Analysis of Composition Complexity, and How to Obtain Smaller Canonical Graphs

Abstract

We discuss an open problem in ROBDD construction 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 construction of (even exponentially) compact partitioned-OBDDs (POBDDs)[17]. Our algorithm which incorporates dynamic partitioning, leads to the most general (and compact) form of POBDDs - graphs with multiple root variables. To show 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 efficient solution for these problems is to find a *compact* representation for Boolean functions on which the basic Boolean operations (*functional manipulation*) can be efficiently carried out.

The requirements of compactness and manipulability are generally conflicting. Currently, Reduced Ordered Binary Decision Diagrams (henceforth ROBDDs) serve as the most popular compromise between these conflicting requirements. ROBDDs are canonical and efficiently manipulable. 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 both the issues of **efficient manipulation** (specifically, we will analyze the complexity of composition) as well as **compact representation** (specifically, we will give a robust algorithm to generate compact POBDDs)[1]

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

[1]In partitioned-OBDDs (POBDDs) [17], a Boolean function $F : B^n \rightarrow B$ is represented by a set of ROBDDs $P_1, \ldots, P_k$ [17]. The Boolean space of the function is partitioned in k, usually disjoint, subspaces (windows): $w_1, \ldots, w_k$, and the ROBDD $P_i, i = 1, \ldots, k$ represents image of the w_i, the i^{th} partition, under the given function F. Thus, an implicit OR of all partitions, i.e., $P_1 \vee \ldots \vee P_k$, 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 [17]. Most ROBDD based algorithms can be used easily for POBDDs.

1. OBDD Manipulation Complexity: First, we answer an important open problem raised by Bryant in his seminal ROBDD paper [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 Representation Complexity: Second, as we understand the complexity of composition, we find 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^{2^k}.

3. POBDD Representation: Practical Algorithms: Our above mentioned observations lead to a robust and efficient POBDD construction algorithm. We will show the effectiveness of our approach through extensive experimental results conducted over actual industrial circuits. Our results show that we can generate orders of magnitude smaller graphs for numerous circuits, both combinational as well as sequential. Note POBDDs are canonical and manipulable. Hence, once our algorithm can be shown to (automatically) 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-BDDs [3, 5, 10] 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 a nothing but a set of ROBDDs, most of the state of the art techniques developed for ROBDDs can easily be used for POBDDs. In this context note that POBDDs can be seamlessly integrated in an ROBDD package with only a minimal change in the manipulation or analysis methods.[2] Thus, development of compelling theoretical and experimental justification for POBDDs has a significant utility.

Observation 2 (Limitations of previous research on POBDDs):

Despite the force of Observation 1, due to unavailability of appropriate algorithms in almost all previous publications on POBDDs, a very restricted form of POBDDs was experimentally investigated. For example, POBDDs [6, 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 as 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).

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. Since analysis of BDDs formed in the process of composition lies in the heart of the theoretical as well as the practical analysis in this paper, 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 complexity of ROBDD composition. Following that, in Section 4 we discuss the cubic complexity of Composition in relation to POBDDs, contrast ROBDDs and POBDDs and introduce an algorithm for POBDD construction. In Sections 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 analysis of sequential circuits. Finally we conclude with Section 7.

2 OBDD Construction using 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], creating the BDDs in the above manner is not only practical but usually

[2]The manipulation algorithms are relatively more different where (universal) quantification is required. Efficient quantification algorithms for POBDDs were developed and explained by [17]).

more efficient than the more prevalent method which consists of building the BDDs starting from the primary inputs using only the BDD *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 BDD for F. Given a circuit representing a Boolean function $f\colon B^n \rightarrow B$, defined over $X_n = \{x_1 \dots x_n\}$, our decomposition strategy consists of introducing new variables based on the increase in the OBDD size during a sequence of OBDD operations or based upon structural analysis as discussed in [14]. We introduce a new variable whenever the total number of nodes in a ROBDD manager increases by a disproportionate measure due to some operation. In our current implementation, the check for memory explosion is done only if the manager size is larger than a predetermined threshold. Also, decomposition points are added when the 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 POBDDs.

2.2 Creating the Composed Representation

At the end of the decomposition phase we obtain a decomposed representation, $f_d(\Psi, X)$, of f where $\Psi = \{\psi_1, \dots, \psi_k\}$ is called a *decomposition set* of the circuit and each $\psi_i \in \Psi$ is a *decomposition point*. Let $\Psi_{bdd} = \{\psi_{1_{bdd}}, \dots, \psi_{k_{bdd}}\}$ represent the array containing the OBDDs of the decomposition points, i.e., each $\psi_i \in \Psi$ has a corresponding OBDD, $\psi_{i_{bdd}} \in \Psi_{bdd}$, in terms of primary input variables as well as (possibly) other $\psi_j \in \Psi$, where $\psi_j \neq \psi_i$. Similarly we represent the array of $\psi_{i_{bdd_{w_i}}}$ by $\Psi_{bdd_{w_i}}$. The composition [4] of ψ_i in $f_d(\Psi, X)$ is denoted by $f_d(\Psi, X).(\psi_i \leftarrow \psi_{i_{bdd}})$ where,

$$f_d(\Psi, X).(\psi_i \leftarrow \psi_{i_{bdd}}) = \overline{\psi_{i_{bdd}}} \wedge f_{d_{\overline{\psi_i}}} + \psi_{i_{bdd}} \wedge f_{d_{\psi_i}} \quad (1)$$

The vector composition of the Ψ in $f_d(\Psi, X)$ is denoted as $f_d(\Psi, X).(\Psi \leftarrow \Psi_{bdd})$ and represents successive composition of ψ_i's into 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 BDDs 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 BDD for function f.

(c) If a tie persists, compose the function which has the smallest BDD.

3 Composition Complexity for OBDDs

Bryant presented two fundamental algorithms, *apply*, and *compose*, for manipulating OBDDs in [4]. The algorithm for *apply* takes as input two OBDDs G_f and G_g representing functions f and g respectively and a binary operator $\odot$ and produces OBDD G_h representing function $f \odot g$ ($\odot$ may be AND, OR, EXOR, ...). Bryant has proved that the size $|G_h|$ of G_h is bounded by $|G_f| * |G_g|$, and has provided a function where (for a given variable order) the above quadratic blow-up can be observed. The algorithm for *compose* can be regarded as replacement by functions; for two functions f (given by BDD G_f) and g (given by BDD G_g) and a variable x_i, the function $f_{x_i=g}$ defined by $h = ite(g, f_{x_i=1}, f_{x_i=0})$ (if g then $f_{x_i=1}$ else $f_{x_i=0}$) has to be represented. If G_h is the BDD that represents h, Bryant showed that $|G_h|$ has an upper bound of $O(|G_f|^2 * |G_g|)$. However, Bryant could not find any worst-case example requiring the above cubic blow-up. Thus, he observed that "([4],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 *ite* 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 $a_o, ..., a_{n-1}$ and $x_o, ..., x_{n-1}$ where $n = 2^k$. The a-variables are control variables describing a number $|a| \in \{0, ..., n-1\}$; the x-variables are data variables addressed by $\vec{a}$. 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:

$a_o, ..., a_{k-1}; b_o, ..., b_{k-1}; x_o, ..., x_{n-1}$, and s.

Let $f(a, b, s, x) = s \wedge MUX(a_o, ..., a_{k-1}, x_o, ..., x_{n-1}) + \bar{s} \wedge MUX(b_o, ..., b_{k-1}, x_o, ..., x_{n-1})$.

Moreover, we define g on $n + k$ variables $c_o, ..., c_{k-1}; x_o, ..., x_{n-1}$ by

$g(c, x) = MUX(c_o, ..., c_{k-1}, x_o, ..., x_{n-1})$.

Both functions are considered as functions on all $n + 3k + 1$ variables in a, b, c, s and x and we investigate for both functions the optimal variable ordering:

$s, a_o, ..., a_{k-1}; b_o, ..., b_{k-1}; c_o, ..., c_{k-1}; x_o, ..., x_{n-1}$.

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 OBDDs for these functions may share the x-nodes and the sinks. The ROBDD size of g is $2n + 1$.

Let $h = f_{s=g} = MUX(c, x) \wedge MUX(a, x) + \overline{MUX}(c, x) \wedge MUX(b, x)$.

By the characterization of the ROBDD size by Sieling and Wegener [20] 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, and c. If $|a| = i, |b| = j$, and $|c| = k$, we obtain the cofactor $x_k x_i + \bar{x}_k x_j$. 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_i. 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 OBDD 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,

$|G_f|^2|G_g| = 18n^3 + 21n^2 + 8n + 1$

and $h = f_{|s=g} = ite(g, f_{s=1}, f_{s=o}) = ite(MUX(c, x), MUX(a, x), MUX(b, x))$ has an ROBDD size of at least $2n^3 - 2n^2 + 2n + 1$. Hence,

$|G_h| = \Theta(|G_f|^2|G_g|)$.

4 Cubic-complexity of Composition and Partitioned-OBDDs

Note that on the right hand side of compose operations of Equation 1, both the disjuncts ($\overline{\psi_{i_{bdd}}} \wedge f_{d_{\overline{\psi_i}}}$ and $\psi_{i_{bdd}} \wedge f_{d_{\psi_i}}$) are mutually orthogonal, and can thus be represented as different partitions in a POBDD. In our method we successively compose the $\psi_{i_{bdd}}$s in f_d. If the graph size increases drastically for some composition (say ψ_j) we can abort the compose operation and instead separately compute each of the two disjuncts in the right hand side of equation 1; each of these disjuncts constitute a separate partition. In other words, we can create two orthogonal partitions instead of continuing with ROBDD composition to completion.

$$Partition1 : \overline{\psi_{i_{bdd}}} \wedge f_{d_{\overline{\psi_i}}}; \;\; Partition2 : \psi_{i_{bdd}} \wedge f_{d_{\psi_i}} \tag{2}$$

Note that each partition can be constructed in quadratic time using BDD *apply*. That is, the complexity of creating such partitions, during the composition of ψ_i in f_d, is only $O(|f_d| \cdot |\psi_{i_{bdd}}|)$. 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_d by $\psi_1, \ldots, \psi_k$, a composition blow-up is observed a total of c times, then we will have produced $c+1$ number of 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 $x_1, \ldots, x_k$ 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 $O(|G|^{2^k} * |g|^{2^k - 1})$. 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 $O(|G| * |g|)$. 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 $O(|G| * |g|^k)$. Thus, the composition complexity of creating POBDDs by our method is bounded by only $O(2^k * |G| * |g|^k)$. (In the above analysis, for simplicity, each decomposition 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 with 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 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 compose 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 folowing parameters: **Max Limit** M, **Min Limit** μ, **Partition Factor** N, **Acceptance Factor** γ, **Multiplication Factor** ϕ. 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 $g_i : i = 1 \dots k$ are the functions to be composed): (1) $|R| \geq M$, (2) $|R| \geq \phi \times (|G| + |g_i|) \wedge |R| \geq \mu$, (3) $|R| \geq \phi \times (|G| + \sum_i |g_i|) \wedge |R| \geq \mu$. Assuming that the partitioning process has produced two graphs P_1, P_2, we accept the partition when the following condition holds: $|P_1| \leq \gamma \times |R| \wedge |P_2| \leq \gamma \times |R|$

4.2.2 Function-Based Partitioning: Some Useful Properties

Although the conventional cube-based partitioning as discussed in [17, 6] 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 (partitions) may have a different variable at the root.

In contrast consider a Boolean function which is partitioned in 2^k partitions $P_1, \dots, P_{2^k}$ using cubes (partial assignments) generated from, say, k input variables $x_1, \dots, x_k$. In such a case, $x_1, \dots, x_k$ are effectively removed from the support set of ROBDDs of each partition. Thus, even when we order each partition separately, $x_1, \dots, x_k$ implicitly remain the root variable of ROBDDs of each partition. As discussed in [6, 3], such a BDD 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]: There exist functions which are compact when represented using POBDDs if different partitions can have different root variables but are exponentially sized for free 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 existed 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 same by applying it 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 512M RAM, and limited the BDD 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, page 10).

Note, that except multiplier C6288, most of the ISCAS-85 benchmark circuits are extremely easy for ROBDD representation. and have no very difficult composition case. Also, C6288 has been proven to be intractable for ROBDDs as well as POBDDs.

5 Combinational Verification

In this section we show how POBDDs fit into a typical Combinational Verification flow [21, 22, 23] 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, 22, 23]). The internal gates that are found to be equivalent are merged, and the given circuit pair (*composite* circuit) is greatly simplified. Many

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

Circuits	ROBDD	POBDDs	
		Σ partitions	# Partitions
Ckt1	222787	23487	2
Ckt2	fails	619634	2
Ckt3	fails	542977	2
Ckt4	fails	2.57M	3
Ckt5	fails	3.32M	4
Ckt6	fails	3.83M	2
Ckt7	fails	3.51M	2
Ckt8	fails	4.48M	2
Ckt9	fails	4.60M	2
Ckt10	fails	5.12M	6

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 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 (see page 12) 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 UltraSparc-IIs with 512M 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 BDDs during a bi-simulation relation based analysis.

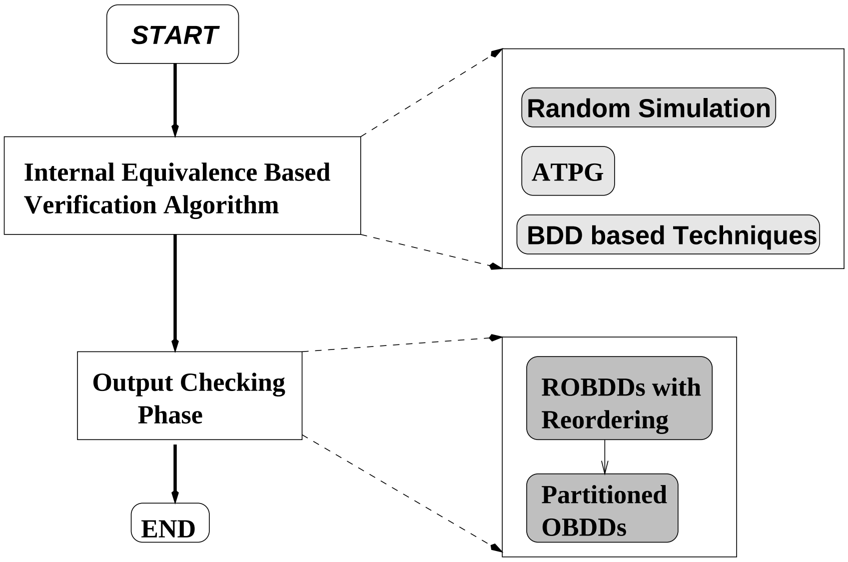

Figure 1: Algorithm Flow.

6.1 Sequential Circuit Equivalence: Notational Framework and Definitions

A sequential circuit consists of a n latches and a combinational logic block. It has m Primary Inputs(PIs) $\vec{x} =< x_1, x_2, \ldots, x_m >$ and l Primary Outputs(POs) $\vec{y} =< y_1, y_2, \ldots, y_l >$. It also has n present-state variables (PSVs) $\vec{q} =< q_1, q_2, \ldots, q_n >$ and n next-state variables (NSVs) $\vec{Q} =< Q_1, Q_2, \ldots, Q_n >$. Each PO is a combinational boolean function of the PIs and the PSVs, that is: $y_i = o_i(\vec{x}, \vec{q}),\ i = 1, 2, \ldots, l$. Each NSV is a combinational boolean function of the PIs and the PSVs: $Q_i = s_i(\vec{x}, \vec{q}),\ i = 1, 2, \ldots, n$. 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 [12, 19, 18, 9, 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 $C_a \Leftrightarrow C_b$ — 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 $\vec{q_a}$ and $\vec{q_b}$ 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 $\vec{q_a}$ and $\vec{q_b}$ 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 Appendix A, which computes the set of pairs $(\vec{q_a}, \vec{q_b})$ of states of C_a and C_b that are equivalent. The central operation of this algorithm is

Table 2: Combinational Verification Results for Some Industrial Test Cases

Circuits	Comb. Verf.–ROBDDs		Comb. Verf.–POBDDs	
	RunTime	Result	RunTime	Result
Ver0	fails	fails	2:26:48	EQUAL
Ver1	fails	fails	51:50	EQUAL
Ver2	fails	fails	1:41:59	EQUAL
Ver3	fails	fails	1:09:45	EQUAL
Ver4	fails	fails	1:10:29	EQUAL
Ver5	1:49	EQUAL	2:48	EQUAL
Ver6	fails	fails	19:35	EQUAL
Ver7	2:03	EQUAL	2:32	EQUAL
Ver8	fails	fails	2:30:20	NOT EQUAL

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 given in Appendix A.

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 A), the BDD 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 Results

In Table 3 we are showing some typical cases where partitioning was invoked during the fixed–point computation. The circuits are some of the harder ones in the 1993 Addendum to the ISCAS–89 benchmarks. During normal compositionwe 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 BDD manager and performing an explicit reordering operation. Our program is written in C++ and utilizes

Table 3: Partitioning Effect During Fixed–Point Algorithm

Circuits	#Latches	New nodes added before composition is aborted[a]	First Partition[b]	Second Partition
s1269	37	> 700000	82797	89454
s4863	104	> 319130	8021	8021
s3271	116	> 331632	102563	102172
prolog	136	> 3M	868634	702299
s3330	132	> 11.8M	1030656	1409214
tc100	40	> 1.2M	337106	405707

[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, likely, at least 5 times larger than the size limit, as calculated in terms of number of nodes, at which we abandon the compose.

[b]This number represents the size of each partition in BDD nodes.

CUDD-2.3.0. The experiments were run on Sun UltraSparc-IIs with 512M of RAM.

7 Conclusion

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 algorithm when partitions are created at the point of composition blow up. We also showed that the sizes of the resulting POBDDs can be exponentially smaller than ROBDDs. Our POBDD algorithm allows multiple root variables, and determines the number of partitions dynamically. Thus, it is more compact as compared to Free-BDDs or (cube-based) POBDDs which were proposed in [17]. We also showed that our techniques are robust and efficient and thereby answer a conjecture in [3] whether POBDDs with above characteristics can be easily realized. Specifically, we demonstrated the utility of the proposed procedures in some well known problems of Boolean function representation, combinational verification and bi-simulation relation based finite state machine analysis. In our experiments we have found even orders of magnitude improvement over ROBDDs for several difficult functions. Since the notion of partitioning is general and can be applied to any BDD representation, our results should be applicable to any decision diagram scheme which allows function composition.

References

[1] J. Burch, E. Clarke, K. McMillan, and D. Dill. “Sequential Circuit Verification Using Symbolic Model Checking”. *Proc. 27th DAC*, pages 46–51, 1990.

[2] J. Gergov and C. Meinel. "Efficient Boolean Manipulation with OBDDs can be extended to FBDDs". *IEEE Trans. on Computers 43*, pages 1197–1209,1994.

[3] B. Bollig, I. Wegener. "Partitioned BDDs vs. Other BDD Models" *Proc. IWLS*, 1997.

[4] R. E. Bryant. "Graph-based Algorithms for Boolean Function Manipulation". *IEEE Transactions on Computers*, C-35:677–691, August 1986.

[5] R. E. Bryant. "Symbolic boolean manipulation with ordered binary decision diagrams". *ACM Computing Surveys*, 24:293–318, September 1992 .

[6] A. Narayan, A. Isles, J. Jain, R. Brayton, A. Sangiovanni-Vincentelli. "Reachability Analysis Using Partitioned-ROBDDs" *Proc. ICCAD*, pages 388–393, 1997.

[7] G. Cabodi, P. Camurati, and S. Quer. "Efficient State Space Pruning in Symbolic Backward Traversal". *Proc. ICCD*, pages 230–235, 1994.

[8] G. Cabodi, S. Quer, and P. Camurati. "Extending Equivalence Class Computation to Large FSMs". *Proc. ICCD*, pages 258–263, 1995.

[9] O. Coudert, C. Berthet, and J. Madre. " Verification of Sequential Machines Using Boolean Functional Vectors". *Formal VLSI Correctness Verification. Elsevier Sc.*, pages 179–196, 1990.

[10] R. Drechsler *et. al.* Efficient representation and manipulation of switching functions based on Ordered Kronecker Functional Decision Diagrams. *DAC*, 1994.

[11] Y.V. Hoskote. "Formal Techniques for Verification of Synchronous Sequential Circuits". *Ph.D. Dissertation, ECE Dept., UT Austin*, 1995.

[12] S.-Y. Huang, K.-T Cheng, K.-C. Chen, and U. Glaeser. "An ATPG-based Framework for Verifying Sequential Equivalence". *Proc. ITC*, pages 865–874, 1996.

[13] J. Jain, J. Bitner, D. S. Fussell, and J. A. Abraham. "Functional partitioning for verification and related problems". *Brown/MIT VLSI Conference*, March 1992.

[14] J. Jain, A. Narayan, C. Coelho, S. Khatri, A. Sangiovanni-Vincentelli, R. Brayton, and M. Fujita. "Decomposition Techniques for Efficient ROBDD Construction". In *Formal Methods in CAD 96*, LNCS. Springer-Verlag, 1996.

[15] Z. Kohavi. *"Switching and Finite Automata Theory"*. McGraw-Hill Book Company, 1978.

[16] A. Narayan, S.P. Khatri, J. Jain, M. Fujita, R. K. Brayton, and A. Sangiovanni-Vincentelli. "A Study of Composition Schemes for Mixed Apply–Compose based Construction of ROBDDs". *Proc. VLSI Design*, pages 249–253, 1996.

[17] A. Narayan, J. Jain, M. Fujita, and A. L. Sangiovanni-Vincentelli. "Partitioned-ROBDDs - A Compact, Canonical and Efficiently Manipulable Representation for Boolean Functions". *ICCAD*, November 1996.

[18] C. Pixley. "A Theory and Implementation of Sequential Hardware Equivalence". *IEEE Trans. on CAD*, pages 1469–1494, 1992.

[19] C. Pixley, V. Singhal, A. Aziz, and R. K. Brayton. "Multi–level Synthesis for Safe Replaceability". *Proc. ICCAD*, pages 442–449, 1994.

[20] D. Sieling and I. Wegener "NC-Algorithms for Operations on Binary Decision Diagrams". *Parallel Processing Letters, 3:3-12, 1993.*

[21] A. Kuehlmann, F. Krohm. "Equivalence Checking Using Cuts and Heaps". *Proc. DAC, 1997.*

[22] W. Kunz, P. Menon. "Muli–level Logic Optimization by Implication Analysis". *Proc. ICCAD, 1994*

[23] R. Mukherjee, J. Jain, K. Takayama, M. Fujita, J. Abraham, D. Fussell. "An Efficient Filter–based Approach for Combinational Verification". *Proc. DATE, pages 132–137, 1999*

Appendix A: Sequential Circuit Equivalence Algorithm

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

7.1 Basic Algorithm Implementation

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 $\vec{x}$, when they are found at $\vec{q}_a$ and $\vec{q}_b$, respectively) will be represented by the predicate $E_1(\vec{q}_a, \vec{q}_b)$.

$$E_1(\vec{q}_a, \vec{q}_b) = \bigwedge_{i=1}^{l} (\forall \vec{x} :\ o_i^a(\vec{x}, \vec{q}_a) \odot o_i^b(\vec{x}, \vec{q}_b)) \tag{3}$$

where $\odot$ is the boolean equivalence (XNOR) operator.

Now, given the states $(\vec{q}_a, \vec{q}_b)$ their successor states for some input $\vec{x}$ are the states $(\vec{Q}_a, \vec{Q}_b)$ where $\vec{Q}_a = \mathbf{s}_a(\vec{x}, \vec{q}_a)$ and $\vec{Q}_b = \mathbf{s}_b(\vec{x}, \vec{q}_b)$. But then the pairs of states $R_i(\vec{q}_a, \vec{q}_b)$ which have successor states not among the i-equivalent states $E_i(\vec{q}_a, \vec{q}_b)$ are the pairs of states given by the following:

$$R_i(\vec{q}_a, \vec{q}_b) = \exists \vec{x}, \vec{Q}_a, \vec{Q}_b :\ \vec{Q}_a \equiv \mathbf{s}_a(\vec{x}, \vec{q}_a) \bigwedge \vec{Q}_b \equiv \mathbf{s}_b(\vec{x}, \vec{q}_b) \bigwedge \neg E_i(\vec{Q}_a, \vec{Q}_b)$$

or if we use composition: $R_i(\vec{q}_a, \vec{q}_b) = \exists \vec{x} : \neg E_i(\mathbf{s}_a(\vec{x}, \vec{q}_a), \mathbf{s}_b(\vec{x}, \vec{q}_b))$

The predicate $\mathcal{E}(\vec{q}_a, \vec{q}_b)$ is now given by:

$$E_{i+1}(\vec{q}_a, \vec{q}_b) = \ E_i(\vec{q}_a, \vec{q}_b) \bigwedge \neg R_i(\vec{q}_a, \vec{q}_b)$$

So we have:

$$E_{i+1}(\vec{q}_a, \vec{q}_b) = E_i(\vec{q}_a, \vec{q}_b) \bigwedge \forall \vec{x} : E_i(\mathbf{s}_a(\vec{x}, \vec{q}_a), \mathbf{s}_b(\vec{x}, \vec{q}_b)) \tag{4}$$

This is a fixed–point algorithm that terminates because the state space is finite. So we get the set $\mathcal{E}(\vec{q}_a, \vec{q}_b)$ of equivalent states when the fixed point $E_k = E_{k-1}$ 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, $C_a \Leftrightarrow C_b$ iff

$$(\forall \vec{q}_a,\ \exists \vec{q}_b : \mathcal{E}(\vec{q}_a, \vec{q}_b)) \bigwedge (\forall \vec{q}_b,\ \exists \vec{q}_a : \mathcal{E}(\vec{q}_a, \vec{q}_b)) \tag{5}$$