Abstract—We present Ordered Kronecker Functional Decision Diagrams (OKFDDs), a graph-based data structure for the representation and manipulation of Boolean functions. OKFDDs are a generalization of Ordered Binary Decision Diagrams and Ordered Functional Decision Diagrams and as such provide a more compact representation of the functions than either of the two decision diagrams. We review basic properties of OKFDDs and study methods for their efficient representation and manipulation. These algorithms are integrated in our OKFDD package PUMA whose implementation is briefly discussed. Finally we point out some applications of OKFDDs, demonstrate the efficiency of our approach by some experiments and discuss a promising extension of the concept to also allow representation and manipulation of word-level functions.

7.1 INTRODUCTION

Decision Diagrams (DDs) are often used in Computer Aided Design (CAD) systems for the efficient representation and manipulation of Boolean functions. The most popular data structure in this context are Ordered Binary Decision Diagrams (OBDDs) [8] that are used in many applications [9]. Nevertheless, some relevant classes of Boolean functions cannot be represented efficiently by OBDDs [5, 38]. As one alternative Ordered Functional Decision Diagrams (OFDDs) [31] have been introduced and in the meantime they are used in various applications of XOR-based logic synthesis (see e.g. [17]). If ease of manipulation and canonicity are not main concerns, still other types of DDs, like Ternary Decision Diagrams [39] and Kronecker Functional Decision Diagrams.
have proven to be useful especially in the area of technology mapping for multi-level XOR-based circuitry.

Recently, Ordered Kronecker Functional Decision Diagrams (OKFDDs) have been introduced as a means for the efficient representation and manipulation of Boolean functions [24]. OKFDDs are a generalization of OBDDs and OFDDs as well and try to combine the advantages of both representations by allowing the use of Shannon decompositions and (positive and negative) Davio decompositions.

From a (more) theoretical point of view it has been shown that there exist certain classes of Boolean functions whose OFDD size is exponentially smaller than the OBDD representation of the same function, and vice versa [4]. Thus, it is useful to consider a representation like OKFDDs, that integrate both, OBDDs and OFDDs. Furthermore, it has been proved that a "restriction" of the OKFDD concept results in families of functions that lose their efficient representations. It follows that OKFDDs in full generality should be considered. On the other hand, based on a formalization of the concept decomposition type it has been shown in [1], that OKFDDs are the most general type of Ordered Decision Diagram (ODDs). In this sense it is interesting and important to also device effective practical algorithms for representing and manipulating Boolean functions with OKFDDs.

In this chapter we review basic algorithmic properties of OKFDDs and study methods for their efficient representation and manipulation. The data structure allows us to dynamically adapt the representation of a Boolean function to a given problem. But, as well-known for OBDDs [8] and OFDDs [5], OKFDDs are also very sensitive to the variable ordering [24]. In addition to the position of a variable in the ordering so-called decomposition types have to be chosen for OKFDDs. Thus, there is a need for heuristics to choose a suitable variable ordering and decomposition type list for OKFDDs.

In [21] first topology-based heuristics have been presented. These heuristics allow a fast construction of OKFDDs from given circuit descriptions. But, in some cases these heuristics fail to determine small graphs. In [24, 17] it has been shown that dynamic variable ordering methods for OBDDs [29, 38] can also be applied to OKFDDs. These algorithms together with other synthesis algorithms are integrated in our OKFDD package PUMA the implementation of which is discussed below.

Finally we point out some applications of the OKFDD concept to logic synthesis and synthesis for testability. We demonstrate the efficiency of our approach by