Meta-BDDs: A Decomposed Representation for Layered Symbolic Manipulation of Boolean Functions

Gianpiero Cabodi

Dip. di Automatica e Informatica
Politecnico di Torino, Turin, Italy
cabodi@polito.it
http://www.polito.it/~cabodi

Abstract. We propose a BDD based representation for Boolean functions, which extends conjunctive/disjunctive decompositions. The model introduced (Meta-BDD) can be considered as a symbolic representation of $k-$Layer automata describing Boolean functions. A layer is the set of BDD nodes labeled by a given variable, and its characteristic function is represented using BDDs. Meta-BDDs are implemented upon a standard BDD library and they support layered (decomposed) processing of Boolean operations used in formal verification problems. Besides targeting reduced BDD size, the theoretical advantage of this form over other decompositions is being closed under complementation, which makes Meta-BDDs applicable to a broader range of problems.

1 Introduction

Binary Decision Diagrams [1] (BDDs) are a core technique for several applications in the field of Formal Verification and Synthesis. They provide compact implicit forms for functions depending on tens to hundreds of Boolean variables.

Many variants of the original BDD type [1] have been proposed to explore possible optimizations and extensions (see for example a survey in [2]). Dynamic variable ordering techniques (sifting [3]) have played a key role to push forward the applicability of BDDs and to face the ordering dependent memory explosion problem. Partitioned [4] and decomposed [5] forms have also been followed as a divide–and–conquer attempt to scale down the complexity of symbolic operations.

This paper follows the latter trend. We propose a decomposed representation for Boolean functions, which extends conjunctive/disjunctive decompositions. One of the limitations of conjunctive (disjunctive) decompositions is that they are biased to the zeroes (ones) of a Boolean function. Let us consides for instance a conjunctive form $f = \bigwedge_i f_i$, each one of the f_i components describes a subset of the zeroes (the OFF–set) of f. Dually for disjunctive forms. Both forms are not closed under negation: the negation of a conjunctive form is disjunctive (and vice–versa), so the application requires both forms, and practical/heuristic simplification rules, unless all formulas can be put in positive normal form.

[1] Reduced Ordered BDDs (ROBDDs), or simply BDDs whenever no ambiguity arises

G. Berry, H. Comon, and A. Finkel (Eds.): CAV 2001, LNCS 2102, pp. 118–130, 2001.

Our work proposes a decomposed form evenly oriented to represent both the zeroes and the ones of a Boolean function. Besides looking at a compact format, we look for efficient symbolic manipulation in the decomposed form. Our solution can be canonical, it is closed under negation, and it supports standard Boolean operations and quantifiers, so it may be applied to BDD based combinational and sequential verification problems. To find the most suitable way of describing this new decomposed form, we adopt an automaton model, which has recently been proposed to describe Boolean functions within an explicit reachability framework [6]. We see a BDD (and the related Boolean function) as an automaton, and we describe it through a set of BDDs. We thus use the term *Meta–BDD* for the decomposed form.

In the sequel, we will briefly overview some preliminary concepts and related works, then we will introduce Meta-BDDs and the related symbolic manipulations. We will finally present some experimental results attained with a prototype implementation.

2 Preliminaries and Related Works

Binary Decision Diagrams (BDDs) [1,2] are directed acyclic graphs providing a canonical representation of Boolean functions. Starting from a non reduced Ordered BDD (OBDD), the Reduced OBDD (ROBDD [1]) for a given Boolean function is obtained by repeatedly applying two well known reduction rules: (1) *Merging rule* (two isomorphic subgraphs are merged), and (2) *Deletion rule* (a BDD node whose two branches point to the same successor is deleted).

Simple graph algorithms, working *depth–first* on BDDs, implement many operators: APPLY, ITE (if–then–else), and existential/universal quantifiers are well–known examples. BDDs have been widely used in verification problems to represent functions as well as sets, by means of their characteristic functions. Operations on sets are efficiently implemented by Boolean operations on their characteristic functions. The notation χ_A is usually adopted for the *characteristic function* of a set A. For instance, let A, B be two sets, and χ_A, χ_B their characteristic functions, we write:

$$\chi_{A\cup B} = \chi_A \vee \chi_B, \chi_{A\cap B} = \chi_A \wedge \chi_B, \chi_{A-B} = \chi_A \wedge \neg\chi_B$$

For sake of simplicity, we make a little abuse of notation in the rest of this paper, and we make no distinction between the BDD representing a set, the characteristic function of the set and the set itself.

2.1 State Sets Represented by $k-$Layer DFAs

A given Boolean function $f(x) : \{0,1\}^k \rightarrow \{0,1\}$ is represented by Holzmann and Puri [6] as a Deterministic Finite Automaton (DFA). They introduce *$k-$Layer DFAs* to describe sets of states within a verification framework based on explicit reachability. An automaton accepts input strings of length k. In the Boolean case, $\{0,1\}$ is the input alphabet, the automaton accepts a set $S \subseteq \{0,1\}^k$ of $k-$tuples, and each layer in the automaton corresponds to an input bit of the function describing a state set.

A $k-$Layer DFA has one initial and two terminal states, the accepting terminal state 1, and the rejecting terminal state 0. The automaton is minimized

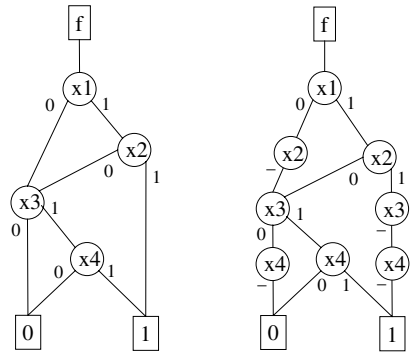

Fig. 1. *A BDD (a) and a k−Layer DFA (b) for the same Boolean function. The deletion rule is not applied to the k−Layer DFA.*

if states which have exactly the same successors are merged together. The automaton describing a Boolean function has a close relationship with the BDD representing it, with a variable ordering corresponding to the input string ordering. A k−Layer DFA is minimized by only using the merging rule, whereas the deletion rule is avoided, to keep input strings of length k.

As an example, Figure 1 shows the BDD (a) and the DFA (b) for the same Boolean function. Given a BDD variable ordering corresponding to the layers, the two representations have similar shapes, but no implicit variables are present in the k−Layer DFA.

2.2 McMillan's Conjunctive Decomposition

McMillan's canonical conjunctive decomposition [5] is another relevant starting point for this work. The automaton representation is proposed by McMillan, too. He sees a BDD representing a set of states 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"*.

A function $f(x_1, ..., x_n)$ is decomposed as $f = \bigwedge_{i=1}^{n} f_i$, the i-th conjunctive component being defined as $f_i = f^{(i)} \downarrow f^{(i-1)}$. The $f^{(i)}$ functions are the projections of f onto growing sets of variables $f^{(i)}(x_1, ..., x_i) = \exists(x_{i+1}, ..., x_n).f(x)$ and $\downarrow$ is the generalized cofactor "constrain" operator [7,8]. Conjunctive components have growing support ($f_i = f_i(x_1, ..., x_i)$), and the representation is canonical given a variable order for projections (which is not necessarily the same as the BDD variable order). Cofactoring is a major source of BDD size reduction for this representation, due to the BDD simplification properties of the generalized cofactor operator: the BDD representing $g \downarrow f$ is often (not always [2] !) smaller

[2] Since "constrain" may introduce new variables (and BDD nodes) in the cofactored term, simplification is not always achieved. Other variants of generalized cofactor have been introduced to expecially address simplification tasks. An example is the

than the BDD of g, and the decomposition $f^{(i)} = f^{(i-1)} \wedge (f^{(i)} \downarrow f^{(i-1)})$ exploits this fact, expecially in cases of factors with *disjoint supports* or supports including *conditionally independent variables*.

Conjunction, disjunction and projection (existential quantification) algorithms are also proposed in [5], in order to use the decomposition in symbolic model checking problems that can be put in positive normal form (with negation only allowed on literals). They can be summarized as follows.

Conjunction. is the simplest operation. The result of a conjunction fg is computed in two steps. An intermediate result t is first evaluated by conjoining the couples of corresponding components f_i and g_i bottom-up (with decreasing i), and applying a "reduction" [3] process:

$$\begin{aligned} t_n &= f_n g_n \\ t_{i-1} &= f_{i-1} g_{i-1} \exists x_i . t_i \end{aligned}$$

The intermediate result is then "normalized" top-down (by increasing i) for canonicity (and BDD simplification): $h_i = t_i \downarrow h_1 \downarrow ... \downarrow h_{i-1}$. The decomposed conjunction thus results in a linear number of conjunction and projection operations, and a quadratic number of cofactor operations.

Disjunction. is a less natural operation for conjunctive decompositions. The i-th component of $h = f \vee g$ should be evaluated by taking into account all components of the operands from 1 to i:

$$\begin{aligned} t_i &= \textstyle\bigwedge_{j=1}^{i} f_j \vee \bigwedge_{j=1}^{i} g_j \\ h_i &= t_i \downarrow h_1 \downarrow ... \downarrow h_{i-1} \end{aligned}$$

This is not efficient, because of the explicit computation of conjunctions and delayed normalization. So a more efficient computation, with interleaved normalization, is proposed:

$$h_i = \bigwedge_{j=1}^{i} (f_j \downarrow h_1 \downarrow ... \downarrow h_{i-1}) \vee \bigwedge_{j=1}^{i} (g_j \downarrow h_1 \downarrow ... \downarrow h_{i-1})$$

resulting in a quadratic number of conjunction and cofactor, and a linear number of disjunction operations (which is more complex than the previous conjunction case).

Projection (Existential Quantification $h = \exists S.f.$) has the same problems as disjunction, and it is computed in a similar way: $h_i = \exists S.(\bigwedge_{j=1}^{i}(f_j \downarrow h_1 \downarrow ... \downarrow h_{i-1}))$ Again a quadratic number of conjunction and cofactor operations (and a linear number of quantifications) is required.

"restrict" cofactor [9], which locally abstracts from f variables not found in g. But some nice properties of "constrain" are lost, and canonicity of conjunctive decomposition is not guaranteed.

[3] Reduction is a term we bring from breadth-first BDD manipulation (indicating a postponed application of merging and deletion rules) [10]. It was not used in [5]

2.3 Incompletely Specified Boolean Functions

An incompletely specified Boolean function is a function defined over a subset of $\{0,1\}^n$. The domain points where the function is not defined are called *don't care* set, whereas the points where the function is defined as true or false are called *ON-set* and *OFF-set*, respectively (the union of ON-set and OFF-set is called *care* set).

Given an incompletely specified function f, we will use the notation $f.on$ for the ON-set, $f.off$ for the OFF-set, and $f.dc$ for the don't care set. Two of them are enough to completely characterize f, since they have null mutual intersections and their union is the domain space. So f might be represented by the couple $f = (f.on, f.off)$, being $f.dc = \neg(f.on \vee f.off)$. Another way to represent f is the interval of completely specified functions $f = [f.on, \neg f.off]$.

3 Meta-BDDs. Describing a BDD by a Layered Set of BDDs

This section defines Meta-BDDs. They are **not** a new type of Decision Diagram for Boolean functions. We introduce them as a layered set of BDDs used to describe a Boolean function. We view a BDD as a DFA, and we use other BDDs to describe it by layers of variables, and to symbolically encode breadth–first computations of BDD operators.

We can also view Meta-BDDs as an extension of McMillan's canonical conjunctive decomposition [5]. Our representation is more general, since it includes conjunctive as well as disjunctive decompositions, and it is closed under Boolean negation. It is canonical under proper conditions.

Let us define the $i-$th layer as the set of nodes labeled by the x_i variable. We characterize the layer with the BDD paths reaching terminals (either 1 or 0) from x_i nodes. In the automaton view of BDDs, this means that the accepting or rejecting final state is decided when testing the x_i variable. In the case of Figure 1, there is no path to terminal nodes from the x_1 layer, there is one path to terminal 1 from the x_2 layer ($x_1x_2 = 11$), 3 paths to 0 at the x_3 layer ($x_1x_2x_3 = \{000, 010, 100\}$), 3 paths to 1 ($x_1x_2x_3x_4 = \{0011, 0111, 1011\}$) and 3 paths to 0 ($x_1x_2x_3x_4 = \{0010, 0110, 1010\}$) from the x_4 layer.

We describe a layer of a given function f by means of a function capturing the zeroes (paths to 0) and ones (paths to 1) of f at that layer. More specifically, we encode the i-th layer of f with an incompletely specified function f_i, such that the ON-set ($f_i.on$) is the set of ones of f at the i-th layer, and the OFF-set ($f_i.off$) is the set of zeroes of f at the i-th layer. As a consequence, the don't care set ($f_i.dc$) is the set of ones/zeroes reached by f at other layers.

We informally introduce the $Meta$ representation of f (Meta-BDD if symbolically encoded by BDDs), as the set of f_i layers that completely characterize f. For each layer we represent the two sets of paths leading to the 1 and 0 terminals at that layer. In the case represented in Figure 1, this leads to the Meta form of Figure 2(a).

A more accurate and formal definition of Meta-BDDs is obtained by introducing the Meta operator $<>^{\mathcal{M}}$, working on incompletely specified Boolean functions.

$$f_1 = (0,0)$$
$$f_2 = (x_1x_2, 0)$$
$$f_3 = (0, \neg(x_1x_2)\neg x_3)$$
$$f_4 = (\neg(x_1x_2)x_3x_4, \neg(x_1x_2)x_3\neg x_4)$$

(a)

$$f_1 = (0,0)$$
$$f_2 = (x_1x_2, 0)$$
$$f_3 = (0, \neg x_3)$$
$$f_4 = (x_4, \neg x_4)$$

(b)

Fig. 2. *Layered Meta representations of f. Each $f_i = (f_i.on, f_i.off)$ is exactly defined only for the ones and zeroes of f at the corresponding layer (a). Upper layers are used to simplify lower f_is by means of cofactoring (b).*

Definition 1. *Given two incompletely specified Boolean functions f and g, $h = < f, g >^{\mathcal{M}}$ is an incompletely specified function, such that the ON-set (OFF-set) of h is the ON-set (OFF-set) of f augmented with the portion of the ON-set (OFF-set) of g not covered by the care set of f:*

$$< f, g >^{\mathcal{M}} \stackrel{def}{=} \mathrm{ITE}(\neg f.dc, f, g) \tag{1}$$

The above definition can be rewritten as

$$< f, g >^{\mathcal{M}} = h \text{ such that } \begin{cases} h.on &= f.on \vee g.on \wedge \neg f.off \\ h.off &= f.off \vee g.off \wedge \neg f.on \\ h.dc &= f.dc \wedge g.dc \end{cases}$$

The operator returns the argument in the unary case ($< f >^{\mathcal{M}} = f$) and it is associative

$$< < f, g >^{\mathcal{M}}, h >^{\mathcal{M}} = < f, < g, h >^{\mathcal{M}} >^{\mathcal{M}} = < f, g, h >^{\mathcal{M}}$$

We are now ready to formally define the Meta decomposition of f in n components.

Definition 2. *The Meta decomposition of a Boolean function f is an ordered set of components that produces f if given as argument to the $<>^{\mathcal{M}}$ operator:*

$$f^{\mathcal{M}}_{[1,n]} \stackrel{def}{=} < f_1, f_2, ..., f_n >^{\mathcal{M}} = f$$

A Meta-BDD is a BDD representation of a Meta decomposition.

We adopt $[i, j]$ subscripts to indicate intervals of components, and we optionally omit them if clear from the context (we use $f^{\mathcal{M}}$ instead of $f^{\mathcal{M}}_{[1,n]}$).

Applying equation (1) and associativity, a Meta decomposition can be recursively written as

$$f^{\mathcal{M}}_{[i,j]} = \mathrm{ITE}(\neg f_i.dc, f_i, f^{\mathcal{M}}_{[i+1,j]})$$

which leads to the following expanded expression for f:

$$f = \mathrm{ITE}(\neg f_1.dc, f_1, \mathrm{ITE}(\neg f_2.dc, f_2, \mathrm{ITE}(...\mathrm{ITE}(\neg f_{n-1}.dc, f_{n-1}, f_n))))$$

The inspiring idea of this decomposition is that each component contributes a new piece to the ON-set and the OFF-set of f. In other words, f is progressively approximated and finally reached by a sequence of incompletely specified functions $f^{\mathcal{M}}_{[1,1]} \preceq f^{\mathcal{M}}_{[1,2]} \preceq ... \preceq f^{\mathcal{M}}_{[1,n]} = f$ ordered by the precedence relation

$$f^{\mathcal{M}}_{[1,i]} \preceq f^{\mathcal{M}}_{[1,j]} \iff (f^{\mathcal{M}}_{[1,i]}.on \subseteq f^{\mathcal{M}}_{[1,j]}.on) \wedge (f^{\mathcal{M}}_{[1,i]}.off \subseteq f^{\mathcal{M}}_{[1,j]}.off)$$

The functions have non decreasing ON-set and OFF-set, and the last one coincides with f.

We have a degree of freedom in selecting the sequence of functions converging to f. Starting from canonical conjunctive decomposition, we adopt the idea of projecting f onto growing sets of variables, so that $f^{\mathcal{M}}_{[1,i]}$ captures the ones and zeroes of f at the upper i layers (i.e. the BDD paths reaching 0 or 1 *forall* variables at the lower levels $(x > x_i)$). We thus choose $f^{\mathcal{M}}_{[1,i]}(x_1, ..., x_i)$ such that

$$f^{\mathcal{M}}_{[1,i]}(x_1, ..., x_i).on = \forall(x > x_i).f$$
$$f^{\mathcal{M}}_{[1,i]}(x_1, ..., x_i).off = \forall(x > x_i).\neg f$$

The $f^{\mathcal{M}}_{[1,i]}$ function is represented by the first i terms of the Meta decomposition $f_1, ..., f_i$. The definition does not provide a rule to uniquely compute the f_i components, given f. In fact, we have here another degree of freedom, as the f_i term is partially "covered" by the previous ones $(< i)$. So we might leave it partially unspecified, or better exploit this fact (as in [5]) and simplify the lower layers by cofactoring them with the don't care set of the upper ones: $f_i = f^{\mathcal{M}}_{[1,i]} \downarrow f^{\mathcal{M}}_{[1,i-1]}.dc$.

Since the don't care set of $f^{\mathcal{M}}_{[1,i-1]}$ is the intersection of the don't care sets of the first $i-1$ components ($f^{\mathcal{M}}_{[1,i-1]}.dc = \bigwedge_{j=1}^{i-1} f_j.dc$,)we avoid computing it and we apply to f_i a chain of cofactors: $f_i = f^{\mathcal{M}}_{[1,i]} \downarrow f_1.dc \downarrow f_2.dc \downarrow ... \downarrow f_{i-1}.dc$. This would simplify the representation of Figure 2(a) to the form (b), which is obviously simpler.

Meta-BDDs and Conjunctive Decompositions. Meta decompositions include McMillan's conjunctive decomposition as the particular case with $f^{\mathcal{M}}_{[1,i]}.on = 0$ for all $i < n$ and $f^{\mathcal{M}}_{[1,i]}.off = \forall(x > x_i).\neg f = \neg\exists(x > x_i).f$. Given the above assumptions, the i-th component of $f^{\mathcal{M}}$ is $f_i = (0, \neg\exists(x > x_i).f \downarrow \exists(x > x_{i-1}).f)$, where the OFF-set is the complement of McMillan's generic conjunctive term.

Variable Ordering and Grouping. The ordering applied to the definition of a Meta-BDD is not required to be the same as the BDD ordering. Moreover, layers can be extended to groups of variables, i.e. each x_i in the previous definitions is a set of variables instead of a single variable. This has the advantage of reducing the number of layers in the decomposition, where each layer includes a set of variables (and the corresponding edges to terminals).

In our implementation, we observed best performance when using the same order for variables and layers, and grouping variables. For the cases we addressed (100 to 200 state variables), reasonable group sizes are 10 to 30 variables. Dynamic variable ordering is supported, provided that variable layers are rebuild each time a new variable order is produced. The overhead we experienced for this transformation is low compared to sifting time, and to the overall cost of image computations, since rebuilding variable groups is a linear operation and

transforming a Meta-BDD from old layers to new ones is a variant of the BDD to Meta-BDD transformation (described in the next section).

Meta-BDDs and Canonicity. Meta-BDDs are canonical under conditions similar to McMillan's decomposition, i.e. that layer simplifications are done through constrain cofactor. Canonicity guarantees constant time equality check, but our experience in sequential reachability shows that we may give it up whenever non canonical representations produce memory reduction.

This is often the case for Meta-BDDs, where a **conditional** application of **reduction and constrain simplification** may filter out the decompositions producing benefits and abort the bad ones. This is a major point for the efficiency in our implementation where reductions and cofactorings are controlled by BDD size based heuristic decisions.

We also experienced the "restrict" cofactor [9], with worse results on the average, compared with controlled application of constrain. A possible reason for this fact is that restrict guarantees good local optimizations of individual functions, but operations involving restricted functions may blow up when combining terms with different restrict optimizations.

4 Symbolic Operations on Meta-BDDs

We describe in this section how basic Boolean operators can be applied to Meta-BDD decompositions. In particular, we will concentrate on the operations required by sequential verification tasks: standard logic operators and quantifiers. Our procedures are here proposed for the canonical case, and we omit for simplicity heuristic decision points for conditional application of reductions and cofactor simplifications.

First of all, Meta BDDs provide a constant time Not operation whose result is again a Meta-BDD. In fact, since Boolean negation swaps zeroes with ones, $\neg f^{\mathcal{M}}$ is computed by simply swapping the $(f_i.on, f_i.off)$ couples.

Going to BDD/Meta-BDD conversions and Apply-like operations, we operate them through a layered process, which is inspired by [10], where BDD operations are performed through a breadth–first two phase (Apply-Reduce) technique. But our method is implicit, we operate breadth–first through layer–by–layer iterations on the T-ROBDD implicit structure, represented by BDDs.

Figure 3 shows how we convert a BDD to a Meta-BDD. We initially assign all terminal edges to the bottom layer, i.e. we initialize all Meta-BDD components to 0, except the last one. Then we perform reduction and constrain simplification.

Reduction and constrain simplification are shown in Figure 4. MetaReduce is a bottom–up process which finds BDD nodes with both cofactors pointing to the same terminal. The merged edges are moved to the upper layers. MetaConstrain (Figure 4(b)) operates the cofactor based simplification. The reduction and constrain operations are here represented by dedicated procedures, as post–processing steps of Meta-BDD operations. For best performance, they can be integrated within breadth–first manipulations, and operated by layers as soon as possible.

```
BDD2Meta (f)
    for i = 1 to n − 1
        f_i ← (0, 0)
    f_n ← (f, ¬f)
    f^M ← (f_1, ..., f_n)
    MetaReduce(f^M)
    MetaConstrain(f^M)
    return f^M
```

Fig. 3. *Converting a BDD to Meta-BDD. All terminal edges are initially assigned to the bottom (n−th) layer. They are moved to the proper upper layers by the reduction procedure. constrain simplification is finally operated*

```
MetaReduce (f^M)
    for i = n − 1 downto 1
        f_i.on ← f_i.on ∨ ∀(x > x_i).f_{i+1}.on
        f_i.off ← f_i.off ∨ ∀(x > x_i).f_{i+1}.off
        f_{i+1} ← f_{i+1} ↓ f_i.dc
```

(a)

```
MetaConstrain (f^M)
    for i = 1 to n − 1
        for j = i + 1 to n
            f_j ← f_j ↓ f_i.dc
```

(b)

Fig. 4. *Reduction (a). Terminal edges are moved upward by a bottom-up iterative process. Move is achieved by adding the reduced part to f_i, then deleting it from $f_{i+1} using cofactor$. Constrain based simplification (b). A double iteration is operated to avoid explicit computation of $f_i^{\mathcal{M}}.dc$.*

As an example of Apply operation, we show the conjunction (MetaAnd) procedure. Disjunction is obtained for free in terms of complementation and conjunction. The proposed algorithm is based on the following theorem

Theorem 1. *Let $f^{\mathcal{M}}_{[j,i]}$ and $g^{\mathcal{M}}_{[j,i]}$ be Meta decompositions, then*

$$f^{\mathcal{M}}_{[j,i]} \wedge g^{\mathcal{M}}_{[j,i]} =< f^{\mathcal{M}}_{[j,i-1]} \wedge g^{\mathcal{M}}_{[j,i-1]}, r_i >^{\mathcal{M}}$$

with r_i computed as:

$$r_i.on = \bigvee_{l=1}^{i} f_l.on \wedge \bigvee_{l=1}^{i} g_l.on$$
$$r_i.off = f_i.off \wedge \neg \bigvee_{l=1}^{i-1} g_l.on \vee g_i.off \wedge \neg \bigvee_{l=1}^{i-1} f_l.on$$

Proof Sketch. Let us consider conjunction as a symbolic breadth–first visit of the product automaton of f and g. The set of paths reaching 0 at the $i-$th layer is given by the paths where either f or g are 0 at the $i-$th layer, and they are not 1 at the upper ones ($< i$) [4] . The set of paths to 1 is given by the paths where both automata (f and g) reach 1 at one of the first i layers.

Figure 5(a) shows our algorithm for conjunction. f' and g' are used to collect the overall ON-sets of the upper components ($\bigvee_l f_l.on$ and $\bigvee_l g_l.on$ in Theorem 1). Explicit computation of the above terms would be in contrast with

[4] Due to the cofactor based simplification, $f_i.off$ ($g_i.off$) could intersect the onset of upper components

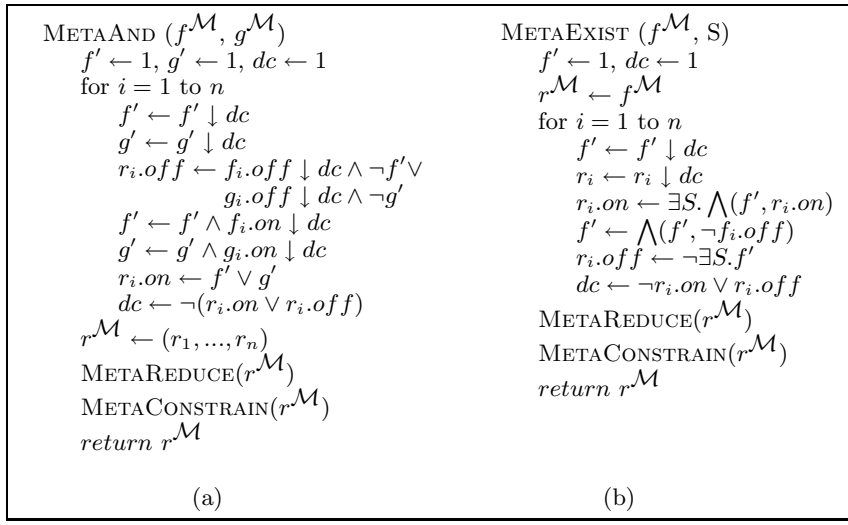

Fig. 5. *Breadth–first computation of Boolean And (a) and existential quantification (b). The layers of the result are computed through top–down layered visits of the operands*

the purposes of the decomposed representation. We thus interleave the layer computations with cofactoring based simplifications, which allow us iteratively projecting f' and g' on decreasing subsets of the domain space. Cofactoring is done both to keep BDD sizes under control, and to achieve a preliminary reduction. Full bottom-up reduction and final constrain simplification are explicitly called as last steps.

Existential quantification (MetaExist procedure) is shown in Figure 5(b). Computation is again top-down, and based on the following theorem

Theorem 2. *Let $f^{\mathcal{M}}_{[i,j]}$ be a Meta decomposition, then*

$$\exists S. f^{\mathcal{M}}_{[i,j]} =< \exists S. f_i, \exists S. (\neg f_i.off \wedge f^{\mathcal{M}}_{[i+1,j]}) >^{\mathcal{M}}$$

Proof Sketch. Let us again concentrate on the layered automaton view of $\exists S. f^{\mathcal{M}}_{[i,j]}$. The existential quantification [5] of the first component ($\exists S. f_i$) captures all ones and zeroes reached at the i-th layer of the non reduced result (other ones/zeroes might be hoisted up when reducing lower levels). The ones and zeroes at lower layers ($> i$) are computed working with $f^{\mathcal{M}}_{[i+1,n]}$ (lower layers of the operand). But spurious ones introduced by cofactor transformations could produce wrong (overestimated) ones, so we need to force 0 within the OFF-set of the i-th component.

The algorithm of Figure 5(b) uses f' to accumulate the filtering function (conjunction of complemented OFF-sets). We do not represent f' as a mono-

[5] We compute the existential quantification of an incompletely specified function as $\exists s. f = (\exists S. f.on, \forall S. \neg f.off)$

lithic BDD, since this would again be a violation of our primary goal (decomposition). We thus use "clustered" BDDs (partitioned conjunctions performed under threshold control), and we also interleave layer computations with cofactor simplifications (as in MetaAnd).

Existential quantification is by far the most important (and expensive) operation in symbolic reachability analysis. Due to its combined usage with conjunction within image/preimage computations, BDD packages provide the so called "relational product" or "and-exist" operator, a recursive procedure specifically concieved to avoid the explicit intermediate BDD generated as a result of conjunction before existential quantification. We did the same with Meta-BDDs, and we implemented a MetaAndExist procedure (not shown here) which properly integrates the previously shown MetaAnd and MetaExist algorithms.

5 Experimental Results

The presented technique has been implemented and tested within a home-made reachability analysis tool, built on top of the Colorado University Decision Diagram (CUDD) package [11]. The experiments shown here are limited to reachability analysis of Finite State Machines, as a first and general framework, unrelated from the verification of specific properties. Our main goal is to prove that the sequential behavior of the circuits presented can be analyzed with relevant improvements by using Meta-BDDs. Combinational verification as well as BDD based SAT checks are other possible applications of Meta-BDDs.

We present data for a few ISCAS'89-addendum [12] benchmarks and some other circuits [13,15]. They have different sizes, within the range of circuits manageable by state-of-the-art reachability analysis techniques. We only report here data for the circuits we could traverse with some gain. The benchmark circuits we tried without any significant result are: s1269, s1423, s1512, s5378. We argue this is mainly due to the fact that no relevant cases of independent or conditionally independent variables are present in the state sets of those circuits. Table 1

Table 1. *Comparing BDDs and Meta-BDDs in reachability analysis. 266 MHz Pentium II, memory limit 400 MB, time limit 36000 s.*

Circuit	FF	D	States	BDDs			Meta-BDDs			$\frac{\|R\|}{\|R^M\|}$
				BDD_{pk} [Knodes]	Mem [MB]	Time (Sift) [s]	BDD_{pk} [Knodes]	Mem [MB]	Time (Sift) [s]	
s3271	116	16	$1.31 \cdot 10^{31}$	-	-	Time-out	782	214	7973 (1190)	7.7
s3330	132	16	$7.27 \cdot 10^{17}$	-	Mem-out	-	4534	356	22345 (17532)	4.2
FIFOs	142	63	$5.01 \cdot 10^{21}$	1169	45	3691(3232)	183	24	3215 (170)	13
queue	82	45	$3.43 \cdot 10^{11}$	387	27	1873(1750)	132	21	921 (350)	19
$Rotator_{16}$	32	2	$1.00 \cdot 2^{32}$	65	14	25 (23)	12	5	1 (0)	1
$Rotator_{32}$	64	2	$1.00 \cdot 2^{64}$	-	Mem-out	-	390	17	831 (602)	1
$Spinner_{16}$	33	2	$1.00 \cdot 2^{33}$	30	5	7 (4)	7	4	2(1)	1
$Spinner_{32}$	65	2	$1.00 \cdot 2^{65}$	-	Mem-out	-	244	20	417 (331)	1

collects statistics on the circuit used, and the results obtained. For each circuit it first shows some common statistics: the number of latches (FF), the sequential depth (D), and the number of reached states (States). We then compare traversals based on the same image heuristic (IWLS95 by Ranjan et al. [14]), with

standard BDDs and Meta-BDDs. Since conjunctively partitioned transition relations are not critical in terms of BDD size, we use Meta-BDDs only for state sets (and intermediate product of image computations). For both techniques we show peak live BDD nodes (BDD_{pk}), maximum memory usage (Mem) and CPU time (Time) with explicit indication of sifting time. We finally show the ratio BDD vs. Meta-BDD size for reachable state sets ($|R|/|R^M|$).

s3271 and s3330 are known to be hard to traverse circuits, both for time and memory costs. FIFO is a freely modified version of the example used in [5], whereas queue is a queue model from the NuSMV [15] distribution. Rotator [13] has two stages. An input register (subscript 16/32 is register size) is fed by primary inputs. An output register stores a rotated copy of the inputs register. The number of rotated bits is determined by a five bits control input. All states are reachable, but image computation is exponentially complex since the early quantification scheme pays the dependence of the output (input) register bits from all input register (primary input) bits. Spinner [13] is a similar circuit, where the input register can be loaded with the output register, too. These are both cases in which conditional independence can be efficently factored out by Meta-BDDs, in order to achieve relevant gains in intermediate image steps (even though no gains are shown in reachable states).

In all cases Meta-BDDs were able to "compress" reachable state sets and to produce overall improvements. The first two circuits could not be completed with standard BDDs in the adopted experimental setup.

Memory gains are clearly visible from peak BDD nodes, memory usage, and reachable state sets size ratio [6]. The overhead introduced to work with the decomposed form is visible in the reduced ratio sifting time vs. total time (except for the larger example, s3330 where sifting still dominates) and time reductions are mainly due to the smaller BDDs involved in computations.

6 Conclusions and Future Work

We propose a BDD based decomposition for Boolean functions, which extends conjunctive/disjunctive decompositions and may factor out variable independances and/or conditional independances with gains not achievable by standard BDDs.

Our work includes and extends [5], by proposing a representation closed under negation and applicable to a wider range of BDD based problems, and by exploring non-canonicity in terms of heuristically controlled decomposition and simplification steps. Experimental results on benchmark and home made circuits show relevant gains agaist standard BDDs in symbolic FSM traversals. Future works will investigate heuristics, and application to real verification tasks.

Acknowledgement

The author thanks Fabio Somenzi for the FIFOs, Rotator and Spinner source descriptions.

[6] The ratio $|R|/|R^M|$ is 1 in the case of Rotator and Spinner because the reachable state set is the entire state space (a constant function both with BDDs and MEta-BDDs). The BDD gains in those circuits are related to intermediate image BDDs.

References

1. R. E. Bryant. Graph–Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C–35(8):677–691, August 1986.
2. R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary–Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.
3. R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proc. IEEE/ACM ICCAD'93*, pages 42–47, San Jose, California, November 1993.
4. J. Jain, J. Bitner, J. A. Abraham, and D. S. Fussel. Functional Partitioning for Verification and Related Problems. In *Brown/MIT VLSI Conference*, pages 210–226, March 1992.
5. K. L. McMillan. A conjunctively decomposed boolean representation for symbolic model checking . *Proc. CAV'96, Lecture Notes in Computer Science 1102, Springer Verlag*, pages 13–25, August 1996.
6. G.J. Holzmann and A. Puri. A minimized automaton representation of reachable states. *Software Tools for Technology Transfer,Springer Verlag*, 3(1), 1999.
7. O. Coudert, C. Berthet, and J. C. Madre. Verification of Sequential Machines Based on Symbolic Execution. In *Lecture Notes in Computer Science 407, Springer Verlag*, pages 365–373, Berlin, Germany, 1989.
8. H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit Enumeration of Finite State Machines Using BDDs. In *Proc. IEEE ICCAD'90*, pages 130–133, San Jose, California, November 1990.
9. O. Coudert and J. C. Madre. A Unified Framework for the Formal Verification of Sequential Circuits. In *Proc. IEEE ICCAD'90*, pages 126–129, San Jose, California, November 1990.
10. R.K. Brayton R.K. Ranjan, J.V. Sanghavi and A. Sangiovanni-Vincentelli. High performance bdd package based on exploiting memory hierarchy. *Proc. ACM/IEEE DAC'96*, pages 635–640, June 1996.
11. F. Somenzi. CUDD: CU Decision Diagram Package – Release 2.3.0. Technical report, Dept. of Electrical and Computer Engineering, University of Colorado, Boulder, Colorado, October 1998.
12. MCNC Private Communication.
13. 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.
14. R. K. Ranjan, A. Aziz, R. K. Brayton, B. Plessier, and C. Pixley. Efficient BDD Algorithms for FSM Synthesis and Verification. In *IWLS'95: IEEE International Workshop on Logic Synthesis*, Lake Tahoe, California, May 1995.
15. A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifyer. In *Proc. CAV'99, Lecture Notes in Computer Science 1633, Springer Verlag*, pages 495–499, July 1999.