Dynamic Minimization of Word-Level Decision Diagrams *

Stefan Höreth

Dept. of Electrical and Computer Engineering
Darmstadt University of Technology
64283 Darmstadt, Germany
http://www.rs.e-technik.tu-darmstadt.de/~sth

Rolf Drechsler

Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg i. B., Germany
drechsle@informatik.uni-freiburg.de

Abstract

*Word-Level Decision Diagrams (WLDDs), like *BMDs and K*BMDs, have recently been introduced as a data structure for verification. The size of WLDDs largely depends on the chosen variable ordering, i.e. the ordering in which variables are encountered, and on the decompositions carried out in each node. In this paper we present a framework for dynamic minimization of WLDDs. We discuss the difficulties with previous techniques if applied to WLDDs and present a new approach that efficiently adapts both variable ordering and decomposition type choice. Experimental results demonstrate that this method outperforms "classical" reordering with respect to run-time and representation size during dynamic minimization of word-level functions.*

1 Introduction

Most formal approaches in verification nowadays make use of function representation by *Decision Diagrams* (DDs). In this context *Ordered Binary Decision Diagrams* (OBDDs) [5] have intensively been studied and frequently applied. Unfortunately, OBDDs fail for some functions with high practical relevance, like multipliers [6]. For this, several extensions of the basic OBDD concept have been proposed over the last few years (see e.g. [22, 17]).

Graph types that allow to represent word-level functions, i.e. functions with a Boolean range and an integer domain, have gained large attention. These DDs are addressed as *Word-Level DDs* (WLDDs) in the following. Examples of WLDDs are e.g. EVBDDs [22], MTBDDs [11, 2], *BMDs [7], HDDs [12], and K*BMDs [15]. In the meantime WLDDs have been integrated in verifications tools [1, 8] and are also used for symbolic model checking [13, 10]. In [21] HDDs have been applied to verification of circuits at the register transfer level. One of the major problems in this setting is decomposition type choice for variables in the circuit, i.e. the decision how to partition the HDD into MTBDDs and BMDs. Unlike bit-level DDs, good decomposition type choice in WLDDs is often essential for verification of HDL constructs.

In this paper we give a framework for dynamic minimization of WLDDs. We address the problems of dynamic variable reordering [18, 24] and dynamic decomposition type choice [14]. A direct extension of the reordering idea used in OBDDs is not possible in all cases (see e.g. [20] for *BMDs or [15] for K*BMDs). The main problem is that an exchange of neighbouring variables is not a local operation. In [20, 16] a first step in this direction has been done, i.e. for *BMDs a restriction has been proposed that allows exchange of variables as a local operation. But, by this restriction graph-size may increase by up to a factor of two. In the following, we show that this drawback is avoided if we loose canonicity during dynamic minimization. Nodes that have to be "repaired" are marked only and the repair run is fully integrated in the reordering algorithm. By this we get significant run-time improvements in comparison to the straightforward approach and larger size reductions than [16]. Another advantage of our method is that it can be generalized to other WLDDs, like K*BMDs. (Notice that our approach can also be applied to *PHDDs [9].)

Furthermore, we present an efficient method to automatically adapt the decomposition type in a WLDD (where this is allowed). The choice of a good decomposition formula is completely integrated in dynamic reordering. Thus, our approach can be seen as an automated tool for verification of arithmetic circuits,

*Copyright 1998 EDAA. Published in the Proceedings of DATE'98, February 23-25, 1998 in Paris, France. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from EDAA.

i.e. integer and floating point, on the word-level.

The algorithms proposed in this work have been implemented on the basis of a hybrid graph manipulation package [20]. Dynamic reordering techniques are supported for 18 different graph types in the package and are applied even if multiple graph types are used together in hybrid operations.

The paper is structured as follows: We review basic notations and definitions in Section 2. The problems with dynamic minimization of WLDDs are discussed in Section 3. In Section 4 we show how these problems are avoided and we propose our new algorithms. Our experiments are described in Section 5. Finally, the results are summarized.

2 Preliminaries

In this section notations and definitions are given that are important for understanding the paper. We give a brief overview on DDs.

All DDs are graph-based representations, where at each (non-terminal) node labeled with a variable x a decomposition of the function represented by this node into two subfunctions (the *low*-function and the *high*-function) is performed. In the following, we assume that the underlying graph is *ordered* and *reduced*, i.e. variables occur in the same order on all paths in the DD and functions represented by nodes of the graph are unique.

For bit-level DDs the following three decomposition have been considered:

$$\begin{array}{rcll} f & = & \overline{x}\, f_{low} \oplus x\, f_{high}, & \text{Shannon } (S) \\ f & = & f_{low} \oplus x\, f_{high}, & \text{positive Davio } (pD) \\ f & = & f_{low} \oplus \overline{x}\, f_{high}. & \text{negative Davio } (nD) \end{array}$$

Function f is represented at node v, while f_{low} (f_{high}) denotes the function represented by the *low*-edge (*high*-edge) of v. $\oplus$ is the Boolean Exclusive OR operation. The recursion stops at terminal nodes labeled with 0 or 1. If at a node a decomposition of type S (D) is carried out this node is called a S-node (a D-node).

In this paper, we consider the same three decompositions for word-level functions, i.e. functions of the form $f : \mathbf{B}^n \to \mathbf{Z}$:

$$\begin{array}{rcl} f & = & (1-x) \cdot f_{low} + x \cdot f_{high}, \\ f & = & f_{low} + x \cdot f_{high}, \\ f & = & f_{low} + (1-x) \cdot f_{high}. \end{array}$$

The notation S, pD and nD is used analogously to the bit-level. x still denotes a Boolean variable, but the values of the functions are integer numbers and they are combined with the usual operations (addition, subtraction, and multiplication) in the ring $\mathbf{Z}$ of integers.

Which decomposition is used, i.e. bit- or word-level, becomes clear from the context.

Decomposition types are associated to the n Boolean variables $x_1, x_2, \ldots, x_n$ with the help of a *Decomposition Type List* (DTL) $d := (d_1, \ldots, d_n)$ where $d_i \in \{S, pD, nD\}$, i.e. d_i provides the decomposition type for variable x_i ($i = 1, \ldots, n$).

Based on the notations and definitions above we now introduce the functions represented by an edge in a DD. The edge function f_e is obtained from the function of the node through multiplication or addition of integer values.

For MTBDD [11, 2], BMD [7], HDD [12], EVBDD [22], *BMD [7], p*BMD [20], and K*BMD [15] the corresponding functions f_e are given in Table 1 (a, m are integer numbers).

graph type	edge function
MTBDD, BMD, HDD	$f_e = f$
EVBDD	$f_e = a + f$
*BMD, p*BMD	$f_e = m \cdot f$
K*BMD	$f_e = a + m \cdot f$

Table 1: Functions represented by edges.

Edge-values are obtained from the node representation during the graph reduction phase. Let $f_{low} = a_{low} + m_{low} \cdot \tilde{f}_{low}$ and $f_{high} = a_{high} + m_{high} \cdot \tilde{f}_{high}$. Then the edge weights of a K*BMD are obtained from the representation of f_{low}, f_{high} depending on the decomposition carried out at the node:

$$\begin{array}{rcll} a & = & a_{low}, & \\ |m| & = & \gcd(m_{low}, a_{high} - a_{low}, m_{high}), & (S\text{-node}) \\ |m| & = & \gcd(m_{low}, a_{high}, m_{high}). & (D\text{-node}) \end{array}$$

(Here gcd denotes the greatest common divisor.) The sign of the multiplicative weight m is taken from the first non-zero value of the arguments to gcd.

Note that all remaining DDs from Table 1 are obtained by further restricting the K*BMD reduction rules and DTL.

Of course, the size of the representation for a given function f depends on the chosen variable ordering and DTL. There are examples which, for a fixed function, provide exponential gaps between representations with differing variable order and DTL [5, 3]. On the other hand, finding good variable orderings and DTLs turns out to be difficult [4] and therefore a lot of heuristics have been developed to find at least reasonable orderings and DTLs, the most successful being *Dynamic Variable Ordering* (DVO) [18, 24] and DVO with DTL change [14].

3 Dynamic Variable Ordering for WLDDs

Dynamic Variable Ordering (DVO) [18, 24, 17, 23] is the state-of-the-art method for finding good orderings for OBDDs and OFDDs. The basic operation is an exchange of neighbouring variables, that is a local operation in these cases, i.e. only pointers must be redirected (see Figure 1). In this section we show that a direct extension of this concept to WLDDs is not possible in all cases, since the exchange of adjacent variables may change edge-values in upper levels, i.e. it is not a local operation.

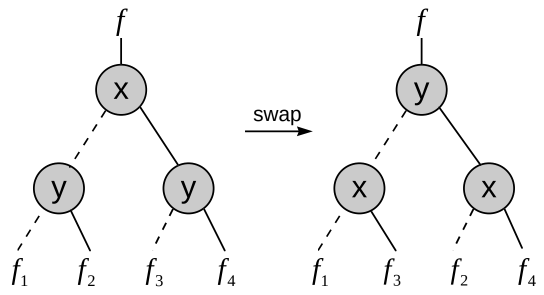

Figure 1: Exchange of adjacent variables.

3.1 WLDDs without Edge-Values

First we consider WLDDs that do not allow edge-values, i.e. MTBDDs, BMDs, and HDDs. It can easily be shown, that an exchange of neighbouring variables is a local operation for these graphs. Moreover, the exchange is even independent of decomposition types.

More precisely, this means that for an exchange of variables only inner pointers f_2 and f_3 (see Figure 1) must be redirected. Function f does not change by this modification, independent of decomposition types for x and y.

3.2 WLDDs with Edge-Values

Augmenting WLDDs with edge-values often results in more compact representations. Unfortunately, exchanging variables now becomes more complicated. We prove that an exchange of neighbouring variables in a K*BMD can complement the sign of a multiplicative edge label at the root node, but not its (absolute) value. Please refer to Figure 2 for an example.

Since reduction rules differ for S-nodes and D-nodes, we consider the situation for decompositions of type S and pD for variables x and y, respectively (extension to other decompositions is straightforward).

In a first step edge-values in the K*BMD are "moved" below variables x and y. Due to additive and multiplicative values at the pointers we obtain:

$$\begin{aligned} f_1 &= a_1 + m_1 \cdot \tilde{f}_1, \quad f_2 = a_2 + m_2 \cdot \tilde{f}_2, \\ f_3 &= a_3 + m_3 \cdot \tilde{f}_3, \quad f_4 = a_4 + m_4 \cdot \tilde{f}_4. \end{aligned}$$

Conversely, additive and multiplicative weight of $f = a + m \cdot \tilde{f}$ are obtained from f_1, f_2, f_3 and f_4 according to the K*BMD reduction rules:

$$\begin{aligned} a &= a_1, \\ |m| &= \gcd(a_3 - a_1, a_2, a_4, m_1, m_2, m_3, m_4). \end{aligned}$$

After moving edge-values below variables x and y, the K*BMD has no marked edges in the upper part and variables can be exchanged simply by swapping pointers f_2 and f_3. Finally, edge-values are restored. For the root node, now one obtains:

$$\begin{aligned} a' &= a_1, \\ |m'| &= \gcd(a_3 - a_1, a_2, a_4 - a_2, m_1, m_2, m_3, m_4). \end{aligned}$$

Since $\gcd(a, a + b) = \gcd(a, b)$, we get:

$$|m'| = \gcd(a_3 - a_1, a_2, a_4, m_1, m_2, m_3, m_4) = |m|.$$

This implies that absolute values do not differ before and after the exchange. However, the sign of the multiplicative edge-value will probably change since it is obtained from edge-values of successors of the root node, which might also change (see Figure 2). Therefore, variable exchange is not a local operation for K*BMDs.

Notice, that all results discussed in this section for K*BMDs directly transfer to *BMDs.

4 Dynamic Minimization of WLDDs

In this section we show that even though exchanging neighbouring variables in a WLDD with edge-values is not a local operation, DVO can be carried out efficiently. We propose two alternatives and give a comparison to underline the differences. Furthermore, we discuss an efficient way of dynamically changing the DTL and incorporate this exchange in DVO.

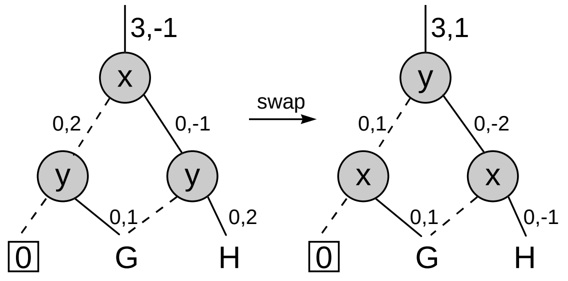

Figure 2: The sign of the multiplicative edge-value at the root of the K*BMD changes independent of decomposition types for x and y.

4.1 Positive K*BMDs

As has been demonstrated above, the only difficulty that might occur is the sign at root nodes. Therefore, it is sufficient to restrict multiplicative edge-values to positive numbers. By this modification exchanging adjacent variables becomes a local operation for K*BMDs[1]. The resulting WLDDs are called pK*BMDs and p*BMD, respectively, in the following. For these WLDDs all classical DVO methods, like sifting, can be applied without any restrictions.

4.2 Sifting of K*BMDs

In the following, we describe an algorithm for dynamic minimization of K*BMDs, similar to sifting [24]. As a consequence of the observation made in Subsection 3.2 exchanging variables is not a local operation. The idea of our method is to loose canonicity (temporarily), i.e. the K*BMD is not completely reduced with respect to reduction rules. Instead, during dynamic reordering we mark all nodes where canonicity might be violated. In a second phase we perform a reduction run to restore the K*BMD.

We now describe the algorithm in more detail: Our sifting-method does not repair each node separately. Instead, nodes are marked only. (For this, one additional bit is needed for each node.)

1. The starting point of our algorithm is a (completely reduced) K*BMD. While sifting a variable **down**, mark all nodes where the sign of incoming edges changes.

2. The next sequence moving the variable **up** incorporates marks of successors to a node (and possibly complements the mark at the current node).

3. Finally, a repair run is carried out which adjusts signs of edge-values in the graph above the last variable position from the **up** phase. Then all marks are cleared.

By this algorithm, canonicity of the graph is restored after reordering of each variable.

4.3 Comparison

To support reordering techniques for (K)*BMDs, either the data structure must be changed to p(K)*BMDs or graphs must be repaired after variable exchanges. Both methods have some advantages and disadvantages:

- Multiplication with a constant only requires constant time for (K)*BMDs, while this is only true for positive numbers in p(K)*BMD. For multiplication with negative numbers a traversal of the graph may become necessary.

- Dynamic reordering for (K)*BMDs needs a reduction run after a sequence of exchanges. This is not needed for p(K)*BMDs. Furthermore, exchanges of variables cannot be randomly distributed over K*BMDs, i.e. reordering is restricted to heuristics like sifting (in order to be efficient).

- Since each (K)*BMD might occur negated and non-negated the relation between the sizes of p(K)*BMDs and (K)*BMDs representing the same function is as follows:

$$||(K)^*BMD|| \leq ||p(K)^*BMD|| \leq 2 \cdot ||(K)^*BMD||$$

In the following section these effects will be demonstrated from a practical point of view.

4.4 Dynamic Change of Decomposition Type

The simplest method to change the decomposition type of a node is to perform a synthesis operation on its successors. But since the synthesis operations have exponential worst case behaviour, this approach might become very time and space consuming.

Instead, we make use of a method proposed for OKFDDs, called DTL-sifting [14]: During dynamic reordering variable x is moved to the bottom level of the WLDD by exchange of neighbouring variables. In the bottom level there exists only nodes marked by x that directly point to constant nodes. Then the decomposition type of these nodes is changed as follows: The type of the node is changed and the corresponding modifications on the edges are performed. In some cases an additional DFS-run must be used to restore canonicity of the (p)K*BMD, i.e. edge-labels have to be changed. An analysis similar to the one in Subsection 3.2 shows that this DFS is needed, if a transformation to (or from) nD is performed, since in this case incoming edges are affected. (The details are left out due to page limitation.) The transformation between S and pD in contrast is a local operation. (For this, we restrict DTL-sifting in the following to S and pD only. Experiments have shown that this is sufficient in our cases.) Then sifting (as described in Subsection 4.2) starts again with the new decomposition type.

5 Experimental Results

In this section we describe experiments carried out on a Sun UltraSparc-170 workstation with 256 MByte of main memory.

[1]This technique has been proposed for *BMDs in [20, 16]

Circuit			pK*BMD				K*BMD					
			Initial		Sift		Initial		“classical” Sift		Sift	
	#in	#out	size	time	size	time	size	time	size	time	size	time
alu4	14	8	3394	1.8s	2124	4.8s	2971	1.6s	1670	8.3s	1670	4.0s
apex1	45	45	2598	4.7s	1558	10.0s	1702	4.4s	1076	150.7s	1076	10.9s
apex3	54	50	1119	62.4s	808	13.7s	880	60.2s	623	338.9s	623	17.8s
apex6	135	99	1631	1.8s	540	45.7s	1086	1.6s	461	3642.1s	461	61.2s
dalu	75	16	4357	9.7s	2222	26.4s	2976	7.3s	968	623.0s	968	26.2s
div8	16	9	8528	5.3s	7933	19.6s	6942	4.4s	6537	35.7s	6537	19.9s
e64	65	65	4031	4.5s	189	28.5s	2077	3.7s	126	403.3s	126	24.8s
frg2	143	139	9681	13.6s	862	65.1s	5855	10.9s	521	4284.1s	521	77.4s
i3	132	6	88780	27.5s	370	108.5s	45448	18.8s	193	4185.8s	193	98.0s
i4	192	6	168046	182.9s	1748	346.1s	87240	129.0s	$> 2h$		850	278.9s
i9	88	63	4165	9.1s	1434	33.4s	2380	7.7s	881	1023.7s	881	36.1s
k2	45	45	2689	5.0s	1729	11.3s	1875	4.6s	1181	166.8s	1181	12.4s
seq	41	35	2559	25.4s	1323	12.4s	1309	19.3s	662	139.5s	662	10.9s
too_large	38	3	21601	16.1s	1813	33.4s	10806	10.5s	915	147.3s	915	20.9s
x3	135	99	1535	2.0s	550	46.2s	1180	1.7s	437	3558.2s	437	61.3s
mean	81	46	21648	24.8s	1680	53.7s	11648	19.0s	1161	17m17s	1140	50.7s

Table 2: Sifting for (p)K*BMDs and ”classical” sifting.

To support reordering techniques for K*BMDs, either the multiplicative edge-value must be restricted to positive numbers (pK*BMD) or the reordering algorithm must be modified (see Subsection 4.2). Table 2 compares both approaches. The name of the benchmark is given in the first column. We first constructed an OBDD. Then the OBDD is transformed into a (p)K*BMD by constructing a weighted sum over all outputs. (For more details on output encodings see [7].) By this a (p)K*BMD with a single output is obtained. The DTL for the (p)K*BMD consisted of type *pD* only. The size (given by the number of nodes) of the resulting (p)K*BMD and the time needed for the transformation are shown in columns Initial. Then we applied a minimization heuristic similar to sifting [24]. The resulting graph-sizes and run-times are reported in columns Sift. Analogously to OBDDs the reduction in size can be up to 98% [24, 23]. Column “classical” Sift gives the results for a straightforward extension of the sifting heuristic to the word-level, i.e. variables are exchanged and a reduction is carried out to retain canonicity in each step.

As can be seen the initial size of K*BMDs is always smaller then for pK*BMDs. This can range up to a factor of two. The times to compute initial representations as well as times for sifting do not vary too much and are relatively independent of the K*BMD-model. But run-times needed for “classical” sifting of K*BMDs are much larger. E.g. for circuit *i4* the straightforward approach failed, while reordering based on pK*BMDs and our new concept terminated in about 4 CPU minutes. Thus, our new reordering method unifies the advantages of the two approaches, i.e. small representation sizes and fast run-times. (Notice that the results also directly transfer to *BMDs.)

In a second series of experiments we compare sifting and DTL-sifting starting from the same initial K*BMD. The results are given in Table 3. The WLDDs are derived from OBDDs as described above. Again, initial graph-sizes and construction times are given in column Initial. The K*BMD sizes after sifting and DTL-sifting are given in column Sift and DTL-Sift, respectively, followed by the run-times of the algorithms. Furthermore, the number of variable exchanges and the maximum graph-size during minimization are reported in columns #swap and peak, respectively.

DTL-sifting reduces graph-size over sifting by up to a factor of 4 at comparable run-time (see circuit *div8*). The average reduction by DTL-sifting is more than a factor of 2 over the sifting heuristic, while time requirements are less then twice the time needed for sifting. The peak size during minimization is about the same for both techniques.

The large size reduction obtained by DTL-sifting also has a significant impact on run-time behaviour of graph construction algorithms. In Table 4 sifting and DTL-sifting have already been applied during the graph construction phase. (Notice that for these circuits the K*BMDs could not be constructed using the initial variable ordering.) (DTL-)sifting occurred whenever the total number of nodes increased by more than a factor of two over the last minimization. Some examples have only been finished partially (the number of outputs are then given in brackets in column

Circuit			Initial		Sift				DTL-Sift			
	#in	#out	size	time	size	time	#swap	peak	size	time	#swap	peak
alu4	14	8	2971	1.6s	1670	4.0s	307	3102	678	6.7s	672	3102
apex1	45	45	1702	4.4s	1076	10.9s	3759	1870	464	17.2s	7776	1870
apex3	54	50	880	60.2s	623	17.8s	5383	986	211	32.4s	11688	986
apex6	135	99	1086	1.6s	461	61.2s	35120	1090	426	111.8s	70357	1181
dalu	75	16	2976	7.3s	968	26.2s	9758	4128	380	40.9s	21881	4128
div8	16	9	6942	4.4s	6537	19.9s	425	8859	1505	17.6s	940	7367
e64	65	65	2077	3.7s	126	24.8s	7309	2081	123	52.6s	15767	3734
frg2	143	139	5855	10.9s	521	77.4s	38731	6056	509	155.6s	82402	6056
i3	132	6	45448	18.8s	193	98.0s	32459	47529	137	190.6s	70604	47535
i4	192	6	87240	129.0s	850	278.9s	69314	93723	888	541.0s	146327	95184
i9	88	63	2380	7.7s	881	36.1s	13395	2502	462	60.8s	28872	2502
k2	45	45	1875	4.6s	1181	12.4s	3682	2332	389	18.3s	7842	2579
seq	41	35	1309	19.3s	662	10.9s	3028	1528	320	18.7s	6553	1528
too_large	38	3	10806	10.5s	915	20.9s	2652	14310	457	34.7s	5456	15854
x3	135	99	1180	1.7s	437	61.3s	35219	1187	395	111.7s	71132	1416
mean	81	46	11648	19.0s	1140	50.7s	17369	12752	490	94.0s	36551	13001

Table 3: Sifting vs. DTL-Sifting.

Circuit			Sift				DTL-Sift			
	#in	#out	size	time	#swap	peak	size	time	#swap	peak
C432	36	7	15288	8m18s	29088	101247	2484	2m13s	30832	12315
C880	60	26	46072	17m45s	96317	103470	28887	12m40s	108647	67781
C3540	50	(12)		$> 2h$		$\geq$ 913080	27836	14m12s	66571	95044
C7552	207	(20)		$> 2h$		$\geq$ 2814941	3531	25m31s	717475	53183
bin2bcd	16	35	16734	3m40s	6369	53511	1653	46s	6304	12473
mcalu32	77	35		$> 2h$		$\geq$ 905891	19758	26m33s	193635	113721

Table 4: Sifting and DTL-Sifting for hard examples.

#out). For the hard test cases of Table 4 improvements obtained by DTL-sifting become even more obvious. For these examples, DTL-sifting is superior to sifting in any respect. This is due to the fact, that sifting cannot change decomposition types. However, both S and pD decompositions are important for word-level functions. As a consequence, the sifting heuristic failed for some examples, while DTL-sifting succeeded.

6 Conclusions

In this paper a framework for dynamic minimization of WLDDs has been presented. We demonstrated that a direct extension of previously used concepts is not possible in all cases.

In order to support reordering techniques for WLDDs like *BMD or K*BMD, either the data structure or reordering algorithms must be changed. A comparison of both approaches has been given in the paper.

We have proposed a modification to the sifting heuristic that temporarily looses canonicity during minimization. Sifting of WLDDs obtains graph-size reductions comparable to the bit-level (up to 98%) and requires negligible overhead to restore canonicity.

Based on this new sifting algorithm, we reexamined a minimization technique for OKFDDs, called DTL-sifting. In contrast to the bit-level, DTL-sifting of word-level functions reduces graph size (over sifting) by more than a factor of two on the average.

Due to this large reduction, DTL-sifting is even faster then sifting for hard examples which require dynamic minimization during graph construction. As a consequence, DTL-sifting succeeded for examples where the sifting heuristic failed.

DTL-sifting is even more promising for representing HDL constructs that combine data part and control (e.g. conditional expressions). Typically, both require different decompositions which are chosen automatically by DTL-sifting. Another direction for future research will be the integration of the presented word-level concepts into BFS-like synthesis techniques, such as MORE [19].

References

[1] L. Arditi. *BMDs can delay the use of theorem proving for verifying arithmetic assembly instructions. In *FMCAD*, pages 34–48, 1996.

[2] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Prado, and F. Somenzi. Algebraic decision diagrams and their application. In *Int'l Conf. on CAD*, pages 188–191, 1993.

[3] B. Becker, R. Drechsler, and R. Enders. On the computational power of bit-level and word-level decision diagrams. In *ASP Design Automation Conf.*, pages 461–467, 1997.

[4] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. on Comp.*, 45(9):993–1002, Sept. 1996.

[5] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[6] R.E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. on Comp.*, 40(2):205–213, 1991.

[7] R.E. Bryant and Y.-A. Chen. Verification of arithmetic functions with binary moment diagrams. In *Design Automation Conf.*, pages 535–541, 1995.

[8] R.E. Bryant and Y.-A. Chen. ACV: an arithmetic circuit verifier. In *Int'l Conf. on CAD*, pages 361–365, 1996.

[9] R.E. Bryant and Y.-A. Chen. *PHDD: an efficient graph representation for floating point circuit verification. In *Int'l Conf. on CAD*, pages 2–7, 1997.

[10] Y. Chen, E. Clarke, P. Ho, Y. Hoskote, T. Kam, M. Khaira, J. O'Leary, and X. Zhao. Verification of all circuits in a floating-point unit using word-level model checking. In *FMCAD*, pages 389–403, 1996.

[11] E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi terminal binary decision diagrams: An efficient data structure for matrix representation. In *Int'l Workshop on Logic Synth.*, pages P6a:1–15, 1993.

[12] E.M. Clarke, M. Fujita, and X. Zhao. Hybrid decision diagrams - overcoming the limitations of MTBDDs and BMDs. In *Int'l Conf. on CAD*, pages 159–163, 1995.

[13] E.M. Clarke and X. Zhao. Word level symbolic model checking - a new approach for verifying arithmetic circuits. Technical Report CMU-CS-95-161, 1995.

[14] R. Drechsler and B. Becker. Dynamic minimization of OKFDDs. In *Int'l Conf. on Comp. Design*, pages 602–607, 1995.

[15] R. Drechsler, B. Becker, and S. Ruppertz. K*BMDs: A new data structure for verification. In *European Design & Test Conf.*, pages 2–8, 1996.

[16] R. Drechsler and S. Höreth. Manipulation of *BMDs. In *Asian and South-Pacific Design Automation Conference*, 1998.

[17] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski. Efficient representation and manipulation of switching functions based on ordered kronecker functional decision diagrams. In *Design Automation Conf.*, pages 415–419, 1994.

[18] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.

[19] A. Hett, R. Drechsler, and B. Becker. MORE: Alternative implementation of BDD packages by multi-operand synthesis. In *European Design Automation Conf.*, pages 164–169, 1996.

[20] S. Höreth. Implementation of a Multiple-Domain Decision Diagram Package. In *CHARME*, Chapman & Hall, pages 185–202, 1997.

[21] G. Kamhi, O. Weissberg, and L. Fix. *Automatic Datapath Extraction for Efficient Usage of HDD.* In *CAV*, LNCS 1254, pages 95–106, 1997.

[22] Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Design Automation Conf.*, pages 608–613, 1992.

[23] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Int'l Conf. on CAD*, pages 74–77, 1995.

[24] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.