IEICE TRANS. INF. & SYST., VOL. E81–D, NO. 7 JULY 1998

PAPER Special Issue on Test and Diagnosis of VLSI

On Properties of Kleene TDDs

Yukihiro IGUCHI†, Tsutomu SASAO††, *Members, and* Munehiro MATSUURA††, *Nonmember*

SUMMARY Three types of ternary decision diagrams (TDDs) are considered: AND_TDDs, EXOR_TDDs, and Kleene_TDDs. Kleene_TDDs are useful for logic simulation in the presence of unknown inputs. Let $N(BDD : f)$, $N(AND\_TDD : f)$, and $N(EXOR\_TDD : f)$ be the number of non-terminal nodes in the BDD, the AND_TDD, and the EXOR_TDD for f, respectively. Let $N(Kleene\_TDD : \mathcal{F})$ be the number of non-terminal nodes in the Kleene_TDD for $\mathcal{F}$, where $\mathcal{F}$ is the regular ternary function corresponding to f. Then $N(BDD : f) \leq N(TDD : f)$. For parity functions, $N(BDD : f) = N(AND\_TDD : f) = N(EXOR\_TDD : f) = N(Kleene\_TDD : \mathcal{F})$. For unate functions,$N(BDD : f) = N(AND\_TDD : f)$. The sizes of Kleene_TDDs are $O(3^n/n)$, and $O(n^3)$ for arbitrary functions, and symmetric functions, respectively. There exist a $2n$-variable function, where Kleene_TDDs require $O(n)$ nodes with the best order, while $O(3^n)$ nodes in the worst order.

key words: *binary decision diagram, ternary decision diagram, logic simulation, ternary logic*

1. Introduction

Various methods exist for representing logic functions. A truth table is the most straightforword method. Another method is a sum-of-products expression (SOP). Binary Decision Diagrams (BDDs) [1] are commonly used in logic synthesis [2], [3], since they can represent complex functions with many variables. Recently, Ternary Decision Diagrams (TDDs) have been developed as an alternative representation of logic functions [4]. TDDs are similar to BDDs, except that each non-terminal node has three children. Some TDDs have terminals other than constant 0 and 1. In this paper, we consider three types of TDDs: AND_TDDs, EXOR_TDDs, and Kleene_TDDs. AND_TDDs represent the sets of the implicants implicitly [5]. They can treat functions for which the conventional cube-based method [6] fails. EXOR_TDDs are useful to minimize AND-EXOR logic expressions [7]. The TDD presented by Jennings (hereafter, we will call it Kleene_TDD) is useful to evaluate logical functions in the presence of unknown inputs [8].

In this paper, we will show some properties of Kleene_TDDs. In Sect. 2, we will introduce a method to evaluate logic functions in the presence of unknown inputs [9]. In Sect. 3, we will introduce BDDs, AND_TDDs, EXOR_TDDs, and Kleene_TDDs. In Sect. 4, we will compare the sizes of TDDs representing various classes of logic functions: symmetric functions, unate functions, and parity functions. In Sect. 5, we will compare the sizes of TDDs for benchmark functions. Also, we show the size of Kleene_TDDs for randomly generated functions.

Manuscript received November 28, 1997.
Manuscript revised February 17, 1998.

†The author is with the Department of Computer Science, Meiji University, Kawasaki-shi, 214–8571 Japan.
††The authors are with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.

2. Evaluation of Logic Functions in the Presence of Unknown Inputs

Let $B = \{0, 1\}$. An n-variable switching function f represents the mapping:

$$f : B^n \rightarrow B.$$

Let $\vec{a} = (a_1, a_2, \ldots, a_n)$ be a binary vector, where $a_i \in B$. We often have to evaluate the value $f(\vec{a})$ for $\vec{a}$, where some a_i are unknown [10]. When we do logic simulation for sequential circuits, we have to consider such inputs. In this section, we will review the method to evaluate f in the presence of unknown inputs.

Let $T = \{0, 1, u\}$, where u is the truth value showing an unknown input. Let $\vec{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be a ternary vector, where $\alpha_i \in T$. If α_i is either 0 or 1 for all i, then $\vec{\alpha} \in B^n$. In this case, $f(\vec{\alpha})$ is either 0 or 1. If $\alpha_i = u$ for some i, then $\vec{\alpha} \in T^n - B^n$. In this case, for some $\vec{\alpha}$, $f(\vec{\alpha})$ is either 0 or 1, but for other $\vec{\alpha}$, $f(\vec{\alpha})$ cannot be determined in general. Therefore, it is convenient to introduce a three-valued logic function, $\mathcal{F} : T^n \rightarrow T$, which is derived from f. Note that f uniquely defines $\mathcal{F}$.

Definition 1: Let $\vec{\alpha} \in T^n$. $A(\vec{\alpha})$ denotes the set of all the binary vectors that are obtained by replacing all u with 0 or 1.

Let s be the number of u's in $\vec{\alpha}$, then the set $A(\vec{\alpha})$ consists of 2^s binary vectors.

Definition 2: Let f be a two-valued logic function, and $\vec{\alpha} \in T^n$.

$$f(A(\vec{\alpha})) = \{f(\vec{a}) \mid \vec{a} \in A(\vec{\alpha})\}.$$

$$\mathcal{F}(\vec{\alpha}) = \begin{cases} 0 & \text{if } f(A(\vec{\alpha})) = \{0\}. \\ 1 & \text{if } f(A(\vec{\alpha})) = \{1\}. \\ u & \text{if } f(A(\vec{\alpha})) = \{0, 1\}. \end{cases}$$

$\mathcal{F}$ is called the regular ternary logic function [11] of f (hereafter, we will call it RT function).

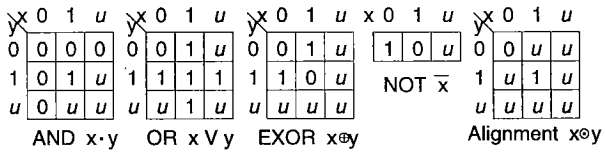

Fig. 1 Ternary AND, OR, NOT, EXOR and Alignment operations.

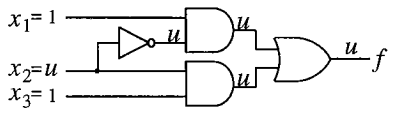

Fig. 2 Realization of expression in Example 1.

Example 1: Consider the expression $f(x_1, x_2, x_3) = x_1\bar{x}_2 \vee x_2x_3$. For the inputs $\vec{\alpha}_1 = (0, 0, u)$, $\vec{\alpha}_2 = (1, u, 1)$, and $\vec{\alpha}_3 = (1, u, u)$, $\mathcal{F}(\vec{\alpha}_1)$, $\mathcal{F}(\vec{\alpha}_2)$, and $\mathcal{F}(\vec{\alpha}_3)$ are derived as follows:

$$f(A(\vec{\alpha}_1)) = \{f(0,0,0), f(0,0,1)\} = \{0\},$$
$$f(A(\vec{\alpha}_2)) = \{f(1,0,1), f(1,1,1)\} = \{1\}, \quad \text{and}$$
$$f(A(\vec{\alpha}_3)) = \{f(1,0,0), f(1,0,1), f(1,1,0), f(1,1,1)\} = \{0,1\}.$$

By Definition 2, we have

$$\mathcal{F}(\vec{\alpha}_1) = 0, \ \mathcal{F}(\vec{\alpha}_2) = 1, \text{ and } \mathcal{F}(\vec{\alpha}_3) = u. \qquad \square$$

When we do gate-level logic simulation, we extend binary operations to ternary logic as shown in Fig. 1. This is the Kleenean strong ternary logic [12]. In this case, signals are evaluated from the primary inputs to the primary outputs by using Fig. 2.

Example 2: Figure 2 shows an AND–OR network that realizes the expression in Example 1. When we evaluate the output f for the input $\vec{\alpha}_2 = (1, u, 1)$ by using a naive method, we have the output u as shown in Fig. 2. However, Example 1 shows that

$$\mathcal{F}(\vec{\alpha}_2) = \mathcal{F}(1, u, 1) = 1.$$

Thus, the naive method does not always produce accurate results. $\square$

Several methods exist to evaluate output values according to Definition 2: including representations using SOPs [9], using BDDs [13], and using Kleene_TDDs [8].

3. BDDs and TDDs

In this section, we will give formal definitions for BDDs and TDDs.

Definition 3: A BDD is a rooted, directed graph with node set V containing two types of nodes:
A non-terminal node v has as attributes an argument index $index(v) \in \{1, \ldots, n\}$, and two children $low(v), high(v) \in V$. *A terminal node* v has as attribute a value$value(v) \in B$.

For any non-terminal node v, if $low(v)$ is also non-terminal, then $index(v) < index(low(v))$. Similarly, if $high(v)$ is also non-terminal, then $index(v) < index(high(v))$. The correspondence between BDDs and Boolean functions is defined as follows:
For a terminal node v:

If $value(v) = 1$, then $f_v = 1$.
If $value(v) = 0$, then $f_v = 0$.

For a non-terminal node v:
Let $\vec{X} = (x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.
If $index(v) = i$, then f_v is a function such that

$$f_v(x_1, \ldots, x_n) = \bar{x}_i \cdot f_{low(v)}(\vec{X}) \vee x_i \cdot f_{high(v)}(\vec{X}).$$

Note that the root node represents the function f itself.

Definition 4: A BDD is a *Reduced Ordered Binary Decision Diagram* (ROBDD) if it contains no node v with $low(v) = high(v)$, and if it does not contain distinct nodes v_1 and v_2 such that the subgraph rooted by v_1 and v_2 are equivalent.

Example 3: Figure 3 (a) shows the ROBDD for the function in Example 1. The number 0(1) attached to each edge incident to v denotes $low(v)(high(v))$. $\square$

TDDs are similar to BDDs, except that each non-terminal node v has three children, $low(v)$, $high(v)$, and $middle(v)$.

Definition 5: A TDD is a rooted, directed graph with node set V containing two types of nodes:
A non-terminal node v has as attributes an argument index $index(v) \in \{1, \ldots, n\}$, and three children $low(v)$, $high(v)$, and $middle(v) \in V$. *A terminal node* v has as attribute a value$value(v) \in T$. For any non-terminal node v, if $low(v)$ is also non-terminal, then $index(v) < index(low(v))$. Similarly, if $high(v)$ is non-terminal, then $index(v) < index(high(v))$. Also, if $middle(v)$ is non-terminal, then $index(v) < index(middle(v))$.

Definition 6: A TDD is a Reduced Ordered Ternary Decision Diagram (ROTDD) if it contains no node v with $low(v) = high(v)$, and if it does not contain distinct nodes v_1 and v_2 such that the subgraph rooted by v_1 and v_2 are equivalent.

Different assignments of operations to the third child produce different TDDs.

Definition 7: Figure 1 shows the ternary operation *Alignment*. Let $a, b \in T$.

$$a \odot b = \begin{cases} a & \text{if } a = b. \\ u & \text{otherwise.} \end{cases}$$

The correspondence between AND_TDD (EXOR_TDD, Kleene_TDD) and a function f is defined as follows:
For terminal node v:

If $value(v) = 1$, then $f_v = 1$.
If $value(v) = 0$, then $f_v = 0$.

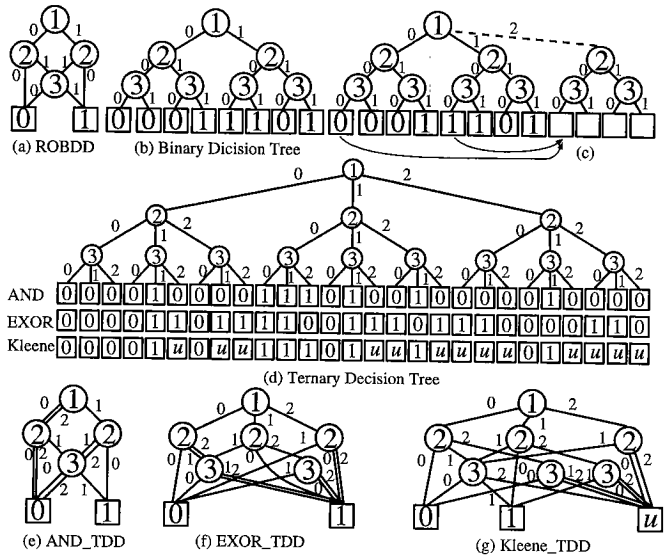

Fig. 3 BDD and TDDs.

For non-terminal node v: If $index(v) = i$, then f_v and $f_{middle(v)}$ are the functions such that $f_v(\vec{X}) = \bar{x}_i \cdot f_{low(v)}(\vec{X}) \vee x_i \cdot f_{high(v)}(\vec{X})$.

- In the case of an AND_TDD:

$$f_{middle(v)}(\vec{X}) = f_{low(v)}(\vec{X}) \cdot f_{high(v)}(\vec{X}).$$

- In the case of an EXOR_TDD:

$$f_{middle(v)}(\vec{X}) = f_{low(v)}(\vec{X}) \oplus f_{high}(\vec{X}).$$

- In the case of a Kleene_TDD:

$$f_{middle(v)}(\vec{X}) = f_{low(v)}(\vec{X}) \odot f_{high(v)}(\vec{X}).$$

If the above relation does not hold, then there is no function for the TDD.

Example 4: Figure 3 shows the AND_TDD, the EXOR_TDD, and the Kleene_TDD for f and $\mathcal{F}$ in Example 1. By expanding the ROBDD in (a), we have the complete binary decision tree (b). In Fig. 3 (c), the new edge at the root node is generated, and the subgraph is created by using the AND operation to derive the AND_TDD. Note that $f_0 = f_{low}$, $f_1 = f_{high}$, and $f_2 = f_{middle}$. In Fig. 3 (c), f_{000} corresponds to the left most terminal node. f_{200} is obtained as the AND of f_{000} and f_{100}. When EXOR is used to generate the middle child, we have an EXOR_TDD. When Alignment is used to generate the middle child, we have a Kleene_TDD. Here, 0, 1, and 2 attached to each edge denote $low(v)$, $high(v)$, and $middle(v)$, respectively. For each subsequent node, create the third node recursively down to the leaves, and we have the complete ternary decision tree shown in (d). By eliminating all the redundant nodes, and sharing all the equivalent subgraphs, we obtain an AND_TDD, an EXOR_TDD, and a Kleene_TDD as shown in (e), (f), and (g), respectively. □

AND_TDDs represent the set of implicants implicitly [5]. EXOR_TDDs are useful to minimize AND-EXOR expressions [7]. Kleene_TDDs are useful to evaluate logic functions in the presence of unknown inputs [8].

Example 5: We can evaluate $\mathcal{F}(\boldsymbol{\alpha})$ in Example 1 by using the Kleene_TDD in Fig. 3 (g). $\mathcal{F}(\boldsymbol{\alpha})$ is obatained by tracing the edges from the root node to a terminal node according to the value of $\boldsymbol{\alpha}$. When the input is $\boldsymbol{\alpha}_1 = (0, 0, u)$, trace the edges, 0, 0, and 2, and reach the terminal node 0. Thus, we have $\mathcal{F}(0, 0, u) = 0$. Similarly, when the input is $\boldsymbol{\alpha}_2 = (1, u, 1)$, trace the edge, 1, 2, 1, and reach the terminal node 1. Thus, $\mathcal{F}(1, u, 1) = 1$. When the input is $\boldsymbol{\alpha}_3 = (1, u, u)$, by tracing 1,2,2, and reach the terminal node u. Thus, we have $\mathcal{F}(1, u, u) = u$. □

4. Complexities of TDDs

In this section, we compare the sizes of TDDs representing various classes of functions: symmetric functions, unate functions, and parity functions.

Definition 8: Let $N(BDD : f)$, $N(AND\_TDD : f)$, and $N(EXOR\_TDD : f)$ be the number of non-terminal nodes in the BDD, the AND_TDD, and the EXOR_TDD for f,respectively. Let $N(Kleene\_TDD : \mathcal{F})$ be the numbers of non-terminal nodes in the Kleene_TDD for $\mathcal{F}$.

Theorem 1:

$$N(BDD : f) \leqq N(AND\_TDD : f),$$
$$N(BDD : f) \leqq N(EXOR\_TDD : f), \text{ and}$$
$$N(BDD : f) \leqq N(Kleene\_TDD : \mathcal{F}).$$

Proof: Remove all the middle edges. Remove the terminal node u. Eliminate all the redundant nodes and share all the equivalent sub-graphs. And, we obtain the BDD for f. This procedure does not increase the number of nodes. Hence, the theorem. □

Theorem 2:

$$N(AND\_TDD : f) \leqq N(Kleene\_TDD : \mathcal{F}).$$

Proof: In the Kleene_TDD for $\mathcal{F}$, replace the terminal node u with the terminal node 0. Next, eliminate all the redundant nodes, and remove all but one equivalent sub-graph. Then, we get the AND_TDD for f. This procedure does not increase the number of nodes. Hence the theorem. □

Theorem 3: Let f be a parity function. Then, we have

$$\begin{aligned} N(BDD : f) &= N(AND\_TDD : f) \\ &= N(EXOR\_TDD : f) \\ &= N(Kleene\_TDD : \mathcal{F}). \end{aligned}$$

Proof: Figure 4 (a) shows the BDD for the parity function $f = a_0 \oplus x_1 \oplus x_2 \oplus \ldots \oplus x_n$, where $a_0 = 0$. Note that $N(BDD : f) = 2n - 1$.

- AND_TDD:
  The parity function returns the value 0(1), when the number of 1's in the inputs is an even (odd) number. Consider two input vectors $\vec{a_1}$ and $\vec{a_2}$, where $\vec{a_1} = (a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_n)$ and $\vec{a_2} = (a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_n)$. Consider the node v for which the paths for $\vec{a_1}$ and $\vec{a_2}$ visit. $low(v)$ corresponds to $\vec{a_1}$ and $high(v)$ corresponds to $\vec{a_2}$. If the number of 1's in $\vec{a_1}$ is an even (odd) number, then the number of 1's in $\vec{a_2}$ is an odd (even) number. Hence, $f(\vec{a_1}) = \overline{f(\vec{a_2})}$. Thus, $f_{low(v)} \cdot f_{high(v)} = 0$ for every non-terminal node v. In the AND_TDD, all the *middle* edges point to the terminal 0 as shown in Fig. 4 (b). Thus, we have $N(AND\_TDD : f) = 2n - 1$.

- EXOR_TDD:
  As is the case of AND_TDD, $f(\vec{a_1}) = \overline{f(\vec{a_2})}$ holds. Hence, $f_{low(v)} \oplus f_{high(v)} = 1$ always holds for every non-terminal node. On the EXOR_TDD, all the *middle* edges are incident to the terminal 1 node as shown in Fig. 4 (c). Thus, we have $N(EXOR\_TDD : f) = 2n - 1$.

- Kleene_TDD:
  As is the case of AND_TDD, $f(\vec{a_1}) = \overline{f(\vec{a_2})}$ holds. Hence, $f_{low(v)} \odot f_{high(v)} = u$ always holds for every non-terminal node. On the Kleene_TDD, all the *middle* edges point to the terminal u as shown in Fig. 4 (d). Thus, we have $N(Kleene\_TDD : \mathcal{F}) = 2n - 1$. □

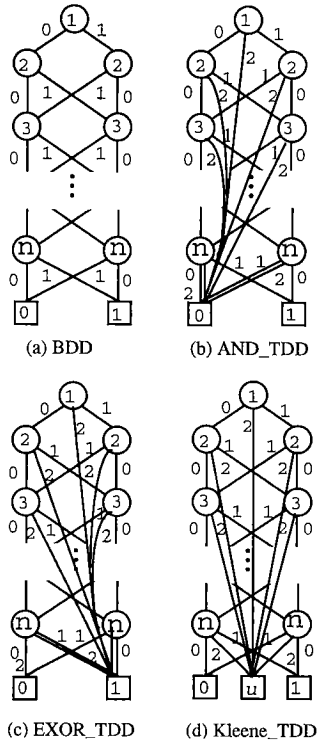

Fig. 4 BDD and TDD for parity function.

Theorem 4: Let f be a unate function. Then, $N(BDD : f) = N(AND\_TDD : f)$.

Proof: For simplicity, we assume that f is a monotone increasing function. For a non-terminal node v with $index(v) = k$, Shannon's expansion of f with respect to the variable x_k is as follows:

$$\begin{aligned} f(x_1, \ldots, x_n) = \bar{x}_k f(x_1, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_n) \\ \vee x_k f(x_1, \ldots, x_{k-1}, 1, x_{k+1}, \ldots, x_n). \end{aligned}$$

Since f is a monotone increasing function, the following inequality holds:

$$\begin{aligned} &f(x_1, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_n) \\ &\quad \leqq f(x_1, \ldots, x_{k-1}, 1, x_{k+1}, \ldots, x_n). \end{aligned}$$

Hence, we have

$$\begin{aligned} &f(x_1, \ldots, 0, \ldots, x_n) \cdot f(x_1, \ldots, 1, \ldots, x_n) \\ &\quad = f(x_1, \ldots, 0, \ldots, x_n). \end{aligned}$$

Therefore, $middle(v)$ is the same node as $low(v)$ for every node v in the AND_TDD. By adding middle edges to the low edges in the BDD, we have the AND_TDD. Hence we have the theorem. Similarly,we have the theorem for unate functions. □

The number of nodes in a complete ternary decision tree for an n-variable function is:

$$1 + 3^1 + 3^2 + \ldots + 3^n = (3^{n+1} - 1)/2.$$

However, in a reduced TDD, only one sub-graph is realized for each sub-function. Thus, the number of nodes can be reduced.

We can state the following:

Lemma 1: All the functions of k or fewer variables can be represented by a Kleene_TDD with at most 3^{3^k} nodes.

Theorem 5: An arbitrary n-variable function can be represented by a Kleene_TDD with at most

$$\min_{k=1}^{n}\left(\frac{3^{k+1}-1}{2}+3^{3^{n-k}}\right)$$

nodes.

Proof: Consider the Kleene_TDD in Fig. 5, where the upper block is the complete ternary decision tree of k variables, and the lower block generates all the functions of $(n-k)$ or fewer variables. The complete ternary decision tree for a k-variable function has

$$1+3^1+3^2+\ldots+3^k=(3^{k+1}-1)/2 \quad nodes.$$

By Lemma 1, the lower block has at most $3^{3^{n-k}}$ nodes. Hence, we have the theorem. □

Corollary 1: An arbitrary n-variable function can be represented by a Kleene_TDD with $O(3^n/n)$ nodes.

Proof: Set $k=n-\log_3 n+\log_3\log_3 n$ in Theorem 5. □

Theorem 6: An arbitrary symmetric function of n-variables can be represented by a Kleene_TDD with $O(n^3)$ nodes.

Proof: Consider the complete ternary decision tree for k variables (Fig. 6). The number of different functions generated in the lower part of the tree is derived as follows:

1. Since f is symmetric, the permutation of the subscripts of f will not change the function: e.g., f_{021120} is equal to f_{001122}. So, the different number of k-variable symmetric functions generated by the complete ternary decision tree is equal to "the number of ways to select k objects from three distinct objects with repetition."

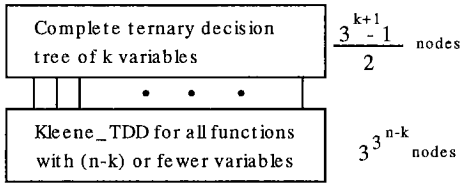

Fig. 5 Representation of an n-variable function by Kleene_TDD.

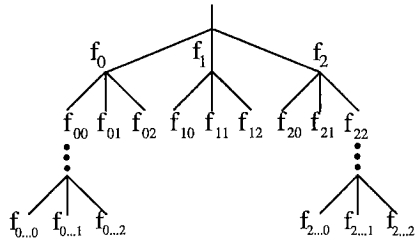

Fig. 6 Complete ternary decision tree for k-variable function.

2. "The number of ways to select k objects from p distinct objects with repetition" is ${}_{p+k-1}C_k$.

3. Since $p=3$, the number of the different symmetric functions is $(k+1)(k+2)/2$.

The total number of non-terminal nodes in the (non-reduced) Kleene_TDD is

$$\sum_{i=0}^{n}\frac{(i+1)(i+2)}{2}=\frac{1}{6}n(n+1)(n+5)+(n+1).$$

Hence, we have the theorem. □

The following is easy to prove:

Theorem 7:

$$N(BDD:f)=N(BDD:\bar{f}).$$
$$N(EXOR\_TDD:f)=N(EXOR\_TDD:\bar{f}).$$
$$N(Kleene\_TDD:f)=N(Kleene\_TDD:\bar{f}).$$

However, in general,

$$N(AND\_TDD:f)\neq N(AND\_TDD:\bar{f}).$$

5. Experiments and Observation

5.1 Sizes of TDDs

We developed TDD algorithms, and generated TDDs for various benchmark functions. Multiple-output functions are represented by shared BDDs and shared TDDs. Note that a node in a TDD requires one more word than that of a BDD in our implementation. The ordering of the inputs variables for BDDs and TDDs are obatined by heuristic algorithms for BDDs [14], [15]. Note that the ordering that minimizes the size of a BDD does not always mininzes the size of the TDD. We used Sun Ultra 170 workstaion with 512 mega bytes of memory.

Table 1 compares the number of non-terminal nodes in BDDs, AND_TDDs, EXOR_TDDs, and Kleene_TDDs. From this table, we observe the following:

1. $N(BDD:f)\leqq N(TDD:f)$ always holds.

2. Except for e64, rd84, t481, xor5, and z5xp1, $N(AND\_TDD:f)<N(EXOR\_TDD:f)$ holds.

3. Except for bc0, chkn, misex3, ts10, x6dn, and xor5, $N(TDD:f)<N(Kleene\_TDD:\mathcal{F})$ holds.

4. The number of nodes in Kleene_TDDs and EXOR_TDD are comparable.

5. TDDs are considerably larger than corresponding BDDs.

Table 1 Sizes of BDDs and TDDs.

function	in	out	BDD	TDD		
				AND	EXOR	Kleene
add6	12	7	47	47	47	106
adr4	8	5	29	29	29	60
alu2	10	6	61	82	142	173
apex1	45	45	1332	6249	47814	18401
apex2	39	3	410	542	3575	3500
apex3	54	50	935	3161	34574	7119
apex5	117	88	1078	3039	3282	4204
apla	10	12	102	139	302	354
bc0	26	11	578	3137	12650	8034
bw	5	28	108	117	144	225
chkn	29	7	273	649	2556	2281
clip	9	5	97	174	259	289
co14	14	1	27	27	39	51
con1	7	2	15	25	35	43
cordic	23	2	75	83	153	271
cps	24	109	987	1457	4808	5653
dc2	8	7	64	72	124	167
dist	8	5	152	323	563	572
dk17	10	11	62	78	147	189
dk27	8	9	25	27	31	49
duke2	22	29	336	522	2176	2555
e64	65	65	1379	1379	1379	2693
ex5	8	63	278	381	444	628
ex5	8	63	278	381	444	628
f51m	8	8	67	87	72	164
in2	19	10	232	377	892	1129
in7	26	10	96	131	252	428
inc	7	9	70	104	172	214
intb	15	7	600	2338	3774	6876
misex1	8	7	36	45	70	92
misex2	25	18	81	81	138	204
misex3	14	14	542	1219	3644	3262
misj	35	14	42	42	72	83
mlp4	8	8	141	322	383	595
pdc	16	40	560	1024	2321	3031
rd53	5	3	23	26	25	39
rd73	7	3	43	52	52	84
rd84	8	4	59	79	72	121
risc	8	31	67	67	76	124
rot8	8	5	75	119	211	227
sao2	10	4	85	114	216	305
seq	41	35	1248	3873	67414	18745
sex	9	14	46	47	62	93
spla	16	46	581	717	2237	2315
sqr8	8	16	233	503	654	895
t481	16	1	32	48	43	66
tial	14	8	686	1732	5241	6558
ts10	22	16	146	146	951	786
vg2	25	8	194	399	865	961
x6dn	39	5	229	637	7597	3177
xor5	5	1	9	9	9	9
z5xp1	7	10	68	79	75	158
9sym	9	1	33	60	70	94

6. xor5 is a parity function. Thus,
$N(BDD : f) = N(AND\_TDD : f)$
$= N(EXOR\_TDD : f) = N(Kleene\_TDD : \mathcal{F})$.

Table 2 shows the sizes of Kleene_TDDs for randomly generated functions. The orderings of the variables in TDDs in Table 2 minimize the size of BDDs. We can observe that the sizes of Kleene_TDDs are $O(3^n/n)$ for n-variable randomly generated functions.

Table 2 also shows the sizes of Kleene_TDDs for the Achilles' heel function: $f = x_1y_1 \vee x_2y_2 \vee \ldots \vee x_ny_n$. The size is $O(3^n)$ when the variable order is $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n$, while $5n-2$ when the variable order is $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$. This shows that the sizes of TDDs greatly depends on the variable ordering.

Table 2 Sizes of Kleene_TDDs for n-variable randamly generated functions and Achilles' heel functions.

n	Random function	$f = x_1y_1 \vee \ldots \vee x_ny_n$	
		worst	optimum
3	–	42	13
4	–	127	18
5	28	378	23
6	68	1123	28
7	139	3342	33
8	295	9967	38
9	592	29778	43
10	1357	89083	48
11	2898	266742	53
12	6165	799207	58
13	12905	2395578	63

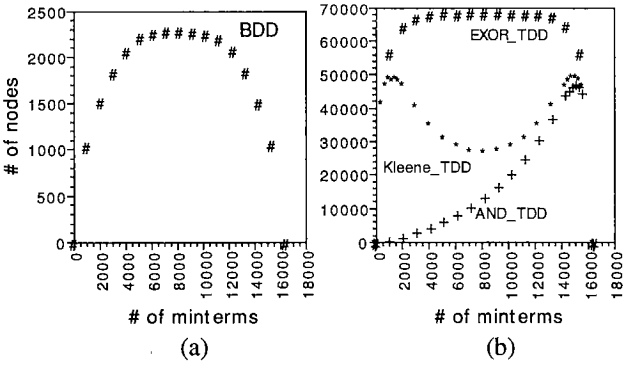

Fig. 7 Dependency on the number of true minterms.

5.2 Dependency on the Number of True Minterms

We generated pseudo-random logic functions of $n = 14$ variables for different number of true minterms $s (0 < s < 2^n)$, and counted numbers of nodes in BDDs and TDDs. Figure 7 (a) shows the numbers of BDD nodes for different s. As shown in Theorem 7, the graph is symmetric with respect to $s = 2^{n-1}$. Also, the number of nodes takes its maximum when $s = 2^{n-1}$. Figure 7 (b) shows the number of TDD nodes. As shown in Theorem 7 the graphs for EXOR_TDDs and Kleene_TDDs are symmetric with respect to $s = 2^{n-1}$. However, the graph for AND_TDDs is asymmetric. Surprisingly, the number of nodes in Kleene_TDDs takes its maximum for two values of s, and takes its local minimum for $s = 2^{n-1}$. This is quite different from the case of BDDs.

5.3 CPU Time

Although Kleene_TDDs are greater than BDDs, Kleene_TDD-based logic simulation are more efficient than BDD-based one.

To evaluate the performance of Kleene_TDD based simulation, we did the following experiments: In logic simulation for design verification, expected outputs for networks are 1 or 0, rather than u. For a given logic function, input vectors can be generated as follows:

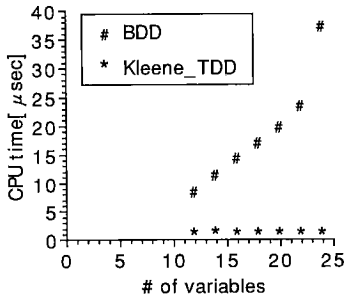

Fig. 8 Comparison of CPU time for adr6~12.

- To verify the ON set.
  Let F be an irredundant sum-of-products expression (ISOP) for f. For each product p_j of F, we have a corresponding input vector $\vec{\alpha}$ such that $f(\vec{\alpha}) = 1$ as follows:

  $\vec{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, where

  $$\alpha_i = \begin{cases} 0 : p_j \text{ contains the literal } \bar{x}_i, \\ 1 : p_j \text{ contains the literal } x_i, \\ u : \text{otherwise.} \end{cases}$$

- To verify the OFF set.
  Let $\bar{F}$ be an ISOP for $\bar{f}$.
  For each product q_j of $\bar{F}$, we have a corresponding input vector $\vec{\beta}$ such that $f(\vec{\beta}) = 0$ as follows:

  $\vec{\beta} = (\beta_1, \beta_2, \ldots, \beta_n)$, where

  $$\beta_i = \begin{cases} 0 : q_j \text{ contains the literal } \bar{x}_i, \\ 1 : q_j \text{ contains the literal } x_i, \\ u : \text{otherwise.} \end{cases}$$

Note that the total number of input vectors is equal to the sum of products in F and $\bar{F}$. In this experiment, we used these as example data for Kleene_TDD based simulation. In the practical logic simulation, we may not use all the product terms of ISOPs for f and $\bar{f}$.

Figure 8 compares simulation time for adders (adr6~adr12) for the vecters generated in the above method. This figure shows that Kleene_TDD-based simulation is faster than BDD based one. TDDs are generated from BDDs. We need extra time to generate TDDs. Even if we condiser this time, Kleene_TDD based simulation is often faster. In the case of adders, the time for generation of TDDs is negligibly small. In the case of apex1, the generating of the TDD from the BDD required 300 millisecond. In this case, if the number of the patterns in the simulation is greater than 40,000, then the total cpu time is shorter.

5.4 Observation

The facts in 5.2 can be interpreted as follows:

1. In AND_TDDs, 1-paths correspond to the implicants of f. The average number of implicants of n-variable functions with s true minterms is given by [16].

   $$G(n,s) = \sum_{k=0}^{n} 2^{n-k} \binom{n}{k} \frac{\binom{w-w_k}{s-w_k}}{\binom{w}{s}},$$

   $$where \;\; w = 2^n, \;\; and \;\; w_k = 2^k.$$

   $G(n,s)$ takes its maximum when s is near to 2^n. Suppose that the number of nodes in the TDD is monotone increasing with the number of paths. Then the graph for AND_TDD has similar shape as the graph for $G(n,s)$.

2. In a Kleene_TDD, 1-paths correspond to the implicants of f, and 0-paths correspond to the implicants of $\bar{f}$. Thus, a Kleene_TDD denotes implicants of f and $\bar{f}$ at the same time. The average number of implicants for f and $\bar{f}$ is given by

   $$G(n,s) + G(n, 2^n - s).$$

   Thus, the shape of the graph is symmetric with respect to $s = 2^{n-1}$, and the graph takes its maximum for two values of s.

6. Conclusion and Comments

In this paper, we introduced three types of TDDs: AND_TDDs, EXOR_TDDs, and Kleene_TDDs. We compared complexities of TDDs for various classes of functions: parity functions, unate functions, and symmetric functions. For parity functions, $N(BDD : f) = N(AND\_TDD : f) = N(EXOR\_TDD : f) = N(Kleene\_TDD : \mathcal{F})$. For unate functions, $N(BDD : f) = N(AND\_TDD : f)$. The sizes of Kleene_TDDs are $O(3^n/n)$ and $O(n^3)$ for arbitrary functions, and symmetric functions, respectively. We compared the sizes of TDDs for benchmark functions. We also show that there exist a $2n$–variable function where Kleene_TDDs require $O(n)$ nodes with the best order, while $O(3^n)$ nodes in the worst order.

Acknowledgements

This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science, Culture and Sports of Japan, and On-Leave Faulty Research Grant of Meiji University.

References

[1] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, Aug. 1986.
[2] T. Sasao, ed., "Logic Synthesis and Optimization," Kluwer Academic Publishers, 1993.
[3] S. Minato, "Graph-based representation of discrete functions," in [4].
[4] T. Sasao and M. Fujita, eds., "Representations of Discrete

Functions," Kluwer Academic Publishers, 1996.
[5] T. Sasao, "Ternary decision diagrams and thier applications," in [4].
[6] D.L. Dietmeyer, Logic Design of Digital Systems, Allyn and Bacon, Inc., Boston, 1971.
[7] T. Sasao, "Optimization of pseudo-Kronecker expressions using multiple-place decision diagrams," IEICE Trans. vol.E76-D, no.5, 1993.
[8] G. Jennings, "Symbolic incompletely specified functions for correct evaluation in the presence of indeterminate input values," 28th Hawaii Int'l Conf. on System Science (vol.I: Architecture), pp.23–31, Jan. 1995.
[9] M. Mukaidono, "Evaluation of logic functions in the presence of unknown input," SIG Notes, DA18-1, Information Processing Society of Japan, 1983.
[10] M. Abramovivi, M.A. Breuer, and A.D. Friedman, "Digital Systems TESTING and Testable DESIGN," pp.43–46, Computer Science Press 1990.
[11] M. Mukaidono, "Regular ternary logic functions—Ternary logic functions suitable for treating ambiguity," IEEE Trans. Comput., vol.C-35, no.2, pp.179–183, Feb. 1986.
[12] S.C. Kleene, "Introduction to Metamathematics," Wolters-Noordhoff, North-Holland Publishing, 1952.
[13] R.E. Bryant, "Boolean analysis of MOS circuits," IEEE Trans. CAD of Integrated Circuits, vol.CAD-6(4), pp.634–649, July 1987.
[14] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchanges of variables," ICCAD-91, pp.472–475, 1991.
[15] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," 1993 International Workshop on Logic Synthesis, pp.3a1–3a12, 1993.
[16] F. Mileto and G. Putzolu, "Average values of quantities appearing in Boolean function minimization," IEEE Trans. Elec. Comput., vol.EC-13, no.4, pp.87–92, April 1964.
[17] Y. Iguchi, S. Sasao, and M. Matsuura "On properties of Kleene TDDs," Asia and South Pacific Design Automation Conference, Proc. ASP-DAC'97, pp.473–476, Jan. 1997.

Yukihiro Iguchi was born in Tokyo, and received the B.E., M.E., and Ph.D. degree in electronic engineering from Meiji University, Kanagawa Japan, in 1982, 1984, and 1987, respectively. He is now a lecturer of Meiji University. His research interest includes logic design, switching theory, and reconfigurable systems. In 1997, he spent a year at Kyushu Institute of Technology.

Tsutomu Sasao received the B.E., M.E., and Ph.D. degrees in Electronic Engineering from Osaka University, Osaka Japan, in 1972, 1974, and 1977, respectively. He was with Osaka University, IBM T.J. Watson Research Center and Naval Postgraduate School in Monterey, California. Now, he is a Professor of Kyushu Institute of Technology, Iizuka, Japan. He has published six books on switching theory and logical design including "Logic Synthesis and Optimization," and "Representation of Discrete Functions," Kluwer 1993 and 1996. He has served Program Chairman for the IEEE International Symposium on Multiple-Valued Logic many times. He received the NIWA Memorial Award, and a Distinctive Contribution Awards from IEEE Computer Society MVL-TC. He is a Fellow of IEEE, and also is an Editor of IEEE Transactions on Computers.

Munehiro Matsuura was born on January 1, 1965 in Kitakyushu City, Japan. He studied at the Kyushu Institute of Technology from 1983 to 1989. He has been working as a Technical Assistant at the Kyushu Institute of Technology since 1991. He have implemented several logic design algorithms under the direction of Professor Tsutomu Sasao. His interests include decision diagrams and exclusive-OR based circuit design.