設 計 自 動 化 74－1
（1995．3．8）

枝刈り機能付き BDD パッケージとその応用

越智 裕之

広島市立大学 情報科学部 情報工学科

近年、二分決定グラフ (BDD) を用いて集合を操作する手法を用いてこれまで解けなかった組合せ最適化問題を解くアルゴリズムが種々提案されつつある。しかしながら与えられた問題の規模が大きくなればグラフのサイズが大きくなり過ぎて取り扱うことができなくなる傾向があり、BDD の応用の障害となっている。本稿では BDD を生成する APPLY 操作そのものに枝刈り (pruning) の機能を組み込んだ APPRUNE 操作を提案する。本稿では閾値関数を評価関数とする枝刈り機能を持つ APPRUNE 操作を例にとってそのアルゴリズムを考え、さらにこれを実装したパッケージを用いた一般化リードマラー論理式の最小化の実験を行なった。

BDD Package with Pruning and its Application

Hiroyuki OCHI

Dept. of Computer Engineering, Faculty of Information Sciences,
Hiroshima City University
151-5, Ozuka, Numata-Cho, Asa-Minami-Ku, Hiroshima, 731-31, JAPAN
ochi@ce.hiroshima-cu.ac.jp

Recently, various algorithms for solving combinatorial optimization problems using BDD-based set manipulation technique are proposed. The ability of such algorithms, however, are limited by the size of BDDs which grow larger for the size of the problems. In this paper, an extension of APPLY operation, named APPRUNE operation, is proposed, which performs APPLY operation (BDD construction) and pruning simultaneously. An algorithm for APPRUNE operation which performs pruning with respect to a threshold function is considered as a prototype, and experimental results for solving exact minimization problem for generalized Reed-Muller expression will be shown.

1 Introduction

Recently, Binary Decision Diagrams (BDDs) [1] have attracted much attention because they makes us possible to manipulate Boolean functions efficiently in terms of time and space. As our understandings of BDDs has deepened, the range of applications has broadened. Besides Boolean functions, we are often faced with manipulating sets of combinations in VLSI CAD problems. By mapping a set of combinations into the Boolean space, they can be represented as a characteristic function using a BDD. This method enables us to implicitly manipulate a huge number of combinations, which have never been practical before. BDD-based set representation is more efficient than conventional methods. However, it can be inefficient at times because BDDs were originally designed to represent Boolean functions.

Minato et al. proposed a new type of BDD, named *0-suppressed BDD* (ZBDD), which has been adapted for set representation [2]. For example, Sasao et al. implemented their exact minimization algorithm for *exclusive-or sum-of-products expression* using a conventional BDD package and a ZBDD package, and shown that ZBDD is more efficient to represent sets of large number of combinations of Boolean functions than conventional BDD [3]. However, there are still many search problems which can be solved much better by using backtrack technique than by manipulating ZBDDs (or BDDs). A major reason is that ZBDDs grow too large because they represent all the solutions, even when we need only one of them. However, there are yet many cases that intermediate ZBDDs grow too large though the final ZBDD is very small. Part of the reason seems that *pruning* is not supported in ZBDD package. To avoid generating unnecessary nodes, we have to construct a ZBDD which represent the set defined by an evaluation function, which can grow so large that the effect of pruning is spoiled.

In this paper, "APPRUNE" (APply + PRUNE) operation is proposed. APPRUNE operation is an extension of APPLY operation; it checks evaluation function to reduce both the number of recursive calls and the number of generated nodes. To show the prototype of the APPRUNE operation, in this paper, a threshold function is taken as an example of the evaluation function for pruning.

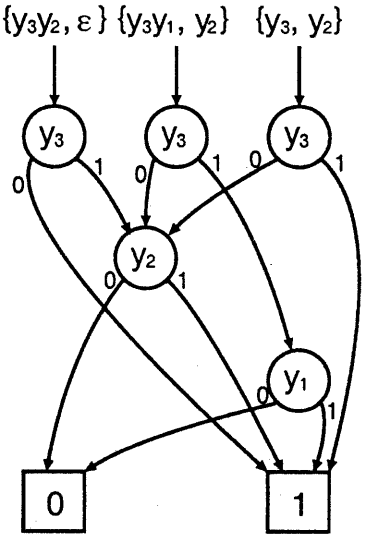

Figure 1: 0-supressed BDD

To evaluate the developed ZBDD package with APPRUNE operation with respect to a threshold function, an exact minimization algorithm for *generalized Reed-Muller expression* [4] is efficiently implemented on a conventional ZBDD package, and then modified to utilize APPRUNE operation. From experimental results, it seems that APPRUNE operation reduces both time and space.

2 Preliminary

2.1 0-suppressed BDD

A *0-suppressed BDD* (ZBDD) [2] is a directed acyclic graph with (at most) two sink nodes labeled by '0' and '1'. Non-terminal nodes are labeled by a variable (or symbol), say y_i, and has exactly two outgoing edges, say '0'-edge and '1'-edge, labeled by '0' and '1', respectively. A variable appear only once in every path in the graph, and variables ap-

pear according to a total order for a whole graph. There is no pair of isomorphic nodes. There is no node whose '1'-edge is directed to '0' sink node. An example of ZBDD is shown in Fig. 1.

Every path from a non-terminal node to the '1' terminal node corresponds to a combination of variables; a variable is contained in the combination iff a '1'-edge of a node labeled by the variable is contained in the path. A non-terminal node represents a set of combinations corresponding to all the paths from the node to the '1' sink node. A '0' sink node represents an empty set ϕ, and '1' sink node itself represents a set $\{\varepsilon\}$, where ε is a combination of no variable. In the following, a node which represents a set P is also denoted by P.

2.2 Conventional APPLY operation

Let us denote the variable labeled to a node P by P.var, and denote by P.succ0 and P.succ1 the successors of '0'- and '1'-edge, respectively, of the node P. To implement a ZBDD (or BDD) package, at least every node require memory space to hold informations .var, .succ0, and .succ1.

Let us consider an intersection operation Intsec() as an example of APPLY operation of a ZBDD package. To guarantee that there is no pair of isomorphic nodes, a hash table named *uniq-table* is introduced. The hash keys for the uniq-table are .var .succ0 and .succ1 of a node. (The other hash table, named computed-table, is discussed later.)

Given sets, say P and Q, represented by a ZBDD, Intsec(P,Q) returns a ZBDD which represents the set $P \cap Q$. Intsec() is performed by the following recursive algorithm:

```
Intsec(P,Q) {
  if (P==φ) return φ;
  if (Q==φ) return φ;
  if (P==Q) return P;
  if (P.var> Q.var) return Intsec(P.succ0,Q);
  if (P.var< Q.var) return Intsec(P,Q.succ0);
  R0=Intsec(P.succ0,Q.succ0);
  R1=Intsec(P.succ1,Q.succ1);
  if (R1==φ) return R0;
  R= search a node with (P.var,R0,R1)
       in uniq-table;
  if (R exist) return R;
  R= generate a node with (P.var,R0,R1);
  append R to the uniq-table;
  return R;
}
```

3 APPRUNE operation

3.1 Basic idea of APPRUNE operation

To solve exhaustive search problem efficiently using ZBDD package, it seems effective to extend APPLY operation to perform *pruning*. Let us call such operation "APPRUNE" (APply + PRUNE).

To implement pruning, an evaluation function should be given. In addition, the evaluation function itself should not be time-consuming. To implement pruning in APPLY operation, evaluation function should be computed easily from local information around a node which is under test for pruning.

As an example of an evaluation function for pruning, let us consider a threshold function $TH(y;w,\tau)$ $(y \in \{0,1\}^n)$ defined as follows:

$$TH(y_0,\ldots,y_{n-1};w_0,\ldots,w_{n-1},\tau) = \begin{cases} 1 & \text{if } \left(\sum_{i=0}^{n-1}(w_i y_i)\right) < \tau \\ 0 & \text{otherwise} \end{cases}$$

In the following, let us assume that every weight (w_i) is non-negative constant, and the given threshold value (τ) is positive. $TH(y;w,\tau)$ is computed by the following recursive algorithm:

```
TH(y_0,...,y_{n-1},τ) {
  if (n==0) return 1;
  if (y_{n-1}==0)
    return TH(y_0,...,y_{n-2},τ);
  if (τ ≤ w_{n-1})
    return 0;
  else
    return TH(y_0,...,y_{n-2},τ − w_{n-1});
```

}

Given a set P represented by a ZBDD, and let T be a set $\{y|TH(y;w,\tau)=1\}$. A ZBDD which represents a set $P \cap T$ is constructed by PruneTH(P,τ), where PruneTH() is implemented as follows:

```
PruneTH(P, τ) {
  if (P==φ) return φ;
  if (P==ε) return ε;
  R0=PruneTH(P.succ0,τ);
  if (τ ≤ w_{P.var}) return R0;
  R1=PruneTH(P.succ1,τ − w_{P.var});
  if (R1==φ) return R0;
  R= search a node with (P.var,R0,R1)
       in uniq-table;
  if (R exist) return R;
  R= generate a node with (P.var,R0,R1);
  append R to the uniq-table;
  return R;
}
```

If we introduce an extra space, say .val, for every node to save the maximum value of the weighted sums for all combinations in the corresponding set, and assume that .val of every existing node have been defined, the above algorithm is improved as follows:

```
PruneTH(P, τ) {
  if (P==φ) return φ;
  if (P==ε) return ε;
  if (P.val< τ) return P;
  R0=PruneTH(P.succ0,τ);
  if (τ ≤ w_{P.var}) return R0;
  R1=PruneTH(P.succ1,τ − w_{P.var});
  if (R1==φ) return R0;
  R= search a node with (P.var,R0,R1)
       in uniq-table;
  if (R exist) return R;
  R= generate a node with (P.var,R0,R1);
  append R to the uniq-table;
  R.val=max(R0.val,R1.val+w_{P.var});
  return R;
}
```

As an example of binary APPRUNE operation of ZBDD, let us consider an Intsec() operation with pruning with respect to the threshold function. IntsecTH() is implemented as follows, where IntsecTH(P,Q,τ) returns a ZBDD that represents a set $P \cap Q \cap T$, and $T=\{y|TH(y;w,\tau)=1\}$:

```
IntsecTH(P, Q, τ) {
  if (P==φ) return φ;
  if (Q==φ) return φ;
  if (P==Q) return PruneTH(P, τ);
  if (P.var> Q.var)
    return IntsecTH(P.succ0,Q, τ);
  if (P.var< Q.var)
    return IntsecTH(P,Q.succ0,τ);
  R0=IntsecTH(P.succ0,Q.succ0,τ);
  if (τ ≤ w_{P.var}) return R0;
  R1=IntsecTH(P.succ1,Q.succ1,τ − w_{P.var});
  if (R1==φ) return R0;
  R= search a node with (P.var,R0,R1)
       in uniq-table;
  if (R exist) return R;
  R= generate a node with (P.var,R0,R1);
  append R to the uniq-table;
  R.val=max(R0.val,R1.val+w_{P.var});
  return R;
}
```

3.2 Computed-table for APPRUNE operation

In order to avoid ever making multiple recursive calls of APPLY operation on the same pair of arguments, a hash based cache, called *computed-table*, is introduced to the conventional BDD and ZBDD packages. The hash keys for the computed-table is a binary operator and two arguments. Every slot of the table should hold, together with the computed result, an operator, and two arguments in order to check miss hit because of hash confliction.

Computed-table is also important for APPRUNE operation. Let us consider IntsecTH() again. Every slot of the table should hold a threshold value in addition to an operator, two arguments, and the computed result. A naive choice of the hash keys are an operator, a threshold value and two arguments, however, this considerably decreases the hit probability.

A good approach is to choose only an operator and two arguments as the hash keys; Computed-table is utilized as follows:

if (operator and arguments hit) {
  if (table[h].τ < τ) ignore the table;
  if (table[h].τ==τ) *return* table[h].R;
  if (table[h].τ > τ)
    return PruneTH(table[h].R,τ);
}

where table[h].τ and table[h].R are the threshold value and the computed result, respectively, read from the slot of the computed-table, and τ is the threshold value of the current operation.

4 Application

As an application of developed 0-suppressed BDD package with APPRUNE operation, an exact minimization algorithm for generalized Reed-Muller expressions (GRM) is implemented. A novel algorithm for exact GRM minimization have been proposed and evaluated using conventional BDD package by Sasao and Debnath [4]. In this section, a modified algorithm which is suitable for implementation using ZBDD package is presented.

4.1 ESOP minimization

Let us introduce ternary representation of a product term, such as 000, 010, and 020 for $\bar{x}_0\bar{x}_1\bar{x}_2$, $\bar{x}_0x_1\bar{x}_2$, and $\bar{x}_0\bar{x}_2$, respectively. A minterm corresponds to a ternary representation without '2' in any digit.

Arbitrary product terms combined by *exclusive-ors* is called an *exclusive-or sum-of-products expression* (ESOP). Let us denote an n-variable ESOP using ternary representation of product terms as follows:

$$\bigoplus_{p\in\{0,1,2\}^n} (y_p p) \quad (y_p \in \{0,1\}) \qquad (1)$$

The problem of finding a minimum ESOP for a given single-output Boolean function corresponds to finding a minimum cost solution of the *Helliwell Equation* $H(y) = 1$. $H(y)$ for an n-variable Boolean function $f(x)$ is defined as follows:

$$H(y_{0\ldots0}, \ldots, y_{2\ldots2}) = \bigwedge_{a\in\{0,1\}^n} \left(1 \oplus f(a) \oplus \bigoplus_{p\in P_a} y_p\right), \qquad (2)$$

where $P_a \subset \{0,1,2\}^n$ is the set of all product terms that cover a minterm a. For every solution (a set of values of ys) of the Helliwell equation, expression 1 gives the corresponding ESOP for $f(x)$. Fig. 2 shows $H(y)$ for $n = 3$.

Using a transformation $\psi_1 \wedge \psi_2 = \psi_1 \wedge (1 \oplus \psi_1 \oplus \psi_2)$ repeatedly to the right side of equation 2, we have:

$$H(y_{0\ldots0}, \ldots, y_{2\ldots2}) = \bigwedge_{b\in\{0,2\}^n} \left(1 \oplus f(b) \oplus \bigoplus_{p\in Q_b} y_p\right), \qquad (3)$$

where $f(x_i = 2) \stackrel{def}{=} f(x_i = 0) \oplus f(x_i = 1)$, and $Q_b \subset \{0,1,2\}^n$ is the set of all product terms that cover exactly one of minterm(s) covered by a product term b. Fig. 3 shows $H(y)$ for $n = 3$.

4.2 GRM minimization

Let $V(p)$ be the set of variables appear in a product term p. An ESOP is a generalized Reed-Muller expression (GRM) iff there is no pair, say (p, q), of product terms in ESOP such that $V(p) = V(q)$. An ESOP denoted by expression 1 is a GRM iff $G(y) = 1$, where

$$G(y_{0\ldots0}, \ldots, y_{2\ldots2}) = \bigwedge_{b\in\{0,2\}^n} \left(\left(\sum_{p \text{ s.t. } V(p)=V(b)} y_p\right) \le 1\right). \qquad (4)$$

From the equation $H(y) \wedge G(y) = 1$ and equations 3 and 4, an algorithm for generating GRM for a given Boolean function f is derived.

4.3 Efficient implementation of GRM minimization

For efficient implementation using BDDs or ZBDDs, ordering of variables of the diagram and ordering of evaluation of expressions should be carefully chosen.

It seems good to chose a variable ordering which satisfies the following property:

$$
\begin{aligned}
H(y) \;=\;& 1 \oplus f(0,0,0) \oplus y_{000} \oplus y_{002} \oplus y_{020} \oplus y_{022} \oplus y_{200} \oplus y_{202} \oplus y_{220} \oplus y_{222} \\
\wedge\;& 1 \oplus f(0,0,1) \oplus y_{001} \oplus y_{002} \oplus y_{021} \oplus y_{022} \oplus y_{201} \oplus y_{202} \oplus y_{221} \oplus y_{222} \\
\wedge\;& 1 \oplus f(0,1,0) \oplus y_{010} \oplus y_{012} \oplus y_{020} \oplus y_{022} \oplus y_{210} \oplus y_{212} \oplus y_{220} \oplus y_{222} \\
\wedge\;& 1 \oplus f(0,1,1) \oplus y_{011} \oplus y_{012} \oplus y_{021} \oplus y_{022} \oplus y_{211} \oplus y_{212} \oplus y_{221} \oplus y_{222} \\
\wedge\;& 1 \oplus f(1,0,0) \oplus y_{100} \oplus y_{102} \oplus y_{120} \oplus y_{122} \oplus y_{200} \oplus y_{202} \oplus y_{220} \oplus y_{222} \\
\wedge\;& 1 \oplus f(1,0,1) \oplus y_{101} \oplus y_{102} \oplus y_{121} \oplus y_{122} \oplus y_{201} \oplus y_{202} \oplus y_{221} \oplus y_{222} \\
\wedge\;& 1 \oplus f(1,1,0) \oplus y_{110} \oplus y_{112} \oplus y_{120} \oplus y_{122} \oplus y_{210} \oplus y_{212} \oplus y_{220} \oplus y_{222} \\
\wedge\;& 1 \oplus f(1,1,1) \oplus y_{111} \oplus y_{112} \oplus y_{121} \oplus y_{122} \oplus y_{211} \oplus y_{212} \oplus y_{221} \oplus y_{222}
\end{aligned}
$$

Figure 2: Helliwell Equation for $n = 3$

$$
\begin{aligned}
H(y) \;=\;& 1 \oplus f(0,0,0) \oplus y_{000} \oplus y_{002} \oplus y_{020} \oplus y_{022} \oplus y_{200} \oplus y_{202} \oplus y_{220} \oplus y_{222} \\
\wedge\;& 1 \oplus f(0,0,2) \oplus y_{000} \oplus y_{001} \oplus y_{020} \oplus y_{021} \oplus y_{200} \oplus y_{201} \oplus y_{220} \oplus y_{221} \\
\wedge\;& 1 \oplus f(0,2,0) \oplus y_{000} \oplus y_{002} \oplus y_{010} \oplus y_{012} \oplus y_{200} \oplus y_{202} \oplus y_{210} \oplus y_{212} \\
\wedge\;& 1 \oplus f(0,2,2) \oplus y_{000} \oplus y_{001} \oplus y_{010} \oplus y_{011} \oplus y_{200} \oplus y_{201} \oplus y_{210} \oplus y_{211} \\
\wedge\;& 1 \oplus f(2,0,0) \oplus y_{000} \oplus y_{002} \oplus y_{020} \oplus y_{022} \oplus y_{100} \oplus y_{102} \oplus y_{120} \oplus y_{122} \\
\wedge\;& 1 \oplus f(2,0,2) \oplus y_{000} \oplus y_{001} \oplus y_{020} \oplus y_{021} \oplus y_{100} \oplus y_{101} \oplus y_{120} \oplus y_{121} \\
\wedge\;& 1 \oplus f(2,2,0) \oplus y_{000} \oplus y_{002} \oplus y_{010} \oplus y_{012} \oplus y_{100} \oplus y_{102} \oplus y_{110} \oplus y_{112} \\
\wedge\;& 1 \oplus f(2,2,2) \oplus y_{000} \oplus y_{001} \oplus y_{010} \oplus y_{011} \oplus y_{100} \oplus y_{101} \oplus y_{110} \oplus y_{111}
\end{aligned}
$$

Figure 3: Alternative form of Helliwell Equation for $n = 3$

- ys are sorted by the number of variables appear in the corresponding product term.
- y_p and y_q are placed as near as possible, if $V(p) = V(q)$.

A variable ordering for $n = 3$ which satisfies the above property is $y_{000} > y_{001} > y_{010} > y_{011} > y_{100} > y_{101} > y_{110} > y_{111} > y_{002} > y_{012} > y_{102} > y_{112} > y_{020} > y_{021} > y_{120} > y_{121} > y_{200} > y_{201} > y_{210} > y_{211} > y_{022} > y_{122} > y_{202} > y_{212} > y_{220} > y_{221} > y_{222}$.

A novel ordering of evaluation have been proposed in [4]. Fig. 4 shows the algorithm for $n = 3$ based on the ordering of evaluation.

5 Experiments

Let us assume that there is a preprocessor that computes a near-minimal GRM for a given Boolean function. Let the number of product terms of the GRM derived from the preprocessor be t_0. The task of exact minimization program can be limited to finding a GRM whose number of product terms is less than t_0.

Two versions of programs are implemented and compared:

(a) Without APPRUNE operation
A ZBDD which represents a set $T = \{y | TH(y; 1, t_0) = 1\}$ is explicitly generated, then a ZBDD which represents a set $\phi'_{2\ldots2} = \phi_{2\ldots2} \cap T$ is generated, then $\phi'_{2\ldots2}$ is used instead of $\phi_{2\ldots2}$ in the succeeding steps of the algorithm shown in the previous section.

(b) Using APPRUNE operation
The algorithm shown in the previous section is implemented, using IntsecTH(P, Q, t_0) for every $P \cap Q$ appear in the algorithm.

1. Construct a ZBDD for

$$\phi_{222} = (1 \oplus f(2,2,2) \oplus y_{000} \oplus y_{001} \oplus y_{010} \oplus y_{011} \oplus y_{100} \oplus y_{101} \oplus y_{110} \oplus y_{111}) \wedge \mathrm{AtMostOne}(y_{000}, y_{001}, y_{010}, y_{011}, y_{100}, y_{101}, y_{110}, y_{111}),$$

where $\mathrm{AtMostOne}(y_0, \ldots, y_{k-1}) = 1$ iff $(\sum_{i=0}^{k-1} y_i) \leq 1$.

2. Construct ZBDDs for

$$\phi_{220} = (1 \oplus f(2,2,0) \oplus y_{000} \oplus y_{002} \oplus y_{010} \oplus y_{012} \oplus y_{100} \oplus y_{102} \oplus y_{110} \oplus y_{112}) \wedge \mathrm{AtMostOne}(y_{002}, y_{012}, y_{102}, y_{112}) \wedge \phi_{222},$$
$$\phi_{202} = (1 \oplus f(2,0,2) \oplus y_{000} \oplus y_{001} \oplus y_{020} \oplus y_{021} \oplus y_{100} \oplus y_{101} \oplus y_{120} \oplus y_{121}) \wedge \mathrm{AtMostOne}(y_{020}, y_{021}, y_{120}, y_{121}) \wedge \phi_{222}, \text{ and}$$
$$\phi_{022} = (1 \oplus f(0,2,2) \oplus y_{000} \oplus y_{001} \oplus y_{010} \oplus y_{011} \oplus y_{200} \oplus y_{201} \oplus y_{210} \oplus y_{211}) \wedge \mathrm{AtMostOne}(y_{200}, y_{201}, y_{210}, y_{211}) \wedge \phi_{222}.$$

3. Construct ZBDDs for

$$\phi_{200} = (1 \oplus f(2,0,0) \oplus y_{000} \oplus y_{002} \oplus y_{020} \oplus y_{022} \oplus y_{100} \oplus y_{102} \oplus y_{120} \oplus y_{122}) \wedge \mathrm{AtMostOne}(y_{022}, y_{122}) \wedge \phi_{220} \wedge \phi_{202},$$
$$\phi_{020} = (1 \oplus f(0,2,0) \oplus y_{000} \oplus y_{002} \oplus y_{010} \oplus y_{012} \oplus y_{200} \oplus y_{202} \oplus y_{210} \oplus y_{212}) \wedge \mathrm{AtMostOne}(y_{202}, y_{212}) \wedge \phi_{220} \wedge \phi_{022}, \text{ and}$$
$$\phi_{002} = (1 \oplus f(0,0,2) \oplus y_{000} \oplus y_{001} \oplus y_{020} \oplus y_{021} \oplus y_{200} \oplus y_{201} \oplus y_{220} \oplus y_{221}) \wedge \mathrm{AtMostOne}(y_{220}, y_{221}) \wedge \phi_{202} \wedge \phi_{022}.$$

4. Construct a ZBDD for

$$\phi_{000} = (1 \oplus f(0,0,0) \oplus y_{000} \oplus y_{002} \oplus y_{020} \oplus y_{022} \oplus y_{200} \oplus y_{202} \oplus y_{220} \oplus y_{222}) \wedge \phi_{200} \wedge \phi_{020} \wedge \phi_{002}.$$

5. Find a minimum cost solution satisfying $\phi_{000} = 1$, and output the corresponding GRM.

Figure 4: GRM minimization algorithm for $n = 3$

From the results of preliminary experiments, (1) even program (a) seems outperforms existing ones, and (2) program (b) seems even better than program (a). The detailed experimental results will be available at the oral presentation.

6 Conclusion

In this paper, APPRUNE (APply + PRUNE) operation for 0-suppressed BDD (ZBDD) package was proposed to reduce time and space to solve exact optimization problems. To show the prototype of the APPRUNE operation, a threshold function was taken as an example of the evaluation function for pruning, and an exact minimization algorithm for generalized Reed-Muller expression (GRM) is implemented using ZBDD package with and without APPRUNE operation.

From experimental results, (1) the developed program for GRM minimization seems outperforms existing ones even if APPRUNE operation is not introduced, and (2) time and space seem to be reduced even more by using APPRUNE operation.

Future work includes developing AP-

PRUNE operations with respect to various kind of evaluation functions which are useful for various problems.

Acknowledgment

The author would like to thank Prof. T. Sasao and Mr. D. Debnath of Kyushu Institute of Technology for their valuable discussions.

References

[1] R. E. Bryant : "Graph-based algorithms for Boolean function manipulation", IEEE Trans. on Computers, vol. C-35, no. 8, pp. 677-691, Aug. 1986.

[2] S. Minato : "Zero-suppressed BDDs for set manipulation in combinatorial problems", Proc. 30th ACM/IEEE Design Automation Conference (DAC'93), pp. 272-277, June 1993.

[3] T. Sasao and M. Matsuura : "A method to derive exact minimum AND-EXOR expressions using Binary Decision Diagrams", Technical Report of IEICE, VLD93-58, Oct. 1993, (in Japanese).

[4] T. Sasao and D. Debnath : "An exact minimization algorithm for generalized Reed-Muller expressions", Proc. IEEE Aaia-Pacific Conference on Circuits and Systems (APCCAS'94), pp. 460-465, Dec. 1994.