A Conjunctively Decomposed Boolean Representation for Symbolic Model Checking

K. L. McMillan

Cadence Berkeley Labs
1919 Addison St., suite 303
Berkeley, CA 94704-1144
mcmillan@cadence.com

Abstract. A canonical boolean representation is proposed, which decomposes a function into the conjunction of a sequence of components, based on a fixed variable order. The components can be represented in OBDD form. Algorithms for boolean operations and quantification are presented allowing the representation to be used for symbolic model checking. The decomposed form has a number of useful properties that OBDD's lack. For example, the size of conjunction of two independent functions is the sum of the sizes of the functions. The representation also factors out dependent variables, in the sense that a variable that is determined by the previous variables in the variable order appears in only one component of the decomposition. An example of verifying equivalence of sequential circuits is used to show the potential advantage of the decomposed representation over OBDD's.

1 Introduction

Symbolic model checking, and related finite-state verification techniques use heuristically compact boolean representations, such as ordered binary decision diagrams (OBDD's), to implicitly represent sets and relations (notably the transition relation of a model, and its set of reachable states). The implicit representation may be compact even thought the number of states or transitions is very large, thus allowing systems with very large state spaces to be verified automatically. However, in many cases the OBDD representation is not compact. To a first approximation, the OBDD representing a set of states can be thought of as a finite state automaton that reads the values of the state variables in some fixed order, and finally accepts or rejects the given valuation. Figuratively speaking, this automaton must "remember" some amount of information about the variables seen so far, in order to decide whether the remaining variable assignments are consistent with those already seen. Hence, to obtain a compact representation, the variable order must be such that the mutual information across any cut through the order is small. This implies that state variables that strongly correlate each other must be nearby in the variable order. Often, however, this is not possible. For example, in a protocol, a state variable representing the contents of a message buffer is likely to be correlated with both the state of the sender and the state of the receiver. Since all pairs of senders and receivers cannot generally be made close in the variable order, there is no suitable place for the variable representing the message buffer.

In other cases, the relationships between variables are not fixed, but vary according to some global control information. For example, suppose we wish to verify that two hardware implementations of a bounded FIFO queue are equivalent (see figure 1). This can be done by building a single model in which the two implementations run in parallel, and verifying that the outputs always agree. Let one implementation be a "shift register", in which the most recent item is always stored in location 0, and all items shift over when a new item is inserted. Let the other implementation be a "ring buffer", where a "head pointer" points to the oldest item, and the items themselves remain fixed. Given the state of the head pointer, there is a one-to-one correspondence between locations in the two implementations. However, since the head pointer is not fixed, we cannot fix an OBDD variable order that will put related state variables together.

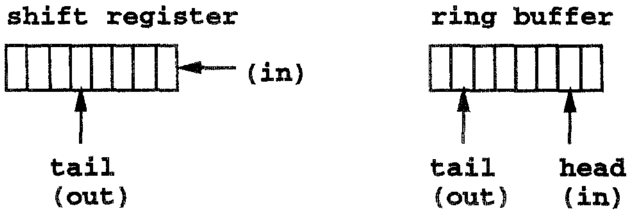

Fig. 1. Two implementations of FIFO queue.

This paper introduces a canonical boolean representation that may be compact in such cases, where OBDD's are not. The intuition behind this representation is that many state variables, such as the message buffers in a protocol, or the data items in the ring buffer, have a property of "conditional independence". That is, once a core set of state variables is fixed, the remaining variables are not mutually correlated. For example, once we the contents of the shift register and the head pointer are fixed, the contents of the ring buffer are determined, and hence uncorrelated. Similarly, the contents of the message buffers in a protocol may provide no mutual information, once the states of the communicating processes are fixed. The representation introduced here decomposes the representation of a boolean function into the conjunction of sequence of components. Each component, which may be represented as an OBDD, fixes the possible values of just one state variable, *given* a feasible assignment to the previous variables. The variable order used for this decomposition may be distinct from the OBDD variable order. The decomposed form has the property that conditionally independent variables are "factored out" into separate components, thus eliminating the need to find a suitable place for these variables in a global OBDD variable order. Among other things, this implies that the conjunction of functions with independent support is additive (not true for OBDD's), as is the conjunction of components in the transition relation of a state machine.

There are a number of examples of conjunctive forms in the literature. Hu and Dill use a technique of checking conjunctive properties where fixed points are computed in parallel, and each conjunct is used to simplify the other conjuncts at each iteration [HD93]. This can yield a more compact representation than an explicit conjunction, but the original decomposition of the problem must be provided by the user. Also, the representation is not canonical, as it is here. Burch and Long use implicitly conjoined transition relations, but do not decompose the representation of the set of reached states, as we do here [BCL91]. Their representation is also not canonical. Jain also describes a canonical disjunctive representation [JABF92], which has a conjunctive dual. It is not directly related to the current method, however, as is it obtained by dividing the truth table into *ad hoc* regions and using one OBDD for each region.

This paper is organized as follows: Section 2 defines the decomposed representation, and proves some useful theorems about the size of the representation for certain classes of functions. Section 3 introduces algorithms for conjunction, disjunction and existential quantification (projection) on the decomposed form. Section 4 discusses model checking using the above algorithms, and provides performance results for verifying the equivalence of the two FIFO queue implementations mentioned above. We find that as the depth of the queues is increased, the size of the decomposed representation for the reachable states increases quadradically, while the OBDD representation increases exponentially.

In this paper, most of the proofs have been omitted due to space limitations.

2 Conjunctive decompositions

Let f be a boolean function of independent boolean variables $V = (v_1, \ldots, v_n)$. We will use the notation $f^{(i)}$, where $0 \leq i \leq n$, to stand for the projection of f onto $(v_1, \ldots, v_i)$. That is,

$$f^{(i)} = \exists(v_{i+1}, \ldots, v_n).\ f$$

In addition, we will use the notation $f|g$, where f and g are two boolean functions, denote the "generalized cofactor" of f relative to g. This function, which can be read as "f given g", agrees with f whenever g is true [CBM89, TSL+90]. Those values where g is false are mapped to the "nearest" point where g is true, according to a distance measure on truth assignments. Thus, if two functions agree wherever g holds, then their cofactors relative to g are equal:

$$f \wedge g = f' \wedge g \quad \text{iff} \quad f|g = f'|g$$

We will use the projection and cofactor operations to decompose a boolean function f into a vector of boolean functions $(f_1, \ldots, f_n)$, where

$$f_i = f^{(i)}|f^{(i-1)}$$

Intuitively, the component f_i determines the set of possible values of variable v_i, *given* a feasible evaluation of the variables $(v_1, \ldots, v_{i-1})$. We will show that the function f is equal to the conjunction of the components f_i:

$$f = \bigwedge_{i=1}^{n} f_i$$

2.1 Generalized cofactor

If f and g are two boolean functions of boolean variables $(v_1, \ldots, v_{i-1})$, then $f|g$ is a boolean function whose value is obtained for a given truth assignment x by finding the "nearest" truth assignment to x that satisfies g, and evaluating f at this point. For this purpose, the distance between two truth assignments x and y is determined by treating their boolean difference (exclusive-or) as a binary number. To be more precise,

Definition 1. Let A be the set of truth assignments $V \rightarrow \{0, 1\}$, and let $W = (w_1, \ldots, w_n)$ be a permutation of V. For any $x, y \in A$, let

$$d(x, y) = \Sigma_{i=1}^{n} 2^{n-i}(x(w_i) \oplus y(w_i))$$

Notice that we have an arbitrary choice of the order W on the variables that defines the distance between truth assignments. Also note for future reference that we have weighted the variables so that w_1 is the most significant, and w_n is the least significant.

Definition 2. Let B be the boolean algebra 2^A. For any $x \in A$, and $g \in B$, where $g \not\equiv 0$, let $x \rightarrow g$ be the unique $y \in g$ minimizing $d(x, y)$.

That is, $x \rightarrow g$ is the nearest point to x that satisfies g. Note that $x \rightarrow g$ is uniquely defined for $g \not\equiv 0$, because the boolean difference between x and any other truth assignment is a unique number. This lets us define the generalized cofactor as follows:

Definition 3. For any $f, g \in B$, and any $x \in A$:

- if $g \not\equiv 0$, then $(f|g)(x) = f(x \rightarrow g)$,
- else $f|g = 0$.

As an example, suppose that $W = (v_1, v_2, v_3)$, that $f = v_3$, $g = (\neg v_1) \wedge (v_2 \vee v_3)$, and that we want to evaluate $f|g$ at the truth assignment $x = (1, 0, 0)$. The truth assignments satisfying g are $(0, 0, 1)$, $(0, 1, 0)$ and $(0, 1, 1)$, of which the nearest to x is $y = (0, 0, 1)$, yielding a distance of $d(x, y) = 5$. The value of $(f|g)(x)$ is thus $f(y) = 1$.

We note that generalized cofactor, as defined above, is exactly the "constrain" operator on OBDD's [CBM89] in the special case when the OBDD variable order is W. In the sequel, however, we will not assume that this is the case.

2.2 Properties of generalized cofactor

We will rely on a variety of properties of the generalized cofactor operation in defining the conjunctive decomposition and in constructing algorithms on decompositions. One very important property of $f|g$ is that it agrees with f everywhere that g is true. Another is that, if two function f and f' agree wherever g is true, then $f|g = f'|g$. That is, $f|g$ is independent of the value of f anywhere that g is false. Letting juxtaposition denote conjunction, and $|$ associate to the left, we also have:

Theorem 4. *1.* $fg = f'g$ iff $f|g = f'|g$.
2. $(f|g)g = fg$
3. *if* $g \neq 0$, *then* $g|g = 1$
4. $f|g|g = f|g$

We also note that generalized cofactor distributes over pointwise operators:

Theorem 5. *For any operator* $\cdot$, *such that* $(f \cdot g)(x) = f(x) \cdot g(x)$*:*

$$(f \cdot g)|h = (f|h) \cdot (g|h)$$

The following theorem is key to the algorithms on conjunctive decompositions, since it allows us, in certain cases, to cofactor relative to a conjunction of functions without explicitly forming the conjunction:

Theorem 6. *For any* $f, g, h \in B$, *if* $g = g|h$, *then* $f|(gh) = f|h|g$.

The name "generalized cofactor" derives from the following property [TSL+90]:

Theorem 7 Touati, et al.. *For any* $f \in B$ *and* $v_i \in V$,

- $f|v_i = f|_{v_i=1}$
- $f|\bar{v_i} = f|_{v_i=0}$

We say that a function f depends on v_i when $f|_{v_i=0} \neq f|_{v_i=1}$. The support of a function is the set of variables on which it depends. Two functions are said to be independent when their supports are disjoint. When two functions are independent, then cofactoring one by the other has no effect:

Theorem 8. *If* f *and* g *have independent support, then* $f|g = f$.

An immediate corollary of this result and theorem 6 is that cofactoring with respect to two independent functions can be done in either order, without affecting the result:

Corollary 9. *If* g *and* h *have independent support, then* $f|(gh) = f|g|h = f|h|g$.

In addition, we can show that projection distributes over cofactor in the much the same way it distributes over conjunction:

Corollary 10. *If* g *is independent of* v_i, *then* $\exists v_i.(f|g) = (\exists v_i.f)|g$.

2.3 Definition of decomposition

We are now ready to define our canonical conjunctive decomposition of a boolean function:

Definition 11. For all $f \in B$, for all $1 \leq i < n$, let $f_i = f^{(i)} | f^{(i-1)}$

We will refer to the functions $(f_1, \ldots, f_n)$ as the *components* of f (relative to V and W). We now show that a function is equal to the conjunction of its components:

Theorem 12. $f = \wedge_{i=1}^{n} f_i$

Proof. We take as our inductive hypothesis that $f^{(j)} = \wedge_{i=1}^{j} f_i$, for all $1 \leq j \leq n$. For the case where f is identical to false, this clearly holds, since all the components f_i are also false. Otherwise, in the base case we have $f_1 = f^{(1)} | f^{(0)} = f^{(1)} | 1 = f^{(1)}$. For the inductive step we have:

$$\begin{aligned}
\wedge_{i=1}^{j} f_i &= (\wedge_{i=1}^{j-1} f_i) \wedge f_j & (1)\\
&= f^{(j-1)} \wedge (f^{(j)} | f^{(j-1)}) & (2)\\
&= f^{(j-1)} \wedge f^{(j)} & (3)\\
&= f^{(j)} & (4)
\end{aligned}$$

Note equation 3 is a case of theorem 4, part 2. That is, $f^{(j)} | f^{(j-1)}$ agrees with $f^{(j)}$ where $f^{(j-1)}$ is true.

Theorem 12 implies that the vector $(f_1, \ldots, f_n)$ is a canonical representation of f, given a fixed V and W. That is, each function has exactly one decomposition, and no two functions have the same decomposition.

There are a number of useful facts about this representation, independent of the component representation and of the choice of permutation W, that defines the generalized cofactor operation. For example, if a function $f \neq 0$ does not depend on some variable v_i, then the corresponding component f_i is identical to true. That is, if f is independent of v_i, then $f^{(i)} = f^{(i-1)}$. Hence $f_i = f^{(i)} | f^{(i)} = 1$, by theorem 4. More generally, we can show that the the ith component of f constrains only variable v_i. That is:

Theorem 13. *If $f \neq 0$, then $\exists v_i . f_i = 1$.*

2.4 Decompositions and disjointness

If two functions f and g have disjoint support, then the components of their conjunction can be obtained by simply taking the conjunction of the corresponding components of f and g, regardless of V or W. Since disjointness implies that every component must be identically true in either f or g or both, it follows that the size of the conjunction is less than or equal to the sum of the sizes of f and g.

Theorem 14. *If f and g have independent support, then*

- *$(fg)_i = f_i$ when f depends on v_i, and*

- *$(fg)_i = g_i$ when g depends on v_i, and*
- *otherwise $(fg)_i = 1$.*

From the above, it follows immediately that the size of fg is bounded by the sum of the sizes of f and g. This result is independent of the underlying representation of the components.

Corollary 15. *If f and g have independent support, then*

$$\Sigma_i|(fg)_i| \leq \Sigma_i|f_i| + \Sigma_i|g_i|$$

It is worth noting that the OBDD representation [Bry86] has this property only in case the OBDD variable order separates the supports of f and g.

2.5 Decompositions and dependent variables

We now consider the special case where the permutation W is the identity (that is, the order of the components f_i is the same as the order that determines the distance measure for generalized cofactor). In this case, if a given variable v_i is functionally determined by its predecessors $v_1 \ldots v_{i-1}$ in the variable order, then we can show that variable v_i appears only in component f_i.

Definition 16. Given a function f, a variable v_i is *functionally determined* by a set of variables $S \subseteq V$ when any two truth assignments agreeing on S must also agree on v_i. If this condition holds, we write $f : S \rightarrow v_i$.

Theorem 17. *If $W = (v_1, \ldots, v_n)$ and $f : (v_1, \ldots, v_{i-1}) \rightarrow v_i$, then f_j depends on v_i only if $j = i$.*

The fact that the decomposed representation is capable of factoring out dependent variables is useful for verifying certain kinds of sequential circuits, as we will observe. It is also a heuristic argument for using $W = V$ in practice.

2.6 Decompositions and conditional independence

The following result generalizes the previous results on independent functions and dependent variables. We will say two variables are *conditionally independent*, relative to a function f, when fixing the value of the preceding variables in the order makes the choice of values of the two variables independent. For example, suppose the function f is $(a \Rightarrow b)(a \Rightarrow c)$. If we fix the value of a, then our choices for b and c become independent. Assuming that the variable order W is (a, b, c), it follows that b and c are conditionally independent. From this we can infer that b occurs only in component f_2, while c occurs only component f_3. That is, conditionally independent variables factor out in the decomposition. In general, we have the following result:

Theorem 18. *Let $f, g \in B$, such that $f^{(i)} = g^{(i)}$, and f and g have disjoint support over $v_{i+1} \ldots v_n$. Then*

- *$(fg)_j = f_j$ when f depends on v_j, and*

- *$(fg)_j = g_j$ when g depends on v_j, and*
- *otherwise $(fg)_j = 1$.*

Once again, the conjunction of f and g requires additive space. Note that the result for disjoint functions (theorem 14) is the special case where $i = 0$, while the fact that dependent variables factor out (theorem 17) is the special case where v_i is independent of later variables because its value is fixed.

3 Algorithms

To use our decomposed form as a representation for symbolic model checking, we need algorithms for computing boolean combinations and for existential quantification (projection) over boolean variables.

3.1 Logical conjunction

We begin with the algorithm for conjunction. It should be noted at the outset that in general it is not the case that $(fg)_i = f_i g_i$ (though this is true for the case when f and g are independent). In general, it may be the case that, though $f^{(i)}$ and $g^{(i)}$ are both true for a given assignment to $(v_1, \ldots, v_i)$, the assignments to the remaining variables that make them true may be different, and hence $(fg)_i$ may be false. Thus $(fg)_i$ may be stronger than $f_i g_i$.

To avoid this problem, we first compute appropriate approximations k_i to $(fg)_i$ for all i. These terms are computed by conjoining the terms $f_i g_i$ in descending sequence, projecting out v_i at each stage. This "early quantification" step is justified by the fact that the remaining terms in the descending sequence do not depend on v_i, and prevents computing an explicit conjunction of all the terms, which would defeat the purpose of a decomposed representation.

Next, we must "normalize" the representation by cofactoring each approximation to k_i by $(fg)^{(i-1)}$. Since we have no direct representation of the latter, we obtain the desired effect by cofactoring each k_i by the preceding components $(fg)_1 \ldots (fg)_{i-1}$ in sequence. This result derives from the following lemma:

Lemma 19. *For any functions x and h,*

$$x|(\wedge_{j=1}^{i} h_j) = x|h_1|h_2|\cdots|h_i$$

Proof.

$$x|(\wedge_{j=1}^{i} h_j) = x|(h_i \wedge h^{(i-1)}) \tag{5}$$

$$= x|h^{(i-1)}|h_i \tag{6}$$

$$= x|(\wedge_{j=1}^{i-1} h_j)|h_i \tag{7}$$

which by induction gives us the lemma. Equation 6 is a case of theorem 6, while equations 5 and 7 are by theorem 12.

We will state the conjunction algorithm formally in terms of a theorem:

Theorem 20. *Let $h = fg$ and let*

$$k_n = f_n g_n \tag{8}$$
$$k_{i-1} = f_{i-1} g_{i-1} \exists v_i . k_i \tag{9}$$

Then

$$h_i = k_i | h_1 | h_2 | \cdots | h_{i-1} \tag{10}$$

A conjunction operation on the decomposed representation involves $O(n)$ conjunction operations on the underlying representation, $O(n)$ one-variable projection operations, and $O(n^2)$ cofactor operations. The latter is unfortunate, but seems to be necessary in order to avoid explicit construction of the terms $(fg)^{(i)}$.

3.2 Logical disjunction

We now consider computing logical disjunction of two functions represented by their components. First, we should note that in general $(f \vee g)_i \neq f_i \vee g_i$. Consider, for example, computing $h = f \vee g$, where $f = \bar{a}\bar{b}\bar{c}\bar{d}\ldots$ and $g = abcd\ldots$. The components of these functions are, respectively, $f_1 = \bar{a}$, $f_2 = \bar{b}$, *etc.*, and $g_1 = a$, $g_2 = b$, *etc.* Thus, the disjunction of f_i and g_i is 1, for every i, which is clearly wrong. The problem here is that because we are forming a disjunction, h_i is "defined" over a potentially larger domain than f_i and g_i. To correct this problem, we would like to broaden the domains of f_i and g_i before taking the disjunction. That is, we would like to compute:

$$f_i' = f^{(i)} | h^{(i-1)}$$
$$g_i' = g^{(i)} | h^{(i-1)}$$

However, we would like to do this without explicitly computing $f^{(i)}$, $g^{(i)}$ and $h^{(i-1)}$. This leads us to the following algorithm:

Theorem 21. *Let $h = f \vee g$ and*

$$f_1' = f_1 \qquad g_1' = g_1$$
$$f_{i+1}' = f_{i+1}(f_i' | h_i) \qquad g_{i+1}' = g_{i+1}(g_i' | h_i)$$

Then $h_i = f_i' \vee g_i'$.

3.3 Projection

The approach to existential quantification over boolean variables is very similar to the disjunction algorithm. The algorithm is as follows:

Theorem 22. *Let $h = \exists S.f$, where $S \subset V$, and*

$$f_1' = f_1 \tag{11}$$
$$f_{i+1}' = f_{i+1}(f_i' | h_i) \tag{12}$$

Then $h_i = \exists S.f_i'$.

The above algorithm is effective in practice for projecting out small numbers of variables. However, if we consider the limiting case, where $S = V$, we see that $h = 1$, and therefore $f'_n = f$, which clearly defeats the purpose of the decomposition. For projecting out a large number of variables, an effective strategy is to successively project out small groups of variables, in descending order. In this way, each step tends to simplify the problem for the next step.

3.4 Implementing the algorithms with OBDD's

Ordered binary decision diagrams (OBDD's) are a particularly effective representation for the components of a function because of the efficient algorithms for conjunction and disjunction [Bry86] and for generalized cofactor [CBM89]. The OBDD representation for a function is determined by a permutation $U = (u_1, \ldots, u_n)$ on the boolean variables. In the special case where $U = W$, there is a quadratic-time algorithm for generalized cofactor on OBDD's. In the case $U = V = W$, we can also show that the size of the decomposed representation of f is never larger than n times the size of the direct OBDD representation of f:

Theorem 23. *If $U = V = W$, then $|f_i|_{OBDD} \leq |f|_{OBDD}$*

4 Symbolic model checking and decompositions

In symbolic model checking, we use a boolean formula to represent the transition relation of a model, and we use fixed point iterations to evaluate formulas in certain modal logics relative to this model. The most important operation in these iterations is computing the image of some set of states, relative to the transition relation. The transition relation is represented by using a set of variables $v_1, \ldots, v_n$ to represent the "pre-state", and a corresponding set $v'_1, \ldots, v'_n$ to represent the "post-state". A boolean formula over these variables characterizes the set of transitions. In other words, a set of states is represented thus: $S = \lambda V.\chi_S$, while a transition relation is represented thus: $R = \lambda(V, V').\chi_R$. The forward image of S w.r.t. R is

$$\text{Image}(R, S) = \lambda V'.\exists V.(S(V) \wedge R(V, V'))$$

while the reverse image is

$$\text{Image}(R^{-1}, S) = \lambda V.\exists V'.(S(V') \wedge R(V, V'))$$

Evaluating images thus requires conjunction, projection and variable substitution. The fixed point computations required to compute, for example, the set of states reachable from set S, also use the disjunction operation. Negation is not strictly needed, since all formulas can be put in positive normal form, in which negation applies only to literals.

Thus, we have all of the operations necessary to do symbolic model checking based on the component representation of functions. It is necessary only to choose appropriate orders V, W and U. We note that transition relations are often of the form $\chi_R = \wedge_{i=1}^{n} C_i(v_i', v_1, \ldots, v_n)$. That is, each post-state variable is typically constrained relative to the pre-state variables, but the post-state variables are independent given a valuation of the prestate variables. In addition, for each i, $\exists v_i'.C_i = 1$. That is, the transition relation does not constrain the pre-state variables in any way. If this is the case, then there is a distinct advantage to using the order $V = (v_1, \ldots, v_n, v_1', \ldots, v_n')$. In this case, by theorem 18, the component in the decomposition corresponding to v_i' is exactly R_i, while all the components corresponding to v_i are equal to 1. That is, the conjunction of the transition relation parts is formed essentially for free. This makes it unnecessary to the "conjunctive partitioning" technique to avoid an explosion in the size of the transition relation [BCL91].

4.1 Example

One of the advantages of the decomposed representation is the fact that conditionally independent variables are "factored out". As an example of this phenomenon, we consider verifying the equivalence of the two FIFO queue implementations of figure 1. The basic technique is to compute the reachable states of the two running in parallel [CBM89]. As mentioned previously, there is no fixed correspondence between locations in the two queues. However, once we fix the shift register contents and the ring buffer "head pointer", the ring buffer data elements become independent (since they are either uninitialized, or determined by the corresponding shift register element). This suggests that in the variable order V control should precede shift register data, which in turn should precede ring buffer data (or the roles of the two implementations could be reversed). In this case, when representing the set of reachable states, each component corresponding to a ring buffer data bit is a linear-size OBDD, which in essence reads the value of the head pointer, then compares the ring buffer bit to the corresponding shift register bit. As a result, the overall size of the representation is quadratic in the number of data bits.

On the other hand, since there is no fixed correspondence between the data bits, there is no interleaving of the bits that will yield a small OBDD for the reachable state set. This is illustrated in the graphs of figures 2–4. In these graphs, the ordinal axis is the number of data bits in each queue (the queues are one bit wide, however essentially the same results apply to wider queues). In the first graph, we see the size of the decomposed representation of the transition relation. This is the same as the size of the conjunctively partitioned transition relation, for reasons mentioned above. The second figure shows the size of the decomposed representation for the largest state set obtained in the reachable states iteration (which happens to be last iteration in all cases). This is well fit to a quadratic curve, as expected. The third figure shows the size of the OBDD representation of the reachable states. Note that the scale here is two orders of magnitude larger than the previous graph. This graph shows the expected exponential explosion, since the OBDD representation must in essence record the entire contents of one queue in order to compare it to the other queue.

It should also be noted here that there exists a compact "free BDD" [GM94] representation for the reachable state set in our example. However using free BDD's would require the user to provide the correct $O(n^2)$ DAG that determines the free BDD "type". Using decompositions, the simple heuristic "control before data" is sufficient.

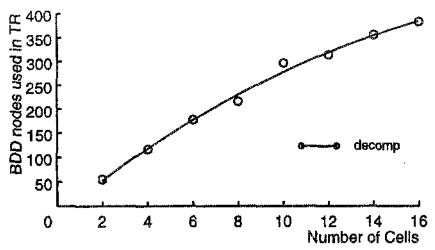

Fig. 2. Space used to represent the transition relation for queue example.

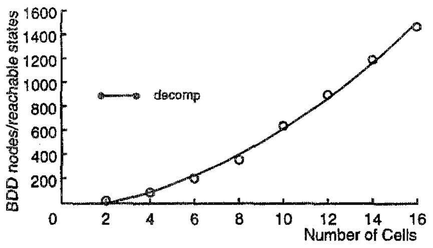

Fig. 3. Space used for decomposed representation of the reachable states for queue example.

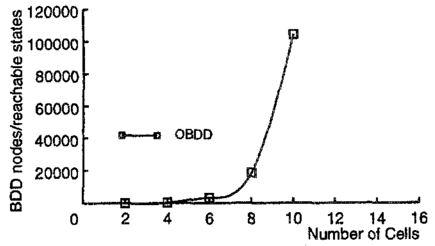

Fig. 4. Space used for OBDD representation of the reachable states for queue example.

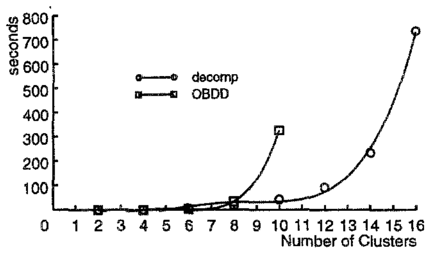

Fig. 5. Time used for computation of the reachable states for queue example.

Finally, figure 5 shows the CPU time in seconds used to compute the reachable state set using both representations. Here, we find the CPU time increasing rapidly in both cases (although the decomposed representation is more efficient as we increase the number of bits). In the decomposition case, it is unclear whether this is an exponential expansion or a fairly high order polynomial (though the difference may be of no practical interest). The algorithms operating on decompositions are not necessarily polynomial, even when measured relative to the result. Therefore, it is possible that exponential time is actually being used. On the other hand, one expects a factor n in the number of iterations due to increasing diameter of the state space. In addition to this, each iteration involves a conjunction, which uses n^2 OBDD operations, each of which is proportional to the transition relation component size ($O(\log n)$) and the state set component size $O(n)$. This would imply at least time proportional to $O(n^5)$, which fits the available data. From a practical point of view, however, it appears that any gains made in space in using the decomposed representation might be offset by losses in time. The question of improving the time performance of the algorithms (at least heuristically) needs to be addressed.

5 Conclusions

We have seen that a boolean representation conjunctively decomposed using generalized cofactor provides a canonical form that exploits "conditional independence" between variables. This property can provide a more compact representation than OBDD's alone, especially in the case when the correspondence between state variables is not fixed, but varies as a function of control. Algorithms for logical operations and projection on this form were described, making the representation usable for symbolic model checking.

The most important practical problem that remains to be solved regarding decompositions is the time required to apply $O(n^2)$ OBBD operations for each operation on a decomposition (where n is the number of variables). The number of variables could, for example, be reduced by grouping them into many-valued variables, though this could make the representation exponentially larger. Also, tight bounds on the complexity of the algorithms should be obtained.

Acknowledgements: This work benefited greatly from discussions with Robert Kurshan of AT&T Bell Labs.

References

[BCL91] Jerry R. Burch, Edmund M. Clarke, and David E. Long. Symbolic model checking with partitioned transition relations. In A. Halaas and P. B. Denyer, editors, *Proceedings of the IFIP International Conference on Very Large Scale Integration*, Edinburgh, Scotland, August 1991.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[CBM89] Olivier Coudert, Christian Berthet, and Jean Christophe Madre. Verification of synchronous sequential machines based on symbolic execution. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1989.

[GM94] J. Gergov and C. Meinel. Efficient boolean manipulation with obdd's can be extended to fbdd's. *IEEE Transactions on Computers*, 43(10):1197–209, Oct. 1994.

[HD93] A. J. Hu and D. L. Dill. Efficient verification with bdds using implicitly conjoined invariants. In C. Courcoubetis, editor, *Computer Aided Verification. 5th International Conference, CAV '93*, pages 3–14, Berlin, Germany, 1993. Springer-Verlag.

[JABF92] J. Jain, J. A. Abraham, J. Bitner, and D. S. Fussell. Probabilistic verification of boolean functions. *Formal Methods in System Design*, 1(1):61–115, July 1992.

[TSL$^+$90] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using BDD's. In *ICCAD*, pages 130–133, 1990.