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 \(N(Kleene.TDD : F)\) be the number of non-terminal nodes in the Kleene.TDD for \(F\), where \(F\) is the regular ternary function corresponding to \(f\). Then \(N(BDD : f) \leq N(TDD : f)\). For parity functions, \(N(BDD : f) = N(AND.TDD : f) = N(EXOR.TDD : f) = N(Kleene.TDD : F)\). 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 straightforward 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 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.

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 \(\vec{a} = (a_1, a_2, \ldots, a_n)\) be a ternary vector, where \(a_i \in T\). If \(a_i\) is either 0 or 1 for all \(i\), then \(\vec{a} \in B^n\). In this case, \(f(\vec{a})\) is either 0 or 1. If \(a_i\) is \(u\) for some \(i\), then \(\vec{a} \in T^n - B^n\). In this case, for some \(a_i, f(\vec{a})\) is either 0 or 1, but for other \(a_i, f(\vec{a})\) cannot be determined in general. Therefore, it is convenient to introduce a three-valued logic function, \(F : T^n \rightarrow T\), which is derived from \(f\). Note that \(f\) uniquely defines \(F\).

Definition 1: Let \(\vec{a} \in T^n\). \(A(\vec{a})\) 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 \(\vec{a}\), then the set \(A(\vec{a})\) consists of \(2^s\) binary vectors.

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

\[ f(A(\vec{a})) = \begin{cases} f(\vec{a}) & \text{if } A(\vec{a}) = \{0\}. \\ 1 & \text{if } f(A(\vec{a})) = \{1\}. \\ u & \text{if } f(A(\vec{a})) = \{0, 1\}. \end{cases} \]

\(F(\vec{a})\) is called the regular ternary logic function [11] of \(f\) (hereafter, we will call it RT function).
Example 1: Consider the expression
\[ f(x_1, x_2, x_3) = x_1 \lor x_2 \lor x_2 x_3. \]
For the inputs \( \bar{a}_1 = (0, 0, u) \), \( \bar{a}_2 = (1, u, 1) \), and \( \bar{a}_3 = (1, u, u) \), \( F(\bar{a}_1) \), \( F(\bar{a}_2) \), and \( F(\bar{a}_3) \) are derived as follows:

\[
egin{align*}
F(\bar{a}_1) &= \{ f(0,0,0), f(0,0,1) \} = \{ \}, \\
F(\bar{a}_2) &= \{ f(1,0,1), f(1,1,1) \} = \{ 1 \}, \quad \text{and} \\
F(\bar{a}_3) &= \{ f(1,0,0), f(1,0,1), f(1,1,0), f(1,1,1) \} = \{ 0, 1 \}.
\end{align*}
\]

By Definition 2, we have
\[ F(\bar{a}_1) = 0, \quad F(\bar{a}_2) = 1, \quad \text{and} \quad F(\bar{a}_3) = u. \]

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 \( \bar{a}_2 = (1, u, 1) \) by using a naive method, we have the output \( u \) as shown in Fig. 2. However, Example 1 shows that
\[ F(\bar{a}_2) = F(1, u, 1) = 1. \]

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

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 \( \text{index}(v) \in \{ 1, \ldots, n \} \), and two children \( \text{low}(v), \text{high}(v) \in V \). A terminal node \( v \) has as attribute a value \( \text{value}(v) \in B \).

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

For a terminal node \( v \):

\[
\begin{align*}
\text{If } \text{value}(v) &= 1, \quad \text{then } f_v = 1, \\
\text{If } \text{value}(v) &= 0, \quad \text{then } f_v = 0.
\end{align*}
\]

For a non-terminal node \( v \):

Let \( \bar{X} = (x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \).

If \( \text{index}(v) = i \), then \( f_v \) is a function such that
\[ f_v(x_1, \ldots, x_n) = x_i \cdot f_{\text{low}(v)}(\bar{X}) \lor x_i \cdot f_{\text{high}(v)}(\bar{X}). \]

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 \( \text{low}(v) = \text{high}(v) \), 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 \( \text{low}(v) = (\text{high}(v)) \).

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

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 \( \text{index}(v) \in \{ 1, \ldots, n \} \), and three children \( \text{low}(v), \text{high}(v), \) and \( \text{middle}(v) \in V \). A terminal node \( v \) has as attribute a value \( \text{value}(v) \in T \).

For any non-terminal node \( v \), if \( \text{low}(v) \) is also non-terminal, then \( \text{index}(v) < \text{index}(\text{low}(v)) \). Similarly, if \( \text{high}(v) \) is non-terminal, then \( \text{index}(v) < \text{index}(\text{high}(v)) \). Also, if \( \text{middle}(v) \) is non-terminal, then \( \text{index}(v) < \text{index}(\text{middle}(v)) \).

Definition 6: A TDD is a Reduced Ordered Ternary Decision Diagram (ROTDD) if it contains no node \( v \) with \( \text{low}(v) = \text{high}(v) \), 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 \( \& \). Let \( a, b \in T \).

\[
a \& b = \begin{cases} 
a & \text{if } a = b, \\
u & \text{otherwise.}
\end{cases}
\]

The correspondence between AND.TDD (EXOR.TDD, Kleeene.TDD) and a function \( f \) is defined as follows:

For terminal node \( v \):

\[
\begin{align*}
\text{If } \text{value}(v) &= 1, \quad \text{then } f_v = 1, \\
\text{If } \text{value}(v) &= 0, \quad \text{then } f_v = 0.
\end{align*}
\]
For non-terminal node $v$: If $\text{index}(v) = i$, then $f_v$ and $f_{\text{middle}(v)}$ are the functions such that
\[ f_v(\vec{x}) = x_i \cdot f_{\text{low}(v)}(\vec{x}) \lor x_i \cdot f_{\text{high}(v)}(\vec{x}). \]

- In the case of an AND.TDD:
  \[ f_{\text{middle}(v)}(\vec{x}) = f_{\text{low}(v)}(\vec{x}) \cdot f_{\text{high}(v)}(\vec{x}). \]

- In the case of an EXOR.TDD:
  \[ f_{\text{middle}(v)}(\vec{x}) = f_{\text{low}(v)}(\vec{x}) \oplus f_{\text{high}(v)}(\vec{x}). \]

- In the case of a Kleene.TDD:
  \[ f_{\text{middle}(v)}(\vec{x}) = f_{\text{low}(v)}(\vec{x}) \odot f_{\text{high}(v)}(\vec{x}). \]

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 $f_0 = f_{\text{low}}$, $f_1 = f_{\text{high}}$, and $f_2 = f_{\text{middle}}$. In Fig.3(c), $f_{200}$ corresponds to the leftmost 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 $\text{low}(v)$, $\text{high}(v)$, and $\text{middle}(v)$, 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.

**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 $N(\text{BDD} : f)$, $N(\text{AND.TDD} : f)$, and $N(\text{EXOR.TDD} : f)$ be the number of non-terminal nodes in the BDD, the AND.TDD, and the EXOR.TDD for $f$, respectively. Let $N(\text{Kleene.TDD} : \mathcal{F})$ be the numbers of non-terminal nodes in the Kleene.TDD for $\mathcal{F}$.

**Theorem 1:**

\[
N(\text{BDD} : f) \leq N(\text{AND.TDD} : f), \quad N(\text{BDD} : f) \leq N(\text{EXOR.TDD} : f), \quad \text{and} \quad N(\text{BDD} : f) \leq N(\text{Kleene.TDD} : \mathcal{F}).
\]
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. \( \Box \)

Theorem 2:

\[ N(\text{AND.TDD} : f) \leq N(\text{Kleene.TDD} : F). \]

Proof: In the Kleene.TDD for \( F \), 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. \( \Box \)

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

\[ N(\text{BDD} : f) = N(\text{AND.TDD} : f) = N(\text{EXOR.TDD} : f) = N(\text{Kleene.TDD} : F). \]

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 \( N(\text{BDD} : f) = 2n - 1 \).

- **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 \( \bar{a}_1 \) and \( \bar{a}_2 \), where \( \bar{a}_1 = (a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_n) \) and \( \bar{a}_2 = (a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_n) \). Consider the node \( v \) for which the paths for \( \bar{a}_1 \) and \( \bar{a}_2 \) visit. \( \text{low}(v) \) corresponds to \( \bar{a}_1 \) and \( \text{high}(v) \) corresponds to \( \bar{a}_2 \). If the number of 1's in \( \bar{a}_1 \) is an even (odd) number, then the number of 1's in \( \bar{a}_2 \) is an odd (even) number. Hence, \( f(\bar{a}_1) = f(\bar{a}_2) \). Thus, \( f_{\text{low}(v)} \cdot f_{\text{high}(v)} = 0 \) 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 \( N(\text{AND.TDD} : f) = 2n - 1 \).

- **EXOR.TDD:**
  As is the case of AND.TDD, \( f(\bar{a}_1) = f(\bar{a}_2) \) holds. Hence, \( f_{\text{low}(v)} \oplus f_{\text{high}(v)} = 1 \) 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 \( N(\text{EXOR.TDD} : f) = 2n - 1 \).

- **Kleene.TDD:**
  As is the case of AND.TDD, \( f(\bar{a}_1) = f(\bar{a}_2) \) holds. Hence, \( f_{\text{low}(v)} \lor f_{\text{high}(v)} = 1 \) always holds for every non-terminal node. On the Kleene.TDD, all the middle edges point to the terminal \( v \) as shown in Fig. 4(d). Thus, we have \( N(\text{Kleene.TDD} : F) = 2n - 1 \).

Theorem 4: Let \( f \) be a unate function. Then, \( N(\text{BDD} : f) = N(\text{AND.TDD} : f) \).

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

\[ f(x_1, \ldots, x_{k-1}, x_k=0, x_{k+1}, \ldots, x_n) \]

\[ \lor x_k f(x_1, \ldots, x_{k-1}, 1, x_{k+1}, \ldots, x_n). \]

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

\[ f(x_1, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_n) \leq f(x_1, \ldots, x_{k-1}, 1, x_{k+1}, \ldots, x_n). \]

Hence, we have

\[ f(x_1, \ldots, 0, \ldots, x_n) = f(x_1, \ldots, 1, \ldots, x_n). \]

Therefore, \( \text{middle}(v) \) is the same node as \( \text{low}(v) \) 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. \( \Box \)

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
\[
\frac{\sum_{k=1}^{n} (3^{k+1} - 1)}{2} + 3^{n-k}
\]
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
\[1 + 3^1 + 3^2 + \ldots + 3^k = (3^{k+1} - 1)/2\] 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_{03122}. 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."

2. "The number of ways to select k objects from p distinct objects with repetition" is \(p \choose k = p + k - 1 \choose k\).

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. \(\square\)

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 obtained by heuristic algorithms for BDDs\([14],[15]\). Note that the ordering that minimizes the size of a BDD does not always minimize the size of the TDD. We used Sun Ultra 170 workstation 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. \[N(BDD : f) \leq N(TDD : f)\] always holds.

2. Except for e64, rd84, t481, xor5, and z5xpl, \[N(AND.TDD : f) < N(EXOR.TDD : f)\] holds.

3. Except for bc0, chkn, misex3, ts10, x6dn, and xor5, \[N(TDD : f) < N(Kleene.TDD : f)\] 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.

<table>
<thead>
<tr>
<th>function</th>
<th>in</th>
<th>out</th>
<th>BDD</th>
<th>TDD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>AND</td>
<td>EXOR</td>
</tr>
<tr>
<td>add6</td>
<td>12</td>
<td>7</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>adc4</td>
<td>8</td>
<td>5</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>alu2</td>
<td>10</td>
<td>6</td>
<td>61</td>
<td>82</td>
</tr>
<tr>
<td>apex1</td>
<td>45</td>
<td>45</td>
<td>1332</td>
<td>6249</td>
</tr>
<tr>
<td>apex2</td>
<td>39</td>
<td>3</td>
<td>410</td>
<td>542</td>
</tr>
<tr>
<td>apex3</td>
<td>34</td>
<td>50</td>
<td>935</td>
<td>3161</td>
</tr>
<tr>
<td>apex5</td>
<td>117</td>
<td>88</td>
<td>1078</td>
<td>3039</td>
</tr>
<tr>
<td>apla</td>
<td>10</td>
<td>12</td>
<td>102</td>
<td>139</td>
</tr>
<tr>
<td>bc0</td>
<td>26</td>
<td>11</td>
<td>578</td>
<td>3137</td>
</tr>
<tr>
<td>bvc</td>
<td>5</td>
<td>28</td>
<td>108</td>
<td>108</td>
</tr>
<tr>
<td>chkn</td>
<td>29</td>
<td>7</td>
<td>273</td>
<td>649</td>
</tr>
<tr>
<td>clip</td>
<td>9</td>
<td>5</td>
<td>97</td>
<td>174</td>
</tr>
<tr>
<td>col1</td>
<td>18</td>
<td>31</td>
<td>27</td>
<td>39</td>
</tr>
<tr>
<td>con1</td>
<td>7</td>
<td>2</td>
<td>15</td>
<td>25</td>
</tr>
<tr>
<td>cordic</td>
<td>23</td>
<td>2</td>
<td>75</td>
<td>83</td>
</tr>
<tr>
<td>cps</td>
<td>24</td>
<td>109</td>
<td>987</td>
<td>1457</td>
</tr>
<tr>
<td>dce</td>
<td>8</td>
<td>7</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>dist</td>
<td>8</td>
<td>5</td>
<td>152</td>
<td>323</td>
</tr>
<tr>
<td>dk17</td>
<td>10</td>
<td>11</td>
<td>62</td>
<td>78</td>
</tr>
<tr>
<td>dk27</td>
<td>3</td>
<td>9</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td>dsk2</td>
<td>25</td>
<td>29</td>
<td>336</td>
<td>522</td>
</tr>
<tr>
<td>e64</td>
<td>65</td>
<td>65</td>
<td>1379</td>
<td>1379</td>
</tr>
<tr>
<td>ex5</td>
<td>8</td>
<td>6</td>
<td>278</td>
<td>381</td>
</tr>
<tr>
<td>ex6</td>
<td>8</td>
<td>6</td>
<td>278</td>
<td>381</td>
</tr>
<tr>
<td>f51m</td>
<td>8</td>
<td>8</td>
<td>87</td>
<td>87</td>
</tr>
<tr>
<td>in2</td>
<td>19</td>
<td>10</td>
<td>232</td>
<td>377</td>
</tr>
<tr>
<td>in7</td>
<td>26</td>
<td>10</td>
<td>96</td>
<td>131</td>
</tr>
<tr>
<td>inc</td>
<td>7</td>
<td>9</td>
<td>70</td>
<td>104</td>
</tr>
<tr>
<td>intb</td>
<td>15</td>
<td>7</td>
<td>600</td>
<td>2338</td>
</tr>
<tr>
<td>misex1</td>
<td>8</td>
<td>7</td>
<td>36</td>
<td>45</td>
</tr>
<tr>
<td>misex2</td>
<td>25</td>
<td>18</td>
<td>81</td>
<td>138</td>
</tr>
<tr>
<td>misex3</td>
<td>14</td>
<td>14</td>
<td>542</td>
<td>1219</td>
</tr>
<tr>
<td>misl</td>
<td>35</td>
<td>14</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>mlp4</td>
<td>8</td>
<td>8</td>
<td>141</td>
<td>322</td>
</tr>
<tr>
<td>pde</td>
<td>16</td>
<td>40</td>
<td>560</td>
<td>1024</td>
</tr>
<tr>
<td>rd53</td>
<td>5</td>
<td>3</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>rd73</td>
<td>7</td>
<td>3</td>
<td>43</td>
<td>52</td>
</tr>
<tr>
<td>rd84</td>
<td>8</td>
<td>4</td>
<td>59</td>
<td>79</td>
</tr>
<tr>
<td>rsc</td>
<td>8</td>
<td>5</td>
<td>67</td>
<td>67</td>
</tr>
<tr>
<td>ron8</td>
<td>8</td>
<td>5</td>
<td>75</td>
<td>119</td>
</tr>
<tr>
<td>sso2</td>
<td>10</td>
<td>4</td>
<td>85</td>
<td>114</td>
</tr>
<tr>
<td>seq</td>
<td>41</td>
<td>35</td>
<td>1248</td>
<td>3873</td>
</tr>
<tr>
<td>spla</td>
<td>16</td>
<td>46</td>
<td>581</td>
<td>717</td>
</tr>
<tr>
<td>sqg8</td>
<td>8</td>
<td>16</td>
<td>233</td>
<td>503</td>
</tr>
<tr>
<td>sb8i</td>
<td>16</td>
<td>1</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>tial</td>
<td>14</td>
<td>8</td>
<td>686</td>
<td>1732</td>
</tr>
<tr>
<td>ts10</td>
<td>22</td>
<td>16</td>
<td>146</td>
<td>146</td>
</tr>
<tr>
<td>vg2</td>
<td>25</td>
<td>8</td>
<td>194</td>
<td>399</td>
</tr>
<tr>
<td>x0m5</td>
<td>35</td>
<td>5</td>
<td>229</td>
<td>668</td>
</tr>
<tr>
<td>xor5</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>z3xp1</td>
<td>7</td>
<td>10</td>
<td>68</td>
<td>79</td>
</tr>
<tr>
<td>9sym</td>
<td>9</td>
<td>1</td>
<td>33</td>
<td>60</td>
</tr>
</tbody>
</table>

6. xor5 is a parity function. Thus,
\[ N(BDD : f) = N(AND.TDD : f) = N(EXOR.TDD : f) = N(Kleene.TDD : f). \]

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: \(f = x_1y_1 \lor x_2y_2 \lor \ldots \lor x_ny_n\). 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.

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 \(s = 2^n-1\). 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:
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 $\vec{a}$ such that $f(\vec{a}) = 1$ as follows:

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

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

To verify the OFF set.
Let $F$ 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:

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

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

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 vectors 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 consider 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].

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

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 $f$. 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, $N(BDD : f) = N(AND.TDD : f) = N(EXOR.TDD : f) = N(Kleene.TDD : f)$. For unate functions, $N(BDD : f) = N(AND.TDD : f) = N(AND.TDD : f)$. 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


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 has implemented several logic design algorithms under the direction of Professor Tsutomu Sasao. His interests include decision diagrams and exclusive-OR based circuit design.

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.