2012 IEEE Computer Society Annual Symposium on VLSI

A Fast Head-Tail Expression Generator for TCAM—Application to Packet Classification

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

***Abstract*—This paper presents a method to generate head-tail expressions for Ternary Content Addressable Memories (TCAMs). First, we derive head-tail expressions for interval functions. We introduce a fast prefix sum-of-product (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.**

I. Introduction

Packet classification is a fundamental technology in high-speed internet. This technology is used for many devices such as routers, firewalls, network address translators, and access controllers [3], [18], [17]. A Ternary Content Addressable Memory (TCAM) is the hardware to support high speed packet classification [9], [13]. Due to its high speed, TCAMs have become a *de facto* standard for IP lookup devices in the network industry. Unfortunately, TCAMs dissipate high-power and are expensive [1]. These problems tend to be worse with the growth of the internet [19], [8]. To overcome these drawbacks, reduction of TCAM words is necessary. The reduction of TCAM is related to logic minimization, and a logic minimizer such as Espresso is utilized. Since exact minimization is extremely time consuming [7], a heuristic approach using a ternary trie has been developed [2]. Although, it is faster and requires less memory than the exact minimizer, it still requires a large memory size and execution time.

In a packet classification, ports are often specified by intervals. When an interval is represented by a *prefix sum-of-product* expression (PreSOP), it often requires many products. This phenomenon is called **rule expansion**. Various methods to represent intervals are proposed to suppress rule expansion [5], [8], [11]. Any interval function can be represented by a sum-of-products expression (SOP) with at most $2(n-2)$ products [11], [14], where n is the number of bits to represent the largest value in the interval. Moreover, the number of products is reduced by using SOPs with four-valued variables in [12]. In [8] and [10], output encoding is used to reduce the number of products. With this method, any interval function can be represented with at most n TCAM words. Hardware modification that adds comparators to represent intervals directly [15] is proposed, but this method is expensive to implement.

Table I 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 II. 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 [5]. For example, in Table II, both intervals $(0, 16)$ and $(-1, 15)$ require 4 words. This problem (*i.e.*, rule expansion) is the main subject of the paper.

TABLE I
Example of classification function

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

TABLE II
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

In this paper, 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.

II. Preliminaries

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) of an n-variable function $f(x_{n-1}, x_{n-2}, \cdots, x_0)$ is an SOP where the first consecutive literals occur, while all others are missing.

978-0-7695-4767-1/12 $26.00 © 2012 IEEE
DOI 10.1109/ISVLSI.2012.47

Example 2.1: $f(x_2, x_1, x_0) = x_2 x_1 \bar{x}_0 \vee x_2 \bar{x}_1 \vee \bar{x}_2$ is a PreSOP. While, $f(x_2, x_1, x_0) = \bar{x}_2 \vee \bar{x}_1 \vee x_1 \bar{x}_0$ is not a PreSOP, but is an SOP. ■

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 [18]. 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.

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.

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:

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

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

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

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

Lemma 2.1: 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 2.1: 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 $\bar{x}_3 \bar{x}_2 \bar{x}_1 x_0 \vee \bar{x}_3 \bar{x}_2 x_1 \vee \bar{x}_3 x_2 \vee x_3$.

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

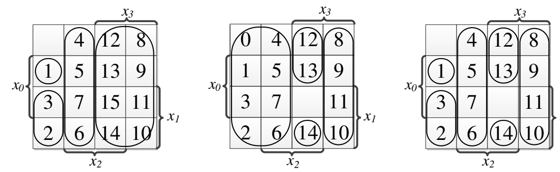
$$LT(n : B) = \bigvee_{i=0}^{n-2} \left(\bigwedge_{j=n-1}^{i+1} x_j^{b_j} \right) \bar{x}_i b_i \vee \bar{x}_{n-1} b_{n-1},$$

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



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

Fig. 1. Maps for Example 2.1

Example 2.2: 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 2.1, the PreSOP for $GT(4 : 0)$ is $\bar{x}_3 \bar{x}_2 \bar{x}_1 x_0 \vee \bar{x}_3 \bar{x}_2 x_1 \vee \bar{x}_3 x_2 \vee x_3$. The number of products is $\sum_{i=0}^{3} \bar{a}_i = 4$. By Lemma 2.2, the PreSOP for $LT(4 : 15)$ is $x_3 x_2 x_1 \bar{x}_0 \vee x_3 x_2 \bar{x}_1 \vee x_3 \bar{x}_2 \vee \bar{x}_3$. The number of products is $\sum_{i=0}^{3} b_i = 4$. And, Theorem 2.1 shows that $IN_0(4 : 0, 15)$ requires $3 + 3 = 6$ products. The PreSOP for $IN_0(4 : 0, 15)$ is $\bar{x}_3 \bar{x}_2 \bar{x}_1 x_0 \vee \bar{x}_3 \bar{x}_2 x_1 \vee \bar{x}_3 x_2 \vee x_3 \bar{x}_2 \vee x_3 x_2 \bar{x}_1 \vee x_3 x_2 x_1 \bar{x}_0$. Fig. 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 $x_0 \bar{x}_1 \vee x_1 \bar{x}_2 \vee x_2 \bar{x}_3 \vee x_3 \bar{x}_0$. ■

III. Realization of Interval Functions on TCAM

A ternary content addressable memory (TCAM) shown in Fig. 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 3.1: Design the TCAM that represents $GT(4 : 0)$. The PreSOP for $GT(4 : 0)$ is

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

Table III shows the corresponding TCAM realization.

TABLE III
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

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

TABLE IV
REALIZATION BASED ON HT.

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

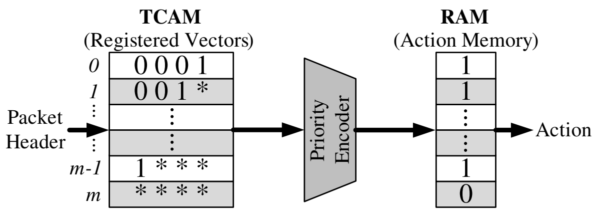


Fig. 2. Realization using TCAM and RAM.

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. 2, the interval function can often be implemented more efficiently. Since $GT(4:0)$ can be represented as

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

f is implemented by the TCAM shown in Table IV. 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.

IV. HEAD-TAIL EXPRESSIONS FOR INTERVAL FUNCTIONS

In this section, we introduce head-tail expressions [6] 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 4.1: A **head-tail expression** (HT) has the form

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

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 4.1: $(\overline{x_1x_2})\cdot(\overline{x_3x_4})\cdot(x_5x_6)\vee(\overline{x_1x_4})\cdot(\overline{x_2x_3})\cdot(\bar{x}_5\bar{x}_6)$ is a head-tail expression. ∎

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

Lemma 4.1: An arbitrary logic function f can be represented by a **head-tail expression** (Eq. (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 4.1: 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:

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

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.

Example 4.2: Let $A=0$. The binary representation of A is $\vec{a} = (0,0,0,0)$. By Theorem 4.1, we have a group of consecutive 0's, where $n=4$, $p=1$, $c_{p-1}=c_0=3$ and $d_0=4$. Thus,

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

Theorem 4.2: 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:

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

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.

Example 4.3: 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 2.1, 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 4.2, 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:

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

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 V
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

Table V(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 V(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. ■

V. FAST PREFIX SOP GENERATOR

In this section, we present a fast PreSOP generator (FP). Various PreSOP generators exist [18], [8], [5]. Fig. 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 2.1, 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 $O(s) \cdot n = O(n^2)$. 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.

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. */
Find the largest index such that $a_s \neq b_s$,
  $s \leftarrow \lfloor \log_2 (A \oplus B) \rfloor$.
For both vectors ($Vector(\vec{a})$ and $Vector(\vec{b})$), perform below.
for $i = 0$; $i < s$; $i++$ **do**
  if $Vector[s-1-i] = B\_flg$ **then**
    $Output[n-1,\ldots,s-i] \leftarrow Vector[n-1,\ldots,s-i]$
    $Output[s-1-i] \leftarrow A\_flg$
  end if
end for
Terminate.

Fig. 3. Pseudocode for FP

VI. 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. 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 2.1 is used to produce the word.
- *Mode 2*: Theorem 4.1 or 4.2 is used to produce the word.

Fig. 4 shows that the time complexity for n-bit DHT is $O(s) \cdot n = O(n^2)$. 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 6.1: 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. 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 2.1: $110111000 \rightarrow 1$, or the factor $(x_8x_7\bar{x}_6x_5x_4x_3\bar{x}_2\bar{x}_1\bar{x}_0)$ (Fig. 5, **Mode 1a**). At $i = 2$, 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. */
Find the largest index such that $a_s \neq b_s$,
$s \leftarrow \lfloor \log_2 (A \oplus B) \rfloor$.
For $i = 0$ to $i = s-1$, iterate the below for $Vector(\vec{a})$ and $Vector(\vec{b})$. Checks $Vector(\vec{a})$ iff $A\_flg = 1$, and checks $Vector(\vec{b})$ iff $B\_flg = 1$.
for $i = 0$; $i < s$; $i++$ **do**
  switch Mode
    case 0:
      if $Vector[i] = B\_flg$ **then**
        Mode $\leftarrow$ 1: Generate no output.
      else
        Mode $\leftarrow$ 0
      end if
    case 1:
      if $Vector[i] = B\_flg$ **then**
        Mode $\leftarrow$ 2: A group of consecutive 0's or 1's is detected. By Theorem 4.1 or 4.2, generate the head factor ($\bar{h}_i$).
      else
        Mode $\leftarrow$ 0: Only a single 0 or 1 is detected. Use Theorem 2.1 to generate a product.
      end if
    case 2:
      if $Vector[i] = B\_flg$ **then**
        Mode $\leftarrow$ 2
      else
        if $Vector[i+1] = B\_flg$ **then**
          Mode $\leftarrow$ 2: Groups of consecutive 0's or 1's are detected. By Theorem 4.1 or 4.2, generate the factor ($\bar{h}_{i-1} \vee g_i$).
        else
          Mode $\leftarrow$ 0: If groups of consecutives 0's or 1's are detected, generate the tail factor (g_i) using Theorem 4.1 or 4.2.
        end if
      end if
  end switch
end for
Terminate.

Fig. 4. Pseudocode for DHT

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 4.2: 110111*** $\rightarrow$ 0, or the factor $(\overline{x_8 x_7 \bar{x}_6 x_5 x_4 x_3})$ (Fig. 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 4.2: 110****** $\rightarrow$ 1, or the factor $(x_8 x_7 \bar{x}_6)$ (Fig. 5, **Mode 2a**, the second output) and terminates. Note that, the corresponding HT is $(x_8 x_7 \bar{x}_6 x_5 x_4 x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0) \vee (\overline{x_8 x_7 \bar{x}_6 x_5 x_4 x_3})(x_8 x_7 \bar{x}_6)$. Table VI shows the produced TCAM pattern. ■

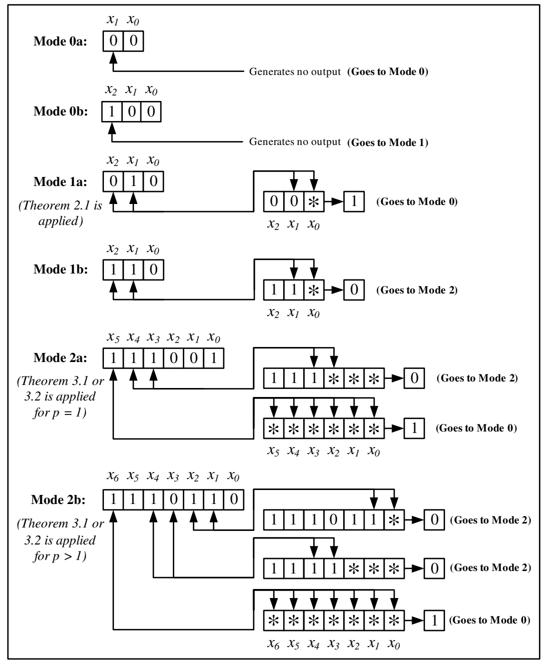


Fig. 5. Steps of DHT in Example 6.1

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

$$GT(9:383) = x_8 x_7$$
$$LT(9:441) = \bar{x}_8 \vee x_8 \bar{x}_7 \vee x_8 x_7 \bar{x}_6 \bar{x}_5 \vee x_8 x_7 \bar{x}_6 x_5 \bar{x}_4 \vee x_8 x_7 \bar{x}_6 x_5 x_4 \bar{x}_3 \vee x_8 x_7 \bar{x}_6 x_5 x_4 x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0$$

Next, obtain the PreSOP for the interval function:

$$GT(9:383) \cdot LT(9:441) = x_8 x_7 \bar{x}_6 \bar{x}_5 \vee x_8 x_7 \bar{x}_6 x_5 \bar{x}_4 \vee x_8 x_7 \bar{x}_6 x_5 x_4 \bar{x}_3 \vee x_8 x_7 \bar{x}_6 x_5 x_4 x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0.$$

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

$$x_8 x_7 \bar{x}_6 (\bar{x}_5 \vee x_5 \bar{x}_4 \vee x_5 x_4 \bar{x}_3) \vee (x_8 x_7 \bar{x}_6 x_5 x_4 x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0)$$
$$= (x_8 x_7 \bar{x}_6)(\overline{x_5 x_4 x_3}) \vee (x_8 x_7 \bar{x}_6 x_5 x_4 x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0)$$
$$= (x_8 x_7 \bar{x}_6)(\overline{x_8 x_7 \bar{x}_6 x_5 x_4 x_3}) \vee (x_8 x_7 \bar{x}_6 x_5 x_4 x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0)$$

TABLE VII
COMPARISON OF PERFORMANCE

Data	Number of Rules	PreSOP		SOP		HT		
		Products	Time(μs) FP	Products	Time(ms) Espresso	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

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

Fig. 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.

VII. EXPERIMENTAL RESULTS

Since no benchmark data for packet classifications is available, ClassBench [17] was used to generate classification functions. First, we generated PreSOPs for various functions. Then, we reduced the number of products in PreSOPs by Espresso [4]. Note that Espresso produces SOPs, which often require fewer products than PreSOPs. Table VII compares PreSOPs and SOPs. However, Espresso 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. 4 is only for a single field function, we applied DHT twice to obtain the HT. As shown in Table VII, 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.

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.

VIII. CONCLUSION

In this paper, 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.

ACKNOWLEDGMENT

This research is supported in part by the Grants in Aid for Scientific Research of JSPS. Prof. J. T. Butler's comments were useful to improve the presentation.

REFERENCES

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

[2] S. Ahmad and R. N. Mahapatra, "An efficient approach to on-chip logic minimization, *IEEE Transactions on VLSI*, vol. 15, no. 9, pp. 1040-1050, Sep. 2007.

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

[4] R. K. Brayton, G. D. Hatchtel, C.T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithm for VLSI Synthesis*, Kluwer, 1984.

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

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

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

[8] 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.

[9] 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.

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

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

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

[13] T. Sasao, *Memory-based Logic Synthesis*, Springer, 2011.

[14] 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.

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

[16] I. Syafalni and T. Sasao, "A TCAM simplification for packet classification," *IWLS 2012*, pp. 49-56, June 2012.

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

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

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