To Split or to Conjoin: The Question in Image Computation*

In-Ho Moon
University of Colorado
Boulder, CO
mooni@colorado.edu

James H. Kukula
Synopsys
Beaverton, OR
kukula@synopsys.com

Kavita Ravi
Cadence
New Providence, NJ
kravi@cadence.com

Fabio Somenzi
University of Colorado
Boulder, CO
fabio@colorado.edu

Abstract

Image computation is the key step in fixpoint computations that are extensively used in model checking. Two techniques have been used for this step: one based on conjunction of the terms of the transition relation, and the other based on recursive case splitting. We discuss when one technique outperforms the other, and consequently formulate a hybrid approach to image computation. Experimental results show that the hybrid algorithm is much more robust than the "pure" algorithms and outperforms both of them in most cases. Our findings also shed light on the remark of several researchers that splitting is especially effective in approximate reachability analysis.

1 Introduction

Symbolic techniques for state space exploration have given new impulse to formal verification by making it possible to analyze systems with very many states. However, practical designs lie outside the grasp of symbolic techniques. Researchers have devised more efficient verification procedures by applying abstractions (e.g., [15]). However, the ultimate success of a verification methodology rests on a balanced combination of techniques that decompose a complex problem into simpler ones, and powerful computational techniques that keep the amount of manual intervention within acceptable limits. In this regard one of the important weaknesses of symbolic state exploration techniques is their lack of robustness. Problem instances of seemingly comparable complexity require vastly different computational resources. In this paper we address the issue of robustness in the image and preimage computations. These computations form the core of the iterations of fixpoint computations, and therefore are at the heart of symbolic model checking algorithms [16].

Image computation consists of finding all the successors of a given set of states S according to a set of transitions T. Preimage computation, on the other hand, is concerned with finding all the predecessors of the given states. In symbolic image computation, the sets of states and the transitions are represented by boolean formulae, which in turn are manipulated in the form of Binary Decision Diagrams (BDDs [3]). The formula representing the transitions is called *transition relation*: It depends in general on three sets of variables (for present states, inputs, and next states), and is true of all triples of a, b, and c, such that there is a transition from a to c, labeled by b. The transition relation of synchronous systems is often given as a set of formulae whose conjunction equals the transition relation. The actual conjunction is often postponed as late as possible in the image computation process to avoid the explosion of the BDD sizes.

Two main methods have been proposed for symbolic image computation. The first is based on conjoining the parts of the transition relation and the formula representing the set of states in an order that tries to minimize the size of the intermediate BDDs. This method goes under the name of *partitioned transition relation* approach. We shall refer to it as to the *conjunction* approach to image computation. The second approach is based on recursive case splitting, by considering either the possible values of an input or present state variable (input splitting), or the values of a next state variable (output splitting). This method is known as *transition function* approach, because for deterministic systems it can be implemented by exclusive manipulation of the state transition functions, without recourse to the next state variables. We shall refer to this second method as to the *splitting* approach to image computation. Preimage computation can also be computed by either conjunction or splitting. For the sake of conciseness, we restrict our discussion to image computation, but our conclusions apply to both problems.

In this paper we study the relative merits of the conjunction and splitting approaches and we argue that the single most important factor determining their performance is the structure of the dependence matrix of the transition relation. This observation explains, for instance, why the splitting method has been traditionally found superior to the conjunction method in approximate reachability analysis. In general, we have found the predictions derived from the dependence matrix quite accurate. Based on this observation, we propose a hybrid algorithm that combines, in a unified framework, conjunction, input and output splitting, and a host of auxiliary techniques that can significantly speed up the computation.

Our technique marks a significant improvement over previous work that combined disjunctive and conjunctive decomposition, because the decision of how much splitting should be done is dynamic. This allows much more splitting to be applied when required (on a per-problem or per-image computation basis). By contrast, a technique that split up to a fixed depth would be either inefficient on problems that are more amenable to conjunction, or inefficient on problems that benefit from extensive splitting.

One significant issue in choosing the proper techniques from a large set that will work best for a given problem instance is the overhead incurred by the decision process. We propose an adaptive approach that relies on trends that are characteristic of the fixpoint computations in which image computation is normally used. For instance, the sets S tend to increase in size with successive image computations, at least for the first few iterations. They may then decrease towards the end. Similar simple observations lead to inexpensive strategies for the application of some of the techniques.

Our initial experimental results show that the hybrid approach outperforms both pure methods in most cases, and is substantially more robust. On examples for which one pure method is the clear winner, while the other does not even complete, the hybrid method

*This work was supported in part by SRC contract 98-DJ-620 and NSF grant CCR-99-71195.

DAC 2000, Los Angeles, California

is capable to follow the correct strategy, and completes in times comparable to the best method. Moreover, positive results have been obtained over a range of problems going from exact reachability analysis, to CTL model checking, passing through approximate reachability analysis.

The remainder of this paper is organized as follows. In Section 2 we recall the main results we need about boolean function manipulation with BDDs. In Section 3 we present a unified treatment of image computation and in Section 4 we discuss the relative merit of splitting and conjoining. Section 5 is devoted to our hybrid image computation algorithm, which is compared to prior work in Section 6. In Section 7 we discuss our experimental result, and in Section 8 we discuss future work and conclude.

2 Preliminaries

BDDs are a popular representation of boolean function. Most algorithms that operate on BDDs are directly based on Boole's Expansion Theorem [1].

Theorem 1 *Let f be a boolean function of variables $x_1, \ldots, x_n$. Then:*

$$f(x_1, \ldots, x_n) = x_1 \wedge f(1, \ldots, x_n) \vee \neg x_1 \wedge f(0, \ldots, x_n).$$

The term $f(1, \ldots, x_n)$ is called the *positive cofactor* of f w.r.t. x_1, and is written f_{x_1}. Likewise, $f_{\neg x_1}$ designates the *negative cofactor*. The *existential quantification* of x_1 from f is defined by $f_{x_1} \vee f_{\neg x_1}$ and enjoys the following important property.

Theorem 2 *Existential quantification distributes over conjunction if only one operand depends on the quantified variable. That is:*

$$\exists x.[f(x,y) \wedge g(y)] = \exists x.[f(x,y)] \wedge g(y). \tag{1}$$

The *constrain* operator [10] (denoted by $\downarrow$) plays a significant role in image computation. It is defined as follows:

Definition 1 *Let $f : B^n \to B^m$ and $g : B^n \to B$ be functions of $x_1, \ldots, x_n$ and let $x_1 < x_2 < \cdots < x_n$ be the variable order. Let the distance between two points y and z in B^n be given by*

$$\Delta(y,z) = \sum_{1 \le i \le n} 2^{n-i} \cdot (y_i + z_i) \bmod 2.$$

Then $(f \downarrow g)(x) = f(\mu_g(x))$, where $\mu_g : B^n \to B^n$ maps each point x in B^n to its nearest neighbor y (according to Δ) such that $g(y) = 1$.

The constrain operator enjoys numerous properties [22, 17]. The following are of particular interest to us:

Theorem 3 *Constrain distributes over boolean connectives. (E.g., $(f \wedge g) \downarrow h = (f \downarrow h) \wedge (g \downarrow h)$.)*

Theorem 4 *If f and g have disjoint support, then $f \downarrow g = f$.*

Theorem 5 *Conjunction can be replaced by* constrain *when all the variables of one of the conjuncts are quantified:*

$$\exists x.[f(x,y) \wedge g(x)] = \exists x.[f(x,y) \downarrow g(x)]. \tag{2}$$

Theorem 6 *The result of* constrain *is constant if and only if the operands are related by containment. Specifically:*

$$\begin{aligned} g \le f \quad &\Leftrightarrow \quad f \downarrow g = 1 \\ g \le \neg f \quad &\Leftrightarrow \quad f \downarrow g = 0. \end{aligned}$$

3 Image Computation

Given a set of present state variables $x = \{x_1, \ldots, x_n\}$, a set of input variables $w = \{w_1, \ldots, w_m\}$, and a set of next state variables $y = \{y_1, \ldots, y_n\}$, we can represent a state transition graph by a pair of boolean formulae $\langle T, I \rangle$, where $T(y,w,x)$ is the *transition relation* and $I(x)$ is the set of initial states.

In this paper we consider the state transition graphs of systems derived from the synchronous composition of subsystems. In this case, the transition relation is obtained by conjoining the transition relations of the subsystems.

$$T(y,w,x) = \bigwedge_{1 \le i \le k} T_i(y,w,x), \tag{3}$$

where the subsets of y variables appearing in each T_i form a partition of the next state variables. If the conjunction is not actually taken, and the transition relation of the synchronous product is represented by a set of relations, the set is called *partitioned* transition relation. A common case occurs when each subsystem is deterministic and contains exactly one state variable. Then we can write:

$$T(y,w,x) = \bigwedge_{1 \le i \le n} T_i(y_i,w,x) = \bigwedge_{1 \le i \le n} (y_i \wedge \delta_i(w,x) \vee \neg y_i \wedge \neg \delta_i(w,x)). \tag{4}$$

where δ_i is the next-state function of the i-th state variable. Each T_i is called in this case a *bit relation*.

The set of all bit relations is called a *fully partitioned* transition relation. *Clustering* may transform a fully partitioned transition relation into the form of Equation 3. If all conjunctions are taken, the result is a single formula called the *monolithic* transition relation.

Image computation is the process of finding the successors of a set of states $S(x)$ in the state transition graph described by $T(y,w,x)$. The image of S according to T is given by

$$\exists x, w. \left[T(y,w,x) \wedge S(x) \right].$$

If T is a *monolithic* transition relation, then one conjunction and one quantification suffice. Most BDD packages provide one procedure that combines these two steps. We will refer to such a procedure as *AndExists*. Furthermore, if T is monolithic the w variables can be quantified from T before the conjunction. In case of repeated image computations, this quantification needs only be done once.

It is seldom the case, however, that T can be conveniently represented by a single BDD, even when the input variables are quantified. A partitioned transition relation often requires much less space than a monolithic one, while a clustered transition relation often leads to faster image computations than a fully partitioned relation.

Image computation with partitioned transition relations can be accomplished by a series of *AndExists* operations. Thanks to Theorem 2, after each conjunction, all variables that appear in only one term can be immediately quantified. This early quantification improves the worst case complexity of *AndExists* (exponential in the number of variables of its operands). The *quantification schedule* problem [12, 20, 13] is the one of devising an effective order in which conjunctions are taken to reduce the peak number of variables in the *AndExists* operations.

It is also possible to apply the expansion theorem to decompose image computation. If v is either an input variable ($v = w_j$) or a present state variable ($v = x_j$), we have:

$$\begin{aligned} \exists x, w. \left[\bigwedge_{1 \le i \le k} T_i \wedge S \right] \quad = \quad & \exists x, w. \left[\bigwedge_{1 \le i \le k} (T_i)_v \wedge S_v \right] \vee \\ & \exists x, w. \left[\bigwedge_{1 \le i \le k} (T_i)_{\neg v} \wedge S_{\neg v} \right]. \end{aligned} \tag{5}$$

This expansion is called *input splitting*. If, on the other hand, the expansion is applied with respect to a next state variable, we obtain *output splitting*:

$$\exists x, w. \left[\bigwedge_{1 \le i \le k} T_i \wedge S \right] = y_j \wedge \exists x, w. \left[\bigwedge_{1 \le i \le k} (T_i)_{y_j} \wedge S_{y_j} \right] \vee \neg y_j \wedge \exists x, w. \left[\bigwedge_{1 \le i \le k} (T_i)_{\neg y_j} \wedge S_{\neg y_j} \right]. \quad (6)$$

When the transition relation is fully partitioned, further manipulation based on Theorem 5 yields:

$$\exists x, w. \left[\bigwedge_{1 \le i \le n} T_i \wedge S \right] = y_j \wedge \exists x, w. \left[\bigwedge_{1 \le i \le n, i \ne j} ((T_i \downarrow S) \downarrow \delta_j) \right] \vee \neg y_j \wedge \exists x, w. \left[\bigwedge_{1 \le i \le n, i \ne j} ((T_i \downarrow S) \downarrow \neg \delta_j) \right]. \quad (7)$$

Input and output splitting were initially devised for the case of fully partitioned transition relations, in which, if one exclusively applies splitting (Equations 5 and 7), the next state variables are not needed, for the images can be directly computed in terms of the x variables. The recursive method that results is known as the *transition function* approach to image computation [10, 11, 7].

The application of the expansion theorem leads to a disjunctive decomposition of image computation. Conjunctive decomposition is easy (thanks to Theorem 2) if the conjuncts can be partitioned so that no variables are shared by the various blocks. Detecting conjunctive decomposition is especially important in the case of the transition function approach. When conjunction of the T_i's is taken, early quantification provides most of the benefits brought by conjunctive decomposition.

Theorem 5 can also be used to eliminate $S(x)$ from the list of conjuncts [22]. (This was done in the derivation of Equation 7.) However, it should be noted that the size of the BDD for $f \downarrow g$ is often comparable to the size of the larger between the BDDs for f and g. Therefore, Theorem 5 should be applied selectively. It should also be kept in mind that the validity of Theorem 3 relies on the variable order being the same for all the computations of *constrain*. This may lead to inefficiencies.

Though in this paper we concentrate on image computation, most of the observations we make apply to preimage computation as well. In particular, it is possible to compute a preimage by either a series of *AndExists* or by splitting on input or output variables. Finding a good order for quantification is somewhat different from image computation [12], but the similarities largely outweigh the differences. In Section 7 we present results for computations based on both images and preimages.

4 Splitting versus Conjoining

The effectiveness of the partitioned representation [4, 22] relative to the monolithic representation relies on three mechanisms.

- The conjunction with $S(x)$ may help keep the size of the BDDs small.
- Early quantification can be applied to eliminate variables from the BDDs as soon as possible.
- The image computation problem can be decomposed, thus avoiding the computation of a single BDD for T.

In this section we discuss the relative effectiveness of conjoining and splitting. We shall see that the transition relation at hand has a strong influence on which technique is better.

Definition 2 *The* dependence matrix *of an ordered set of functions $(f_1, \ldots, f_m)$ depending on variables $x_1, \ldots, x_n$ is a matrix D with m rows and n columns such that $D(i, j) = 1$ if function f_i depends on variable x_j, and $D(i, j) = 0$ otherwise.*

The dependence matrix of the set $(T_1, \ldots, T_k, S)$ provides insight on how image computation proceeds [14]. As columns we shall consider only the variables that must be quantified (x and w). If there exists a permutation of rows and columns such that the dependence matrix can be put in *block-diagonal* form, then conjunctive decomposition due to disjoint support is possible. Sparse matrices yield such decompositions more easily than dense matrices. This is a favorable case: If the blocks on the diagonal are small enough, even problems with many state variables can be solved in little time and memory.

If the dependence matrix can be put in *block-triangular* form with a large number of blocks, then there is a good quantification schedule. Problems with this structure are particularly amenable to the application of conjunction. Figure 1 shows the dependence matrix of a 32-bit sequential multiplier, which is a very easy case for conjunction-based image computation. When instead splitting is applied to such problems, the performance of the image cache is critical due to the possible occurrence of identical sub-problems in the difference cases of the split. This in order dictates that the splitting variables be chosen in a fixed order along the different recursion paths to improve image cache hits. This is indeed the case of the sequential multiplier of Figure 1.

When the dependence matrix is close to full, there is no good quantification schedule. Splitting, on the other hand, may be effective, because it may increase the sparsity of the matrix. In the case of the 32-bit rotator whose dependence matrix is shown in Figure 2, the matrix causes considerable difficulty to conjunction-based image computation. The conjunction of the corresponding bit relations results in an exponential blow-up of the BDDs (none of the variables can be quantified early); however, it can be solved easily with splitting. Indeed, if the splitting variable is one of the inputs controlling the amount of rotation, then splitting almost halves the support of each function.

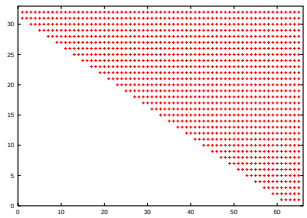

Figure 1: Dependence matrix for a 32-bit sequential multiplier. This matrix, like the ones of Figures 2 and 3, is for the transition relation before clustering and ordering of the bit relations.

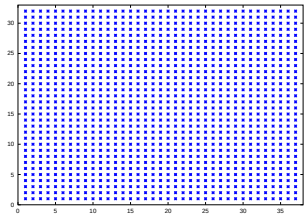

Figure 2: Dependence matrix for a 32-bit rotator.

Approximate reachability analysis [8, 18] is based on the partitioning of the circuit to be traversed into a collection of submachines. It has been observed experimentally that the *transition function* approach to image computation (that is, pure splitting and no

use of y variables) outperforms the pure conjunction approach in this application. An examination of the dependence matrices corroborates the conjecture that the advantage enjoyed by splitting is due to the lack of a good block-triangular form for the submachines. Figure 3 contrasts the dependence matrices for the entire circuit and one of its submachines for the case of *hw_top*. The almost full dependence matrix shown is typical of most submachines in this and other examples. To some extent, this is a consequence of the partitioning algorithms used to generate the submachines [9], which tends to cluster state variables that interact strongly.

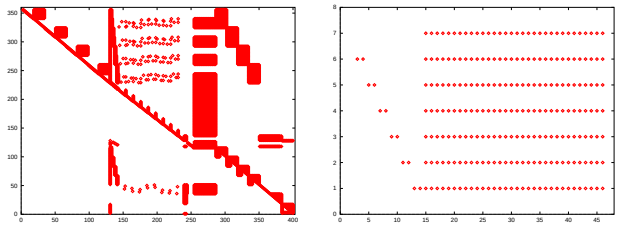

Figure 3: Dependence matrix for Circuit *hw_top* (left) and one of its submachines used in approximate traversal (right).

5 Hybrid Image Computation Algorithm

The analysis of Section 4 suggests that an effective image computation procedure should neither rely exclusively on the conjunction of the transition relation parts, nor employ only splitting. Several auxiliary techniques available to speed up the computation—like the use of Theorem 5 discussed at the end of Section 3—tend to have opposite effects on different problem instances. The challenge is therefore to devise a flexible strategy that deploys the right technique based on the nature of the problem at hand. Since this nature is affected to a large extent by the structure of the transition relation as displayed by the dependence matrix, it is reasonable to assume that in the course of a fixpoint computation trends can be identified to guide the application of a specific technique.

Consider, for instance, the aforementioned application of Theorem 5. As the size of the BDD for $S(x)$ increases in successive fixpoint iterations, the effectiveness of eliminating it by using *constrain* diminishes. This suggests that this algorithmic feature should be enabled at the beginning of the fixpoint computation, and disabled as soon as it is found counterproductive, meaning that the resulting BDD size after *constrain* can be very large compared to the size of the original. On the other hand, when input splitting occurs the size of $S(x)$ usually decreases, thus increasing the probability of success of application of Theorem 5. Our approach is therefore to maintain a minimum depth at which the use of *constrain* should be attempted, and update it based on the success of the technique.

The same approach of keeping track of the minimum recursion depth at which a particular technique should be applied can be used for another technique related to Theorem 5. Suppose $S(x) = c(x) \cdot S(x)$ where $c(x)$ is a *cube*, that is, a conjunction of literals. It is then easily seen that one can constrain all conjuncts with $c(x)$, including $S(x)$. Since $c(x)$ is a cube, *constrain* guarantees that $T_i(y,w,x) \downarrow c(x)$ will have a smaller BDD than $T_i(y,w,x)$. Furthermore, the variables appearing in $c(x)$ will be eliminated from the computation. The smallest $c(x) \geq S(x)$ can be identified in a single pass on the BDD for $S(x)$. However, when $c(x) = 1$ the effort involved in computing it is wasted. Also, the probability of finding $c(x) \neq 1$ decreases with the iteration of the fixpoint, and increases with the recursion depth in case of splitting. By maintaining a minimum depth at which the technique should be applied per every image computation, the check will be quickly turned off if it does not help.

The single most important decision in our hybrid image computation procedure is whether to split or conjoin. Our algorithm is once again based on an adaptive approach that takes into account the structure of the problem instance as primary decision criterion, and relies on the history of the computation to limit the decision overhead.

Suppose the dependence matrix D has m columns and n rows. Suppose also that the rows $d_1, \ldots, d_n$ are permuted in such a way that the conjunction order is bottom-up. (That is, d_n is first conjoined to d_{n-1}; the result is then conjoined to d_{n-2}, and so on.) If $l_j(h_j)$ is the smallest(largest) index i in column j such that $D_{ij} = 1$, respectively, then variable x_j can be quantified as soon as d_i is conjoined. Assuming that one row of D is $S(x)$, we can define the *normalized average lifetime* of the variables in D as

$$\lambda = \frac{\sum_{1 \leq j \leq m}(h_j - l_j + 1)}{n \cdot m}.$$

Clearly, the value of λ depends heavily on the structure of the dependence matrix. A full matrix has a $\lambda = 1$. A lower-triangular D has $\lambda = 0.5$. We start with the matrix as close to the lower triangular form as possible i.e., suited to conjunction. Subsequently, we make decisions to split or conjoin based on the value of λ. A matrix close to the lower triangular form can be derived in several ways, including using heuristics to derive a quantification schedule. Our approach uses the matrix provided by the quantification schedule heuristic of Ranjan *et al.* [20].

Both the value of λ and its rate of change as a result of splitting are relevant in deciding whether to split or conjoin. Our algorithm chooses conjunction whenever $\lambda \leq 0.5 + \varepsilon_1$. The value of ε_1 in our current implementation is 0.1. If on the other hand, $\lambda > 0.5 + \varepsilon_1$ it may be convenient to split. The effectiveness of input splitting is also predicated on its ability to increase the sparsity of D, and specifically to decrease λ. Therefore, except at the top of the recursion, splitting is abandoned in favor of conjunction if λ does not decrease by at least ε_2 as a result of splitting or if the BDD sizes do not decrease significantly. The value of ε_2 in our current implementation is 0.1.

To keep the overhead low, the transition relation is clustered only once, before the fixpoint computation starts. That is, the set of rows of the dependence matrix denote the same clusters through the entire computation. Every time splitting takes place, however, the rows are reordered to try to improve the quantification schedule and to compute more accurate λ. Since the cost of this recomputation is negligible and the maximum number of splits is restricted, we perform this recomputation with every split.

Another reason to switch from splitting to conjunction is that the BDDs for the transition relation may have simplified to the point that there is no serious danger of size explosion. Finally, if the recursion depth reaches a given limit, the algorithm switches to conjunction. This dynamic approach to deciding whether to split or to conjoin has an advantage over fixing the number of splitting variables a priori; it allows a variable splitting depth—which is useful in some cases—without the attendant overhead for the problem instances more amenable to conjunction.

The maximum recursion depth may be dynamically adjusted starting from a fixed limit. If splitting at depth k yields uniformly poor results during one image computation (as measured by the values of λ) then the maximum depth may be decreased for the next image computation. We have not yet experimented with this dynamic approach.

Once our algorithm switches from splitting to conjunction it does not revert to splitting. The dependence matrix is unlikely to change much in structure as a result of the conjunctions being taken. Hence, our primary criteria for deciding which technique to apply are not expected to recommend a return to splitting. The trade-off, in this case, is between the overhead of making this choice dynamically and the ability to revert to splitting when the intermediate results of conjunction grow too large.

When the decision to split has been taken for a given problem or su**b**problem, a choice is made between input splitting and output splitting. The guiding principle is that output splitting with respect to bit relation $T_i(y_i, w, x)$ is especially favorable if $\delta_i(w, x)$ is a cube or the complement of a cube. In that case, one of the two subproblems is significantly simplified. If that condition is not met, then input splitting is preferred.

The choice of the splitting input variable is done in up to three stages. First the variables that appear in the support of the largest number of conjuncts are selected [14]. If they are more than one, then a first tie-breaker based on estimating the simplification of $T(y, w, x)$ and $S(x)$ as a result of splitting [6] is applied. If more than one variable yields the same estimated amount of simplification, then the one closest to the top of the current variable order is selected.

Since we do not use a fixed splitting order along all recursion paths, and we switch to conjunction at a depth bounded by a rather small number (8 in our implementation), we do not cache the subimages computed by splitting. The image cache adds a noninsignificant overhead and because the splits are few, it is likely that the image cache hits may actually occur in the BDD cache also. In our experience, the absence of the image cache has not affected performance negatively.

6 Comparison to Prior Work

Our work is related to the disjunctive partitioning approach of [6, 5, 19]. The common trait is the disjunctive decomposition of the problem. The differences are in the criteria applied to balance conjunction and decomposition. Our algorithm incorporates the analysis of the structure of the problem instance and adapts to the changing conditions as the fixpoint computation proceeds. It is less focused on simply reducing the size of the BDDs, though it does strive to do so. The decision to split based on *normalized average lifetime* increases the chances of support reduction also (affecting worst case BDD sizes in the *AndExists* operations of image computation). Finally, our framework accommodates a large number of optimization techniques for image computation—for instance output splitting—in a uniform manner.

The difficulty posed by traversal of the rotator circuit of Figure 2 is addressed in [21] by the use of hints. Our technique is not as general as symbolic guided search based on hints, but is fully automatic and works equally well on *rotator* and circuits in which departure from breadth-first traversal is not necessary to avoid memory bottleneck. It is obviously possible to combine our hybrid image computation technique with symbolic guided search or other strategies that combine depth-first and breadth-first symbolic searches.

7 Experimental Results

We implemented the hybrid algorithm for image computation in VIS [2]. Experiments were conducted on a 400 MHz Pentium II machine with 1GB of RAM running Linux. We have applied the hybrid algorithm to exact reachability analysis, approximate reachability analysis, and model checking. We compare the hybrid method (denoted by Hybrid) to the conjunction approach using transition relation (denoted by TR) and the splitting approach using transition function (denoted by TF). We used the heuristic of Ranjan *et al.* [20] in VIS as pure conjunction method and used the transition function method in Veritas [8] as pure splitting method. In these experiments, we turned on dynamic variable reordering and we set a data size limit of 750MB. In the hybrid method, we did not used the image cache, whereas the splitting method used it. The splitting method used only input splitting.

Table 1 compares the three methods in the context of exact reachability analysis. The first column shows circuit names and the second column shows the number of latches in each circuit. The rest of the columns compare the three methods in terms of CPU time and peak number of live BDD nodes during each run. In the table, '—' indicates memory out.

As shown in Section 4, the 32-bit rotator does require the splitting method: the BDDs blow up right away with the conjunction method. On the other hand, the 32-bit multiplier really needs the conjunction method. In this case, the splitting method takes a lot more time than the conjunction method. However, for both circuits, the hybrid method is very comparable to the best of the other two.

In the case of the splitting method for *mult32a* and *mm30*, if we always choose the top variable in the BDD as splitting variable, we use the image cache and disable dynamic variable reordering, we can get much better results; (1.6s, 5.7K) and (5.8s, 45.3K) respectively. This is because the BDDs of those circuits have many reconvergent paths which result in many identical subproblems.

The splitting method exhausts memory in most cases due to the overhead of the image cache. In those cases, if we do not use the image cache, we get time-out instead of memory-out due to too many recursive calls. This means that the splitting method is not suitable for exact reachability analysis in general. However, the hybrid method combining the conjunction and splitting method outperformed the conjunction method in most cases in time. Moreover, in the cases of *s3271* and *am2901*, which are known as hard circuits for exact reachability analysis, the conjunction method exhausted memory, whereas the hybrid method completed. We believe that this was possible because the dynamic decision whether to split or conjoin on our hybrid method plays a significant role. In terms of memory usage, the number of peak live nodes of the hybrid method was about same as the one of the conjunction method except for *elevator*, *bpb*, and *s1269*. These data support the claim that even though the hybrid method is not always the best, it is much more robust than the splitting and conjunction methods.

Circuit	FF	Time (seconds)			Peak Live Nodes (K)		
		TR	TF	Hybrid	TR	TF	Hybrid
rotator	64	—	0.1	4.4	—	2.6	24.5
mult32a	32	0.1	31.5	0.3	5.7	24.7	5.7
mm30	90	34.6	—	30.9	81.2	—	81.2
elevator	50	571.0	—	815.0	518.4	—	604.7
bpb	36	720.1	—	203.4	51.2	—	67.6
cps	231	5162.4	—	4735.6	1423.1	—	1449.2
soap	140	143.8	—	180.0	123.1	—	122.9
s1269	37	4844.4	—	891.0	3648.1	—	1576.1
s1512	57	3192.0	—	2016.0	207.8	—	211.6
s3271	116	—	—	4833.5	—	—	1482.1
s3330	132	74303.2	—	10316.1	3669.2	—	3268.4
am2901	68	—	—	15518.4	—	—	1480.6
s5378opt	121	11913.4	—	1767.4	231.5	—	383.8

Table 1: Exact reachability analysis.

Table 2 compares the three methods applied to approximate reachability analysis using LMBM [18]. In this experiment, we used the partitions of [8] for *s1423, s5378, s13207, s15850, s35932* and *s38584*. For the remaining circuits, we set the group size to 8 for automatic state space decomposition [9]. As discussed in Section 4, Table 2 shows that the splitting approach outperforms the conjunction approach in both time and space in most cases. However, as the size of designs grows large (like in *s38417* and *avq*), the splitting approach is likely to exhaust memory due to the overhead of the image cache. Again, if we do not use the image cache in this case, we get time-out instead of memory-out. On the other hand, the hybrid method also outperformed the conjunction method in time in all cases, and the hybrid method is very comparable to the splitting method and in some cases the hybrid method was even faster than the splitting method. Especially for *avq*, only the hy-

brid method was able to complete, whereas the other two methods exhaust memory.

In terms of memory usage in approximate reachability analysis, the splitting method has much fewer peak live BDD nodes than the conjunction method and, even though there are some cases in which the hybrid method takes fewer peak live nodes than the splitting method, in general, we can see that the memory usage of the hybrid method is in between the splitting and conjunction methods. Also, by the nature of the transition function method, which does not need next state variables, we can see that the transition function method consumes much less memory than the transition relation method unless the overhead of the image cache is too high.

Circuit	FF	Time (seconds)			Peak Live Nodes (K)		
		TR	TF	Hybrid	TR	TF	Hybrid
s1423	74	5.0	14.5	4.0	32.2	19.8	34.5
am2901	68	329.7	45.2	34.0	337.9	76.3	185.9
hw_top	356	70.2	32.3	20.3	60.3	60.3	20.9
trainflat	866	435.9	132.1	134.9	9690.1	649.1	620.5
s5378	164	21.9	14.4	9.5	44.5	11.6	37.8
s13207	669	477.6	8.9	34.8	2099.2	20.8	1074.5
s15850	597	495.2	13.4	34.2	118.0	92.6	47.3
s35932	1728	2209.2	110.4	256.0	25.9	29.3	39.9
s38584	1452	2554.9	193.7	551.7	602.2	79.5	76.4
s38417	1636	4235.5	—	709.4	1376.5	—	1374.3
avq	3705	—	—	9482.8	—	—	2381.1

Table 2: Approximate reachability analysis.

We have also implemented preimage computation using the hybrid method. Table 3 shows that the hybrid method in model checking also outperforms the conjunction method in time and space. These are preliminary results obtained on a limited number of cases.

Circuit	FF	Time (seconds)			Peak Live Nodes (K)		
		TR	TF	Hybrid	TR	TF	Hybrid
cps	231	696.0	—	278.6	544.0	—	285.2
ethernet	80	5353.8	—	802.5	455.1	—	388.0
hw_top	356	580.0	—	361.4	165.3	—	132.2

Table 3: Model checking.

8 Conclusions

Image and preimage computation are the crucial steps in model checking. They have been traditionally performed by either iterated conjunction or recursive splitting.

In this paper we have presented a hybrid image computation algorithm that combines the conjunction method with the splitting method. We have also presented a robust and adaptive approach to deciding whether to split or conjoin for a given problem or subproblem, based on the dependence matrix.

By using the dependence matrix, we can predict whether the conjunction method will outperform the splitting method or vice versa. We are thus able to explain why the splitting method outperforms the conjunction method in approximate reachability in most cases.

Our algorithm provides a framework in which various optimization techniques can be applied uniformly and with low overhead. Our experimental results show that the hybrid method outperforms both pure methods in most cases for exact reachability analysis, approximate reachability analysis, and model checking.

References

[1] G. Boole. *An Investigation of the Laws of Thought*. Walton, London, 1854. (Reprinted by Dover Books, New York, 1954).

[2] R. K. Brayton et al. VIS. In *Formal Methods in Computer Aided Design*, pages 248–256. Springer-Verlag, Berlin, November 1996. LNCS 1166.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[4] J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in symbolic model checking. In *Proceedings of the Design Automation Conference*, pages 403–407, San Francisco, CA, June 1991.

[5] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer. Disjunctive partitionining and partial iterative squaring: An effective approach for symbolic traversal of large circuits. In *Proceedings of the Design Automation Conference*, pages 728–733, Anaheim, CA, June 1997.

[6] G. Cabodi, P. Camurati, and S. Quer. Improved reachability analysis of large finite state machines. In *Proceedings of the International Conference on Computer-Aided Design*, pages 354–360, Santa Clara, CA, November 1996.

[7] H. Cho, G. D. Hachtel, S.-W. Jeong, B. Plessier, E. Schwarz, and F. Somenzi. ATPG aspects of FSM verification. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 134–137, November 1990.

[8] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for approximate FSM traversal based on state space decomposition. *IEEE Transactions on Computer-Aided Design*, 15(12):1465–1478, December 1996.

[9] H. Cho, G. D. Hachtel, E. Macii, M. Poncino, and F. Somenzi. Automatic state space decomposition for approximate FSM traversal based on circuit analysis. *IEEE Transactions on Computer-Aided Design*, 15(12):1451–1464, December 1996.

[10] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using boolean functional vectors. In L. Claesen, editor, *Proceedings IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, Leuven, Belgium, November 1989.

[11] O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 126–129, November 1990.

[12] D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation partitions. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV'94)*, pages 299–310, Berlin, 1994. Springer-Verlag. LNCS 818.

[13] R. Hojati, S. C. Krishnan, and R. K. Brayton. Early quantification and partitioned transition relations. In *Proceedings of the International Conference on Computer Design*, pages 12–19, Austin, TX, October 1996.

[14] S.-W. Jeong, B. Plessier, G. D. Hachtel, and F. Somenzi. Variable ordering and selection for FSM traversal. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 476–479, Santa Clara, CA, November 1991.

[15] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, Princeton, NJ, 1994.

[16] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 1994.

[17] K. L. McMillan. A conjunctively decomposed boolean representation for symbolic model checking. In R. Alur and T. A. Henzinger, editors, *8th Conference on Computer Aided Verification (CAV'96)*, pages 13–25. Springer-Verlag, Berlin, August 1996. LNCS 1102.

[18] I.-H. Moon, J. Kukula, T. Shiple, and F. Somenzi. Least fixpoint approximations for reachability analysis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 41–44, San Jose, CA, November 1999.

[19] A. Narayan, A. J. Isles, J. Jain, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Reachability analysis using partitioned ROBDDs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 388–393, November 1997.

[20] R. K. Ranjan, A. Aziz, R. K. Brayton, B. F. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. Presented at IWLS95, Lake Tahoe, CA., May 1995.

[21] K. Ravi and F. Somenzi. Hints to accelerate symbolic traversal. In *Correct Hardware Design and Verification Methods (CHARME'99)*, pages 250–264, Berlin, September 1999. Springer-Verlag. LNCS 1703.

[22] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDD's. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 130–133, November 1990.