is reduced to

$$\begin{aligned} &m_1m_2 + m_1\sum_{j=1}^{u_2}(p_{2j}+1) + m_2\sum_{j=1}^{u_1}(p_{1j}+1) \\ &+ u_2\sum_{j=1}^{u_1}p_{1j} + u_1\sum_{j=1}^{u_2}p_{2j} + u_1u_2, \end{aligned}$$

where u_i is the number of head-tail expressions for f_i. ■

Example 6.11. *Find the number of factors for the function in Example 6.6 when both fields (f_1 and f_2) are represented by head-tail expressions. Consider Lemma 6.7, in f_1 and f_2, we have $m_1 = 0$, $m_2 = 0$, $u_1 = 1$, $u_2 = 1$, $p_{11} = 1$ and $p_{21} = 2$. Thus, the number of factors is*

$$\sum_{i=1}^{2} p_{i1} = 1 + 3 = 4.$$

Example 6.12. *Consider the rule of the two-field classification function $F = f_1\cdot f_2$, where f_1 and f_2 represent $(9800, 42112)$ and $(6438, 36296)$, respectively. Table 6.5 shows the TCAM representations of f_1 and f_2, where f_1 has 11 factors, while f_2 has 12 factors. They are represented by PreSOPs and head-tail expressions, and have the structures in Fig. 6.10. Note that the last row of each column has a universal product that defines the non-covered values as 0's. The universal product is always attached to the last row of the TCAM as a default value and it is* not involved *in the reduction of the TCAM. If the function is represented by a head-tail expression, then the last row should be the*

universal product with the action value 1. In this case, the action value 1 corresponds to Accept *and the action value 0 corresponds to* Discard.

TABLE 6.5: TCAM Words for Example 6.12

Field f_1	Field f_2
101001000****** → 1	1000110111000*** → 1
101000********** → 1	1000110111****** → 0
100*********** → 1	1000111********* → 0
0010011001001000 → 0	1000*********** → 1
0010011001000*** → 0	0001100100100111 → 1
0010011000****** → 0	0001100100100*** → 0
0010011********* → 1	00011001000***** → 0
00100********** → 0	00011000******** → 0
000*********** → 0	00011********** → 1
0*********** → 1	000*********** → 0
********** → 0	0*********** → 1
	********** → 0

In this case, in f_1 *and* f_2*, we have* $m_1 = 3$*,* $m_2 = 2$*,* $u_1 = 2$*, and* $u_2 = 3$*. In the head-tail expressions for* f_1*, we have* $p_{11} = 3$ *and* $p_{12} = 2$*, while in the head-tail expressions for* f_2*, we have* $p_{21} = 2$*,* $p_{22} = 3$*, and* $p_{23} = 1$*. Thus, as derived in Example 6.10, the expression requires*

$$\begin{aligned} & m_1 m_2 + m_1 \sum_{j=1}^{3} (p_{2j} + 1) + m_2 \sum_{j=1}^{2} (p_{1j} + 1) \\ & \quad + 3 \sum_{j=1}^{2} p_{1j} + 2 \sum_{j=1}^{3} p_{2j} + (2)(3) \\ & = (3)(2) + (3)(9) + (2)(7) + (3)(5) + (2)(6) + (2)(3) \\ & = 80 \end{aligned}$$

factors. Note that the straightforward method requires $10 \times 11 = 110$ *factors.*

When both fields are represented by PreSOPs, the binary representations of 9800*,* 42112*,* 6438*, and* 36296 *are*

$$\begin{aligned} \vec{a}_1 &= (0,0,1,0,0,1,1,0,0,1,0,0,1,0,0,0), \\ \vec{b}_1 &= (1,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0), \\ \vec{a}_2 &= (0,0,0,1,1,0,0,1,0,0,1,0,0,1,1,0), \text{ and} \\ \vec{b}_2 &= (1,0,0,0,1,1,0,1,1,1,0,0,1,0,0,0), \end{aligned}$$

respectively. By Theorem 6.1, the numbers of products in PreSOPs for f_1 and f_2, are 13 and 15, respectively. If we consider Property 5.1, then the number of products for the two-field classification rule in the PreSOP is $13 \times 15 = 195$. ■

6.7 Algorithm to Generate Simplified Expressions for Multi-Field Classification Functions

In this section, we present an algorithm to generate simplified expressions for multi-field classification functions called MFHT. For one-field and two-field classification functions, methods using dynamic programming have been proposed in [49].

Fig. 6.11 shows the pseudocode for MFHT for two-field classification function. In this algorithm, head-tail expressions are detected, and a simplified expression is generated directly. The inputs are $\{ListF_1, Act_1\}$ and $\{ListF_2, Act_2\}$, while the output is $\{ListOut, ActOut\}$. $ListF_1$ and $ListF_2$ consists of the factors (products) of head-tail expressions (PreSOPs) for f_1 and f_2, respectively. τ_1 and τ_2 denote the number of factors (products) in $ListF_1$ and $ListF_2$, respectively. First, each action in Act_1 is checked. If $Act_1[i]$ is 1, then a disjoint or a PreSOP product is detected. Therefore, $ListF_1[i]$ and all the factors in $ListF_2$ are concatenated. In Fig. 6.11, the concatenation operation is denoted by $\circ$. Otherwise, a head-tail expression in $ListF_1[i]$ is detected. In this case, if a disjoint/PreSOP product is detected in $ListF_2[j]$, then all the factors in head-tail expression for $ListF_1[i]$ are concatenated to $ListF_2[j]$. size($ListF_1[i]$) denotes the number of factors for the head-tail expression. Lastly, if in both $ListF_1[i]$ and $ListF_2[j]$ are detected as the head-tail expressions, then the simplified expression is directly generated. Fig. 6.11 shows that the time complexity of the algorithm is $\tau_1\tau_2$ steps or $O(r^2)$. After checking $ListF_1[i]$, all factors in $ListF_2$ are concatenated with $ListF_1[i]$.

Consider the case, where the function has three fields. Let the number of reduced factors be τ_1' where the reduced factors are the outputs produced by applying MFHT in $ListF_1$ and $ListF_2$ resulting $ListOut$, and $ListF_3$ stores all factors representing f_3. In this case, we can compute the factors of the function by replacing the data as follows: $ListF_1 \leftarrow ListOut$, and $ListF_2 \leftarrow ListF_3$. Then, for reducing the third factors, it costs at most $\tau_1'\tau_3$ steps or $O(r^2) \cdot r \approx O(r^3)$. This shows that the complexity of MFHT for k-field classification functions is $O(r^k)$.

Example 6.13. *Consider the rule of the two-field classification function $F = f_1 \cdot f_2$, where both f_1 and f_2 represent $(0, 15)$, and they are defined by 4-bit numbers. From Example 6.4, we have the head-tail expressions for f_1 and f_2, and their TCAM representations as shown in Table 6.6(a).*

MFHT for two-field classification function:

/∗ Input: {$ListF_1$, Act_1} and {$ListF_2$, Act_2} that store all the factors and actions for f_1 and f_2, respectively. ∗/

/∗ Output: {$ListOut$, $ActOut$}, concatenated and reduced factors and actions represented by head(H)-tail(T) expressions (HTs) and disjoint products with the total number of factors τ. ∗/

1: Let τ_1 be the number of factors in $ListF_1$, and τ_2 be the number of factors in $ListF_2$.

2: $\tau \leftarrow 0$.
3: **for** $i = 0$; $i < \tau_1$; $i++$ **do**
4: **if** $Act_1[i] == 1$ **then**
/∗ $ListF_1[i]$ is a disjoint/PreSOP product. ∗/
5: **for** $j = 0$; $j < \tau_2$; $j++$ **do**
6: $ListOut[\tau] \leftarrow ListF_1[i] \circ ListF_2[j]$.
7: $ActOut[\tau] \leftarrow Act_2[j]$.
8: $\tau \leftarrow \tau + 1$.
9: **end for**
10: **else**
/∗ An HT is detected in $ListF_1[i]$. ∗/
11: **for** $j = 0$; $j < \tau_2$; $j++$ **do**
12: **if** $Act_2[j] == 1$ **then**
/∗ $ListF_2[j]$ is a disjoint/PreSOP product. ∗/
13: $ListOut[\tau] \leftarrow$ (HT of $ListF_1[i]$) $\circ$ $ListF_2[j]$.
14: $ActOut[\tau] \leftarrow Act_1[i]$.
15: $\tau \leftarrow \tau +$ size($ListF_1[i]$).
16: **else**
/∗ HTs are detected in $ListF_1[i]$ and $ListF_2[j]$. Generate simplified expression from $ListF_1[i]$ and $ListF_2[j]$: ∗/
17: $ListOut[\tau] \leftarrow$ (Hs of $ListF_1[i]$) $\circ$ (T of $ListF_2[j]$).
18: $ActOut[\tau] \leftarrow 0$.
19: $\tau \leftarrow \tau +$ size($ListF_1[i]$) $- 1$.
20: $ListOut[\tau] \leftarrow$ (T of $ListF_1[i]$) $\circ$ (Hs of $ListF_2[j]$).
21: $ActOut[\tau] \leftarrow 0$.
22: $\tau \leftarrow \tau +$ size($ListF_2[j]$) $- 1$.
23: $ListOut[\tau] \leftarrow$ (T of $ListF_1[i]$) $\circ$ (T of $ListF_2[j]$).
24: $ActOut[\tau] \leftarrow 1$.
25: $\tau \leftarrow \tau + 1$.
26: **end if**
27: **end for**
28: **end if**
29: **end for**
30: Terminate.

FIGURE 6.11: Pseudocode for MFHT

First, the first action of the factors in f_1 ($Act_1[0]$) is checked. Since the value of $Act_1[0]$ is 0 (line 4 of Fig. 6.11), a head-tail expression is detected in f_1. Next, the first action of the factors in f_2 ($Act_2[0]$) is checked. A head-tail expression is also detected in f_2.

TABLE 6.6: TCAM Words for Example 6.13

(a) TCAM for Each Field

HT of f_1		HT of f_2	
Hs[a]	0000 → 0	Hs	0000 → 0
	1111 → 0		1111 → 0
T[b]	**** → 1	T	**** → 1

[a]Hs: Head Factors
[b]T: Tail Factor

(b) Simplified TCAM for F

$F = f_1 \cdot f_2$	
Hs	0000**** → 0
	1111**** → 0
	****0000 → 0
	****1111 → 0
T	******** → 1

Thus, we generate the simplified expression directly: 1) concatenate the head factors (Hs) of f_1 and the tail factor (T) of f_2, 2) concatenate T of f_2 and Hs of f_1, and 3) concatenate T of f_1 and f_2. We have the simplified TCAM for F with 5 words as shown in Table 6.6(b).

To assess the effectiveness of MFHT, we also compare MFHT to an algorithm called single-field head-tail generator (SFHT) [51] where each field is represented by head-tail expressions and no simplification is performed among the fields. If SFHT is applied instead of MFHT, F requires $3 \times 3 = 9$ words. Moreover, if we represent both of the fields by PreSOPs, from Example 6.1, we have 6 products for each field and the TCAM for F requires $6 \times 6 + 1 = 37$ words. ■

6.8 Experimental Results

Since the data for access control list (ACL) and firewall (FW) are confidential in nature, no benchmark data are available for packet classifications. ClassBench is software to generate benchmark data for evaluation [59]. First, we generated a five-field random classification function by ClassBench, where the two fields (source port and destination port) are represented by intervals. We also developed an algorithm PreSOPG to generate expression, where each field is PreSOP [53]. Table 6.7 shows that, the SFHT achieved 47.76% reduction, while MFHT achieved 57.85% reduction over PreSOPG for random rules.

TABLE 6.7: Number of TCAM Words for Random Rules

#Rules	PreSOPG	SFHT	MFHT
50000	9791417 (100%)	5114996 (52.24%)	4127403 (42.15%)

Next, we generated five-field ACL and FW functions shown in Table 6.8, by ClassBench. In ACL functions, the source port has only the trivial interval $[0, 65535]$ (which has the size of the interval $C = 65534$). Because the source port field can be represented without a literal, the MFHT produced the same solutions as SFHT. In FW functions where both source and destination ports have non-trivial intervals, MFHT produces solutions with fewer factors than SFHT. MFHT achieved 52% reduction for ACL and FW rules over PreSOPG.

In these experiments, we generated 50000 rules to see the run time of the algorithms. In practice, the numbers of rules are between 100 until 41000 rules depend on applications [26, 49].

TABLE 6.8: Number of TCAM Words for ACL and FW Rules

Data	#Rules	#DIS[a]	#DIP[b]	PreSOPG Words	SFHT Words	MFHT Words
ACL1	49910	1	35	68426	63880	63880
ACL2	48461	1	3	98139	55651	55651
ACL3	49894	1	38	95577	63585	63585
ACL4	49633	1	52	89214	62015	62015
ACL5	39039	1	5	51825	43797	43797
FW1	48442	3	3	167002	59766	56588
FW2	49313	2	1	96723	58795	58795
FW3	47275	3	3	136045	55653	53234
FW4	46774	6	6	313330	84038	79684
FW5	46847	3	4	110542	54074	52343
Total				1226823	601254	589572
Ratio				100%	49%	48.05%

[a]DIS: Distinct Intervals in Source Port ($C > 1$).
[b]DIP: Distinct Intervals in Dest. Port ($C > 1$).

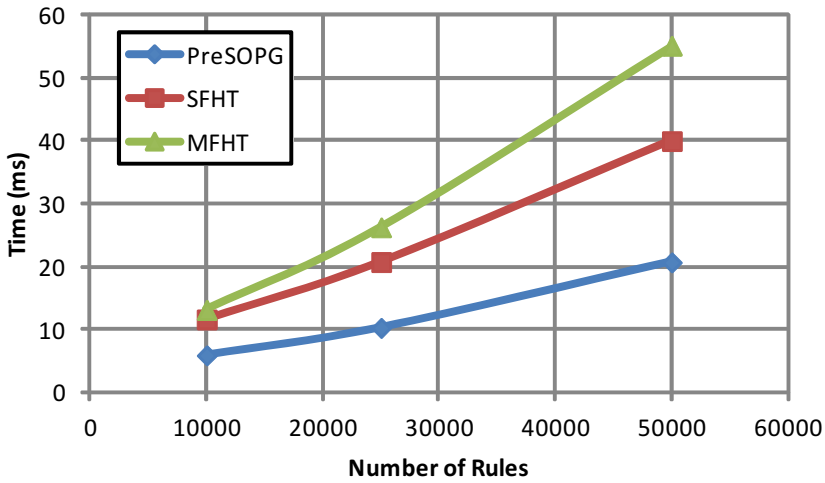


FIGURE 6.12: Comparison of execution times for PreSOPG, SFHT, and MFHT in millisecond

We also compared the execution times of PreSOPG, SFHT, and MFHT for classification functions with $r = 10000$ to 50000 ACL and FW rules. In this case, we generated rules by setting the variable $<$*number of rules*$>$ to 10000, 25000, and 50000 in the ClassBench

program. Fig. 6.12 shows CPU time in millisecond. This shows that the execution times of SFHT and MFHT are close. But, when the number of rules increases, the CPU time for MFHT grows faster than SFHT because MFHT generates simplified expression for multi-field classification functions ($O(r^2)$). Moreover, the average execution time to simplify a single rule for MFHT is nearly 1 microsecond. In the experiments, we used a 2 GHz Intel Core i7 with 8 GB memory and 64 bits OS-X.

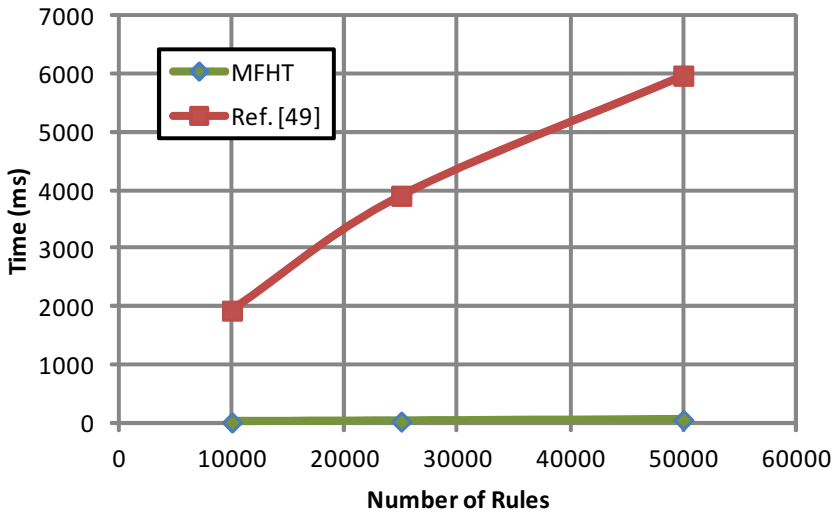


FIGURE 6.13: Comparison of execution times for MFHT and Ref. [49] in millisecond

Lastly, we compared the execution time of MFHT (as in Fig. 6.12) and the execution time of dynamic programming algorithm represented in [49]. The method [49] using dynamic programming produced expressions with the same number of words as MFHT. The algorithm [49] runs in $O(rwn^2)$ worst case time, where r is the number of rules, w is the number of actions, and n is the number of bits. Furthermore, in Fig. 6.13, we can see that MFHT is much faster (100 times) than the algorithm in [49].

6.9 Conclusion

In this chapter, first, we showed a fast method to simplify rules for packet classification in TCAMs. We partitioned rules into groups so that each group has the same source address, destination address and protocol. After that, we simplified rules by removing redundant rules. We have developed a computer program to simplify rules. Experimental results show that this method reduces the size of rules up to 57% of the original specification for ACL5 rules, 73% for ACL3 rules, and 87% for overall rules. This algorithm is useful to reduce TCAMs for packet classification.

Second, we presented a method to reduce the number of factors in a head-tail expression for multi-field classification functions. We derived the head-tail expression for a rule of a

multi-field classification function. Furthermore, we presented an algorithm to generate simplified expressions for multi-field classification functions. Experimental results show that MFHT [54, 56] achieved 58% reduction over PreSOPG for random rules, and 52% reduction for ACL and FW rules. Moreover, MFHT is more than 100 times faster than that of the reference [49].

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we discussed about prefix sum-of-products expressions (PreSOPs), head-tail expressions and their implementation in classification functions.

First, we considered PreSOPs for interval functions and we derived minimum PreSOPs and the numbers of products in the minimum PreSOP (τ_p) to represent an interval functions. Then, we derived the formula $\Psi(n,\tau_p)$ for the number of interval functions that require τ_p products in PreSOPs. And, we showed by experiments that the average number of products in MSOPs is approximated by $\tau = \log_2(B - A)$ where A and B are the endpoints of the intervals. We also showed that the average number of products $\mu(n)$ needed to represent interval functions of an n-bit field by PreSOPs is $\mu(n) \approx n - 2$, with an approximate variance $\sigma^2(n) \approx \frac{n}{2} + 1$. We showed by numerical computations that more than 99.9% of the interval functions of an n-bit field can be represented by PreSOPs with $\lceil \frac{3}{2}n - 1 \rceil$ products, when $n > 12$. These results are useful for estimating the TCAM size for packet classifications.

Second, we proposed a new method to represent interval functions called head-tail expressions. We derived the head-tail expressions for *greater-than* ($GT(n : A)$) and *less-than* ($LT(n : B)$) functions. We found that when there is a special property in the binary representation of A for $GT(n : A)$ function or in the binary representation of B for $LT(n : B)$ function, we can represent the interval functions efficiently (*i.e.,* with a fewer factors). We prove that an HT requires at most n factors to represent any interval function $IN_0(n : A, B)$. By a heuristic algorithm, we obtained average numbers of factors to represent the interval functions in HTs for up to $n = 16$. And, we conjecture that, for sufficiently large n, the average number of factors by HTs to represent

n-variable interval functions is $\frac{2}{3}n - \frac{5}{9}$. We also show that, for $n \geq 10$, HTs generated by our heuristic program require at least 20% fewer factors than MSOPs, on the average.

Third, we derived head-tail expressions to simplify TCAMs for single-field classification functions. We developed a fast prefix sum-of-product generator. And, we proposed DHT to produce head-tail expressions directly from the endpoints (A,B). We used ClassBench to generate benchmark functions of packet classification. And we generate the simplified packet classifier by using DHT. Experimental results showed that DHT is 6×10^5 times faster and produces smaller TCAMs than ESPRESSO-EXACT. DHT is useful for minimizing the TCAM size in packet classification.

Finally, we presented methods to simplify TCAMs for multi-field classification funcitions. First, we showed a quick method to simplify rules for packet classification. We partitioned rules into groups so that each group has the same source address, destination address and protocol. After that, we simplified rules by removing redundant rules. We have developed a computer program to simplify rules. Experimental results show that this method reduces the size of rules up to 57% of the original specification for ACL5 rules, 73% for ACL3 filter, and 87% for overall rules. Second, we use head-tail expressions to reduce TCAM size for multi-field classification functions. We derived the head-tail expression for a rule of a multi-field classification function. Furthermore, we presented an algorithm to generate simplified expressions for multi-field classification functions. Experimental results show that MFHT achieved 58% reduction over Pre-SOPG for random rules, and 52% reduction for ACL and FW rules. Moreover, MFHT is fast. These methods are useful to reduce TCAMs for packet classification.

7.2 Future Work

There are still many open problems in this area of logic minimization, especially for internet applications, which are interesting to explore. The first is finding the optimal head-tail expressions for arbitrary functions: If we come up with a huge set of rules, how to find the optimal (*i.e.,* the smallest) TCAM represented by HTs and prove their optimality are still big challanges. And, the idea, such as combining the fastest and the most efficient algorithms in packet classification into a one comprehensive system (*i.e.,* system-on-chip (SoC)) that is stored in devices, is also a promising research. Moreover, in these days, the network connection speed already reaches the frequency of GHz. Thus, parallel computing may be a solution along with a faster TCAM technology for the next generation high-speed packet classification. Finally, the application of HTs for LSI-CAD, such as logic sythesis in FPGA, since our research results show that HTs

require fewer factors than MSOPs, is another challange to reduce the sizes of a circuits in FPGA blocks.

Acknowledgements

Foremost, I would like to express my sincere gratitude and appreciation to my advisor, Prof. Tsutomu Sasao, for his constant guidance, invaluable assistance, endless patience, and encouragement throughout the progress of this research, without which it could not have been completed. Even, after his retirement from Kyushu Institute of Technology, he still guided me in finishing my doctoral study. He taught me all the technical skills that I currently have; how to set up the research goal, how to formulate and solve the problem, how to write a paper, and how to prepare a technical presentation. I will treasure what I learned from Professor Sasao for the rest of my life.

I would also like to thank Professor Seiji Kajihara for being my second advisor after Professor Sasao's retirement for his helps and supports during my final year of the doctoral program.

It is my great pleasure to thank Professor Xiaoqing Wen and Professor Kazuyuki Nakamura for their invaluable helps and advices since the beginning of my study in here. And also special thanks go to Prof. Jon T. Butler for his valuable comments on my research during his time at Kyushu Institute of Technology.

I would also extend my gratitude to Prof. Xiaoqing Wen, Prof. Kazuyuki Nakamura and Prof. Akihiro Fujiwara who kindly agreed to be members of both the qualification and the thesis committees.

And I would like to thank to the financial support for my doctoral study, the Japanese Government Scholarship (MEXT).

The researches in my thesis were partly supported by the Grant in Aid for Scientific Research of the Japan Society for the Promotion of Science (JSPS), the grant of the MEXT Regional Innovation Cluster Program, and by the Adaptable and Seamless Technology Transfer Program through target-driven R&D, JST.

Thanks to all members of Prof. Sasao Lab; Mr. Matsuura, Dr. Nakahara, Maeta, Kengo, Kai, and Shimada. I have had the pleasure of sharing interesting talk, space and

good time throughout my research. Special thanks to Mr. Munehiro Matsuura for his technical helps during my study, especially for the final year.

Last and most, I want to give thanks to my family, my wife Anita and my baby Aimi-Chan, for their love, unlimited support and patience until now and on.

List of Publications

Journal Publications

1. I. Syafalni and T. Sasao, "On the number of products in prefix SOPs for interval functions," *IEICE Transactions on Information and Systems*, vol. E96-D, no. 5, pp. 1086-1094, May 2013.

2. I. Syafalni and T. Sasao, "Head-tail expressions for interval functions," *IEICE Transactions on Fundamentals*, vol. E97-A, no. 10, October 2014.

International Conference Publications

1. I. Syafalni and T. Sasao," Head-tail expressions for interval functions," *The 17th Workshop on Synthesis And System Integration of Mixed Information technologies* (SASIMI-2012), pp. 94-99, Beppu, Ohita, Japan, March 8-9, 2012.

2. I. Syafalni and T. Sasao, "A TCAM simplification for packet classification," *International Workshop on Logic and Synthesis* (IWLS-2012), pp. 49-56, Berkeley, CA, USA, June 1-3, 2012.

3. I. Syafalni and T. Sasao, "A fast head-tail expression generator for TCAM–Application to packet classification," *IEEE Computer Society Annual Symposium on VLSI* (ISVLSI-2012), pp. 27-32, Amherst, MA, USA, August 19-21, 2012.

4. I. Syafalni and T. Sasao, "A TCAM generator for packet classification", *IEEE International Conference on Computer Design* (ICCD-2013), pp. 322-328, Asheville, NC, USA, October 6-9, 2013.

5. I. Syafalni and T. Sasao, "A fast simplification algorithm for packet classification", *The 18th Workshop on Synthesis And System Integration of Mixed Information Technologies* (SASIMI-2013), pp. 328-333, Sapporo, Japan, October 21-22, 2013.

Domestic Conference in Japan

1. I. Syafalni and T. Sasao, "Comparison of TCAM minimization algorithms for packet classification," *36th Note on Multiple-Valued Logic in Japan*, vol. 36, no. 7, pp. 1-6, Himeji, Japan, September 14-15, 2013.

Bibliography

[1] B. Agrawal and T. Sherwood, "Modeling TCAM power for next generation network devices," *IEEE ISPASS*, pp. 120-129, March 2006.

[2] B. Agrawal and T. Sherwood, "Ternary CAM Power and Delay Model: Extensions and Uses," *IEEE Trans. VLSI Systems*, vol. 16, no. 5, pp. 554-564, May 2008.

[3] S. Ahmad and R. Mahapatra, "TCAM enabled on-chip logic minimization," *DAC*, pp. 678-683, June 2005.

[4] S. Ahmad and R. Mahapatra, "An efficient approach to on-chip logic minimization," *IEEE Transactions on VLSI*, pp. 1040-1050, Sept. 2007.

[5] M. J. Akhbarizadeh et.al., "A TCAM-based parallel architecture for high-speed packet forwarding," *IEEE Transactions on Computers*, vol. 56, no. 1, January 2007.

[6] F. Baboescu et.al., "Packet classification for core routers: Is there an alternative to CAMs?," *IEEE INFOCOM*, vol. 1, pp. 53-63, March 2003.

[7] F. Baboescu and G. Varghese, "Scalable packet classification," *IEEE/ACM TON*, vol.13, no.1, pp. 2-14, Feb. 2005.

[8] A. Bremler-Barr and D. Hendler, "Space-efficient TCAM-based classification using gray coding," *IEEE INFOCOM*, pp. 1388-1396, May 2007.

[9] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers: Boston, MA., 1984.

[10] A. DasGupta, *Fundamentals of Probability: A First Course*, Springer: New York, 2010.

[11] S. Dharmapurikar et.al., "Fast packet classification using bloom filters," *ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, pp. 61-70, December 2006.

[12] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, "Packet classifiers in ternary CAMs can be smaller," *ACM SIGMETRICS*, pp. 311-322, June 2006.

[13] J. F. Gimpel, "The minimization of TANT networks," *IEEE Transactions on Electronic Computers*, vol. 16, no. 1, pp. 18-38, Feb. 1967.

[14] M. G. Gouda and A. X. Liu,"Structured firewall design," *Elsevier Computer Networks*, vol. 51, no. 4, pp. 1106-1120, March 2007.

[15] P. Gupta and N. McKeown, "Packet classification on multiple fields," *ACM SIGCOMM*, pp. 147-160, Sept. 1999.

[16] S. Iyer et.al., "ClassiPl: An architecture for fast and flexible packet classification," *IEEE Network*, vol. 15, no. 2, pp. 33-41, March 2001.

[17] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," *ACM SIGCOMM*, pp. 193-204, August 2005.

[18] H. Liu, "Efficient mapping of range classifier into ternary-CAM," *IEEE HOT Interconnects*, pp. 95-100, August 2002.

[19] H. Liu, "Routing table compaction in ternary CAM," *IEEE Micro*, vol. 22, no. 1, pp. 58-64, January 2002.

[20] A. X. Liu, "Firewall policy verification and trouble shooting," *Elsevier Computer Network*, vol. 53, no. 16, pp. 2800-2809, July 2009.

[21] A. X. Liu, "Diverse firewall design," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 9, pp. 1237-1251, September 2009.

[22] A. X. Liu et. al., "Compressing network access control lists," vol. 22, no. 12, pp. 1969-1977, December 2011.

[23] J. V. Lunteren and T. Engbersen, "Fast and scalable packet classification," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 560-571, May 2003.

[24] R. L. Lysecky and F. Vahid, "On-chip logic minimization," *ACM Design Automation Conference*, pp. 334-337, 2003.

[25] L. Lovasz, J. Pelikan, and K. Vesztergombi, *Binomial Coefficients and Pascal's Triangle*, Springer: New York, 2003.

[26] R. McGeer and P. Yalagandula, "Minimizing rulesets for TCAM implementation," *INFOCOM*, pp. 1314-1322, 2009.

[27] C. R. Meiners, A. X. Liu, and E. Torng, "TCAM razor: A systematic approach towards minimizing packet classifiers in TCAMs," *ICNP*, pp. 266-275, 2007.

[28] C. R. Meiners, A. X. Liu, and E. Torng, "Topological transformation approaches to optimizing TCAM-based packet classification systems," *ACM SIGMETRICS*, pp. 73-84, June 2009.

[29] T. Mishra and S. Sahni, "PETCAM–A power efficient TCAM for forwarding tables," *IEEE Symposium on Computers and Communications*, pp. 224-229, July 2009.

[30] H. Noda et.al., "A cost-efficient high-performance dynamic TCAM with pipelined hierarchical searching and shift redundancy architecture," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 1, pp. 245-253, January 2005.

[31] A. Nottingham and B. Irwin, "Parallel packet classification using GPU co-processors," *SAICSIT '10*, pp 231-241, October 2010.

[32] E. C. Oh and P. D. Franzon, "Technology Impact Analysis for 3D TCAM," *IEEE International Conference on 3D System Integration*, pp. 1-5, September 2009.

[33] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, No. 3, pp. 712-727, March 2006.

[34] R. Panigrahy and S. Sharma, "Sorting and searching using ternary CAMs," *IEEE Micro*, vol. 23, no. 1, pp. 44-53, January 2003.

[35] B. Reusch, "Generation of prime implicant from subfunctions and a unifying approach to the covering problem," *IEEE Trans. Comput.* Vol. C-24, No. 9. Sept. 1975, pp. 924-930.

[36] M. Ruiz-Sanches, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algortihms," *IEEE Network*, vol. 15, no. 2, pp. 8-23, March 2001.

[37] O. Rottenstreich and I. Keslassy, "Worst-case TCAM rule expansion," *INFOCOM Miniconference*, pp. 1-5, March 2010.

[38] O. Rottenstreich and I. Keslassy, "On the code length of TCAM coding schemes," *IEEE ISIT*, pp. 1908-1912, July 2010.

[39] T. Sasao and J. T. Butler, "Worst and best irredundant sum-of-products expressions," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 935-948, Sept. 2001.

[40] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers: Boston, MA., 1999.

[41] T. Sasao, "On the complexity of classification functions," *IEEE ISMVL*, pp. 57-63, May 2008.

[42] T. Sasao, "On the number of products to represent interval functions by SOPs with four-valued variables," *IEEE ISMVL*, pp. 282-287, May 2010.

[43] T. Sasao, *Memory-Based Logic Synthesis*, Springer 2011.

[44] B. Schieber, D. Geist, and A. Zaks, "Computing the minimum DNF representation of boolean functions defined by intervals," *Discrete Applied Mathematics, Elsevier*, vol. 149, Issue 1-3, pp. 154-173, Aug. 2005.

[45] S. Singh et.al., "Packet classification using multidimensional cutting," *ACM SIGCOMM '03*, pp. 213-224, Aug. 2003.

[46] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended TCAMs," *ICNP*, pp. 120-131, Nov. 2003.

[47] X. Sun et.al., "Packet classification consuming small amount of memory," *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, October 2005.

[48] Y. Sun and M. S. Kim, "Tree-based minimization of TCAM entries for packet classification," *IEEE CCNC*, pp. 1-5, January 2010.

[49] S. Suri, T. Sandholm, and P. Warkhede, "Compressing two-dimensional routing tables," *Algorithmica*, 35(4), 287-300, 2003.

[50] I. Syafalni and T. Sasao, "Head-tail expressions for interval functions", *SASIMI 2012*, March 2012.

[51] I. Syafalni and T. Sasao, "A fast head-tail expression generator for TCAM–Application to packet classification," *ISVLSI*, August 2012.

[52] I. Syafalni and T. Sasao, "A TCAM simplification for packet classification," *International Workshop on Logic and Synthesis*, pp.49-56, June 2012.

[53] I. Syafalni and T. Sasao, "On the numbers of products in prefix SOPs for interval functions," *IEICE Transaction on Information and System*, vol. E96-D, no 55, pp. 1086-1094, May 2013.

[54] I. Syafalni and T. Sasao, "A TCAM generator for packet classification", *IEEE International Conference on Computer Design*, pp. 322-328, October 2013.

[55] I. Syafalni and T. Sasao, "A fast simplification algorithm for packet classification", *SASIMI 2013*, pp. 328-333, October 2013.

[56] I. Syafalni and T. Sasao, "Comparison of TCAM minimization algorithms for packet classification," *36th Note on Multiple-Valued Logic in Japan*, vol. 36, no. 7, pp. 1-6, September 2013.

[57] Y. Qi et. al., "Packet classification algorithms: From theory to practice," *IEEE INFOCOM*, pp. 648-656, 2009.

[58] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238-275, 2005.

[59] D. E. Taylor, J. S. Turner, "ClassBench: a packet classification benchmark," *IEEE/ACM TON*, vol. 3, no. 15, pp. 499-511, 2007.

[60] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238-275, 2005.

[61] B. Vamanan et.al., "Efficuts: Optimizing packet classification for memory and throughput," *ACM SIGCOMM*, pp. 207-218, August 2010.

[62] P. Wang et.al., "High-speed packet classification for differentiated services in next-generation networks," *IEEE Transactions on Multimedia*, vol. 6, no. 6, pp. 925-935, December 2004.

[63] P. Warkhede et.al., "Fast packet classification for two-dimensional conflict-free filters," *IEEE INFOCOM*, pp. 1434-1443, April 2001.

[64] H. Yu, "A memory- and time-efficient on-chip TCAM minimizer for IP lookup," *DATE*, pp. 926-931, March 2010.