Recently, BDDs have attracted much attention because they enable us to manipulate Boolean functions efficiently in terms of time and space. There are many cases that the algorithm based on conventional data structures can be significantly improved by using BDDs\cite{MF89, BCMD90}.

As our understanding of BDDs has deepened, their range of applications has broadened. In VLSI CAD problems, we are often faced with manipulating not only Boolean functions but also sets of combinations. By mapping a set of combinations into the Boolean space, they can be represented as a characteristic function by using a BDD. This method enables us to implicitly manipulate a huge number of combinations, which has never been practical before. Two-level logic minimization methods based on implicit set representation have been developed recently\cite{CMF93}, and those techniques for manipulating sets of combinations are also used to solve general covering problems\cite{LS90}. Although BDD-based set representation is generally more efficient than the conventional methods, it can be inefficient at times because BDDs were originally designed to represent Boolean functions.

In this chapter, we propose a new type of BDD that has been adapted for set representation\cite{Min93d}. This idea, called a Zero-suppressed BDD (ZBDD), enables us to represent sets of combinations more efficiently than using conventional BDDs. We also discuss unate cube set algebra\cite{Min94}, which is convenient for describing ZBDD algorithms or procedures. We present efficient methods for computing unate cube set operations, and show some practical applications of these methods.
6.1 BDDs for Sets of Combinations

Here we examine the reduction rules of BDDs when applying them to represent sets of combinations. We then show a problem which motivates us to develop a new type of BDDs.

6.1.1 Reduction Rules of BDDs

As mentioned in Chapter 2, BDDs are based on the following reduction rules:

1. Eliminate all the redundant nodes whose two edges point to the same node. (Fig. 6.1(a))

2. Share all the equivalent subgraphs. (Fig. 6.1(b))

BDDs give canonical forms for Boolean functions when the variable ordering is fixed, and most uses of BDDs are based on the above reduction rules.

It is important how BDDs are shrunk by the reduction rules. One recent paper[LL92] shows that, for general (or random) Boolean functions, node sharing makes a much more significant contribution to storage saving than the node elimination. For practical functions, however, the node elimination is also important. For example, as shown in Fig. 6.2, the form of a BDD does not depend on the number of input variables as long as the expressions of the functions are the same. When we use BDDs, the irrelevant variables are suppressed automatically and we do not have to consider them. This is