Analysis of Composition Complexity and How to Obtain Smaller Canonical Graphs

J. Jain\textsuperscript{*} K. Mohanram\textsuperscript{1} D. Moundanos\textsuperscript{*} I. Wegener\textsuperscript{1} Y. Lu\textsuperscript{8}

Abstract

We discuss an open problem in construction of Reduced Ordered Binary Decision Diagrams (ROBDDs) using composition, and prove that the worst case complexity of the construction is truly cubic. With this insight we show that the process of composition naturally leads to the construction of (even exponentially) compact partitioned-OBDDs (POBDDs) [12]. Our algorithm which incorporates dynamic partitioning, leads to the most general (and compact) form of POBDD - graphs with multiple root variables. To show that our algorithm is robust and practical, we have analyzed some well known problems in Boolean function representation, verification and finite state machine analysis where our approach generates graphs which are even orders of magnitude smaller.

1 Introduction

A large number of problems in VLSI-CAD and other areas of computer science can be formulated in terms of Boolean functions. A central issue in the development of a solution to these problems is to find a compact representation for Boolean functions on which the basic Boolean operations (functional manipulation) can be carried out efficiently.

The requirements of compact representation and ease of manipulation are generally conflicting. Currently, ROBDDs serve as the most popular compromise between these conflicting requirements. ROBDDs are canonical and are efficiently manipulated. For many practical functions, ROBDDs are compact as well. Due to these properties, ROBDDs are the Boolean representation of choice in most CAD applications. In this paper, we will address not only the issue of efficient manipulation (specifically, we will analyze the complexity of composition) but also that of compact representation (specifically, we will give a robust algorithm to generate compact POBDDs).

The thrust of this paper, and thus our results, can be described under the following categories:

1. OBDD Manipulation Complexity: First, we answer an important open problem raised by Bryant in his seminal paper on ROBDDs [4]: Is composition of ROBDDs a cubic time operation or only a quadratic time operation? We present a worst-case example of ROBDD compose which is also a worst-case example for the more general IF-THEN-ELSE (ITE) operation.

2. POBDD\textsuperscript{1} Representation Complexity: Second, we understand the complexity of composition, we note that the actual process of composition contains a local OR operation which leads to cubic-sized ROBDDs but quadratic sized POBDDs! Specifically, we will show that after a series of \(k\) worst-case compositions on an ROBDD of size \(G\), the sizes of POBDDs are provably smaller than the sizes of ROBDDs by at least \(G^{2k}\).

3. POBDD Representation - Practical Algorithms: Our above mentioned observations lead to a robust and efficient algorithm for POBDD construction. We will show the effectiveness of our approach through extensive experimental results conducted on actual industrial circuits. Our results show that we can generate orders of magnitude smaller graphs for several circuits, both combinational as well as sequential. Note that POBDDs are canonical and manipulable. Hence, once our algorithm can be shown to generate smaller graphs in Boolean functions that arise in both combinational and sequential circuits, a wider applicability of the method stands demonstrated.

1.1 Critical comments on the relevance of our POBDD results

Apart from the obvious importance of developing practical algorithms for any representation scheme, we believe that the following observations explain and further accentuate the relevance of this work.

Observation 1: (Lack of practical acceptance of most representation schemes):

Since function manipulation and representation is an important problem, it has been studied extensively over the years. However, most function representation schemes such as EVBDDs, FDDs, OKFDDs, IBDDs, Parity-BOs [3, 5, 9] etc. have one or more of the following disadvantages - the node decomposition used or the ordering rules allowed or the manner of carrying out fundamental functional manipulation operations in these schemes are very "different" from those of ROBDDs. Since significant resources have been spent on developing the state of the art with respect to ROBDDs, researchers and practitioners are often reluctant to switch from ROBDDs to a new Decision Diagram scheme - especially in the absence of very compelling theoretical and/or experimental evidence. Since POBDDs are nothing but a set of ROBDDs, most of the state of the art techniques developed for ROBDDs can be easily used for POBDDs. In fact, POBDDs can be seamlessly integrated into an ROBDD package with only a minimal change in the manipulation or analysis methods. Hence, the demonstration of a compelling theoretical and experimental justification for POBDDs has a significant utility.

\(w_1, \ldots, w_k\), and the ROBDD \(P_i\), \(i = 1, \ldots, k\) represents the image of the \(i\)th window, \(w_i\), under the given function \(F\). Thus, an implicit OR of all partitions, i.e., \(P_1 \lor \ldots \lor P_k\), would represent \(F\) over its complete Boolean space. Different partitions in a POBDD can have different variable orderings. It has been shown that POBDDs provide a compact, canonical and efficiently manipulable representation for Boolean functions [12, 13, 17]. Most ROBDD based algorithms can be adapted easily for POBDDs.

The manipulation algorithms are relatively more different where (universal) quantification is required. Efficient quantification algorithms for POBDDs were developed and explained by [17]).
Observation 2 (Limitations of previous research on POBDDs):

Despite the force of Observation 1, due to the unavailability of appropriate algorithms in almost all previous publications on POBDDs, a very restricted form of POBDDs was experimentally investigated. For example, POBDDs [18, 17] were generated by partitioning the Boolean space by a set of cubes. Thus, practically, the partitions always had a common root variable. The resulting graphs can be called POBDDs with a common root variable. However, this data structure is provably no more compact than free BDDs or typed-Free BDDs [3]. Also, the number of partitions to be generated were always apriori selected - an approach which also has significant theoretical limitations (see Section 4.2.2). In [12] one of the suggested POBBDD algorithm (graph division) did allow POBDDs with different root variables but the suggested algorithm was not explored in detail. In our experiments we have found graph division difficult to automate and often less efficient than cube-based partitioning or monolithic ROBDDs.

We present a robust POBDD construction algorithm which uses the full power of the POBDD model: here the POBDDs are dynamically generated and can have multiple root variables. Thus, they can be exponentially more compact than the cube-based POBDDs as well as free BDDs.

1.2 Organization of the Paper

The outline of the paper is as follows. Note that the analysis of BDDs formed in the process of composition lies at the heart of the theoretical as well as the practical analysis in this paper. Hence, in Section 2 we will briefly discuss a practical method for constructing ROBDDs for a given Boolean circuit using decomposition and composition procedures. In Section 3 we discuss the composition of ROBDDs in the composition of ROBDDs. In Section 4 we discuss the cube complexity of composition in relation to ROBDDs, contrast ROBDDs and POBDDs and introduce an algorithm for POBDD construction. In Section 4 we also show how using POBDDs can generate smaller, canonical graphs for various industrial circuits that are intractable by ROBDDs. In Section 5 we focus on combinational verification and show how POBDDs can help verify many circuits that were proving intractable otherwise. In Section 6 we discuss POBDDs for simplifying the representation of Boolean functions which arise during the analysis of sequential circuits. Finally we conclude with Section 7.

2 OBDD Construction using a Decomposed Representation

In the following we will first discuss how to create ROBDDs from any circuit using a process of decomposition and composition. As shown in [14], the ROBDDs in the above manner is not only practical but usually more efficient than the more prevalent method which consists of building the ROBDDs starting from the primary inputs using only the ROBBD apply.

2.1 Creating a Decomposed Representation

We first decompose the given function \( F \), and then obtain the window functions for creating its POBDD by analyzing the decomposed ROBDD for \( F \). Given a circuit representing a Boolean function \( f : B^n \rightarrow B \), defined over \( X_n = \{ x_1, \ldots, x_n \} \), our decomposition strategy consists of introducing new variables based on the increase in the ROBDD size during a sequence of ROBDD operations or based on structural analysis as discussed in [14]. We introduce a new variable whenever the size of the ROBDD resulting due to some operation is deemed to have blown up. In our current implementation, the new variable is introduced only if the resulting ROBDD size exceeds some predetermined threshold. Also, decomposition points are added when any ROBDD grows beyond another threshold value. This ensures that the decomposition points themselves do not have very large ROBDDs. We find that even a simple size-based decomposition scheme works quite effectively for demonstrating the potential of ROBDDs.

2.2 Creating the Composed Representation

At the end of the decomposition phase we obtain a decomposed representation, \( f_d(\Psi, \Sigma) \), of \( f \) where \( \Psi = \{ \psi_1, \ldots, \psi_n \} \) is called a decomposition set of the circuit and each \( \psi_i \in \Psi \) is a decomposition point. Let \( \Psi_{bd} = \{ \psi_{r1}, \ldots, \psi_{rd} \} \) represent the array containing the ROBDDs of the decomposition points, i.e., each \( \psi_i \in \Psi \) has a corresponding ROBDD, \( \psi_{bd} \in \Psi_{bd} \), in terms of primary input variables as well as possibly other \( \psi_j \in \Psi \), where \( \psi_j \neq \psi_i \). The composition [4] of \( \psi_i \) in \( f_d(\Psi, \Sigma) \) is denoted by \( f_d(\Psi, \Sigma)(\psi_i \leftarrow \psi_{bd}) \). Where,

\[
f_d(\Psi, \Sigma)(\psi_i \leftarrow \psi_{bd}) = \overline{\psi_{bd} \wedge \delta_{\psi_i} + \psi_{bd} \wedge \delta_{\psi_i}}
\]

The vector composition of the \( \Psi \) in \( f_d(\Psi, \Sigma) \) is denoted by \( f_d(\Psi, \Sigma) \cdot (\psi_i \leftarrow \psi_{bd}) \) and represents successive composition of the variables in \( \Psi \) in \( G_d \) by their corresponding \( \psi_{bd} \). Here, \( \cdot \) represents the order in which \( \psi_i \)s are composed in \( f_d \). An algorithm for determining a good order of composition has been proposed in [14]. Briefly their algorithm is as follows:

(a) Compose first the ROBDDs of those decomposition points which are not in the support set of other decomposition points.
(b) Given a choice between two decomposition points, compose the one which introduces the least number of new variables in \( G_f \), the ROBDD for function \( f \).
(c) If a tie persists, compose the function which has the smallest ROBDD.

3 Composition Complexity for ROBDDs

Bryant presented two fundamental algorithms, apply, and compose, for manipulating ROBDDs in [4]. The algorithm for apply takes as input two ROBDDs \( G_f \) and \( G_g \) representing functions \( f \) and \( g \) respectively and a binary operator \( \odot \) and produces OBDD \( G_h \) representing function \( f \odot g \). Bryant has proved that the size \( |G_h| \) of \( G_h \) is bounded by \( |G_f| \cdot |G_g| \) and has provided a function where (for a given variable order) the above cubic blow-up can be observed. The algorithm for compose can be regarded as replacement by functions; for two functions \( f \) given by ROBDD \( G_f \) and \( g \) (given by ROBDD \( G_g \)) and a variable \( x_i \), the function \( f_{x_i=0} \) defined by \( h = \{ i(i, f_{x_i=1}, f_{x_i=0}) \} \) (if then \( f_{x_i=1} \) else \( f_{x_i=0} \)) has to be represented. If \( G_h \) is the ROBDD that represents \( h \), Bryant showed that \( |G_h| \) has an upper bound of \( O(|G_f||G_g|) \). However, Bryant could not find any worst-case example requiring the above cubic blow-up. Thus, he observed that "[14], pp 261) It is unclear whether the efficiency of this algorithm truly has a quadratic dependence on the size of its first argument, or whether this indicates a weakness in our performance analysis."  

We answer Bryant's open problem in the following; we present a worst-case example of compose which is also a worst-case example for the more general \( \odot \) operation.

3.1 OBDD Composition: A Worst-Case Analysis

We prove in the following that the composition truly has a cubic worst-case complexity.

Proof: Let \( MUX(a, x) \) be defined on \( n + k \) variables \( a_0, \ldots, a_{k-1} \) and \( x_0, \ldots, x_{n-1} \), where \( n = 2^k \). The \( a \)-variables are control variables describing a number \( |a| \in \{ 0, \ldots, n - 1 \} \); the \( x \)-variables are data variables addressed by \( a \). Hence, \( MUX(a, x) \) is defined as \( x_{|a|} \). For this multiplexer (or direct storage access function) the ROBDD variable ordering \( (a, x) \) is optimal. The \( a \)-variables may be in arbitrary order. The same holds for the
x-variables. The ROBDD size is $2n + 1$ for this ordering. The complete binary a-tree contains $n − 1$ nodes with $n$ outgoing edges. For the outgoing edge representing the address a the variable $x_a$ is tested. This leads to $n$ x-nodes and 2 sinks.

We define $f$ on $n + 2k + 1$ variables: $a_0, \ldots, a_k; b_0, \ldots, b_k; x_0, \ldots, x_{n−1}$, and $s$. Let $f(a_0, b_0, s, x) = s \land MUX(a_0, a_k, b, x_{n−1}) + s \land MUX(b_0, b_k, x_{n−1})$. Moreover, we define $g$ on $n + k$ variables $c_0, \ldots, c_k; x_0, \ldots, x_{n−1}$ by $g(c, x) = MUX(c_0, c_k; x_0, x_{n−1})$.

Both functions are considered as functions on all $n + 3k + 1$ variables in $a, b, c, x$ and $s$ and we investigate for both functions the optimal variable ordering: $s, a_0, \ldots, a_k; b_0, \ldots, b_k; c_0, \ldots, c_k; x_0, \ldots, x_{n−1}$.

The ROBDD size of $f$ is $1 + 2(n−1) + n + 2 = 3n + 1$. We start with an $s$-node and then realize $MUX(a, x)$ and $MUX(b, x)$. The ROBDDs for these functions may share the $x$-nodes and the sinks. The ROBDD size of $g$ is $2n + 1$. Let $h = f_{s\neg s} = MUX(c, x) \land MUX(a, x) + MUX(c, x) \land MUX(b, x)$.

By the characterization of the ROBDD size by Stiebling and Wegener [22] the ROBDD for $h$ contains at least as many $x$-nodes as there are different cofactors which are obtained by assigning constants to the control variables $a, b, c$. If $[a] = i, [b] = j$, and $[c] = k$, we obtain the cofactor $x_0 x_i + x_2 x_j$. If $i \neq j$, the cofactors depend essentially on all their three variables and are all different. For $i = j$ we obtain the function $x_1$. Hence, we get $n^2(n−1) + n = n^3 − n^2 + n$ different cofactors which have to be represented by different $x$-nodes. These $x$-nodes are reached by edges from the upper part where the control variables are tested. Since the ROBDD has a single source, the ROBDD contains at least $n^3 − n^2 + n$ − 1 nodes where control variables are tested and 2 sinks. This leads to the lower bound $2n^3 − 2n^2 + 2n + 1$ for the ROBDD size of $h$. Hence, $|G_f|^2 |G_h| \geq 18n^3 + 21n^2 + 8n + 1$ and $h = f\neg s = s \land MUX(a, x) + MUX(b, x)$ has an ROBDD size of at least $2n^3 − 2n^2 + 2n + 1$. Hence, $|G_h| = \Theta(|G_f|^2 |G_h|)

4 Cubic-complexity of Composition and Partitioned-ROBDDs

Note that on the right hand side of composite operations of Equation 1, both the dijuncts $(\psi_{\theta_\phi} \land f_{\psi_{\theta_\phi}})$ and $(\psi_{\theta_\psi} \land f_{\psi_{\theta_\psi}})$ are mutually orthogonal, and can thus be represented as different partitions in a POBDD. In our method we successively compose the $\psi_{\theta_\phi}$ in $f$. If the graph size increases drastically for some composition (say $\psi_f$) we can abort the composite operation and instead separately compute each of the two dijuncts in the right hand side of equation 1; each of these dijuncts constitutes a separate partition. In other words, we can create two orthogonal partitions instead of continuing the ROBDD composition to completion.

\[ \psi_{\theta_\psi} \land f_{\psi_{\theta_\psi}} \land \psi_{\theta_\phi} \land f_{\psi_{\theta_\phi}} \]

Note that each partition can be constructed in quadratic time using the apply algorithm. That is, the complexity of creating such partitions, during the composition of $\psi_0$ in $f$, is only $O(|f_0| \cdot |\psi_{\theta_\psi}|)$. We can now individually call the composition routine on each of the partitions. As remaining decomposition points are composed inside any partition, a partitioning is performed each time a blow-up of composition is observed. If in composing $f_0$ by $\psi_1, \ldots, \psi_k$, a composition blow-up is observed a total of $c$ times, then we will have produced $c + 1$ ROBDDs, $P_1, \ldots, P_{c+1}$. Together, $P_1, \ldots, P_{c+1}$ represent the complete Boolean space of $F$ in terms of primary input variables $x_1, \ldots, x_n$ and constitute the POBDD representation of $F$.

4.1 Contrasting the Complexity of ROBDDs with the Partitioned Representation

It is instructive to compare the size of the monolithic ROBDD with the corresponding POBDD that a sequence of $k$ worst case compositions may lead to.

Complexity of ROBDDs: Due to the cubic complexity of ROBDD composition, composing k ROBDDs $g_1, \ldots, g_k$, each of size $|g|$, in an ROBDD $G$ can require $O(|G|^k \cdot |g|^{2k−1})$. The size of POBDDs is bounded by a much smaller polynomial as discussed below.

Complexity of POBDDs: Considering each of the disjuncts in Equation 2 as a separate partition, each partition can be constructed in $O(\frac{|G|}{|g|})$. While composing the remaining $k − 1$ decomposition points, each of the original partitions can be further split into $2^k−1$ partitions. Thus, in the worst case, we are left with $2^k$ partitions. Note that the size of each partition is bounded by $O(\frac{|G|}{|g|^k})$. Thus, the composition complexity of creating POBDDs by our method is bounded by only $O(2^k \cdot \frac{|G|}{|g|^k})$. (In the above analysis, for simplicity, each composition point is assumed to be a function of primary inputs only.)

4.2 Our POBDD Algorithm

With the above theoretical framework, our POBDD algorithm can be summarized as follows. To distinguish it from the cube-based partitioning algorithm [17], where required, we will refer to this algorithm as a function-based partitioning algorithm.

1. Generate functional decomposition points $\psi_1, \ldots, \psi_k$ during the construction of ROBDDs (See Section 2). The output ROBDD $G_D$ is now a decomposed ROBDD.

2. Compose the ROBDDs of $\psi_1, \ldots, \psi_k$ in $G_D$ in the order discussed in Section 2.2.

3. When the ROBDD composite blows up, generate two partitions as per Equation 2.

4. Compose each partition as in Step 2 and Step 3.

4.2.1 Heuristics to Decide When to Partition

In order to describe our procedure that detects the blow up, we define the following parameters: Max Limit $M$, Min Limit $\mu$, Partition Factor $N$, Acceptance Factor $\gamma$, Multiplication Factor $\phi$. We partition when one of the following conditions is met (in the following, R is the resulting graph, $G$ is the graph in which we are composing and $g_1, \ldots, g_k$ are the functions to be composed): (1) $|R| > M$, (2) $|R| \geq \phi \cdot |G| + |\psi_{\theta_\phi}|$, (3) $|\psi_{\theta_\psi}| \geq \phi \cdot (|G| + \sum |g_i|)$. Assuming that the partitioning process has produced two graphs $P_1, P_2$, we accept the partition when the following condition holds: $|P_1| \leq \gamma \times |R| \land |P_2| \leq \gamma \times |R|$

4.2.2 Function-Based Partitioning: Some Useful Properties

Although the conventional cube-based partitioning as discussed in [12, 17, 18] can be combined with the function-based partitioning, it will be instructive to contrast both partitioning styles.

Observation 3: Each partition in Equation 2 can be ordered separately (using, say, dynamic reordering techniques). Thus, each of the final $2^k$ ROBDDs (partition) may have a different variable at the root.

In contrast consider a Boolean function which is partitioned into $2^k$ partitions $P_{1}, \ldots, P_{2k}$ using cubes (partial assignments) generated from, say,
Given $k$ input variables $x_1, \ldots, x_k$. In such a case, $x_1, \ldots, x_k$ are effectively removed from the support set of ROBDDs of each partition. Thus, even when we order each partition separately, $x_1, \ldots, x_k$ implicitly remain the root variables of the ROBDDs of each partition. As discussed in [3, 17], such a ROBDD has almost the same space complexity as typed-Free ROBDDs [2]. Note, it has been proved that there are many functions which require exponentially large space using free BDDs or typed-Free BDDs but have a compact representation using POBDDs [3].

Theorem 1. [3, 17]: There exist functions which are compact when represented using POBDDs if different partitions can have different root variables but are exponentially sized for BDDs as well as for typed-Free BDDs (and thus for cube-based partitioning also).

Observation 4: As we recursively create (traverse) the partitioning tree, along any path, the number of partitions are determined dynamically as partitioning is invoked only when a blow-up is detected. Thus, as a practical matter, we do not face the limitations of selecting a static $k$ as was the case in the cube-based partitioning algorithm of [17].

Theorem 2. [3]: Given any integer $m$, there exist functions for which a POBDD with only $m$ partitions is exponential in size but a POBDD with $m + 1$ partitions is compact.

Thus our paper also answers a very important theoretical question raised in [3] where it was shown that a general POBDD model (that allows different root variables) where the number of partitions can be dynamically determined is one of the most powerful known function representation scheme. It was also conjectured that it may be difficult to realize robust and efficient algorithms to construct such POBDDs in practice. However, in the following, we will show that our POBDD construction procedure which incorporates all of the above desirable features, is indeed practical and efficient. We demonstrate the efficiency of POBDDs by applying them on a large set of intractable industrial circuits.

4.3 Some Experimental Results on Comparing the Sizes of POBDDs and ROBDDs for Intractable Industrial Circuits

In our experiments we took a large set of industrial circuits (ranging in size from 2K gates to 20K gates) whose output ROBDDs are extremely hard to construct. Predictably many of these circuits, when decomposed, had some difficult cases to compose. We ran our experiments (using CUDD-2.3.0) on various workstations with 512MB RAM, and limited the ROBDD manager size to approximately 20 Million nodes. It is interesting to observe that for most of these cases we were able to build the output POBDD even when only two partitions were created. It is easy to see that the worst case cubic bound can theoretically result in a very large ROBDD for even very small composition cases. For example, composing an ROBDD $g$ in an ROBDD $f$, with $|f| = |g| = 1000$ nodes, can result in a graph with close to a billion nodes. As a practical issue, the ROBDD sizes for all decompositions were large enough that it was not feasible to allow the ROBDD composition to carry on to the worst case cubic bound. Thus, the ROBDD composition was terminated when a given threshold size was reached. However, we do find many cases in these circuits which exhibit at least a quadratic blow up during the ROBDD construction (see Table 1). Note, that except for the multiplier C6288, most of the ISCAS-85 benchmark circuits are extremely easy for ROBDD representation and do not contain very difficult composition cases. Also, C6288 has been proven to be intractable for ROBDDs as well as POBDDs.

As far as the time performance of both the methods are concerned, we have found that even when the sizes of the final graphs are similar, POBDDs  

<table>
<thead>
<tr>
<th>Circuits</th>
<th>ROBDD</th>
<th>POBDDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ckt1</td>
<td>222787</td>
<td>23487</td>
</tr>
<tr>
<td>Ckt2</td>
<td>fails</td>
<td>619634</td>
</tr>
<tr>
<td>Ckt3</td>
<td>fails</td>
<td>542977</td>
</tr>
<tr>
<td>Ckt4</td>
<td>fails</td>
<td>2.57M</td>
</tr>
<tr>
<td>Ckt5</td>
<td>fails</td>
<td>3.32M</td>
</tr>
<tr>
<td>Ckt6</td>
<td>fails</td>
<td>4.48M</td>
</tr>
<tr>
<td>Ckt7</td>
<td>fails</td>
<td>5.12M</td>
</tr>
</tbody>
</table>

Table 1: Comparing the ROBDD and POBDD sizes (measured in # of nodes) for some very difficult industrial circuits

run faster since at any given time, smaller graphs are being constructed (reordered). In general, we have noticed a factor of 2 reduction in runtime as well as maximum manager size even when the output POBDD and monolithic ROBDDs were comparable in size. We constructed the ROBDDs using both apply based methods as well as decomposition/composition based procedures.

5 Combinational Verification

In this section we show how POBDDs fit into a typical Combinational Verification flow [21] described in Figure 1. To check the equivalence of circuits $C_1$ and $C_2$, one starts with an internal equivalence based verification core, that incorporates several verification procedures that can determine internal equivalences between the two given circuits. (For more details on this subject please see [21]). The internal gates that are found to be equivalent are merged, and the given circuit pair (composite circuit) is greatly simplified. Many output gates are easily verified in this process. However, some difficult output gates are still left unverified. But since our given pair of circuits has been greatly simplified due to merging of internally equivalent gates, one now attempts to create the ROBDD for the output function by treating each gate in the composite circuit that was found to be internally equivalent as a decomposition point. These decomposition points are composed in a suitable order (see Section 2) to create the required output ROBDD. In
Table 2: Combinational Verification Results for Some Industrial Test Cases

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Comb. Verif. - ROBDDs</th>
<th>Comb. Verif. - POBDDs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RunTime</td>
<td>Result</td>
</tr>
<tr>
<td>Ver0</td>
<td>fails</td>
<td>2:26:48</td>
</tr>
<tr>
<td>Ver1</td>
<td>fails</td>
<td>51:50</td>
</tr>
<tr>
<td>Ver2</td>
<td>fails</td>
<td>1:41:59</td>
</tr>
<tr>
<td>Ver3</td>
<td>fails</td>
<td>1:09:45</td>
</tr>
<tr>
<td>Ver4</td>
<td>fails</td>
<td>1:10:29</td>
</tr>
<tr>
<td>Ver5</td>
<td>1:49</td>
<td>EQUAL</td>
</tr>
<tr>
<td>Ver6</td>
<td>fails</td>
<td>19:35</td>
</tr>
<tr>
<td>Ver7</td>
<td>2:03</td>
<td>EQUAL</td>
</tr>
<tr>
<td>Ver8</td>
<td>fails</td>
<td>2:30:20</td>
</tr>
</tbody>
</table>

| Table 3: Partitioning Effect During Fixed-Point Algorithm |
|-------------|------------------|------------------|
| Circ.       | #FFs  | New nodes created before aborting $^a$ | First Partition | Second Partition |
| s1269       | 37    | > 700000 | 82797         | 89454             |
| tc100       | 40    | > 1.2M   | 337106        | 405707            |
| s4863       | 104   | > 319130 | 8021          | 8021              |
| s3271       | 116   | > 331632 | 102563        | 102172            |
| s3330       | 132   | > 11.8M  | 1030656       | 1409214           |
| s136        | 136   | > 3M     | 886634        | 702299            |

$^a$ This limit represents the maximum number of new nodes that we allow to be created during composition before invoking partitioning. As should be obvious, the actual monolithic graph may be a lot bigger. We estimate that in most cases the graphs are at least 5 times larger than the size limit, as calculated in terms of number of nodes, at which we abandon the compose.

our experiments, in the above verification process, if the ROBDD construction fails (due to the composition blow up), then we automatically create POBDDs. In Columns 2–3 of Table 2 we report the runtime and result of the output verification process using ROBDDs for 9 difficult industrial test cases (ranging from 9K to 16K gates). As we can see from column 2, our internal correspondence based verification method that does not use POBDDs fails for most of these cases. In Columns 4–5 we do the same for POBDDs. In this case we are able to verify all 9 Circuits. Our program was written in C++ and the experiments were run on Sun UltraSpare-Is with 512MB of RAM.

6 Composition Instances and Partitioning During Bi-Simulation Analysis

In this paper we are attempting to show that reducing the complexity of Composition through the use of POBDDs can generate smaller ROBDDs during a bi-simulation relation based analysis.

6.1 Sequential Circuit Equivalence: Notational Framework and Definitions

A sequential circuit consists of $n$ latches and a combinational logic block. It has $m$ primary inputs (PIs) $x = < x_1, x_2, \ldots, x_m >$ and $l$ primary outputs (POs) $y = < y_1, y_2, \ldots, y_l >$. It also has $n$ present-state variables (PSVs) $q = < q_1, q_2, \ldots, q_n >$ and $n$ next-state variables (NSVs) $Q = < Q_1, Q_2, \ldots, Q_n >$. Each PO is a combinational boolean function of the PIs and the PSVs, that is: $y_i = o_i(x, q)$, $i = 1, 2, \ldots, l$. Each NSV is a combinational boolean function of the PIs and the PSVs: $Q_i = a_i(x, q)$, $i = 1, 2, \ldots, n$. To make the notation more compact we will use boldface lower case letters to denote vector boolean functions (VBFs). Over the last 15–20 years there have been many attempts to formalize the notions of equivalence and containment between two sequential machines [11, 20, 19, 8, 15]. In this paper we follow the notion of equivalence as defined below:

Definition 1: (Equivalence) Two sequential circuits $C_a$ and $C_b$ are called equivalent — denoted by $C_a \equiv C_b$ — iff for every state of $C_a$ there is at least one equivalent state of $C_b$ and for every state of $C_b$ there is at least one equivalent state of $C_a$.

Definition 2: The states $q_a$ and $q_b$ of two sequential circuits $C_a$ and $C_b$ respectively are called equivalent iff the output functions of $C_a$ and $C_b$ assume identical values for $q_a$ and $q_b$ and for any pair of states of $C_a$ and $C_b$ reachable from them by any input sequence.

The above definitions directly led to the fixed-point algorithm described in the Appendix, which computes the set of pairs $(q_a, q_b)$ of states of $C_a$ and $C_b$ that are equivalent. The central operation of this algorithm is the composition of the NSFs into the set of $k$-Equivalent states at each step of the fixed-point computation. This is described in Equation 4 in the Appendix.

6.2 Composition Instances and Partitioning During Bi-Simulation Analysis

During the composition of next state functions while computing the fixed-point (see Equation 4, Appendix), the ROBDD sizes often blow up. Thus, it is a very suitable time to introduce functional partitioning instead of carrying out the given composition to completion. Note, after partitioning is invoked, one can proceed with the remaining compositions on each part in isolation. In such an algorithm, several issues need to be considered such as storage management, ordering considerations, exchange of information among partitions before universal quantification etc. which are beyond the scope of this paper. In this discussion we are only trying to demonstrate a way of reducing the complexity of the composition operations commonly encountered in a typical sequential circuit analysis algorithm.

6.3 Preliminary POBDD Results on Using Composition in Bi-Simulation Relation

In Table 3 we are showing some typical cases where partitioning was involved during the fixed-point computation. The circuits are some of the harder ones in the 1993 Addendum to the ISCAS-89 benchmarks (tc100 is an adder-accumulator circuit). During normal composition we monitor the number of new nodes created (nodes that are not obtained by a cache lookup) and when a given limit (see Column 3) is exceeded we invoke partitioning. We then report in Columns 4–5 the number of nodes in each of the two partitions created after moving them into a new ROBDD manager and performing an explicit reordering operation. Our program is written in C++ and utilizes CUDD-2.3.0. The experiments were run on Sun UltraSpare-Is with 512M of RAM.

7 Conclusions

In this paper we discussed an open problem in ROBDD construction using composition and proved that the worst case complexity of composition is truly cubic. We then showed how the composition procedure naturally leads to a POBDD construction algorithm when partitions are created at the point
of composition blow up. We also showed that the sizes of the resulting POBD- DDs can be exponentially smaller than ROBDDs. Our POBDD construction algorithm allows multiple root variables, and determines the number of part-
itions dynamically. Thus, the resulting POBDDs are more compact than
Free-BDDs or (cube-based) POBDDs which were studied in [17]. We also showed that our techniques are robust and efficient and thereby answer a conjecture in [3] whether POBDDs with the above characteristics can be
easily realized. Specifically, we demonstrated the utility of the proposed
procedures in some well known problems of Boolean function representa-
tion, combinational verification and bi-simulation relation based finite state
machine analysis. In our experiments we have found an even orders of mag-
itude improvement over ROBDDs for several difficult functions. Since the
notion of partitioning is general and can be applied to any ROBDD represen-
tation, our results should be applicable to any decision diagram scheme
which allows function composition.

References
Machines Using Boolean Functional Vectors”. Formal VLSI Correctness
switching functions based on Ordered Kronecker Functional Decision
Sequential Circuits”. Ph.D Dissertation, Dept. of Electrical and Computer
Engineering, The University of Texas at Austin, 1995.
based Framework for Verifying Sequential Equivalence”. Proc. ITC,
pages 865–874, 1996.
for verification and related problems”. Brown/MIT VLSI Conference,
of Electrical and Computer Engineering, The University of Texas at
Austin, 1993.
ROBDD Construction”. In Formal Methods in CAD 96, LNCS.
Springer-Verlag, 1996.
A. Sangiovanni-Vincentelli. “A Study of Composition Schemes for
Mixed Apply–Compose based Construction of ROBDDs”. Proc. VLSI
“Partitioned-ROBDDs - A Compact, Canonical and Efficiently Manage-
able Representation for Boolean Functions”. ICCAD, November
1996.
[18] A. Narayan, A. Iles, J. Jain, R. Brayton, and A. Sangiovanni-
Fussell. “An Efficient Filter-based Approach for Combinational Veri-

Appendix
In the following we present the basic implementation of the algorithm
which computes the set of pairs \((\bar{q}_a, \bar{q}_b)\) of states of \(C_a\) and \(C_b\) that are equivalent.

The 1-equivalent states (the pairs of states for which circuits \(C_a\) and \(C_b\)
will produce identical output values for any single input vector \(\bar{x}\), when they are
found at \(q_a\) and \(q_b\), respectively) will be represented by the predicate
\(E_1(\bar{q}_a, \bar{q}_b)\).

\[
E_1(\bar{q}_a, \bar{q}_b) = \bigwedge_{i=1}^{n} \forall \bar{x} \exists \bar{q}_i \cdot \bar{f}_i(\bar{x}, \bar{q}_a) \equiv \bar{f}_i(\bar{x}, \bar{q}_b)
\]

(3)

where \(\equiv\) is the boolean equivalence (XNOR) operator.

Now, given the states \((\bar{q}_a, \bar{q}_b)\) their successor states for some input \(\bar{x}\)
are the states \((\bar{q}_a', \bar{q}_b')\) where \(\bar{q}_a' \equiv s_a(\bar{x}, \bar{q}_a)\) and \(\bar{q}_b' \equiv s_b(\bar{x}, \bar{q}_b)\). But then the pairs of states \(R_i(\bar{q}_a, \bar{q}_b)\) which have successor states not among
the i-equivalent states \(E_i(\bar{q}_a, \bar{q}_b)\) are the pairs of states given by the following:

\[
\exists \bar{x} \exists \bar{q}_a, \bar{q}_b : \bar{q}_a \equiv s_a(\bar{x}, \bar{q}_a) \land \bar{q}_b \equiv s_b(\bar{x}, \bar{q}_b) \land \neg E_1(\bar{q}_a, \bar{q}_b)
\]

or if we use composition: \(R_i(\bar{q}_a, \bar{q}_b) = \exists \bar{x} \equiv E_1(s_a(\bar{x}, \bar{q}_a), s_b(\bar{x}, \bar{q}_b))\)

The predicate \(E_i(\bar{q}_a, \bar{q}_b)\) is now given by:

\[
E_{i+1}(\bar{q}_a, \bar{q}_b) = E_i(\bar{q}_a, \bar{q}_b) \lor \neg R_i(\bar{q}_a, \bar{q}_b)
\]

So we have:

\[
E_{i+1}(\bar{q}_a, \bar{q}_b) = E_i(\bar{q}_a, \bar{q}_b) \lor \forall \bar{x} : E_i(s_a(\bar{x}, \bar{q}_a), s_b(\bar{x}, \bar{q}_b))
\]

(4)

This is a fixed-point algorithm that terminates because the state space
is finite. So we get the set \(E(\bar{q}_a, \bar{q}_b)\) of equivalent states when the fixed
point \(E_{i+1} = E_{i+1-1}\) is reached. Finally, there is a simple proposition we can
formulate which will show us whether the two circuits at hand are equivalent.

More specifically, \(C_a \Leftrightarrow C_b\) if

\[
(\forall \bar{q}_a : \exists \bar{q}_b : E(\bar{q}_a, \bar{q}_b)) \land (\forall \bar{q}_b : \exists \bar{q}_a : E(\bar{q}_a, \bar{q}_b))
\]

(5)