

Tight Integration of Combinational Verification Methods

Jerry R. Burch

Vigyan Singhal

Cadence Berkeley Labs
2001 Addison St, 3rd Floor
Berkeley, CA 94704

Abstract

Combinational verification is an important piece of most equivalence checking tools. In the recent past, many combinational verification algorithms have appeared in the literature. Previous results show that these algorithms are able to exploit circuit similarity to successfully verify large designs. However, none of these strategies seems to work when the two input designs are not equivalent. We present our combinational verification algorithm, with evidence, that is designed to be robust for both the positive and the negative problem instances. We also show that a tight integration of different verification techniques, as opposed to a coarse integration of different algorithm, is more effective at solving hard instances.

1 Introduction

In this paper we study the problem of combinational verification. Given two combinational netlists, which have the same set of inputs and outputs, the problem of combinational verification is to determine if for every possible input combination, each pair of outputs (in the two netlists) evaluates the same boolean value. It is well-known that this problem is coNP-hard. Nevertheless, in practice, the situation is not so hopeless. For most real world equivalence checking problems, the design methodology which is responsible for the derivation of one design from the other virtually assures that, besides the output pairs, many internal net pairs also should be equivalent. Indeed, this was the idea that revolutionized the scope of combinational verification. Berman and Trevillyan [1] presented the basic approach which uses internal equivalent points, which we will refer to as *cutpoints* after [6]. The classic problem in a cutpoint based approach is that of false negatives. Indeed, most of the subsequent work in cutpoint-based methods [2, 9, 6, 14] has been to effectively deal with this problem.

The early papers on practical combinational equivalence checking [2, 7, 9] were all based on single methods and did not have multiple strategies to deal with different classes of circuits. Brand's method used ATPG methods with cutpoints, Kunz *et al.* used recursive learning and Matsunaga used BDDs. Of course, random simulation has been used for verification for a long time. The observations that different methods were strong on different classes of circuits prompted approaches using filters [6, 10]. Here the authors propose a set of combinational verification engines as filters through which the problem instances pass. This approach works well on a large class of circuits; however, if an unsuitable method is applied before the "right" method for the problem, it can be very expensive. The next advancement was to dovetail between these methods so that the problem instance passes through the methods [10] with increasing thresholds. The motivation is that the dovetailing avoids committing to using a method on an instance for which the method is not suited. However, this approach still has limitations which can be understood by comparing the performance of this approach with an imaginary tool which uses an oracle; when presented with an equivalence check problem, this oracle magically picks the best method to solve this problem. One significant contribution of

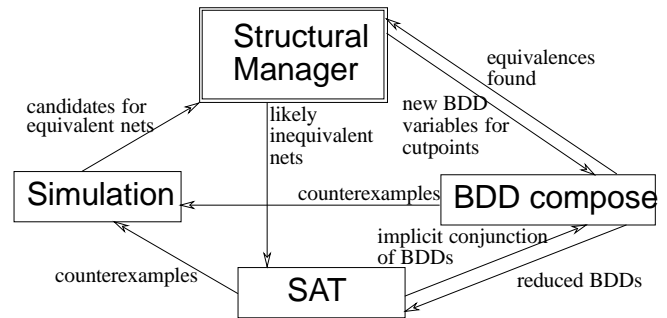


Figure 1: Tight integration between various methods

our approach is that we invoke multiple methods in the inner loop of the algorithm and the information discovered by one method is used by another method. This gives our algorithm the ability to beat the oracle-based algorithm.

We have (1) a BDD-based composition engine (as in [9, 6, 14]), (2) a SAT-based search engine (which could be similar in performance to the ATPG search engine of [2], although we have found it more efficient to solve SAT directly on an implicit conjunction of BDDs), and (3) a simulation engine which simulates interesting vectors when we find them (similar to that in [14]). The key new contribution is a tight integration of these methods so that information found by one method is used by the other methods in showing equivalences using cutpoints (see Figure 1), and in this sense our work is different from that in [10], where the algorithm in its inner-loop just dovetails between different methods, and its performance is bounded by the oracle described above. Previous work has used parts of the tight integration we have, but no one used a tight integration between SAT-based methods, BDD-based composition methods and random simulation. And it is this tight integration which allows our algorithm to be robust. Some examples of our tight integration are: when we decide to switch from BDDs to the SAT algorithm, the SAT algorithm works directly on the BDDs; if the SAT algorithm finds that two internal nets can be equal only if some other nets are constants, this reduces the BDDs that the BDD composition algorithm uses; the counterexamples generated from both the SAT algorithm and the BDD composition algorithm are used for simulation to reduce future candidates for equivalent nets. The structural manager is responsible for creating cutpoints and managing the overall verification problem by making calls to the various methods. We defer more details for later sections of this paper. Through our experiments, we show data to illustrate the importance of a robust checker in the inner loop.

An added advantage of a robust checker is that we have the ability to find "hard" design bugs: we call those bugs "hard" which are unlikely be found by using either random simulation or ATPG directly on the outputs. For example, Kuehlmann and Krohm [6] suggest that since their BDD-based approach does not work well for falsification; it is most effective in conjunction with an engine that is

specifically designed for falsification, namely random simulation. If the equivalence checker fails to solve the problem, random simulation is used, or vice-versa. However, it should be no surprise that random simulation is inadequate to expose those bugs which can be exposed by only looking at tiny fractions of input space. An ATPG-based verification algorithm such as [2] will also have a difficult time with hard bugs because the expected internal net equivalences will mostly disappear in the region of the circuit which lies between the bug location and the output; since Brand relies on finding internal net equivalences, the task can become arbitrarily hard depending on the complexity of the logic in the fanout of the bug.

We have encountered many falsification instances which were not possible to resolve using random simulation. In addition, it can be argued that as combinational equivalence checkers become more mainstream and a standard part of design methodology, designers will use these tools early on in the design process, and this will give rise to an increasingly larger fraction of negative instances. This increases the importance of having a tool which is robust on both positive and negative instances.

Next we present a detailed discussion of previous work and how our approach advances the state-of-the art. This is followed by a discussion of our verification methods in Section 3. Finally, we present experimental data in Section 4.

2 Previous work and new ideas

The central idea behind decomposing the problem into smaller problems by using cutpoints is that if $f_1(x) = g_1(x)$ for all values of the input vector x , and if $f_2(y, z) = g_2(y, z)$ for all values of y and z , then $f_2(f_1(x), z) = g_2(g_1(x), z)$ for all values of x and z . However, the converse is not true, namely, if $f_2(y, z) \neq g_2(y, z)$, we cannot say whether $f_2(f_1(x), z)$ equals $g_2(g_1(x), z)$ or not. This is termed as the *false negative problem*, first introduced in [1]. There can be two possible resolutions of this problem: either f_2 and g_2 are not equivalent (i.e., we have a real negative), or f_2 and g_2 are equivalent (i.e., we have a false negative). In this paper, we will call this resolution step as *cutpoint resolution*.

In a BDD-based approach, cutpoints are introduced, and intermediate functions are represented as $f(X, Y)$ where X is the set of variables denoting primary inputs and Y is the set of variables denoting the cutpoints. Each $y_i \in Y$ is associated with a BDD representing its characteristic function $h_i(X, \{y_1, \dots, y_{i-1}\})$ (notice that the y_i 's are ordered from primary inputs towards the outputs; y_j cannot depend on y_k if $k \geq j$). If two BDDs $f(X, Y)$ and $g(X, Y)$ are not equal, we have a cutpoint resolution problem. The resolution can be obtained by starting the BDD for $[f(X, Y) \neq g(X, Y)]$ and then iteratively composing the cutpoints until the resulting BDD is independent of y_i 's. A composition is achieved by a standard BDD operation, `bdd-compose` [3]; composing y_i into $s(X, Y)$ represents obtaining $\exists y_i : s(X, Y) \wedge (y_i \equiv h_i(X, \{y_1, \dots, y_{i-1}\}))$. If after all compositions, the final BDD represents 0, we have a false negative, else we have an assignment of values to inputs that denotes a real negative. Many heuristics become important in such a compositions based method. The selection of cutpoints is important: van Eijk [14] presents heuristics to select cutpoints so that the cutpoint resolution problem does not occur often or is not too difficult; the intuition is that a node is a good cutpoint if its number of fanouts is large or if it is evenly spaced between other cutpoints. Another important heuristic is to determine which cutpoint to compose next; Matsunaga [9] gives heuristics which determine the order of composition.

The process of successive composition does not work well if the two candidate nets were actually unequal (the real negative case); this is so because it will not be possible to determine this until sufficiently many cut-points have been composed in so that the primary input variables start appearing in the current BDD. One can always

threshold such a composition scheme by a maximum BDD size, but besides being prone to missing a real cut point, such a thresholding may add up to a lot of wastage of time if many cutpoint resolutions turn out to be real negatives. Fortunately, such a situation happened rarely for Matsunaga's experiments; this can be explained by the fact that for all his examples, the designs were equivalent and his data-set was the set of MCNC circuits which are relatively small. In our experience, many resolutions yield real negatives, especially when the designs are not equivalent, and it is important for the equivalence checker to be equally robust for the real negative case.

Kuehlmann's method [6] presents a very graceful tradeoff between memory used and the ability to do cutpoint resolutions, by working on many cutpoint resolutions at the same time, and jumping between these sub-problems depending on the BDD size. However, again, if the circuits are not equal, this algorithm cannot finish unless it is possible to build the BDDs of the outputs in terms of the inputs, which is not possible for many large circuits.

Another approach using cut-points is that by Brand [2]. This algorithm is based on ATPG techniques. To decide the equality of two functions, it is determined if there exists a test pattern that can test the presence of a stuck-at-0 fault at the XOR of the two functions¹. This is applied at intermediate nets of the two circuits, and if a cut-point is found, the function from one circuit is substituted at the place of the cut-point in the other circuit. In general, solving combinational equivalence for the entire circuits using ATPG (or general SAT solutions) for the XOR-gates only at the primary outputs is a bad idea because the only sharing of information between the circuits is at the primary inputs. However, the substitution of nets in Brand's method brings this sharing of information closer to the primary outputs. In spite of all this, in our experience, using ATPG-based methods is not as effective as using BDD-based methods, when the answer is that there is no test. This seems to be similar to the experience of [6] where they suggest that they use Brand's ATPG-based algorithm only when their standard BDD-based algorithm fails. Because Brand's algorithm is not as robust, it has to threshold the ATPG check with a time bound. In addition, if internal pairs of nets are found to be inequivalent, this information is not used.

In our experience, a general statement that can be made is that, for the cut-point resolution of a candidate pair of nets, if the pair of nets is actually equivalent, BDD-based composition methods (such as Matsunaga's) are effective. The intuition is that if the two points are equivalent, after all the composition, the BDD for the XOR of the two pair of nets will turn to be equal to 0, so there is a hope that if the composition order is intelligently chosen, then none of the intermediate BDDs will blow up. On the other hand, if the pair of nets is not equivalent, search-based methods (such as Brand's ATPG solution) are more effective because they are good at searching for one solution, when we know that at least one exists. BDD-based solutions represent all solutions even though we are interested in only one.

Of course, to begin with, there is no way to know in advance whether or not the candidate pair of nets will turn out to be equivalent or not. However, we can often make good guesses, and use these to selectively balance each cutpoint resolution step inside the algorithm between these two different kind of methods: composition and search. Achieving a tight integration between the various approaches by passing information learned in one method to another is extremely useful.

One key idea in our algorithm is the dual of the original idea of Berman and Trevillyan. Let the two functions be $f_2(f_1(x), z)$ and $g_2(g_1(x), y)$, as before. As in [2] and [9], our original candidate

¹In fact, Brand actually tests if this stuck-at-fault can be observed at the primary output, but we ignore this generalization, since that is not the focus of this paper.

pairs for cutpoints come from simulations: if two nets have the same signature, they are a candidate pair, and we can partition all nets into equivalence classes based on their signature. One can say that these candidate pairs from simulation represent false positives for the combinational verification problem; if two nets are in different classes they must not be equivalent, but not vice-versa. For bugs that cannot be easily identified with simulation, the output pair lies in the same equivalence class. Suppose there is hard bug in circuit, so that f_2 and g_2 have the same signature, and f_1 and g_1 have the same signature, even though due to the bug, f_1 and g_1 are functionally not equivalent. We first have to resolve the cutpoint candidate pair corresponding to f_1 and g_1 . Once we do this, at a later point we still have to resolve the false positive comprising of the pair f_2 and g_2 . Notice that this next resolution might be a much tougher problem because the total input cones of f_2 and g_2 might be much larger than the input cones of f_1 and g_1 . Our approach is that once we show that a candidate pair is not equivalent, we use the test pattern obtained to show this inequivalence to resimulate many test vectors which refine many other equivalence classes. In fact, many instances of inequivalence are detected immediately after the resolution of an intermediate net pair which is much closer to the primary inputs than the outputs themselves. This iterative simulation, in effect, helps magnify those sub-spaces in the large input space which are represent meaningful functions for the two circuits. This proves to be much more effective than the initial random simulation. van Eijk [14] has also proposed using such directed simulation. However, his cutpoint resolution only uses BDD composition, and as we will show later, BDD composition is not often well-suited for determining negatives. Many of our real negatives come from a novel SAT-based search which operates directly on the BDDs.

Our cutpoint resolution step is robust. We do not spend any less resources on resolving a cutpoint than resolving the output pair. We will show that, if we instead had an algorithm, which aborts cutpoint resolution of internal nets due to thresholds (e.g., [2, 9, 14]), that may be a *little* faster on some instances but can be *much* slower or may not even be able to solve some other instances designs.

There is another class of combinational verification algorithms, which we have not talked about. These are learning-based algorithms, as in [5] and [7] and resynthesis-based algorithm [11]. However, all available data, from these papers and from [9, 14] and our own experiments, shown later in this paper, suggest that these techniques by themselves do not yield as effective solutions as cut-point based methods. It is possible that they could also be tightly integrated into our equivalence checker, and give us additional benefit.

3 Our Combinational Verification Engine

Our implementation is based on using a fine-grained combination of BDD compose algorithm, search algorithms and test pattern simulation to provide the balance between positive and negative verification. By providing the balance between the positive and negative verification methods at a fine granularity, we can effectively identify hard bugs too.

In this section we present the details of the algorithms in the combinational verification core engine. First we discuss our cutpoint resolution algorithm since that is the key to our checker. We explain the multiple methods used in this cutpoint resolution. Then we outline the global algorithm.

3.1 Robust cutpoint resolution

The key component of our equivalence checking engine is a robust and efficient cutpoint resolution engine. The main difference in our cutpoint resolution from previous algorithms is that we invest all our resources to successfully solve each such sub-problem; each such resolution sub-problem is no less important than the resolution for

the primary output pair. This is in contrast with all previous work which invest a limited amount of resources for these sub-problems and abort these resolutions if the limit is exceeded, saving the resources for the possible false resolution of the primary outputs. This would be the only possible strategy if the cutpoint resolution engine is not robust for both positive and negative cases. As we had argued earlier in Section 2, BDD-based composition methods, as in [9, 14], are powerful if the resolution yields a false negative; however, if we have a real negative, all cutpoints have to be composed which can be expensive. On the other hand, ATPG or SAT-based methods and random simulation, as in [2], are good for showing real negatives, but are not as effective as BDD composition methods when we have a false negative.

Fortunately, we have access to both kinds of methods. We have a BDD composition algorithm, and we balance it with a randomized local search algorithm which is efficient in finding real negatives. This balance allows us to be robust, thus avoiding placing a time bound for false resolutions of intermediate nets to minimize wasted work. Also, we can now claim to have an overall robust algorithm, because before showing the output pair equivalence, we have solved *all* possible cutpoint resolution problems. This would be more robust than an algorithm which may be efficient for most positive verification instances, but may not be able to solve an inequivalent pair of designs because it gave up on the resolution of intermediate nets. Also, it is easy to see that giving up on intermediate net resolutions can be costly even for positive verification instances if the aborted resolution was going to yield a false negative.

The initial set of candidates cutpoint resolution comes from random simulation on the two input designs. Any two nets which have the same signature become candidates. However, in future, if a cutpoint resolution results in a real negative, we use the witness for this negative to do further simulation. This ensures that all nets get interesting test vectors: for example, eventually, unless a net is functionally equivalent to a constant 0 or 1, it will have a signature different from all 0's or all 1's. As we discussed earlier, this iterative simulation is very valuable in preventing future cutpoint resolutions due to real negatives, as well as in finding hard bugs at the outputs.

In the next two subsections we describe the two cutpoint resolution schemes, one based on BDD composition and the other, a new randomized local search algorithm.

3.1.1 BDD composition algorithm

As discussed earlier in Section 2, the BDD composition procedure starts with a root BDD, which represents $[f(X, Y) \neq g(X, Y)]$. Then we start composing BDD variables which represent the cutpoints. If we obtain the BDD which represents 0, we know that we have a false negative, i.e., f is equivalent to g ; otherwise, if we have a BDD which is a function of only input variables and does not depend on any cutpoint variables, any minterm of this BDD represents a witness which shows why f is not equivalent to g . This is the standard termination criterion, as described in [9, 6, 10]. One new and very useful improvement that we have found is to stop if there is a path containing only input variables from the root node of the BDD to the BDD-leaf representing 1. This is an inexpensive check. If this happens, we know that even when we compose all the cutpoint variables that the BDD depends on, the resulting BDD will not represent 0. For example, consider the BDD in Figure 2. The path $\{x_7 = 1, x_1 = 1\}$ to the 1-vertex does not depend to the cutpoint variable y_3 , and hence, this function will evaluate to 1 for this assignment even when we compose the BDD for the cutpoint variable y_3 . We have found this little heuristic to be extremely valuable.

When composing in cutpoint variables, the order of the variables to compose in is very important. Some researchers have found that composing variables in the order of their distance from the potential new cutpoint is best because it guarantees that we do not compose any particular variable more than once [6]. Another heuristic

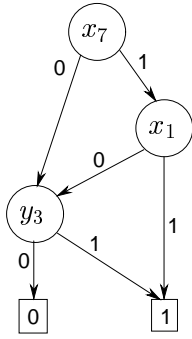


Figure 2: BDD with input variables $\{x_1, x_7\}$ and cutpoint variable $\{y_3\}$.

is to compose a *set* of variables so that there is a large sharing of the newly introduced cutpoint variables in the structural support of the set of selected variables [9]. Our heuristic is different from these two; it is driven by a cost function on the BDD size. This may mean that we often compose a cutpoint variable more than once, but empirically we have found that our heuristic performs better than if we were to compose variables in the order of their distance to the potential cutpoint. In contrast to Matsunaga, we only compose one variable at a time.

The BDD composition algorithm has a BDD size limit and we stop working on BDD composition if this limit is exceeded.

3.1.2 Randomized local search on BDDs

As we discussed earlier, the BDD-based method may not be well suited for the case when the cutpoint resolution will yield a real negative. Instead, we balance the BDD-based composition approach with a new randomized local search algorithm. This algorithm takes a root BDD in terms of variables representing primary inputs and other variables representing cutpoints, and BDDs for each one of the cutpoints and returns a satisfying assignment so that all the BDDs are satisfied. The root BDD represents $s(X, Y)$ and each $y_i \in Y$ is associated with a BDD representing $h_i(X, \{y_1, \dots, y_{i-1}\})$. A satisfying assignment associates a boolean value for each variable in $X \cup Y$ such that $s(X, Y)$ evaluates to 1, and so does the characteristic function BDD for each of the m cutpoints: $\{y_i \equiv h_i(X, \{y_1, \dots, y_{i-1}\})\}$. In order to do a cutpoint resolution between $f(X, Y)$ and $g(X, Y)$, we invoke the search algorithm with $s(X, Y)$ set to $\{f(X, Y) \neq g(X, Y)\}$. For example, Figure 3 shows the root BDD $s(X, Y)$ and 4 BDDs for the characteristic functions; the edges between the BDDs represent the dependency of a BDD on a cutpoint, e.g., the BDD for cutpoint y_3 depends on the cutpoints y_1 and y_2 , in addition to the primary input variables X .

The randomized search algorithm is inspired by the WALK-SAT algorithm for the satisfiability problem described by Selman *et al.* [12]. Given a SAT problem as a conjunction of clauses, this algorithm finds a satisfying assignment which satisfies each one of a set of input clauses.

We begin with some pre-processing on the $(m + 1)$ BDDs. The preprocessing consists of finding constants in the BDDs and simplifying the BDDs with respect to the constants. For a variable $z \in X \cup Y$, a constant $\{z = 0\}$ (or $\{z = 1\}$) can be learned if, for any one of the $(m + 1)$ BDDs, all paths from the root to the BDD-leaf representing 1 pass through the $\{z = 0\}$ (or $\{z = 1\}$) branch. This can be by with a linear time traversal over the BDD. It is easily seen that when we discover a constant $\{z = 0\}$, every satisfying assignment to our SAT problem must have $\{z = 0\}$. When we find a constant we simplify all BDDs with respect to this. Finding constants through one BDD frequently leads to constants in other BDDs. In

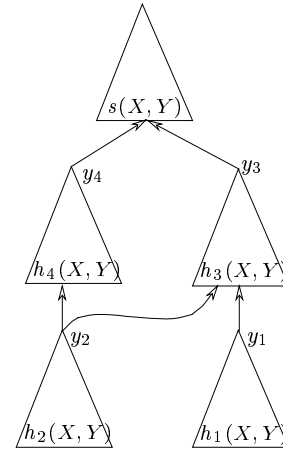


Figure 3: A set of BDDs for the search algorithms

addition, if a variable z occurs only in one BDD $t(X, Y)$, we can replace the BDD by $\exists z : t(X, Y)$; this is because we can choose any value for this variable z without affecting the truth of the other BDDs. Sometimes, this preprocessing turns one of the BDDs into a constant 0 BDD. In that case, we know that the problem is unsatisfiable, and we have solved the cutpoint resolution. More often, the preprocessing just simplifies the BDDs, effectively simplifying the problem.

We start with a random assignment to the variables in $X \cup Y$. Each assignment of variables naturally identifies a path from the root of each BDD to a leaf-vertex; if the leaf-vertex is BDD-leaf 0, we say that the BDD is not satisfied, otherwise the BDD is satisfied. Then at each iteration, with probability p , we take a greedy step towards satisfying the $(p + 1)$ BDDs; with probability $(1 - p)$, we take a random step which helps us escape local minimas which a purely greedy algorithm might get stuck in. The greedy step flips the assignment of one variable so that the cost function is minimized. The naive cost function is the summation over all BDDs of the distance of the current assignment path to a path to the BDD-leaf representing 1, i.e., a satisfying path. This distance is measured by the minimum numbers of variables that need to be flipped in order to satisfy the BDD, and represents how close the current assignment is to an assignment which solves the problem. The random step picks one of the BDDs which is not being satisfied and makes the fewest changes in the current assignment so that this BDD is satisfied.

The trickiest part of this algorithm was discovering the cost function. Our original cost function was the number of the BDDs that are not satisfied by the current assignment. We found this cost function to be too coarse and it would get stuck in plateaus too often (getting trapped in a search space of a plateau is a well-known phenomenon in the SAT community [4]). Then we refined this cost function to the one based on distances to satisfying paths in the BDDs, described above. Unfortunately, this does not work either. The reason is that any input assignment can satisfy all BDDs but one, but can be arbitrarily far from a solution. It is always possible to satisfy each of the m BDDs that come from cutpoints by choosing *any* input assignments, and picking appropriate values for y_i 's starting from the inputs to the outputs. This arbitrary assignment rarely satisfies the SAT problem because the root BDD $s(X, Y)$ rarely evaluates to 1 for an *arbitrary* assignment. We then modified the cost function so that a BDD contributes a maximum value to the cost if some BDD in its transitive fanout is not satisfied; the maximum value a BDD can contribute is the number of variables in its support. In addition, we also modified the cost function so that if a cutpoint variable does not appear on the current assignment path of any BDD, and if all the

BDDs in its transitive fanout are satisfied, this BDD contributes 0 to the cost. For example, in Figure 3, if variable y_1 does not appear on the current assignment path in ($y_3 \equiv h_3(X, Y)$), and both BDDs ($y_3 \equiv h_3(X, Y)$) and $s(X, Y)$ are satisfied, we do not need to satisfy the BDD ($y_1 \equiv h_1(X, Y)$); this is simply because the value of the variable y_1 does not affect the truth of the problem.

The search algorithm terminates after making a pre-determined number of moves, or when the cost becomes 0, whichever happens first. We bound the number of moves spent in this procedure by a number proportional to the sum of the sizes and the number of input BDDs.

It is possible that any ATPG-based algorithm could be substituted in place of our SAT algorithm. However, the biggest advantage of our algorithm that it directly operates on BDDs (as opposed to clauses or the netlist), and thus can closely couple with the BDD composition algorithm, by passing back and forth successively simpler problems (the implicit conjunctions of BDDs). We have found our implementation to be very efficient for most instances when there is a satisfying assignment. Of course the search algorithm is most often successful only when the answer is that there is a satisfying assignment; so we invoke this search algorithm *first* (as opposed to the BDD compose algorithm) when we strongly suspect that the cutpoint resolution will result in a real negative. A heuristic indication of this is when the simulation signature is all-0 or all-1, and when the two nets to be compared have different input supports.

If the SAT algorithm terminates without finding a solution, the simplified BDDs are passed back to the BDD composition algorithm.

3.2 Interleaved simulation with cutpoint resolution

Whenever a cutpoint resolution returns a real negative, we run directed simulation on a primary input vector which shows the real negative. The effect of this step is to refine the future candidates for cutpoints by simulating more interesting test vectors on the circuits. When the real negative comes from the BDD composition algorithm, we use different branches of the BDD which represents all input minterms that shows the real negative; if the negative comes from the SAT algorithm, the input cube which shows the false negative is used. The idea of interleaving simulation with cutpoint resolution has also been discovered by van Eijk [14]; however, since he does not have access to a method which is good at finding negatives, he has to abort his cutpoint resolution using a threshold, and will in general miss real negatives which are easy to detect using SAT or ATPG. We show in Section 4 why integrating a SAT engine gives us more robustness.

3.3 Overall algorithm

The overall algorithm is very straightforward:

```
GenerateInitialCutpointCandidates
foreach class of candidates,
    sorted from inputs to the output pair {
    ResolveCutpoints
    if (TrueNegative) {
        RefineCutpointClasses
        if outputs in different classes
            return(Unequal, test pattern)
    } else {
        Create new cutpoint
        if outputs in this class
            return(Equal)
    }
}
```

The `GenerateInitialCutpointCandidates` call runs initial random simulation and generates classes of cutpoint candidates. Two nets belong to the same class if they have the same signature.

The procedure `ResolveCutpoints` is the key procedure implemented by the integration between BDD composition and the SAT algorithm. Initially BDDs are built for each net in the class; these BDDs are in terms of variables representing previously discovered cutpoints. If structural analysis suggests that the nets are unlikely to be equivalent (e.g., if the signature of this class is all-0 or all-1, or if the structural support of two nets in terms of the circuit inputs is different), the SAT algorithm is invoked, else the BDD composition algorithm is invoked. If the algorithm that is invoked first does not successfully resolve the cutpoint, the other algorithm is invoked. We iterate between the two algorithms. The important point is that, as we discussed in Section 3.1.1 and 3.1.2, information discovered in each of these methods is passed back to the other method. In practice, this gives a robust algorithm for cutpoint resolution. As we discussed earlier in Section 2, previous papers do not multiple methods in the inner loop, nor do they pass information between various methods. Thus, they must exclusively rely on putting a time constraint on the cutpoint resolution.

If `ResolveCutpoints` returns a true negative, then `RefineCutpointClasses` is invoked. Here, the witness for the true negative (i.e., an input test pattern that shows the difference) is used to run simulation again and to refine the classes of cutpoint candidates. Often, if the circuits are not equivalent, this is shown during this simulation step from a cutpoint resolution of an internal net pair. This should be a cheaper test than actually showing that the output pair itself is inequivalent.

4 Experimental Results

For the experiments reported here, we used the public domain BDD package by David Long [8]. We ran our experiments on an Ultra-Sparc-1 with 256 MBytes of memory.

First we report results on verifying MCNC circuits. We verified the original circuit against their optimized version; the latter is obtained by using `script.rugged` from SIS [13] followed by a technology mapping step to the MCNC library of gates. The MCNC circuits are the only designs we can use to compare against published literature. The results appear in Table 1. The gate count is a count of simple gates (AND, OR and NOT gates) in the representation obtained after parsing the circuits. This table is not intended to be an accurate and fair comparison with previous work for at least two reasons: firstly, we do not calibrate our measurements to the hardware environment used by others, and second, we verify a generic logic optimization script whereas others verify redundancy removal. The only objective of Table 1 is to show that even with using the tight integration that is necessary for the hard instances, our algorithm performs similarly as the others for these simple MCNC problem instances. In addition to being small instances, these are all cases instances of positive verification. The tight integration methods described in our paper are really not needed for any of these toy examples.

Table 2 illustrates some of the arguments discussed in our paper. We show that integrating SAT tightly with BDD-based composition methods is able to solve instances that are not possible to solve otherwise. More importantly, we show the advantage of having a robust cutpoint resolution algorithm. The instances are obtained from three different sources: `ckt1`, `ckt2` and `ckt3` are instances of verification of Verilog RTL designs versus their corresponding gate-level versions; `pips` and `s38417` are instances of verification of gate-level unoptimized vs gate-level optimized using a commercial synthesis tool; `fxdfloc` and `sno` are instances of verification of VHDL RTL design versus their corresponding gate-level versions.

Ckt.	orig	opt	ours	[14]	[10]	[11]	[9]	[7]
C2670	1735	2076	3.5	0.8	3.4	58	3.9	159.3
C3540	2643	2802	25.7	3.0	12.6	307	17.4	67.6
C5315	3632	2525	5.3	2.7	8.3	97	14.0	372.8
C6288	6672	7331	12.1	4.3	7.2	69	9.1	21.5
C7552	5690	4467	12.7	34.6	20.8	307	20.6	5583.3

Table 1: Results on MCNC circuits (note that we verify synthesis, while others verify redundancy removal).

Not all of the output pairs are functionally equivalent; in fact, some of the bugs are “hard” in the sense that they could not be found even after more than 10 hours of random simulation time. Notice that even one bug in a design can cause multiple outputs to fail.

Table 2 describes three runs per circuit. The first run, marked as ‘basic’ is running our implementation unmodified. The second run, marked as ‘no SAT’ corresponds to running our implementation if we replace our SAT implementation by a null operation. This run compared with the first one indicates the usefulness of integrating the SAT algorithms in our tool. The third run, marked by ‘aborts’ is used to show how a non-robust cutpoint resolution would affect overall performance. Recall, that many previous papers [2, 9, 14] advocate using a threshold to bound the amount of time spent in cutpoint resolution. Given a circuit, it is hard to guess up front what should be a reasonable threshold; to give the best possible bias to the third run, we pick the 90-th percentile point of the set of cutpoint resolution times taken in the first run (for example, for `ckt1`, there were 496 cutpoint resolutions in the first run, and the time taken by 446-th shortest resolution was 0.61s, so we pick this as the threshold for abort a cutpoint resolution; notice that there is a long tail in the distribution, since the longest time in the first run for `ckt1` is 3.97s). One run (`pi ps`) was timed out after 3 hours. To limit memory usage, our BDD composition algorithm terminates when BDDs reach a threshold size, 20000 BDD nodes for these experiments. This is the reason why three of the other runs failed to finish: the ‘no SAT’ run for `fxdfloc`, the ‘no SAT’ and ‘aborts’ runs for `ckt2`. This BDD size limit is low and is just meant to illustrate the bounds of a finite memory; for these example, increasing the limit to 100000 nodes did not change the answer of any of the test cases, but increased the run-times significantly.

The table shows the total times for the runs and times for cutpoint resolutions. The remaining time is spent in parsing the circuits, in building BDDs before doing cutpoint resolution and in simulation which is performed if a resolution results in a real negative.

In the above table, the circuits are ordered in the order of difficulty, which is measured by the ratio of the total run time (of the ‘basic’ run) to the number of outputs. For the two easiest instances, all three runs are close enough that it seems that neither the SAT algorithm nor the robustness of the cutpoint resolution is important; in fact, the time for cutpoint resolution is only a small fraction of the total time, i.e., the time to build BDDs and run simulations dominates.

Without the SAT algorithm, we would have been unable to do two of the five harder instances, `ckt2` and `fxdfloc`. Also, not only is the ‘no SAT’ run for `ckt3` slower than for the ‘basic’ run, it is also the case that 4 of the cutpoint resolutions aborted due to BDD size limits; this means that had one of these 4 pairs been an output pair, we would be aborting on an output pair. So this circuit is an example of a case where a cutpoint resolution was harder than all output resolutions, something that we conjecture does not happen very often. Surprisingly, on `pi ps`, the ‘no SAT’ run was about 16% faster than the ‘basic’ run. Upon investigation, we found that for this instance, the difference in time was completely explained by the fact that somehow the BDD composition gave “better” real negatives than SAT; the negatives were better because simulation on these negatives refined many more equivalence classes than the neg-

atives from SAT. This resulted in fewer future cutpoint resolutions (that would have ended with real negatives) and caused fewer future simulations. Overall, we can conclude that using the SAT algorithm incurred a minor penalty sometimes, but on the other hand, it enabled the verification of some instances which would not have been possible otherwise.

The effects of using a robust cutpoint resolution are clear from looking at the ‘aborts’ runs for the five harder instances. To begin with, we chose a nice threshold, using an imaginary oracle and picking the 90-th percentile of the times from the ‘basic’ runs (notice that we use this threshold for the cutpoint resolutions of only internal nets, and not for the output pairs). Picking the 90-th percentile times cuts out the long cutpoint resolution times, e.g., for `ckt3`, the maximum time for a cutpoint resolution was reduced from 13.83s to 0.76s. However, these five instances show that having a non-robust but fast cutpoint resolution does not really help. Invariably, we needed to perform more resolutions. For two instances, `pi ps` and `ckt2`, we were not able to verify the instances. For another circuit `ckt3`, the run was about 25% slower because of the time wasted in aborts as well as the extra resolutions that were needed. For the other two circuits, we were faster, but by not a lot. Overall, we learn that a non-robust resolution which uses a threshold is unlikely to avoid losing most of the gains obtained by aborting at a threshold; in fact, sometimes thresholding on internal nets may prevent us from verifying the outputs.

We do not have experimental data to show that our interleaved simulation with cutpoint resolution is very effective in reducing candidates for future cutpoint resolutions as well as in finding the bugs. This result should not be that surprising. Also, all the bugs found in Table 2, except for some found in the ‘aborts’ runs, were found by simulating a witness which showed the inequivalence of a potential internal (non-output) cutpoint.

5 Conclusions

We have presented an implementation of a combinational equivalence checker that has tight integration of various verification methods, some of which are good at showing equivalences and some of which are good at showing inequivalences. The tight integration is achieved by carefully interleaving the various methods as well as passing information learned from one method to another. We have presented a novel SAT algorithm which runs directly on BDDs so as to achieve a tight coupling. Tight integration helps us achieve a cutpoint resolution mechanism which is robust and does not need to abort. Having a robust cutpoint resolution step directly affects the robustness of the entire algorithm.

References

- [1] C. L. Berman and L. H. Trevillyan. Functional Comparison of Logic Designs for VLSI Circuits. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 456–459, 1989.
- [2] D. Brand. Verification of Large Synthesized Designs. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 534–537, 1993.

Ckt.	size (gates)	outputs		Run	Cutpoint resolutions					Total time	finished?
		total	equal		all			abort			
					#	time	max	#	time		
smo	11068	1474	1317	basic	97	9.0	0.86	0	0.0	122.8	Yes
	vs			92	10.2	0.79	0	0.0	130.7	Yes	
	8816			aborts	105	7.3	0.30	9	2.7	124.7	Yes
s38417	21872	1742	1742	basic	112	12.7	2.02	0	0.0	160.9	Yes
	vs			110	10.7	1.32	0	0.0	158.4	Yes	
	26956			aborts	142	7.7	0.16	33	5.2	168.6	Yes
pips	19501	2537	2537	basic	589	214.6	8.49	0	0.0	592.7	Yes
	vs			516	169.4	8.54	0	0.0	496.3	Yes	
	10147			aborts	691	354.0	1.09	160	174.4	> 3hr	NO
ckt2	26380	1059	1020	basic	729	197.6	47.00	0	0.0	598.8	Yes
	vs			730	223.5	51.28	2	88.6	614.5	NO	
	32451			aborts	776	254.1	0.28	65	178.98	665.5	NO
ckt1	31611	519	505	basic	496	105.7	3.97	0	0.0	404.2	Yes
	vs			486	111.7	4.08	0	0.0	389.6	Yes	
	23745			aborts	494	103.9	0.61	6	3.7	390.4	Yes
ckt3	59129	409	409	basic	709	383.5	13.83	0	0.0	992.6	Yes
	vs			704	598.4	56.85	4	171.7	1195.3	Yes	
	21683			aborts	778	497.4	0.76	94	71.3	1236.8	Yes
fxdfloc	2996	1	0	basic	25	9.4	5.62	0	0.0	123.1	Yes
	vs			20	65.0	32.82	1	32.8	150.5	NO	
	1161			aborts	25	4.8	0.80	3	2.4	106.5	Yes

Table 2: Data on larger circuits; please refer to Section 4 for a detailed analysis.

- [3] R. E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35:677–691, August 1986.
- [4] J. Gu, P. W. Purdom, J. V. Franco, and B. W. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey. In D. Du, J. Gu, and P. M. Pardalos, editors, *Satisfiability Problems: Theory and Applications*. American Mathematical Society, 1997.
- [5] J. Jain, R. Mukherjee, and M. Fujita. Advanced Verification Techniques Based on Learning. In *Proc. Design Automation Conf.*, pages 420–426, 1995.
- [6] A. Kuehlmann and F. Krohm. Equivalence Checking using Cuts and Heaps. In *Proc. Design Automation Conf.*, pages 263–268, 1997.
- [7] W. Kunz, D. K. Pradhan, and S. Reddy. A Novel Framework for Logic Verification in a Synthesis Environment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 15(1):20–36, January 1996.
- [8] David E. Long. BDD Manipulation Library. Public software. Carnegie Mellon University, Pittsburgh, PA, June 1993. <ftp://emc.cs.cmu.edu/pub/bdd/bddlib.tar.z>.
- [9] Y. Matsunaga. An Efficient Equivalence Checker for Combinational Circuits. In *Proc. Design Automation Conf.*, pages 629–634, 1996.
- [10] R. Mukherjee, J. Jain, K. Takayama, M. Fujita, J. A. Abraham, and D. S. Fussell. FLOVER: Flitering Oriented Combinational Verification Approach. In *Workshop Notes of International Workshop on Logic Synthesis*, Tahoe City, CA, 1997.
- [11] D. K. Pradhan, D. Paul, and M. Chatterjee. VERILAT: Verification Using Logic Augmentation and Transformations. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 88–95, 1996.
- [12] B. Selman, H. Leveque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proc. of AAAI*, pages 440–446, 1992.
- [13] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Sequential Circuit Design Using Synthesis and Optimization. In *Proc. Intl. Conf. on Computer Design*, pages 328–333, Cambridge, MA, October 1992.
- [14] C. A. J. van Eijk. *Formal Methods for the Verification of Digital Circuits*. PhD thesis, Eindhoven University of Technology, Dept. of Electrical Engineering, Eindhoven, Netherlands, 1997.