SAT-Based Image Computation with Application in Reachability Analysis

Aarti Gupta, Zijiang Yang, Pranav Ashar, and Anubhav Gupta

NEC USA CCRL
4 Independence Way, Princeton, NJ 08540
{agupta, jyang, ashar, anubhav}@ccrl.nj.nec.com

Abstract. Image computation finds wide application in VLSI CAD, such as state reachability analysis in formal verification and synthesis, combinational verification, combinational and sequential test. Existing BDD-based symbolic algorithms for image computation are limited by memory resources in practice, while SAT-based algorithms that can obtain the image by enumerating satisfying assignments to a CNF representation of the Boolean relation are potentially limited by time resources. We propose new algorithms that combine BDDs and SAT in order to exploit their complementary benefits, and to offer a mechanism for trading off space vs. time. In particular, (1) our integrated algorithm uses BDDs to represent the input and image sets, and a CNF formula to represent the Boolean relation, (2) a fundamental enhancement called BDD Bounding is used whereby the SAT solver uses the BDDs for the input set and the dynamically changing image set to prune the search space of all solutions, (3) BDDs are used to compute all solutions below intermediate points in the SAT decision tree, (4) a fine-grained variable quantification schedule is used for each BDD subproblem, based on the CNF representation of the Boolean relation. These enhancements coupled with more engineering heuristics lead to an overall algorithm that can potentially handle larger problems. This is supported by our preliminary results on exact reachability analysis of ISCAS benchmark circuits.

1 Introduction

Image and pre-image computation play a central role in symbolic state space traversal, which is at the core of a number of applications in VLSI CAD like verification, synthesis, and testing. The emphasis in this paper is on reachability analysis for sequential system verification. For simplicity of exposition, we focus only on image computation; the description can be easily extended to pre-image computation as well.

1.1 BDD-based Methods

Verification techniques based on symbolic state space traversal [7, 9] rely on efficient algorithms based on BDDs [4] for computing the image of an input set over a Boolean relation. The input set in this case is the set of present states

P, and the Boolean relation is the transition relation T, i.e. the set of valid present-state, next-state combinations. (For hardware, it is convenient to also include the primary inputs in the definition of T). The use of BDDs to represent the characteristic function of the relation, the input, and the image set, allows image computation to be performed efficiently through Boolean operations and variable quantification. As an example of its application, the set of reachable states can be computed by starting from a set P which denotes the set of initial states of a system, and using image computation iteratively, until a fixpoint is reached.

A number of researchers have proposed the use of *partitioned* transitioned relations [6, 21], where the BDD for the entire transition relation is not built a priori. Typically, the partitions are represented using multiple BDDs, and their conjunction is interleaved with early variable quantification during image computation. Many heuristics have been proposed to find a good quantification schedule, i.e. an ordering of the conjunctions which minimizes the number of peak variables [11, 19]. There has also been an interest in using disjunctive partitions of the transition relations and state sets [8, 17, 18], which effectively splits the image computation into smaller subproblems.

The BDD-based approaches work well when it is possible to represent the sets of states and the transition relation (as a whole, or in a usefully partitioned form) using BDDs. Unfortunately, BDD size is very sensitive to the number of variables, variable ordering, and the nature of the logic expressions being represented. In spite of a large body of work, the purely BDD-based approach has been unreliable for designs of realistic size and functionality.

1.2 Combining BDDs with SAT-based Methods

An alternative, used extensively in testing applications [13], is to represent the transition relation in Conjunctive Normal Form (CNF) and use Boolean Satisfiability Checking (SAT) for various kinds of analysis. SAT solver technology has improved significantly in recent years with a number of sophisticated packages now available, e.g. [16]

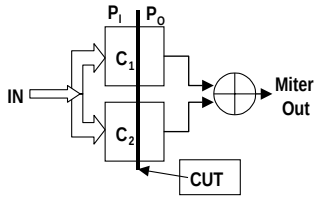


Fig. 1. Miter Circuit for Combinational Verification

For checking equivalence of two given combinational circuits C1 and C2, a typical approach is to prove that the XOR of their corresponding outputs, called

the miter circuit output, can never evaluate to 1, as shown in Figure 1. This proof can be provided either by building a BDD for the miter, or by using a SAT solver to prove that no satisfying assignment exists for the miter output. In cases where the two methods fail individually, BDDs and SAT can also be combined, for example, in the manner shown in Figure 1. A cut is identified in the miter circuit to divide the circuit into two parts: the part P_I of the circuit between the circuit inputs and the cut, and the part P_O of the circuit between the cut and the output. A BDD is built for P_O, while P_I is represented in CNF. A SAT solver then tries to enumerate all valid combinations at the cut using the CNF for P_I, while checking that it is not contained in the on-set of the BDD for P_O [12]. Enumerating the valid combinations at the cut corresponds exactly to computing the image of the input set over the Boolean relation corresponding to P_I. Other ways of combining BDDs and SAT for equivalence checking have also been proposed [5].

For property checking, the effectiveness of SAT solvers for finding bugs has also been demonstrated in the context of bounded model checking and symbolic reachability analysis [1, 2, 22]. The common theme is to convert the problem of interest into a SAT problem, by devising the appropriate propositional Boolean formula, and to utilize other non-canonical representations of state sets. However, they all exploit the known ability of SAT solvers to find a single satisfying solution when it exists. To our knowledge, no attempt has been made to formulate the problems in a way that a SAT solver is used to find *all* satisfying solutions.

In our approach to image computation, we use BDDs to represent state sets, and a CNF formula to represent the transition relation. All valid next state combinations are enumerated using a backtracking search algorithm for SAT that exhaustively visits the entire space of primary input, present state and next state variables. However, rather than using SAT to enumerate each solution all the way down to a leaf, we invoke BDD-based image computation at intermediate points within the SAT decision procedure, which effectively obtains all solutions below that point in the search tree. In a sense, our approach can be regarded as SAT providing a disjunctive decomposition of the image computation into many subproblems, each of which is handled in the standard way using BDDs. In this respect, our work is closest to that of Moon et al. [17], who independently formulated a decomposition paradigm similar to ours. However, there are significant differences in the details, and we defer that discussion to Section 7.

We start by providing the necessary background on a typical SAT decision procedure in the next section. Our proposed algorithm for image computation is described in detail in the sections that follow. Towards the end, we provide experimental results for reachability analysis, which validate the individual ideas and the overall approach proposed by us, and describe some of our work in progress.

2 Background: Satisfiability Checking (SAT)

The Boolean Satisfiability (SAT) problem is a well-known constraint satisfaction problem with many applications in computer-aided design, such as test generation, logic verification and timing analysis. Given a Boolean formula, the objective is to either find an assignment of 0-1 values to the variables so that the formula evaluates to true, or establish that such an assignment does not exist. The Boolean formula is typically expressed in Conjunctive Normal Form (CNF), also called product-of-sums form. Each sum term (clause) in the CNF is a sum of single literals, where a literal is a variable or its negation. An n-clause is a clause with n literals. For example, (vi+vj'+vk) is a 3-clause. In order for the entire formula to evaluate to 1, each clause must be satisfied, i.e., evaluate to 1.

```
Initialize(); // all var made "free"
do {
   Implications();
   status = Bound();
   if (status == contradiction)
     if (active_var->assigned_val ==
           active_var->first_val)
        active_var->assigned_val =
            !active_var->first_val;
     else
        prev_var = Backtrack();
        if (prev_var == NULL)
            soln = no_soln;
            return;
        endif;
     endif
   else
     active_var = Next_free_var();
     if (active_var == NULL)
         soln = found;
         return;
     endif;
     active_var->first_val =
         active_var->assigned_val = Val();
   endif;
} While ();
```

Fig. 2. Backtracking Search Procedure for SAT

The complexity of this problem is known to be NP-Complete. In practice, most of the current SAT solvers are based on the Davis-Putnam algorithm [10]. The basic algorithm begins from an empty assignment, and proceeds by assigning a 0 or 1 value to one free variable at a time. After each assignment, the algorithm determines the direct and transitive implications of that assignment on other variables, typically called *bounding*. If no contradiction is detected during the implication procedure, the algorithm picks the next free variable, and repeats the procedure. Otherwise, the algorithm attempts a new partial assignment by complementing the most recently assigned variable for which only one value has been tried so far. This step is called *backtracking*. The algorithm terminates

either when all clauses have been satisfied and a solution has been found, or when all possible assignments have been exhausted. The algorithm is complete in that it will find a solution if it exists.

Pseudo code for the basic Davis-Putnam search procedure is shown in Figure 2. The function and variable names have obvious meanings. This procedure has been refined over the years by means of enhancements to the `Implications()`, `Bound()`, `Backtrack()`, `Next_free_var()` and `Val()` functions. The GRASP work [15] proposed the use of non-chronological backtracking by performing a conflict analysis, and addition of conflict clauses to the database in order to avoid repeating the same contradiction in the future.

3 Image Computation

The main contribution in our paper is the novel algorithm for image computation by combining BDD- and SAT-based techniques in a single integrated framework. In relationship to current SAT solvers, our contributions are largely specific to their use for image computation. They are orthogonal to the most advanced features found in state-of-the-art SAT algorithms like GRASP [16], and indeed add to them.

3.1 Representation Framework

Our representation framework consists of using BDDs to represent the input and image sets, and a CNF formula to represent the Boolean relation. This choice is motivated by the fact that BDD-based methods frequently fail because of their inability to effectively manipulate the BDD(s) for the transition relation, in its entirety or in partitioned form. Furthermore, since BDDs for the input and image sets might also become large for complex systems, we do not require that a single BDD be used to represent these sets. Any disjunctively decomposed set of BDDs will work with our approach. This setup is shown pictorially in Figure 3. For our current prototype, we use a simple "chronological" disjunctive partitioning, such that whenever the BDD size for a set being accumulated crosses a threshold, a new BDD is created for storing future additions to the set. We are exploring use of alternative representations to manage these sets.

3.2 Image Computation Using CNF-BDDs

The standard image computation formula is shown below in Equation (1), where x, y, and w denote the set of present state, next state, and primary input variables, respectively; $P(x)$ denotes the input set, and $T(x, w, y)$ denotes the transition relation.

$$Image(P, T)(y) = \exists x, w. P(x) \wedge T(x, w, y) \tag{1}$$

In our framework, $P(x)$ and $Image(y)$ are represented as (multiple) BDDs, while T is represented as a CNF formula in terms of x, w, y and some additional

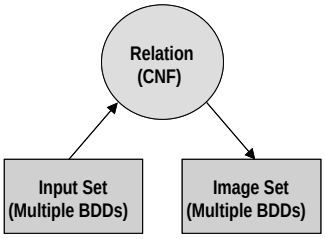


Fig. 3. The CNF-BDDs Representation Setup

variables z, which denote internal signals in the circuit. We compute the image set by enumerating all solutions to the CNF formula T, and recording only the combinations of y variables, while restricting the values of x variables to those that satisfy $P(x)$. Note that by restricting the x variables to satisfy $P(x)$, we are effectively performing the conjunction in the above formula. This restriction is performed by what we call *BDD Bounding*. Essentially, during the SAT search procedure, any partial assignment to the x variables that does not belong to the on-set of the BDD(s) $P(x)$ is pruned immediately [12]. Note also, that by enumerating *all* (not a single) solution to the CNF formula, and by considering combinations of only y variables among these solutions, we are effectively performing a quantification over all the other variables (x, w, z).

We also use $Unreached(y)$ as a care-set for the image set. In applications such as reachability analysis where image computation is performed iteratively, this set can be computed as the negation of the current set of reached states. Again, by using BDD(s) to represent $Unreached(y)$, we can obtain additional pruning of the SAT search space by performing BDD Bounding against this image care-set. To summarize, we use the following equation for image computation:

$$Image(P, T)(y) = \exists x, w, z.P(x) \wedge T(x, w, z, y) \wedge Unreached(y) \qquad (2)$$

4 BDD Bounding

A naive approach for performing BDD Bounding is to enumerate each complete SAT solution up to the leaf of the search tree, and then check if the solution satisfies the given BDD(s). This is obviously inefficient since the number of SAT solutions may be very large.

In our setup, the x/y variables are shared between the input/image set BDD(s) and the CNF formula. Therefore, whenever a value is set to or implied on one of these variables in SAT, we can check if the intersection of the partial assignment with the given BDD(s) is non-null. If it is indeed non-null, the SAT procedure can proceed forward. Otherwise it must backtrack, since no solution consistent with the conjunctions can be found under this subtree. In our earlier work on combinational verification, we had called this the *Early Bounding* approach [12], and had demonstrated a significant reduction in the number of

```
Bound(sat,lit,input_bdd) {
   if (lit_is_input_set_variable(lit)){
     new_bdd =
     project_variable_in_bdd(input_bdd,lit);
     if (bdd_equal_zero(new_bdd))
       return contradiction;
     else
       input_bdd = new_bdd;
   } // rest of the procedure is unchanged
}
```

Fig. 4. Pseudo-code for BDD Bounding

backtracks due to pruning off large subspaces of the search tree. Note that the smaller the bounding set, the greater the pruning, and the faster the SAT solver is likely to be.

4.1 Bounding Against the Image Set: A Positive Feedback Effect

In addition to bounding against the input set $P(x)$ and the image care-set $Unreached(y)$, a fundamental speed-up in our image computation procedure can be obtained by also bounding against the BDDs of the currently computed image set denoted $Current(y)$. Note that $Unreached(y)$ does not change during a single image computation, while $Current(y)$ is updated dynamically, as new solutions for the image set are enumerated. Therefore, if a partial assignment over y variables is contained in $Current(y)$, it implies that any extension to a full assignment has already been enumerated. Therefore, it serves no purpose for the SAT solver to explore further, and it can backtrack. As a result, a positive feedback effect is created in which the larger the image set grows, the faster the SAT solver is likely to be able to go through the remaining portion of the search space.

4.2 Implementation Details: Bounding the x Variables

For BDD Bounding against $P(x)$, we modify the `Bound()` function of Figure 2, so that it checks the satisfaction of a partial assignment on x variables with the on-set of the BDDs for $P(x)$. Again, if the partial assignment has a null intersection with each BDD, the SAT solver is made to backtrack, just as if there were a contradiction.

The pseudo-code for the `Bound()` procedure for a single BDD is shown in Figure 4. In this procedure, the initial argument is the BDD for $P(x)$, and the recursive argument maintains its projected version down the search tree. The `project_variable_in_bdd()` procedure, can be easily implemented by using either the `bdd_substitute()` or the `bdd_cofactor()` operations available in standard BDD packages [20]. Unfortunately, these standard BDD operations represent a considerable overhead in terms of unique table and cache lookups. In our implementation, we use a simple association list to keep track of the x variable assignments. Projecting (un-projecting) a variable is accomplished simply by setting (un-setting) its value in the association – a constant-time operation.

However, checking against a `zero_bdd` requires a traversal of the argument BDD, by taking branches dictated by the variable association. If any path to a `one_bdd` is found, the traversal is terminated, and the BDD is certified to be not equal to the `zero_bdd`. In the worst case, this takes time proportional to the size of the BDD. As a further enhancement, for each BDD node, we associate a value indicating the presence or absence of a path to a `one_bdd` from that node. This bit must be modified only if the value of a variable below this node is modified. As a result, the average complexity of the emptiness check is likely better than the BDD size.

4.3 Implementation Details: Bounding the y Variables

During the SAT search, if the intersection of a partial assignment over y variables and $(Unreached(y) \wedge !Current(y))$ is non-null, exploration should proceed further down that path, otherwise it can backtrack.

```
/* Each BDD node has a user-defined field that indicates if the sub-graph
below it is tautologically one, zero or neither (TWO_BDD) */
/* B_array is an array of BDD nodes, V_array is the array of variables and
their values */
    bdd_equal_zero(B_array, V_array) {
 /* Leaves */
 if (any BDD in B_array is ZERO_BDD)
    return ZERO_BDD;
 if (each BDD in B_array is ONE_BDD)
    return ONE_BDD;
 if (only one BDD in B_array is TWO_BDD)
    return TWO_BDD;

 /* Non-leaf case */
 NewB_array = remove_one_bdds(B_array);
 if (is_in_cache(NewB_array, &Val))
    return Val;

 /* get top var from all the BDDs in NewB_array */
 Top_var = get_top_var(NewB_array);
 ThenB_array = get_then(NewB_array, Top_var);
 ElseB_array = get_else(NewB_array, Top_var);
 Proj_value = get_proj_val(Top_var, V_array);
 if (Proj_value == ONE) {
    Val = bdd_equal_zero(ThenB_array, V_array);
 } else if Proj_value == ZERO) {
    Val = bdd_equal_zero(ElseB_array, V_array);
 } else {
    Val0 = bdd_equal_zero(ThenB_array, V_array);
    Val1 = bdd_equal_zero(ElseB_array, V_array);
    if (Val0 == Val1) {
      Val = Val0;
    } else {
        Val = TWO_BDD;
      }
    }
    insert_in_cache(newB_array, Val);
    return Val;
 }
}
```

Fig. 5. Determining Emptiness of the Product of Multiple BDDs with Projections

Recall that we allow use of a disjunctive partitioning of the reached state set $R = \cup_i R_i$. Therefore, both the $Unreached$ and $!Current$ sets can be repre-

sented as product of BDDs, i.e. $Unreached(y) = \cap_i !R_i(y)$, and $!Current(y) = \cap_i !Current_i(y)$. Rather than performing an explicit product of the multiple BDDs, the partial assignment over y variables is projected separately onto each BDD. Then the the multiple BDDs are traversed in a lock-step manner by using a modified `bdd_equal_zero()` procedure, to determine if there exists a path in their product that leads to a `one_bdd`. The pseudo code for this is shown in Figure 5, where the given procedure assumes that projection of variable values onto the individual BDDs has already been carried out. In the actual implementation, the projection of variables and detection of emptiness are done in a single pass, along with handling of complemented BDD nodes. The worst-case complexity is that of actually computing a complete product, but in practice the procedure terminates as soon as any path to `one_bdd` is found.

5 BDDs at SAT Leaves

So far we have explained our algorithm for image computation in terms of enumerating all solutions of the CNF formula using SAT-solving techniques, while performing BDD Bounding where possible in order to prune the search space. This still suffers from some drawbacks of a purely SAT-based approach, i.e. solutions are enumerated one-at-a-time, without any reuse. To some extent this drawback is countered by examining partial solutions (cubes) for inclusion and for pruning, but we can actually do better.

It is useful in this regard to compare a purely SAT-based approach vs. a purely BDD-based approach. In essence, both work on the same search space of Boolean variables – SAT solvers use an explicit decision tree, while BDD operations work on the underlying DAGs. A BDD-based approach is more suitable for capturing all solutions simultaneously. However, due to the variable ordering restriction, it can suffer from a size blowup in the intermediate/final results. On the other hand, a SAT decision tree has no variable ordering restriction, and can therefore potentially manage larger problems. However, since it is not canonical, many subproblem computations may get repeated.

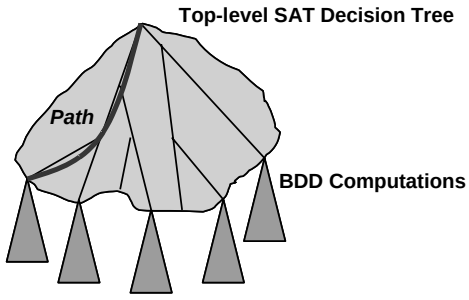


Fig. 6. BDDs at SAT Leaves

In order to combine the relative advantages of both, we use a SAT decision tree to organize the top-level search space. Within this tree, along any path, rather than using the SAT-solver to explore the tree further, we can invoke a BDD-based approach to compute all solutions in the sub-tree under that path. This integrated scheme, which we call *BDDs at SAT Leaves*, is illustrated pictorially in Figure 6. In a sense, the SAT decision tree can be regarded as a disjunctive partitioning of a large problem at the root into smaller subproblems at the leaves, each of which can be handled by a purely BDD-based approach.

5.1 Leaf Subproblem: BDD-based Image Computation

The formulation of the BDD subproblem to be solved at each leaf of the SAT decision tree is shown below:

$$New(y) = Path(y') \wedge \exists x", w", z". P(x)|_{Path(x')} \wedge Unsat(x", w", z", y") \quad (3)$$

This computes the image set solutions New from a sub-tree rooted at the end of a path in the SAT decision tree. Here, for a set of variables v, the assigned set is denoted v', and the unassigned set is denoted $v"$. $Path(x')/Path(y')$ denote the BDDs representing the partial assignment of x/y variables along the path. $Unsat(x", w", z", y")$ denotes the product of all unsatisfied clauses at the end of the path, projected by the assigned variables along that path, expressed in terms of the unassigned variables appearing in the original CNF formula. Finally, $P(x)|_{Path(x')}$ denotes the restriction of the set $P(x)$ to the partial assignments of x along the path.

Note that in this equation, the part following the existential quantification is identical in formulation to a standard purely BDD-based approach. The difference is only in the *granularity* of the Boolean relation $Unsat$, and its conjunctive decomposition. In a standard approach, the Boolean relation $T(x, w, y)$ is a transition relation, expressed in terms of the present state, primary input, and next state variables only. Furthermore, its conjunctive decomposition is typically based on splitting the next state variables.

In our approach, the Boolean relation is expressed as a CNF formula over the set of present state, primary input, next state, *and* intermediate variables denoting signals on internal gates that implement the next state logic of the sequential circuit. Furthermore, the conjunctive decomposition follows the structural decomposition of the circuit into gates. Though this finer-grained approach must handle more number of Boolean variables than the standard approach, it also allows a greater potential for early quantification, which has been noted to help overcome the blowup during image computation (described in detail in the next section).

Another benefit of using the fine-grained CNF partitions is that there is no penalty for performing pre-image computations. Many researchers have noted that backward symbolic traversal is less efficient than forward traversal. This is partly due to having to handle the typically irregular unreachable part of

the state space. Furthermore, most methods use partitions based on splitting the next state (y) variables, while sharing the present state (x) variables. This scheme is good for performing image computations with early quantification of x variables, but it does not work very well for pre-image computations where the y variables need to be quantified. In contrast, our fine-grained CNF formulation is symmetric with respect to the x and y variables. Therefore, our method can be applied equally well for image as well as pre-image computations.

5.2 Leaf Subproblem: Quantification Schedule

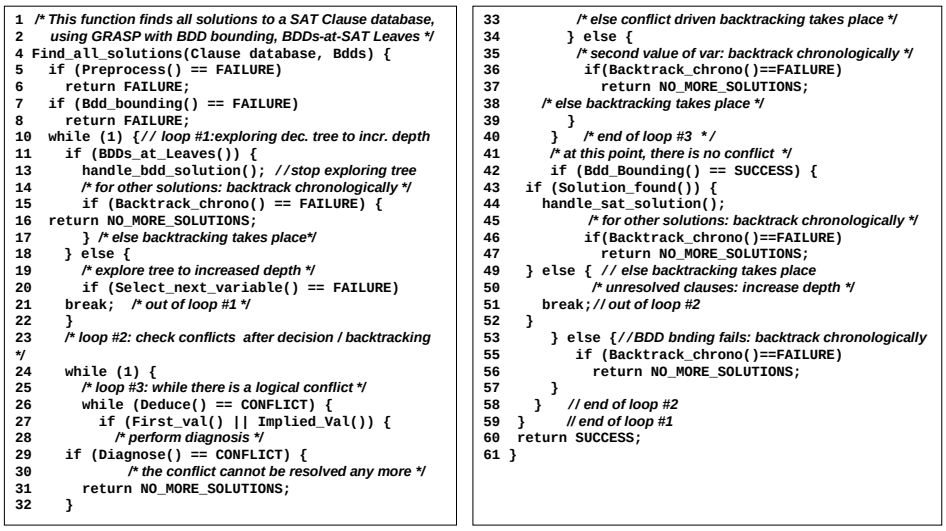
In practice, it is important to choose a good quantification schedule, i.e. an ordering on the conjunctions of the partitions that avoids intermediate blowup during image computation. Typically, a good schedule tries to minimize the number of active variables in a linearized schedule, by analyzing the variable support sets of the individual partitions [11, 17, 19].

```
Leaf_Image_Computation() {
   B = {projected P(x), projected unsat_clauses};     // set of BDDs
   do {
      v = min_cost_variable(B);    // choose variable v
      C = {b | b ∈ B, b depends on v};    // gather conjuncts for v
      c = and_smooth(C, v);    // quantify v along with conjunction
      replace(B, C, c);    // replace conjuncts C in B by c
   } while (variables_to_be_quantified);
   c = and(B);
   new = and(path(y),c);
}
```

Fig. 7. Leaf Image Computation

For each leaf image computation, the pseudo-code for the quantification schedule is shown in Figure 7. We start with a collection B of BDDs consisting of the projected $P(x)$ (and potentially $Unreached(y)$), and a BDD for every projected unsatisfied clause. Next, we heuristically select a variable v to be quantified. We greedily choose the minimum cost variable, where cost is estimated as the product of the individual BDD sizes that the variable appears in. Once v is selected, we gather in set C all conjuncts that v appears in. This is followed by conjunction and quantification of v in C, and this result replaces the set C in B. (Since the y variables cannot be quantified, we never choose them.) This basic loop is iterated until no more variables can be quantified. The remaining BDDs (with only y variables) are conjoined together, and the result is conjoined with $path(y)$ (the cube of assigned y variables), to give the set of new image solutions corresponding to that path.

Note that this formulation does not depend on a live variable analysis over a linearized schedule but considers the actual BDD sizes for selection. Therefore, it



```
1  /* This function finds all solutions to a SAT Clause database,
2     using GRASP with BDD bounding, BDDs-at-SAT Leaves */
4  Find_all_solutions(Clause database, Bdds) {
5    if (Preprocess() == FAILURE)
6      return FAILURE;
7    if (Bdd_bounding() == FAILURE)
8      return FAILURE;
10   while (1) {// loop #1:exploring dec. tree to incr. depth
11     if (BDDs_at_Leaves()) {
13       handle_bdd_solution(); //stop exploring tree
14       /* for other solutions: backtrack chronologically */
15       if (Backtrack_chrono() == FAILURE) {
16   return NO_MORE_SOLUTIONS;
17       } /* else backtracking takes place*/
18     } else {
19       /* explore tree to increased depth */
20       if (Select_next_variable() == FAILURE)
21     break;   /* out of loop #1 */
22     }
23     /* loop #2: check conflicts  after decision / backtracking
*/
24     while (1) {
25       /* loop #3: while there is a logical conflict */
26       while (Deduce() == CONFLICT) {
27         if (First_val() || Implied_Val()) {
28           /* perform diagnosis */
29     if (Diagnose() == CONFLICT) {
30             /* the conflict cannot be resolved any more */
31       return NO_MORE_SOLUTIONS;
32     }
```

```
33           /* else conflict driven backtracking takes place */
34         } else {
35           /* second value of var: backtrack chronologically */
36            if(Backtrack_chrono()==FAILURE)
37              return NO_MORE_SOLUTIONS;
38      /* else backtracking takes place */
39         }
40       }   /* end of loop #3  */
41       /* at this point, there is no conflict  */
42       if (Bdd_Bounding() == SUCCESS) {
43    if (Solution_found()) {
44      handle_sat_solution();
45            /* for other solutions: backtrack chronologically */
46           if(Backtrack_chrono()==FAILURE)
47             return NO_MORE_SOLUTIONS;
49    } else { // else backtracking takes place
50            /* unresolved clauses: increase depth */
51      break;// out of loop #2
52    }
53       } else {//BDD bnding fails: backtrack chronologically
55          if (Backtrack_chrono()==FAILURE)
56            return NO_MORE_SOLUTIONS;
57       }
58     }   // end of loop #2
59   }     // end of loop #1
60   return SUCCESS;
61 }
```

Fig. 8. Complete Image Computation Procedure

is better able to balance the computation in the form of a tree of conjunctions, rather than a linear series of conjunctions. In our experiments, this heuristic performed far better than others based on variable supports.

6 The Complete Image Computation Procedure

Our complete procedure for enumerating all solutions of the image set is shown in Figure 8. It is based on the publicly available GRASP SAT-solver [16]. We start by describing its original skeleton. After the initial preprocessing, the procedure consists of an outer loop #1 (line 10) that explores the SAT decision tree to increased depth if necessary. The inner loop #2 (line 24) is used primarily to propagate constraints and check for conflicts after either a decision variable is chosen, or after backtracking takes place to imply a certain value on a variable. Loop #3 (line 26) actually performs the deduction to check for contradictions and tries to resolve the conflict using diagnosis until there is no more conflict. In GRASP, clauses are added to record causes of all backtracking operations, including those used to enumerate multiple solutions.

The completeness argument for our procedure with respect to finding all solutions of the image set is based largely on the completeness of the original procedure in GRASP [14]. The additions we have made to the original procedure consist of introducing the techniques of *BDDs at SAT Leaves* (lines 11-18), and *BDD Bounding* (lines 7-8, lines 42-57). The only other modification we have made is to perform conflict analysis only if the value of the decision variable is the first value being tried, or if its second value has been implied (line 27).

The correctness of finding all BDD solutions at the leaves, and of pruning the search space when BDD Bounding fails follows from the arguments described in the previous sections. Note that in both these cases, we perform a chronological backtracking (lines 14-17, lines 54-56) in order to search for the next solution. In case BDD Bounding succeeds (line 42), we check whether a solution is found, i.e. whether all clauses are satisfied. If they are, a SAT solution has been found, which is handled in the usual way, followed by chronological backtracking to find the next solution (lines 45-48).

The reason for the modification (line 27) is that we do not wish to add clauses to record the causes of chronological backtracking. In our modified GRASP algorithm, chronological backtracking takes place after a solution has been found, or after BDD Bounding fails. However, when clauses for chronological backtracking are not recorded, GRASP's conflict analysis during diagnosis becomes incomplete, and it may be erroneous to perform non-chronological backtracking based on this conflict analysis. Therefore, if the second value of a variable is implied by some clause (either an original, or a conflict clause), we do allow diagnosis to take place. Otherwise, we disable non-chronological backtracking by not performing any diagnosis at all. Instead, we perform a simple chronological backtracking (lines 34-39). Note that performing chronological backtracking instead of non-chronological backtracking can at most affect the procedure's efficiency, not its completeness.

7 Why SAT?

As mentioned earlier, there has been recent interest in using disjunctive decompositions of the image computation problem using purely BDDs, with substantially improved practical results [8, 17]. Our use of a SAT decision tree to split the search tree, and use of the BDD-based image computations at its leaves to perform the conjoining, results in a similar decomposition. However, in our view, SAT provides many more advantages than just a disjunctive decomposition, which also differentiate our approach from the rest.

In particular, it allows us to easily perform implications of a variable decision (splitting). In principle, deriving implications can be done in non-SAT contexts as well, e.g. directly on circuit structure, using BDDs etc. However, to our best knowledge, this has not been done in practice for image computation. By using a standard state-of-the-art SAT package [16], we are utilizing the years of progress in this direction, as well as in related techniques of efficient backtracking and conflict analysis, which all help toward pruning the underlying search space. Our use of BDD Bounding is an additional pruning technique, which allows us to perform early backtracking without even invoking a BDD-based leaf computation.

Another difference of our approach from the rest is in the granularity of our underlying search space. Since we focus on the CNF formula for the transition relation, which is derived directly from a gate-level structural description of the design, we obtain a very fine-grained partition of the relation, which is also sym-

metric with respect to image and pre-image computations. This allows us to split the overall into much finer partitions, where decision (splitting) variables can also be internal signals. We use both BDD-based and SAT-based criteria for selection of these variables, e.g. estimate of cofactor BDD sizes [8], number of clauses a variable appears in, etc. We are also exploring SAT-based criteria targeted towards finding multiple, and not single, solutions. For each partition itself, the finer level of granularity allows us to exploit the benefits of early quantification to a greater degree. This is reflected in our BDD-based quantification schedule algorithm, which uses different criteria (actual BDD sizes) for selecting the variable to be quantified, and is organized as a tree of conjunctions, rather than a linear series.

Finally, our aim is to combine SAT and BDDs in a seamless manner in order to facilitate a smooth and adaptive tradeoff between time and space for solving the image computation problem. In our algorithm, the move from SAT to BDDs occurs when a BDD subproblem is triggered. Ideally, we would like to do this whenever we could be sure that the BDDs would not blow up. However, there seems to be no simple measure to predict this a priori. We are currently experimenting with several heuristics based on number of unassigned variables, size of the projected $P(x)$ set etc. We have also implemented a simple timeout mechanism for the BDD subproblem, which allows us to return back to SAT, in order to perform some more splits (unlike [17]). Since CNF formulas and BDDs are entirely interchangeable, the boundary between SAT and BDDs is somewhat arbitrary. In principle, it is possible to freely intermix CNFs and BDDs for various parts of the circuit, and perform required analysis on the more appropriate representation. Our approach is a step in this direction.

8 Experiments

We have implemented an initial prototype of our image computation algorithm based on the CUDD BDD package [20] and the GRASP SAT solver [16]. This section describes our experimental results on some ISCAS benchmark circuits known to be difficult for reachability analysis. All experiments were run on an UltraSPARC workstation, with a 296 MHz processor, and 768 MB memory.

Since our main contribution here is to make the core step of image computation more robust, we only focus on experiments for exact reachability analysis. Our algorithm can be easily adapted and enhanced in many orthogonal directions such as its use in approximate reachability analysis, invariant checking, and model checking. We are currently working on porting this prototype to VIS [3] in order to use its infrastructure for such applications, and also to have access to a wider set of benchmarks.

A comparison of our prototype, which we call the CNF-BDD prototype, with VIS [3] is shown in Table 1. It shows results for performing an exact reachability analysis using pure breadth-first traversal on some benchmark circuits known to be difficult to handle in practice. The circuit name and number of latches are shown in Columns 1 and 2, respectively. For our approach, a measure of circuit

Ckt	#FF	#Vars	#Steps	#Reached	Vis Time	CNF BDD				
				States	(s)	Time (s)	Mem (MB)	Peak (M Nodes)	Leaves	Bounds
s1269	37	624	10	1.13E+09	3374	1877	46	0.6	15150	5278
s1512	57	866	1024	1.66E+12	2362	6337	28	0.34	13289	3565
s1423	74	748	11 (p)	7.99E+09	7402	2425	40	1.16	217	329
			13 (p)	7.96E+10	--	5883	274	6.86	330	452
s5378	164	2978	6 (p)	2.47E+16	31346	19024	197	2.4	668	348
			8 (p)	2.36E+17	--	29230	202	2.4	981	421

Table 1. Results for Exact Reachability Analysis

complexity is the number of variables appearing in our CNF representation of the transition relation – shown in Column 3. The number of steps completed is shown in Column 4, where a "(p)" indicates partial traversal. Column 5 reports the number of states reached. In Column 6, we report the CPU time taken (in seconds) by VIS. For these experiments we used a timeout of 10 hours. However, sometimes VIS runs into memory limitations before the timeout itself, indicated as "–" in this column. Columns 7 through 11 provide numbers for our CNF-BDD prototype. Column 7 reports the CPU time taken (in seconds). Columns 8 and 9 report the memory used, and the number of peak BDD nodes, as reported by the CUDD package. To give an idea of the efficiency of our modified SAT solver, we report the number of BDDs subproblems (Leaves), and the number of backtracks due to BDD Bounding (Bounds) in Columns 10 and 11, respectively. In all our experiments, we did not observe any non-chronological backtracking in the SAT solver. Therefore, the total number of backtracks during image computation for all steps is the sum of Columns 10 and 11.

It can be seen that the performance of our CNF-BDD prototype is better than VIS in 3 of 4 circuits. For s1512, our prototype is worse likely due to the large number of steps – 1024, and the overhead in every step of re-starting the SAT solver. For s1269, our prototype performs better than VIS by about a factor factor of 2. For both of these circuits, the CNF-BDD numbers are worse than those reported recently by Moon et. al [17] – 891 sec. for s1269, and 2016 sec. for s1512. The real gains from our approach can be seen in the more difficult circuits s1423 and s5378, where neither VIS nor our prototype can perform complete traversal. For s1423, VIS was able to complete up to step 11. For the same number of steps, our prototype is faster than VIS by more than a factor of 3. In fact, our prototype is able to complete two additional steps of the reachability computation in the time allotted compared to VIS. Similarly, for s5378, our prototype is faster than VIS up to step 6, and is able to complete 2 additional steps in the time allotted. As can be seen from the Columns showing Mem and Peak, our approach seems not to be memory bound yet, for these experiments. On the other hand, VIS gets memory bound in the allotted time. Furthermore, the number of backtracks is also well under control, with BDD Bounding being very effective in pruning the SAT search space. For all our experiments, the overhead of computing the BDD Bounding information was negligible.

s1423	Nano Trav				CNF-BDD				
Step	#Reached	Time(s)	Peak	Live	Time(s)	Peak	Live	Leaves	BoundB
1	545	2	14308	2627	0.7	22484	249	2	12
2	3345	2.5	19418	3847	4.9	31682	337	8	14
3	55569	5.6	32704	4010	4.1	42924	550	9	18
4	3.92E+05	14.1	35770	6766	5.6	50078	1352	13	22
5	2.08E+06	39.4	55188	15773	8.4	50078	2629	9	19
6	8.49E+06	81.4	74606	19824	11.6	64386	6562	12	20
7	3.37E+07	391.7	17696	35849	20.8	82782	13061	10	19
8	1.11E+08	867.4	320908	64126	27.4	123662	33262	8	17
9	4.90E+08	3050	888118	199237	125.4	224840	93492	34	61
10	1.68E+09	7991.3	1634178	413060	521.8	531440	253527	58	72
11	7.99E+09	18705.2	4027702	650065	1357.1	1168146	585034	51	55
12	2.30E+10	--	--	--	2100.8	4259696	1538532	33	44
13	7.96E+10	--	--	--	1694.3	6865796	4098655	80	79
					5882.9			327	452

Table 2. Results on Memory Usage for Circuit s1423

To contrast the memory requirements of our approach with a standard purely BDD-based approach, we conducted detailed experiments for s1423. At this time, we were not able to change the VIS interface in order to extract the memory usage statistics we needed. Therefore, these experiments were conducted using a stand-alone traversal program called "nanotrav", which is distributed along with the CUDD package [20]. In general, nanotrav performs worse than VIS, since it does not include sophisticated heuristics for early quantification or clustering of the partitioned transition relation. This limitation does affect the memory requirements to some extent, but we present Table 2 mainly to show the overall trend. In this table, the number of steps and reached states is shown in Columns 1 and 2. For nanotrav, Column 3 reports the CPU time taken (in seconds). The number of peak nodes and live nodes at the end of each image computation are shown in Columns 4 and 5, respectively. Columns 6, 7, 8 report the same for our CNF-BDD approach. We also report the number of BDDs at SAT Leaves (Leaves), and the number of backtracks due to BDD Bounding (Bounds) in Columns 9 and 10, respectively.

As can be seen clearly from this table, the memory requirements continue to grow with the number of steps. However, this growth is at a faster pace for nanotrav than it is for the CNF-BDD prototype. At the 11th step, the peak BDD nodes for nanotrav are greater than for CNF-BDD by a factor of 4. Not surprisingly, the CNF-BDD can complete two more reachability steps in about the same time. Furthermore, the CNF-BDD prototype is still not limited by memory while performing the 14th step, but is forced to time out after 10 hours.

In our approach, we can tradeoff between SAT solvers and BDDs by dynamically changing the conditions for triggering BDDs at SAT leaves. However, this only provides a memory-time tradeoff for the purpose of image computation. It does not reduce the memory requirements in using monolithic BDDs for representing state sets. As described in the earlier sections, our approach is completely suited for handling state sets in the form of disjunctive partitions (multiple BDDs). We are currently working on extending our prototype in this direction.

9 Conclusions and Ongoing Work

In conclusion, we have presented an integrated algorithm for image computation which combines the relative merits of SAT solvers and BDDs, while allowing a dynamic interaction between the two. In addition, the use of a fine-grained CNF formula allowed us to explore the benefits of early quantification in the BDD subproblems. This can potentially find application in purely BDD-based approaches as well.

Apart from extending our prototype to handle other applications such as approximate traversal, invariant checking and CTL model checking, a number of enhancements to the basic image computation strategy are possible. Specifically, we are considering various forms of partitioning including disjunctive partitioning of the reached set, exploiting disjoint partitions in the CNF formula (and maximizing this effect through appropriate variable selection), and partitioning the circuit structurally. The SAT-BDD approach works best when it has a large reached set to bound against. We are experimenting with the use of an under-approximate traversal as a preprocessing step to generate a large reached set, before starting an exact traversal. Similarly, the SAT-BDD framework naturally allows exploration of various combinations of DFS, BFS, and hybrid search techniques targeted at finding bugs.

References

1. P. A. Abdulla, P. Bjesse, and N. Een. Symbolic reachability analysis based on SAT-solvers. In *Tools and Algorithms for the Analysis and Construction of Systems (TACAS)*, 2000.
2. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Analysis and Construction of Systems (TACAS)*, volume 1579 of *Lecture Notes in Computer Science*, 1999.
3. R. K. Brayton et al. VIS: A system for verification and synthesis. In R. Alur and T. Henzinger, editors, *Proceedings of the Internation Conference on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 428–432, June 1996.
4. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug. 1986.
5. J. Burch and V. Singhal. Tight integration of combinational verification methods. In *Proceedings of the International Conference on Computer-Aided Design*, pages 570–576, 1998.

6. J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in symbolic model checking. In *Proceedings of the 28th Design Automation Conference*, pages 403–407, June 1991.
7. J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design*, 13(4):401–424, Apr. 1994.
8. G. Cabodi, P. Camurati, and S. Quer. Improving the efficiency of BDD-based operators by means of partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(5):545–556, May 1999.
9. O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines using symbolic execution. In *Proceedings of the Internatiocal Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 365–373, June 1989.
10. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–205, 1960.
11. D. Geist and I. Beer. Efficient model checking by automatic ordering of transition relation partitions. In *Proceedings of the Internation Conference on Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 299–310, 1994.
12. A. Gupta and P. Ashar. Integrating a Boolean satisfiability checker and BDDs for combinational verification. In *Proceedings of the VLSI Design Conference*, Jan. 1998.
13. T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, Jan. 1992.
14. J. P. Marques-Silva. *Search Algorithms for Satisfiability Problems in Combinational Switching Circuits*. PhD thesis, EECS Department, University of Michigan, May 1995.
15. J. P. Marques-Silva and K. A. Sakallah. Grasp: A new search algorithm for satisfiability. In *Proceedings of the International Conference on Computer-Aided Design*, pages 220–227, Nov. 1996.
16. J. P. Marquez-Silva. Grasp package. http://algos.inesc.pt/~jpms/software.html.
17. I.-H. Moon, J. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *Proceedings of the Design Automation Conference*, pages 23–28, June 2000.
18. A. Narayan, A. J. Isles, J. Jain, R. K. Brayton, and A. Sangiovanni-Vincentelli. Reachability analysis using partitioned ROBDDs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 388–393, 1997.
19. R. K. Ranjan, A. Aziz, R. K. Brayton, B. F. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. In *International Workshop for Logic Synthesis*, May 1995. Lake Tahoe, CA.
20. F. Somenzi et al. CUDD: University of Colorado Decision Diagram Package. http://vlsi.colorado.edu/~fabio/CUDD/.
21. H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using BDDs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 130–133, 1990.
22. P. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining decision diagrams and SAT procedures for efficient symbolic model checking. In *Proceedings of the Internation Conference on Computer-Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 124–138, 2000.