A TCAM Generator for Packet Classification

Infall Syafalni
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820-8502, Japan
infall@aries01.cse.kyutech.ac.jp

Tsutomu Sasao
Department of Computer Science
Meiji University
Kawasaki, Kanagawa 214-8571, Japan
sasao@cs.meiji.ac.jp

***Abstract*—In the internet, packets are classified by source and destination addresses and ports, as well as protocol type. Ternary content addressable memories (TCAMs) are often used to perform this operation. This paper shows a method to reduce the number of words in TCAM for multi-field classification functions. We use head-tail expressions to represent a multi-field classification rule. Furthermore, 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.**

***Index Terms*—Logic design, Head-tail expressions, Multi-field classification functions, TCAMs.**

I. Introduction

Logic minimization in packet classification is quite different from that of large-scale integrated circuit (LSI) design. A ternary content addressable memory (TCAM) is used instead of a two-level AND-OR circuit. The optimization is more complicated than that of sum-of-products expressions (SOPs) [4]. 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 [2], [6].

A. Packet Classification

In the internet, packets are classified by source address (SA), destination address (DA), source port (SP), and destination port (DP), as well as the protocol type (PO). Table I 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 I
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 [1], rules are applied from the top to the bottom. Thus, in Table I, 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 I can be considered as a five-field classification function.

Note that all the fields can be represented by intervals [7]. 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 $[7, 7]$.

B. TCAM

A content addressable memory (CAM) simultaneously compares the inputs vector with the entire list of registered vectors [5]. TCAM is a *de facto* standard in routers and devices for packet classification. Fig. 1 shows an example of a TCAM circuit [8]. 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.

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 $X = 8x_3 + 4x_2 + 2x_1 + x_0$. If 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 II(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 II(a) consists of all *don't cares*. Thus, the TCAM requires 7 words.

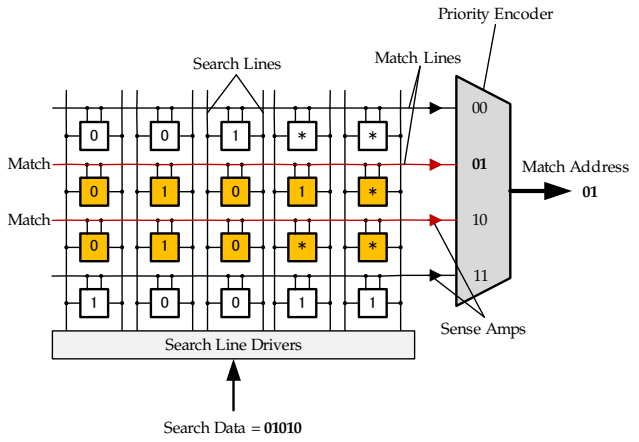


Fig. 1. TCAM Circuit

TABLE II
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 II(b), which uses only three words. In this case, the entries for the static random access memory (SRAM) also must be modified.

Rules for packet classification are modified frequently, thus a quick logic minimizer is desirable. Moreover, TCAMs dissipate high power and are expensive. To overcome these problems, the reduction of the number of words is crucial.

In this paper, 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. 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.

II. DEFINITION AND BASIC PROPERTIES

A. Prefix Sum-of-Products

Definition 2.1: ${x_i}^{a_i}$ denotes x_i when $a_i = 1$, and $\bar{x}_i$ when $a_i = 0$. x_i and $\bar{x}_i$ are **literals** of a variable x_i. The AND of literals is a **product**. The OR of products is a **sum-of-products expression** (SOP).

Definition 2.2: A **prefix SOP** (PreSOP) is an SOP consisting of products having the form $x^*_{n-1}x^*_{n-2}\dots x^*_{m+1}x^*_m$, where x^*_i is x_i or $\bar{x}_i$ and $m \le n-1$.

Example 2.1: $f(x_2, x_1, x_0) = \bar{x}_2\bar{x}_1x_0 \vee \bar{x}_2x_1 \vee x_2$ is a PreSOP. $f(x_2, x_1, x_0) = x_0 \vee x_1 \vee x_2$ is an SOP, for the same function, but is not a PreSOP. ■

In general, an SOP requires fewer products than a PreSOP to represent the same function [7]. However, in the internet communication area, PreSOPs are used instead of SOPs, since PreSOPs can be quickly generated from the binary decision tree of the functions [13].

B. Interval Functions

Definition 2.3: 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 $C = B - A - 1$.

Definition 2.4: An n-input **open interval function** is

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

An n-input **greater-than** (GT) function is

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

An n-input **less-than** (LT) function is

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

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

Lemma 2.1: A GT function can be represented by the PreSOP

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

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

Example 2.2: Consider the case of $n = 4$ and $A = 0$. The binary representation of A is $\vec{a} = (0, 0, 0, 0)$. Thus, $GT(4 : 0) = x_3 \vee \bigvee_{i=2}^{0} \left(\bigwedge_{j=3}^{i+1} \bar{x}_j \right) x_i = x_3 \vee \bar{x}_3x_2 \vee \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3\bar{x}_2\bar{x}_1x_0$. ■

Lemma 2.2: An LT function can be represented by the PreSOP

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

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

Theorem 2.1 [11]: 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 $\sum_{i=0}^{t-2} (\bar{a}_i + b_i)$.

Proof: See the reference [11]. □

The optimality of $GT(n : A)$, $LT(n : B)$, and $IN_0(n : A, B)$ functions represented by PreSOPs has been discussed in the reference [11].

Example 2.3: Let $A = 0$ and $B = 15$. The binary representations of A and B are $\vec{a} = (0, 0, 0, 0)$ and $\vec{b} = (1, 1, 1, 1)$, respectively. Thus, by Theorem 2.1, the PreSOP for $IN_0(4 : 0, 15)$ is $\bar{x}_3 x_2 \vee \bar{x}_3 \bar{x}_2 x_1 \vee \bar{x}_3 \bar{x}_2 \bar{x}_1 x_0 \vee x_3 \bar{x}_2 \vee x_3 x_2 \bar{x}_1 \vee x_3 x_2 x_1 \bar{x}_0$. Table II(a) shows the PreSOP realization of $IN_0(4 : 0, 15)$. ■

C. Classification Functions

Definition 2.5 [7]: A **classification function** with k **fields** is a mapping $F : P_1 \times P_2 \times \cdots \times P_k \to \{0, 1, 2, \cdots, r\}$, where $P_i = \{0, 1, \cdots, 2^{t_i} - 1\}$ $(i = 1, 2, \cdots, k)$. F is specified by a set of r rules. A **rule** consists of k fields, and each field is specified by an interval of t bits.

III. TCAM Simplification

A. Head-Tail Expression for Interval Functions

First, we use head-tail expressions (HTs) to efficiently represent interval functions [10]. We use HTs that were introduced to design NAND networks [3].

Definition 3.1: A **head-tail expression** has a form

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

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 paper, (product) and $\overline{(\text{product})}$ are called **factors**.

Example 3.1: $(\overline{x_2 x_1 x_0}) \cdot (\overline{\bar{x}_2 x_1 x_0}) \cdot (x_4 x_3) \vee \overline{(x_2 \bar{x}_1)} \cdot \overline{(\bar{x}_2 x_1)} \cdot (\bar{x}_4 \bar{x}_3)$ is a head-tail expression. ■

Section II-B showed that the number of products in a PreSOP for IN_0 function is $\sum_{i=0}^{t-2} (\bar{a}_i + b_i)$. Thus, in the worst case, an IN_0 function requires $2(n-1)$ products. However, by a head-tail expression, we often can represent the function with fewer TCAM words (factors) compared to that of in a PreSOP.

Lemma 3.1: An arbitrary interval function f can be represented by a **head-tail expression** Eq. (1).

Theorem 3.1 [10]: 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 $p + 1$ factors:

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

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.

Proof: See Appendix. □

Example 3.2: Let $A = 0$. The binary representation of A is $\vec{a} = (0, 0, 0, 0)$. By Theorem 3.1, we have a group of consecutive 0's, where $n = 4$, $p = 1$, $c_{p-1} = c_0 = 3$ and $d_0 = 4$. Thus,

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

where the number of factors is $p + 1 = 2$. ■

Theorem 3.2 [10]: 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:

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

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.

Proof: See Appendix. □

Example 3.3: Let $B = 15$. The binary representation of B is $\vec{b} = (1, 1, 1, 1)$. By Theorem 3.2, we have a group of consecutive 1's where $p = 1$, $c_{p-1} = c_0 = 3$ and $d_0 = 4$. Thus,

$$\begin{aligned} LT(4 : 15) &= \left(\overline{\bigwedge_{j=n-1}^{c_0+1} x_j^{b_j} \bigwedge_{i=c_0}^{c_0+1-d_0} x_i} \right) \cdot \left(\bigwedge_{j=n-1}^{c_0+1} x_j^{b_j} \right) \\ &= (\overline{x_3 x_2 x_1 x_0}) \cdot (1). \end{aligned}$$

Thus, $IN_0(4:0,15)$ can be represented by:

$$GT(4:0)\cdot LT(4:15) = (\overline{\bar{x}_3\bar{x}_2\bar{x}_1\bar{x}_0})\cdot(\overline{x_3x_2x_1x_0})\cdot(1).$$

Table II(b) shows the head-tail expression realization of $IN_0(4:0,15)$. The first factor $(\overline{\bar{x}_3\bar{x}_2\bar{x}_1\bar{x}_0})$ corresponds to the first word, the second factor $(\overline{x_3x_2x_1x_0})$ corresponds to the second word, and the last factor (1) corresponds to the last word in Table II(b). ■

B. 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 3.1: 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:

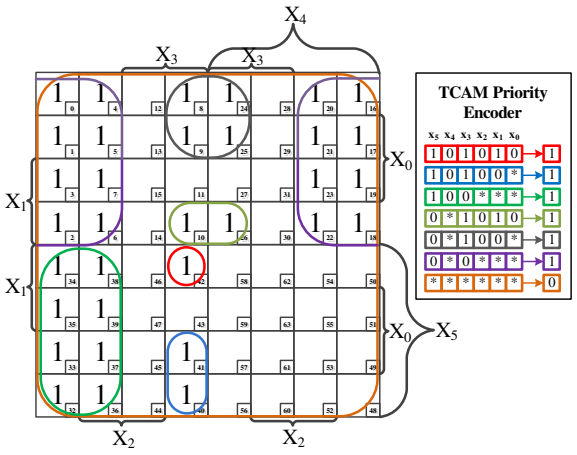
TABLE III
TCAM WORDS FOR EXAMPLE 3.4

Field f_1	Field f_2	Results
PreSOP $[0,2]$	PreSOP $[0,10]$	Products
$10 \to 1$	$1010 \to 1$	$101010 \to 1$
$0* \to 1$	$100* \to 1$	$10100* \to 1$
	$0*** \to 1$	$100*** \to 1$
		$0*1010 \to 1$
		$0*100* \to 1$
		$0*0*** \to 1$
		$****** \to 0$
Head-Tail Expr. $[0,2]$	Head-Tail Expr. $[0,10]$	Factors
$11 \to 0$	$1011 \to 0$	~~$11011 \to 0$~~
$** \to 1$	$11** \to 0$	~~$1111** \to 0$~~
	$**** \to 1$	$11**** \to 0$
		$**1011 \to 0$
		$**11** \to 0$
		$****** \to 1$

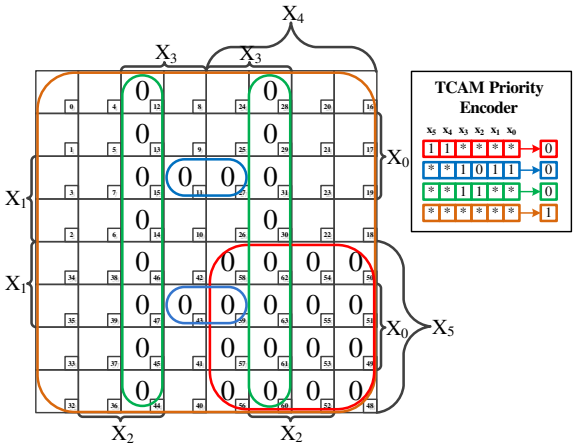
Example 3.4: Consider the rule of a two-field classification function $F = f_1 \cdot f_2$, where f_1 and f_2 represent intervals $[0,2] = (-1,3)$ and $[0,10] = (-1,11)$, respectively. In this case, by Theorem 2.1 or Lemma 2.2, their PreSOPs are

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

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 III. In this case, the action value 1 corresponds to *Accept*, while the action value 0 corresponds to *Discard*. Furthermore, Fig. 2(a) shows the map.



(a) PreSOP



(b) Head-Tail Expression

Fig. 2. Maps for two-field classification rule

However, if we represent these functions by head-tail expressions (Theorem 3.2), we have

$$\begin{aligned} f_1 &= IN_0(2:-1,3) = LT(2:3) \qquad (2) \\ &= (\overline{x_1x_0})\cdot(1) \\ f_2 &= IN_0(4:-1,11) = LT(4:11) \qquad (3) \\ &= (\overline{x_3\bar{x}_2x_1x_0})\cdot(\overline{x_3x_2})\cdot(1) \end{aligned}$$

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

Furthermore, we can generate the simplified expression directly from Eq. (2) and Eq. (3)

$$F = f_1 \cdot f_2 = (\overline{x_5x_4})\cdot(\overline{x_3\bar{x}_2x_1x_0})\cdot(\overline{x_3x_2})\cdot(1).$$

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. 2(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.

IV. Algorithm to Generate Simplified Expr. 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 [9].

Fig. 3 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. 3, 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. 3 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 $O(r^k)$.

TABLE IV
TCAM Words for Example 4.1

(a) TCAM for Each Field

HT of f_1		HT of f_2	
Hs[a]	0000 → 0 1111 → 0	Hs	0000 → 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

MFHT for two-field classification function:

```
/* Input: {ListF1, Act1} and {ListF2, Act2} that store all the factors and actions
   for f1 and f2, respectively. */
/* Output: {ListOut, ActOut}, concatenated and reduced factors and actions rep-
   resented by head(H)-tail(T) expressions (HTs) and disjoint products with the total
   number of factors τ. */
1: Let τ1 be the number of factors in ListF1, and τ2 be the number of factors in
   ListF2.
2: τ ← 0.
3: for i = 0; i < τ1; i++ do
4:     if Act1[i] == 1 then
           /* ListF1[i] is a disjoint/PreSOP product. */
5:         for j = 0; j < τ2; j++ do
6:             ListOut[τ] ← ListF1[i] ∘ ListF2[j].
7:             ActOut[τ] ← Act2[j].
8:             τ ← τ + 1.
9:         end for
10:    else
           /* An HT is detected in ListF1[i]. */
11:        for j = 0; j < τ2; j++ do
12:            if Act2[j] == 1 then
                   /* ListF2[j] is a disjoint/PreSOP product. */
13:                ListOut[τ] ← (HT of ListF1[i]) ∘ ListF2[j].
14:                ActOut[τ] ← Act1[i].
15:                τ ← τ + size(ListF1[i]).
16:            else
                   /* HTs are detected in ListF1[i] and ListF2[j]. Generate simplified
                   expression from ListF1[i] and ListF2[j]: */
17:                ListOut[τ] ← (Hs of ListF1[i]) ∘ (T of ListF2[j]).
18:                ActOut[τ] ← 0.
19:                τ ← τ + size(ListF1[i]) − 1.
20:                ListOut[τ] ← (T of ListF1[i]) ∘ (Hs of ListF2[j]).
21:                ActOut[τ] ← 0.
22:                τ ← τ + size(ListF2[j]) − 1.
23:                ListOut[τ] ← (T of ListF1[i]) ∘ (T of ListF2[j]).
24:                ActOut[τ] ← 1.
25:                τ ← τ + 1.
26:            end if
27:        end for
28:    end if
29: end for
30: Terminate.
```

Fig. 3. Pseudocode for MFHT

Example 4.1: 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 3.3, we have the head-tail expressions for f_1 and f_2, and their TCAM representations as shown in Table IV(a).

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. 3), 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. 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 IV(b).

To assess the effectiveness of MFHT, we also compare MFHT to an algorithm called single-field head-tail generator (SFHT) [10] 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 2.3, we have 6 products for each field and the TCAM for F requires $6 \times 6 + 1 = 37$ words. ■

V. 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 [12]. 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 develop an algorithm PreSOPG to generate expression, where each field is PreSOP [11]. Table V shows that, the SFHT achieved 47.76% reduction, while MFHT achieved 57.85% reduction over PreSOPG for random rules.

TABLE V
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 VI, 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 [4], [9].

TABLE VI
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 ($C > 1$).

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 filter>* to 10000, 25000, and 50000 in the ClassBench program. Fig. 4 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 an 2 GHz Intel Core i7 with 8 GB memory and 64 bits OS-X.

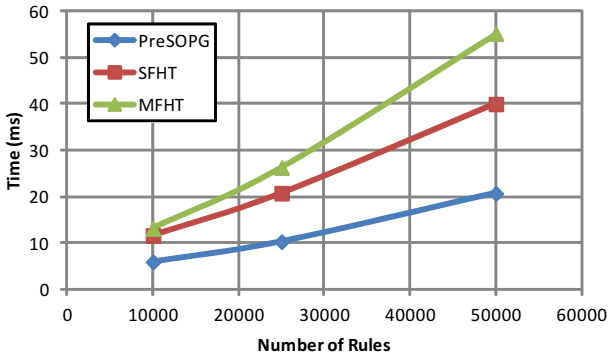


Fig. 4. Comparison of execution times for PreSOPG, SFHT, and MFHT in millisecond

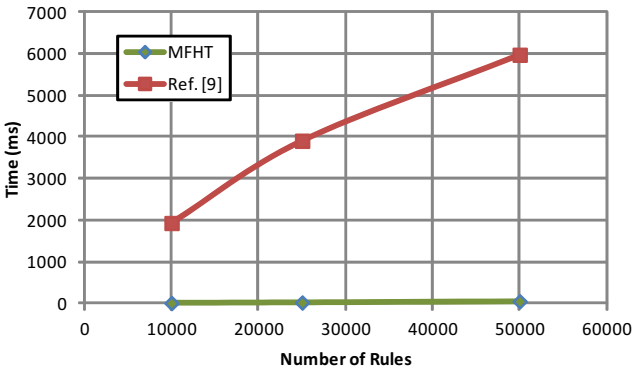


Fig. 5. Comparison of execution times for MFHT and Ref. [9] in millisecond

Last, we compared the execution time of MFHT (as in Fig. 4) and the execution time of dynamic programming algorithm represented in [9]. The method [9] using dynamic programming produced expressions with the same number of words as MFHT. The algorithm [9] 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. 5, we can see that MFHT is much faster (100 times) than the algorithm in [9].

VI. Conclusion

In this paper, we presented a method to reduce the number of factors in a head-tail expression for multi-field classification functions. We derived the PreSOP and 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 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 [9].

ACKNOWLEDGMENTS

This work is partially supported by the Japan Society for the Promotion of Science (JSPS), Grant in Aid for Scientific Research, and by the Adaptable and Seamless Technology Transfer Program through target-driven R&D, JST.

APPENDIX

Lemma A.1: Greater-than (GT) or less-than (LT) functions can be represented by a head-tail expression:

$$\bigvee_{i=0}^{p-1}(\bar{h}_i)(g_i)$$

where $p \leq n$.

Lemma A.2: Let $\vec{a} = (a_{n-1}, a_{n-2}, \cdots, a_1, a_0)$ be the binary representation of integer A. 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 a head-tail expression with two factors:

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

where d is the number of consecutive 0's in $\vec{a}$ and $d > 0$.

Lemma A.3: If $h_0 \subset g_0 \subset h_1 \subset g_1 \subset \cdots \subset h_{p-2} \subset g_{p-2} \subset h_{p-1} \subset g_{p-1}$, 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:

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

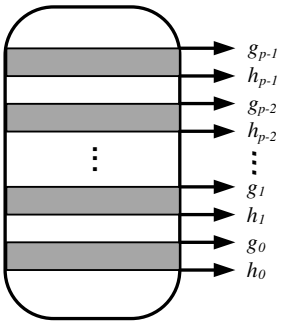


Fig. 6. Map for Lemma A.3

Proof: The grey area in the map of Fig. 6 indicates the covering of Z. Thus, we have the lemma. □

Proof of Theorem 3.1: Lemma A.1 and Lemma A.2 are used to represent p terms of head-tail expressions. Each group of consecutive 0's in $\vec{a}$ can be represented by:

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

$$\cdots$$

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

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 A.2 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:

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

by applying Lemma A.3, $2p$ factors can be reduced to only $p+1$ factors. Thus, we have the theorem. □

The proof of Theorem 3.2 is similar to that of Theorem 3.1.

REFERENCES

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

[2] R. K. Brayton, et al., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer, 1984.

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

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

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

[6] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

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

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

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

[10] I. Syafalni and T. Sasao, "A Fast Head-Tail Expression Generator for TCAM–Application to Packet Classification," *ISVLSI*, August 2012.

[11] 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.

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

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