On Properties of Kleene TDDs

Yukihiro IGUCHI
Dept. of Computer Science
Meiji University
Kawasaki 214-71, Japan
e-mail: iguchi@cs.meiji.ac.jp

Tsutomu SASAO Munehiro MATSUURA
Dept. of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820, Japan
sasao@cse.kyutech.ac.jp matsuura@cse.kyutech.ac.jp

Abstract— 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 Kleenean 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.

I. 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)[2] are commonly used in logic synthesis[8, 9], since they can represent complex functions with many variables. Recently, Ternary Decision Diagrams (TDDs) have been developed as an alternative representation of logic functions[10]. 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[11]. They can treat functions for which the conventional cube-based method[4] fails. EXOR_TDDs are useful to minimize AND-EXOR logic expressions[12]. The TDD presented by Jennings (hereafter, we will call it Kleene_TDD) is useful to evaluate logical functions in the presence of unknown inputs[5].

In this paper, we will show some properties of Kleene_TDDs.

II. 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 $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ be a binary vector, where $a_i \in B$. We often have to evaluate the value $f(\boldsymbol{a})$ for $\boldsymbol{a}$, where some a_i are unknown[1]. 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 $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be a ternary vector, where $\alpha_i \in T$. If α_i is either 0 or 1 for all i, then $\boldsymbol{\alpha} \in B^n$. In this case $f(\boldsymbol{\alpha})$ is either 0 or 1. If $\alpha_i = u$ for some i, then $\boldsymbol{\alpha} \in T^n - B^n$. In this case, for some $\boldsymbol{\alpha}$, $f(\boldsymbol{\alpha})$ is either 0 or 1, but for other $\boldsymbol{\alpha}$, $f(\boldsymbol{\alpha})$ cannot be determined. 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 $\mathcal{F}$.

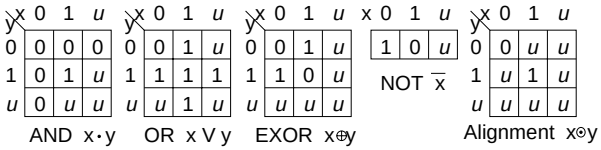

Fig. 2.1. Ternary AND, OR, NOT, EXOR, and Alignment operations

Definition 2.1 *Let $\boldsymbol{\alpha} \in T^n$. $A(\boldsymbol{\alpha})$ denotes the set of all the binary vectors that are obtained by replacing all u with 0 or 1.*

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

Definition 2.2 *Let f be a two-valued logic function, and $\boldsymbol{\alpha} \in T^n$.*

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

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

Example 2.1 *Consider the expression $f(x_1, x_2, x_3) = x_1\bar{x}_2 \vee x_2x_3$. For the inputs $\boldsymbol{\alpha}_1 = (0, 0, u)$, $\boldsymbol{\alpha}_2 = (1, u, 1)$, and $\boldsymbol{\alpha}_3 = (1, u, u)$, $\mathcal{F}(\boldsymbol{\alpha}_1)$, $\mathcal{F}(\boldsymbol{\alpha}_2)$, and $\mathcal{F}(\boldsymbol{\alpha}_3)$ are derived as follows:*

$f(A(\boldsymbol{\alpha}_1))=\{f(0,0,0), f(0,0,1)\} = \{0\}$,
$f(A(\boldsymbol{\alpha}_2))=\{f(1,0,1), f(1,1,1)\} = \{1\}$, *and*
$f(A(\boldsymbol{\alpha}_3))=\{f(1,0,0), f(1,0,1), f(1,1,0), f(1,1,1)\} = \{0,1\}$.

By Definition 2.2, we have

$\mathcal{F}(\boldsymbol{\alpha}_1) = 0$, $\mathcal{F}(\boldsymbol{\alpha}_2) = 1$, *and* $\mathcal{F}(\boldsymbol{\alpha}_3) = u$. ∎

When we do gate-level logic simulation, we extend binary operations to ternary logic as shown in Fig.2.1. This is the Kleenean strong ternary logic[6]. In this case, signals are evaluated from the primary inputs to the primary outputs by using Fig.2.1.

Example 2.2 *Fig. 2.2 shows an AND–OR network that realizes the expression in Example 2.1. When we evaluate the output f for the input $\boldsymbol{\alpha}_2 = (1, u, 1)$ by using a naive method, we have the output u as shown in Fig.2.2. However, Example 2.1 shows that*

$$\mathcal{F}(\boldsymbol{\alpha}_2) = 1.$$

Thus, the naive method does not always produce accurate results. ∎

Several methods exist to evaluate output values according to Definition 2.2: including representations using SOPs, using BDDs[3], and using Kleene_TDDs[5].

ASP-DAC '97
0-89791-851-7$5.00

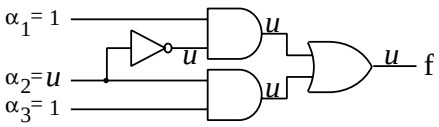

Fig. 2.2. A naive method to evaluate the function

III. BDDs and TDDs

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

Definition 3.1 *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:

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

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

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

Note that the root node represents the function f itself.

Definition 3.2 *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 3.1 *Fig. 3.1(a) shows the ROBDD for the function in Example 2.1. The number 0(1) attached to each edge incident to v denotes low(v)(high(v)).* ∎

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

Definition 3.3 *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 3.4 *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 3.5 *Fig. 2.1 shows the ternary operation Alignment. Let* $a, b \in T$.

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

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

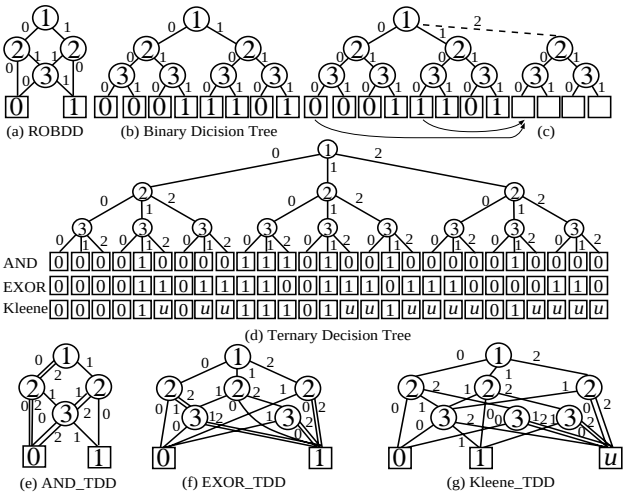

Fig. 3.1. BDD and TDDs

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

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

- *In the case of an AND_TDD:*
$$f_{middle(v)}(\boldsymbol{X}) = f_{low(v)}(\boldsymbol{X}) \cdot f_{high(v)}(\boldsymbol{X}).$$
- *In the case of an EXOR_TDD:*
$$f_{middle(v)}(\boldsymbol{X}) = f_{low(v)}(\boldsymbol{X}) \oplus f_{hogh}(\boldsymbol{X}).$$
- *In the case of a Kleene_TDD:*
$$f_{middle(v)}(\boldsymbol{X}) = f_{low(v)}(\boldsymbol{X}) \odot f_{high(v)}(\boldsymbol{X}).$$

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

Example 3.2 *Fig. 3.1 shows the AND_TDD, the EXOR_TDD, and the Kleene_TDD for f and $\mathcal{F}$ in Example 2.1. By expanding the ROBDD in (a), we have the complete binary decision tree (b). In Fig. 3.1(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. In other words, 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 sub-graphs, 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[11]. EXOR_TDDs are useful to minimize AND-EXOR expressions[12]. Kleene_TDDs are useful to evaluate logic functions in the presence of unknown inputs[5].

Example 3.3 *We can evaluate $\mathcal{F}(\boldsymbol{\alpha})$ in Example 2.1 by using the Kleene_TDD in Fig.3.1(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 $\mathcal{F}(1, u, u) = u$.* ∎

IV. Complexities of TDDs

Definition 4.1 *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 4.1

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

Theorem 4.2

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

Theorem 4.3 *Let f be a parity function. Then, we have*

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

Theorem 4.4 *Let f be a unate function. Then, $N(BDD:f) = N(AND\_TDD:f)$.*

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 4.1 *All the functions of k or fewer variables can be represented by an Kleene_TDD with at most 3^{3^k} nodes.*

Theorem 4.5 *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.

Corollary 4.1 *An arbitrary n-variable function can be represented by a Kleene_TDD with $O(3^n/n)$ nodes.*

Theorem 4.6 *An arbitrary symmetric function of n-variables can be represented by a Kleene_TDD with $O(n^3)$ nodes.*

Theorem 4.7

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

However, in general,

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

V. Experiments and Observation

A. 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. Table 5.1 compares the number of non-terminal nodes in BDDs, AND_TDDs, EXOR_TDDs, and Kleene_TDDs. From this table, we observe the following:

1. Except for e64, rd84, t481, xor5, and z5xp1, $N(AND\_TDD:f) < N(EXOR\_TDD:f)$ holds.
2. TDDs are considerably larger than corresponding BDDs.
3. xor5 is a parity function. Thus, $N(BDD:f) = N(AND\_TDD:f) = N(EXOR\_TDD:f) = N(Kleene\_TDD:\mathcal{F})$.

Table 5.2 shows the sizes of Kleene_TDDs for randomly generated functions. We can observe that the sizes of Kleene_TDDs are $O(3^n/n)$ for n-variable randomly generated functions.

Table 5.2 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 5.1
Sizes of BDDs and TDDs

function	in	out	BDD	TDD		
				AND	EXOR	Kleene
9sym	9	1	33	60	70	94
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
cordic	23	2	75	83	153	271
cps	24	109	987	1457	4808	5653
duke2	22	29	336	522	2176	2555
e64	65	65	1379	1379	1379	2693
ex5	8	63	278	381	444	628
misex1	8	7	36	45	70	92
misex2	25	18	81	81	138	204
misex3	14	14	542	1219	3644	3262
pdc	16	40	560	1024	2321	3031
rd84	8	4	59	79	72	121
sao2	10	4	85	114	216	305
seq	41	35	1248	3873	67414	18745
spla	16	46	581	717	2237	2315
t481	16	1	32	48	43	66
vg2	25	8	194	399	865	961
xor5	5	1	9	9	9	9
z5xp1	7	10	68	79	75	158

TABLE 5.2
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

B. 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. Fig. 5.1(a) shows the numbers of BDD nodes for different s. As shown in Theorem 4.7, the graph is symmetric with respect to $s = 2^{n-1}$. Also, the number of nodes takes its maximum when $s = 2^{n-1}$. Fig. 5.1(b) shows the number of TDD nodes. As shown in Theorem 4.7 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

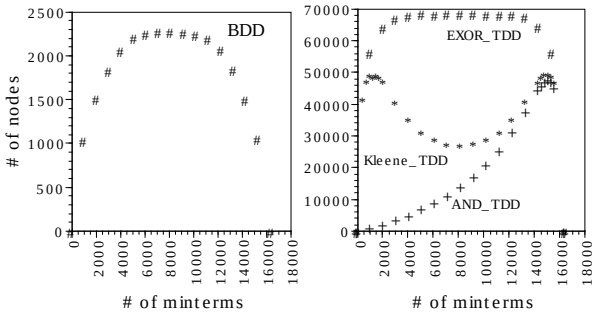

Fig. 5.1. Dependency on the number of true minterms

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.

C. CPU time

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

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:

- 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 $\boldsymbol{\alpha}$ such that $f(\boldsymbol{\alpha}) = 1$ as follows:
  $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, 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 $\boldsymbol{\beta}$ such that $f(\boldsymbol{\beta}) = 0$ as follows:
  $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_n)$, 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 $\bar{F}$.

Fig.5.2 compares simulation time for adders (adr6$\sim$ adr12) for the vecters generated in the above method. This figure shows that Kleene_TDD-based simulation is faster than BDD based one.

D. Observation

The facts in 5.B 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 [7].
   $$G(n,s) = \sum_{k=0}^{n} 2^{n-k} \binom{n}{k} \frac{\binom{w-w_k}{s-w_k}}{\binom{w}{s}},$$
   $$where \quad w = 2^n, \quad and \quad w_k = 2^k.$$

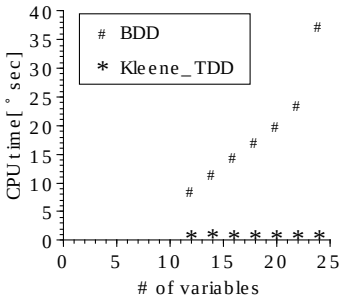

Fig. 5.2. Comparison of CPU time for adr6$\sim$12

   $G(n,s)$ 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.

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] M. Abramovivi, M. A. Breuer and A. D. Friedman, *Digital Systems TESTING and Testable DESIGN*, pp.43-46, Computer Science Press 1990.

[2] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, no. 8, pp. 677-691, Aug. 1986.

[3] R. E. Bryant, "Boolean analysis of MOS circuits," *IEEE Trans. on CAD of Integrated Circuits*, CAD-6(4), pp.634-649, July 1987.

[4] D. L. Dietmeyer, *Logic Design of Digital Systems*, Allyn and Bacon, Inc., Boston, 1971.

[5] 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.

[6] S. C. Kleene, *Introduction to Metamathematics*, Wolters-Noordhoff, North-Holland Publishing, 1952.

[7] 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.

[8] S. Minato, "Graph-based representation of discrete functions," in [10].

[9] T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[10] T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.

[11] T. Sasao, "Ternary decision diagrams and thier applications," in [10].

[12] T. Sasao, "Optimization of pseudo-Kronecker expressions using multiple-place decision diagrams," *IEICE Trans.* vol. E76-D, No. 5, 1993.