Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Basing Decisions on Sentences in Decision Diagrams

Yexiang Xue*
Department of Computer Science
Cornell University
yexiang@cs.cornell.edu

Arthur Choi and **Adnan Darwiche**
Computer Science Department
University of California, Los Angeles
{*aychoi,darwiche*}*@cs.ucla.edu*

Abstract

The Sentential Decision Diagram (SDD) is a recently proposed representation of Boolean functions, containing Ordered Binary Decision Diagrams (OBDDs) as a distinguished subclass. While OBDDs are characterized by total variable orders, SDDs are characterized by dissections of variable orders, known as vtrees. Despite this generality, SDDs retain a number of properties, such as canonicity and a polytime `Apply` operator, that have been critical to the practical success of OBDDs. Moreover, upper bounds on the size of SDDs were also given, which are tighter than comparable upper bounds on the size of OBDDs. In this paper, we analyze more closely some of the theoretical properties of SDDs and their size. In particular, we consider the impact of basing decisions on sentences (using dissections as in SDDs), in comparison to basing decisions on variables (using total variable orders as in OBDDs). Here, we identify a class of Boolean functions where basing decisions on sentences using dissections of a variable order can lead to exponentially more compact SDDs, compared to OBDDs based on the same variable order. Moreover, we identify a fundamental property of the decompositions that underlie SDDs and use it to show how certain changes to a vtree can also lead to exponential differences in the size of an SDD.

Introduction

A new representation of Boolean functions was recently proposed, called the Sentential Decision Diagram, with a number of interesting properties (Darwiche 2011). First, the notion of decisions performed on variables in Ordered Binary Decision Diagrams (OBDDs) is generalized to a notion of decisions performed on sentences in Sentential Decision Diagrams (SDDs). Second, as total variable orders characterize OBDDs (Bryant 1986), a special type of ordered trees, called vtrees, characterize SDDs. Despite this generality, SDDs are still able to maintain a number of properties that have been critical to the success of OBDDs in practice. For example, SDDs support an efficient `Apply` operation as in OBDDs, and they are also canonical, under restrictions similar to reductions in OBDDs.

On the theoretical side, an upper bound was identified on the size of SDDs (based on treewidth) that is tighter than the corresponding upper bound on the size of OBDDs (based on pathwidth) (Darwiche 2011). In this paper, we investigate more closely some further theoretical properties of SDDs. In particular, we examine the impact that branching on sentences can have on the size of SDDs, as opposed to branching on variables in OBDDs.

First, we consider a way to obtain vtrees for an SDD, by *dissecting* variable orders for OBDDs. We identify a class of Boolean functions, where certain variable orders lead to exponentially large OBDDs, but where certain dissections of the same variable order lead to SDDs of only linear size, suggesting that the ability to branch on sentences may be a powerful one. In the process, we provide a more general result, where we give a simple algorithm for constructing SDDs of linear size, when the Boolean function of interest corresponds to a tree-structured circuit.

Next, we identify a fundamental property of the decompositions that underlie SDDs and use it to prove further properties of SDDs. For example, we show that simply swapping a pair of children in a vtree can sometimes lead to exponential differences in SDD size.

These results have an interesting implication on the potential of SDDs in practice, as a generalization of OBDDs. This implication is based on three observations. First, since OBDDs with a particular variable order correspond to SDDs with a restricted type of vtree, the search space over SDDs has embedded in it the search space of OBDDs. Next, effective dynamic variable re-ordering heuristics have been critical to the practical success of OBDDs. Finally, dissecting a variable order can result in an exponentially more compact SDD. As a result, effective algorithms that dynamically search for good vtrees (i.e., variable orders and their dissections) could potentially further extend the reach and practical use of decision diagrams.

Technical Preliminaries

We start with some technical and notational preliminaries. Upper case letters (e.g., X) will be used to denote variables and lower case letters to denote their instantiations (e.g., x). Bold upper case letters (e.g., $\mathbf{X}$) will be used to denote sets

*Part of this research was conducted while the author was a visiting student at the University of California, Los Angeles.

of variables and bold lower case letters to denote their instantiations (e.g., $\mathbf{x}$).

A *Boolean function* f over variables $\mathbf{Z}$ maps each instantiation $\mathbf{z}$ to 0 or 1. The *conditioning* of f on instantiation $\mathbf{x}$, written $f|\mathbf{x}$, is a *sub-function* that results from setting variables $\mathbf{X}$ to their values in $\mathbf{x}$. A function f *essentially depends* on variable X iff $f|X \neq f|\neg X$. We write $f(\mathbf{Z})$ to mean that f can only essentially depend on variables in $\mathbf{Z}$. A *trivial* function maps all its inputs to 0 (denoted false) or maps them all to 1 (denoted true).

Consider a Boolean function $f(\mathbf{X}, \mathbf{Y})$ with disjoint sets of variables $\mathbf{X}$ and $\mathbf{Y}$. If

$$f(\mathbf{X}, \mathbf{Y}) = (p_1(\mathbf{X}) \wedge s_1(\mathbf{Y})) \vee \ldots \vee (p_n(\mathbf{X}) \wedge s_n(\mathbf{Y}))$$

then the set $\{(p_1, s_1), \ldots, (p_n, s_n)\}$ is called an $(\mathbf{X}, \mathbf{Y})$-*decomposition* of function f as it allows one to express function f in terms of functions on $\mathbf{X}$ and on $\mathbf{Y}$ (Pipatsrisawat and Darwiche 2010). The ordered pairs (p_i, s_i) are called *elements* of the decomposition. Moreover, if $p_i \wedge p_j = \mathsf{false}$ for $i \neq j$, each p_i is consistent ($\neq \mathsf{false}$), and the disjunction of all p_i is valid ($= \mathsf{true}$), then we call $\{(p_1, s_1), \ldots, (p_n, s_n)\}$ an $(\mathbf{X}, \mathbf{Y})$-*partition* of function f (Darwiche 2011). In this case, each p_i is called a *prime* and each s_i is called a *sub*. We say an $(\mathbf{X}, \mathbf{Y})$-partition is *compressed* iff its subs are distinct, i.e., $s_i \neq s_j$ for $i \neq j$. Finally, the size of a decomposition, or partition, is the number of its elements. Note that by definition, false can never be a prime in an $(\mathbf{X}, \mathbf{Y})$-partition. In addition, if true is prime, then it is the only prime.

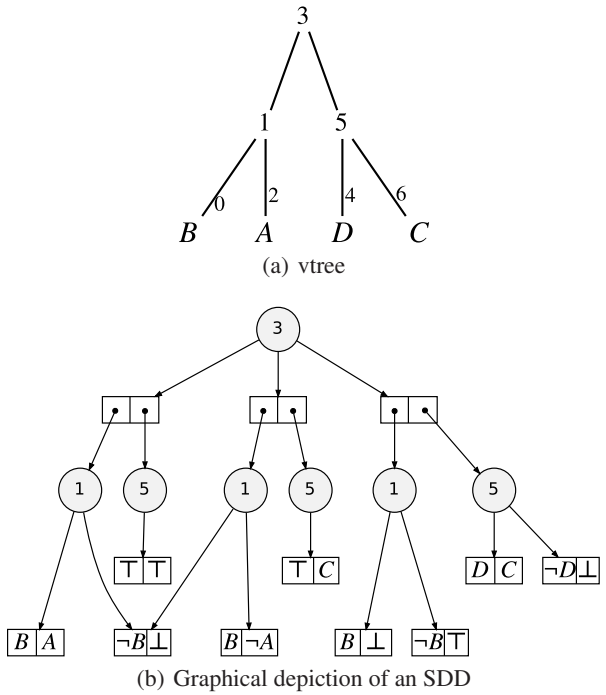
For example, $\{(A, B), (\neg A, \mathsf{false})\}$ and $\{(A, B)\}$ are both (A, B)-decompositions of $f = A \wedge B$. However, only the first is an (A, B)-partition. Decompositions $\{(\mathsf{true}, B)\}$ and $\{(A, B), (\neg A, B)\}$ are both (A, B)-partitions of $f = B$, while only the first is compressed.

Note that $(\mathbf{X}, \mathbf{Y})$-partitions generalize Shannon decompositions, which fall as a special case when $\mathbf{X}$ contains a single variable. OBDDs result from the recursive application of Shannon decompositions, leading to decision nodes that branch on the states of a single variable (i.e., literals). As we show next, SDDs result from the recursive application of $(\mathbf{X}, \mathbf{Y})$-partitions, leading to decision nodes that branch on the state of a set of variables (i.e., arbitrary sentences).

Sentential Decision Diagrams (SDDs)

As total variable orders characterize OBDDs, SDDs are characterized by vtrees. A *vtree* for a set of variables $\mathbf{X}$ is an ordered, full binary tree whose leaves are in one-to-one correspondence with the variables in $\mathbf{X}$. Figure 1(a) depicts a vtree for variables A, B, C and D. As is customary, we will often not distinguish between a node v and the subtree rooted at v, referring to v as both a node and a subtree. The vtree was originally introduced in (Pipatsrisawat and Darwiche 2008), but without making a distinction between the left and right children of a node. However, we make such a distinction when dealing with SDDs by using v^l and v^r to denote the left and right children of node v.

We can use a vtree to recursively decompose a Boolean function f, starting at the root of a vtree. Consider node $v = 3$ in Figure 1(a), which is the root. The *left* subtree



(a) vtree

(b) Graphical depiction of an SDD

Figure 1: Function $f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$.

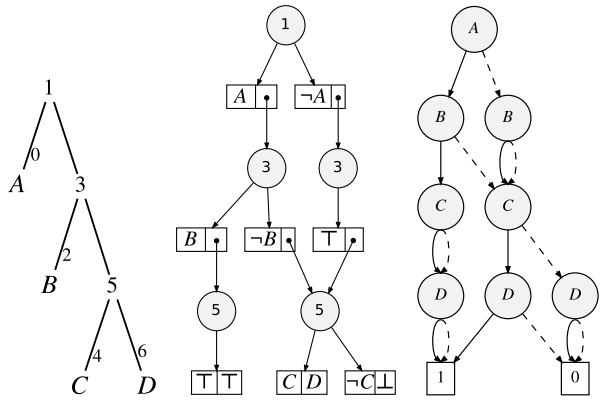
contains variables $\mathbf{X} = \{A, B\}$ and the *right* subtree contains $\mathbf{Y} = \{C, D\}$. Decomposing function f at node $v = 3$ amounts to generating an $(\mathbf{X}, \mathbf{Y})$-partition of function f. If $\{(p_1, s_1), \ldots, (p_n, s_n)\}$ is an $(\mathbf{X}, \mathbf{Y})$-partition of function f at node $v = 3$, then each prime p_i will be further decomposed at node $v^l = 1$ and each sub s_i will be further decomposed at node $v^r = 5$. The process continues until we have constants or literals.

Next, we provide a formal definition of an SDD. If we denote an SDD by α, then we denote the Boolean function that SDD α represents by $\langle \alpha \rangle$. Formally, we say that α is an SDD that is *normalized for a vtree* v iff α and v fall under one of the three following cases:

- $\alpha = \bot_v$ or $\alpha = \top_v$ and v is a leaf.[1]
  Semantics: $\langle \bot_v \rangle = \mathsf{false}$ and $\langle \top_v \rangle = \mathsf{true}$.
- $\alpha = X$ or $\alpha = \neg X$ and v is a leaf with variable X.
  Semantics: $\langle X \rangle = X$ and $\langle \neg X \rangle = \neg X$.
- $\alpha = \{(p_1, s_1), \ldots, (p_n, s_n)\}$, v is an internal node, primes $p_1, \ldots, p_n$ are SDDs that are normalized for left child v^l, subs $s_1, \ldots, s_n$ are SDDs that are normalized for right child v^r, and $\langle p_1 \rangle, \ldots, \langle p_n \rangle$ is a partition (mutually exclusive, exhaustive, and $\langle p_i \rangle \neq \mathsf{false}$ for all p_i).
  Semantics: $\langle \alpha \rangle = \bigvee_{i=1}^{n} \langle p_i \rangle \wedge \langle s_i \rangle$.

A constant or literal SDD is called *terminal.* Otherwise, it is called a *decomposition.* The size of SDD α is obtained by summing the sizes of all its decompositions.

[1] Although the terminal SDDs representing true and false are distinct for each leaf node, we will omit the subscript v in $\top_v$ and $\bot_v$ when it is clear from the context.



(a) A vtree (b) An SDD (c) An OBDD

Figure 2: A vtree, SDD and OBDD for $(A \wedge B) \vee (C \wedge D)$.

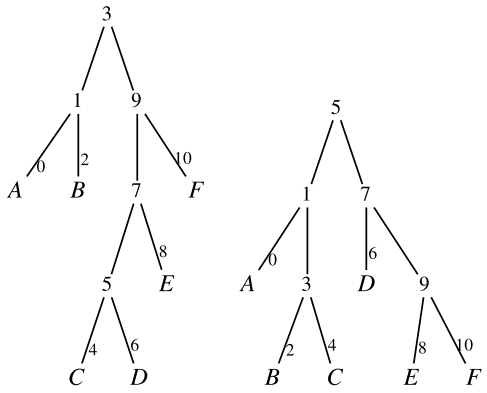


Figure 3: Two vtrees that result from dissecting the order $\langle A, B, C, D, E, F \rangle$.

SDDs can be notated graphically as in Figure 1(b), where a decomposition is represented by a circle with outgoing edges pointing to its elements, and an element is represented by a paired box $\boxed{p \mid s}$, where the left box represents the prime and the right box represents the sub. A box will either contain a terminal SDD or point to a decomposition SDD. Here, the number contained in each circle is the vtree node that the corresponding decomposition is normalized for.

Normalized SDDs are canonical given that they are compressed (Darwiche 2011). An SDD is *compressed* if all of its decompositions are compressed. Normalized SDDs also support a polytime `Apply` operation, allowing one to combine two SDDs using any Boolean operator.

SDDs and OBDDs

A vtree is said to be *right-linear* if each left-child is a leaf. The vtree in Figure 2(a), for example, is right-linear. The compressed SDD in Figure 2(b) is normalized for this right-linear vtree. Every decomposition in this SDD has the form $\{(X, \alpha), (\neg X, \beta)\}$, which is a Shannon decomposition, or of the form $(\top, \beta)$. This is not a coincidence as it holds for all compressed SDDs that are normalized for right-linear vtrees. In fact, such SDDs correspond to quasi-reduced OBDDs in a precise sense; see Figure 2(c).[2] Consider a quasi-reduced OBDD that is based on the variable order induced by the given right-linear vtree. Every decomposition in the SDD corresponds to a decision node in the OBDD and every decision node in the OBDD corresponds to a decomposition or literal in the SDD. In a quasi-reduced OBDD, a literal is represented by a decision node with 0 and 1 as its children, e.g., the literal D in Figure 2(c). However, in a compressed SDD, a literal is represented by a terminal SDD, e.g., the literal D in Figure 2(b).

[2] A quasi-reduced OBDD is a minimal-size OBDD that mentions every variable in its variable order in all paths from root to leaf. Quasi-reduced OBDDs are also canonical, and are at most a factor $n + 1$ larger than the corresponding reduced OBDD (Wegener 2000). Quasi-reduced OBDDs can also be found by repeated applications of the merging rule (without the elimination rule) from a complete binary decision tree.

Upper Bounds

A CNF with n variables and pathwidth pw is known to have an OBDD of size $O(n2^{pw})$ (Prasad, Chong, and Keutzer 1999; Huang and Darwiche 2004; Ferrara, Pan, and Vardi 2005). Pathwidth pw and treewidth w are related by $pw = O(w \log n)$; see, e.g., (Bodlaender 1998). Hence, a CNF with n variables and treewidth w has an OBDD of size polynomial in n and exponential in w (Ferrara, Pan, and Vardi 2005). As for SDDs, (Darwiche 2011) showed that the size of an SDD need only be *linear* in n and exponential in w. BDD-trees are also canonical and come with a treewidth guarantee. Their size is linear in n but at the expense of being *doubly* exponential in treewidth (McMillan 1994). Hence, SDDs come with a tighter treewidth bound than BDD-trees.

Dissecting an OBDD

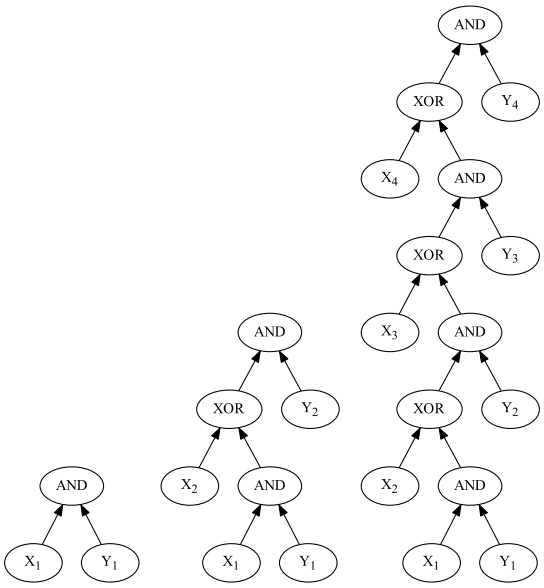
One can view a vtree as the result of dissecting a total variable order in the following sense.

Definition 1 (Dissection) *We say that a vtree T induces a variable order π if and only if a left-right traversal of vtree T visits leaves (variables) in the same order as π. In this case, we also say that vtree T is a dissection of order π.*

Figure 3 depicts two different vtrees that result from dissecting the same variable order. Consider now an OBDD α with respect to a variable order π and an SDD β with respect to a dissection of order π. We will say in this case that SDD β is a dissection of OBDD α. Our main goal in this section is to answer the following question: Can dissecting an OBDD lead to an exponential reduction in its size? The answer is affirmative as given by the following theorem.

Theorem 1 *There exists a class of Boolean functions f, a corresponding variable order π, and a corresponding vtree T that dissects order π, such that the quasi-reduced (or reduced) OBDD induced by π is exponentially larger than the normalized and compressed SDD induced by T.*

We will now consider a class of functions that satisfies the conditions of Theorem 1. Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ and $\mathbf{Y} =$



(a) Formula f_1 (b) Formula f_2 (c) Formula f_4

Figure 4: Circuit realizations of a function $f_n(\mathbf{X}, \mathbf{Y})$ that satisfies Theorem 1.

$\{Y_1, \ldots, Y_n\}$. The function $f_n(\mathbf{X}, \mathbf{Y})$ is defined inductively as follows:

- If $n = 1$, then $f_1(X_1, Y_1) = X_1 \wedge Y_1$.
- If $n > 1$, then $f_n(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$

$$= [f_{n-1}(X_1, \ldots, X_{n-1}, Y_1, \ldots, Y_{n-1}) \oplus X_n] \wedge Y_n$$

where $\oplus$ is the exclusive or operator.

Figure 4 depicts circuit realizations of function $f_n(\mathbf{X}, \mathbf{Y})$ for $n = 1, 2, 4$. We start with a number of observations about this function before we prove the main result.

Lemma 1 *$f_n|\mathbf{xy} = \bigoplus_{i=k+1}^{n} x_i$, where $k = 0$ if $y_i = \mathsf{true}$ for all i; otherwise, k is the largest index where $y_k = \mathsf{false}$.*

Proof If $y_i = \mathsf{true}$ for all i, then $f_n|\mathbf{xy}$ simplifies to $x_1 \oplus \ldots \oplus x_n$. If k is the largest index such that $y_k = \mathsf{false}$, then $y_j = \mathsf{true}$ for all $j > k$. In this case, $f_k|\mathbf{xy} = 0$, $f_{k+1}|\mathbf{xy} = x_{k+1}$ and, hence, $f_n|\mathbf{xy} = x_{k+1} \oplus \ldots \oplus x_n$. □

Lemma 2 *For every pair of instantiations $\mathbf{x} \neq \mathbf{x}^\star$, there is an instantiation $\mathbf{y}$ such that $f_n|\mathbf{xy} \neq f_n|\mathbf{x}^\star\mathbf{y}$.*

Proof Let k be the largest index for which instantiations $\mathbf{x}$ and $\mathbf{x}^\star$ disagree on variable X_k. Consider the instantiation $\mathbf{y}$ that sets $Y_1, \ldots, Y_{k-1}$ to false and $Y_k, \ldots, Y_n$ to true. By Lemma 1,

$$f_n|\mathbf{xy} = x_k \oplus \ldots \oplus x_n \quad \text{and} \quad f_n|\mathbf{x}^\star\mathbf{y} = x_k^\star \oplus \ldots \oplus x_n^\star.$$

Since $x_k \neq x_k^\star$ and $x_i = x_i^\star$ for $i > k$, we must have $f_n|\mathbf{xy} \neq f|\mathbf{x}^\star\mathbf{y}$. □

Corollary 1 *For every pair of instantiations $\mathbf{x} \neq \mathbf{x}^\star$, $f_n|\mathbf{x} \neq f_n|\mathbf{x}^\star$ and, hence, the conditioning of f_n on variables $X_1, \ldots, X_n$ generates 2^n distinct sub-functions.*

The first part of Theorem 1 follows from the following corollary, which itself follows immediately from the Sieling and Wegener bound (Sieling and Wegener 1993).

Corollary 2 *A quasi-reduced (or reduced) OBDD for function f_n with respect to any variable ordering that starts with variables $X_1, \ldots, X_n$ must have at least 2^n nodes.*

Thus, the variable order $\pi = \langle X_n, \ldots, X_1, Y_1, \ldots, Y_n \rangle$, for example, yields an OBDD with at least 2^n nodes.

We will now show the second part of Theorem 1, in which we dissect such an OBDD, obtaining an SDD whose size is only linear in n. In fact, we show a more general result: for any function that is represented by a tree-structured circuit, there is an SDD whose size is linear in the number of circuit variables. The result is based on the following theorem.

Theorem 2 *Let $f(\mathbf{X})$ and $g(\mathbf{Y})$ be two Boolean functions where $\mathbf{X} \cap \mathbf{Y} = \emptyset$. If $\circ$ is a Boolean operator, then function $f \circ g$ has the following $(\mathbf{X}, \mathbf{Y})$-partition:*

$$\{(f, \mathsf{true} \circ g), (\neg f, \mathsf{false} \circ g)\}.$$

Moreover, the $(\mathbf{X}, \mathbf{Y})$-partition is compressed when operator $\circ$ is commutative and not trivial, and function g is not trivial.

Proof If $\mathbf{x}$ is an instantiation that satisfies function f, then $(f \circ g)|\mathbf{x} = \mathsf{true} \circ g$. Otherwise, if $\mathbf{x}$ is an instantiation that satisfies $\neg f$, then $(f \circ g)|\mathbf{x} = \mathsf{false} \circ g$. Hence,

$$\{(f, \mathsf{true} \circ g), (\neg f, \mathsf{false} \circ g)\}$$

is an $(\mathbf{X}, \mathbf{Y})$-partition of function $f \circ g$. For example, when $\circ = \wedge$, we have

$$\{(f, \mathsf{true} \wedge g), (\neg f, \mathsf{false} \wedge g)\} = \{(f, g), (\neg f, \mathsf{false})\}.$$

When $\circ = \oplus$, we have

$$\{(f, \mathsf{true} \oplus g), (\neg f, \mathsf{false} \oplus g)\} = \{(f, \neg g), (\neg f, g)\}.$$

By definition, the $(\mathbf{X}, \mathbf{Y})$-partition is compressed when $\mathsf{true} \circ g \neq \mathsf{false} \circ g$, which holds when operator $\circ$ is commutative and not trivial, and function g is not trivial.[3] □

Given Theorem 2, we can construct an SDD of linear size, for any function that has a tree-structured circuit, as follows. Assuming that each gate has two inputs, we simply use the circuit structure as our vtree;[4] see the circuit in Figure 4(c) and the vtree in Figure 7(a). Note here that there is a vtree node for each primary input and gate of the circuit. We can now construct the SDD recursively, as shown in Algorithm 1. Constructing SDDs for primary inputs is the base case here. Given that we have SDDs for the two inputs of a gate, we can construct an SDD for the gate immediately using Theorem 2. Algorithm 1 appeals to two additional results. The first result, from (Darwiche 2011), says that one can negate an $(\mathbf{X}, \mathbf{Y})$-partition by simply negating its subs.

[3] Suppose function g is not trivial. Then $\mathsf{true} \circ g = \mathsf{false} \circ g$ implies that $1 \circ 1 = 0 \circ 1$ and $1 \circ 0 = 0 \circ 0$. Since $1 \circ 0 = 0 \circ 1$, we then have $1 \circ 1 = 0 \circ 1 = 1 \circ 0 = 0 \circ 0$. This implies that operator $\circ$ is trivial, which is a contradiction. Hence, $\mathsf{true} \circ g \neq \mathsf{false} \circ g$.

[4] If the circuit contains a gate with more than two inputs, the gate can be replaced with a sub-circuit over just binary gates.

Algorithm 1 `sdd-tree`(v)

input: A tree-structured circuit with primary output v. Each gate is non-trivial and has exactly two inputs.

output: A pair of SDDs $(\alpha, \alpha^\star)$ representing the function of circuit v and its negation.

main:

1: **if** v represents a primary input X **then**
2: **return** $(X, \neg X)$
3: **else**
4: $v^l, v^r \leftarrow$ the two sub-circuits feeding into gate v
5: $(\beta, \beta^\star) \leftarrow$ `sdd-tree`(v^l)
6: $(\gamma, \gamma^\star) \leftarrow$ `sdd-tree`(v^r)
7: $\circ \leftarrow$ Boolean operator corresponding to gate v
8: $\eta_1 \leftarrow$ `Apply`(true, γ, $\circ$) {returns γ, $\gamma^\star$, $\top$ or $\bot$}
9: $\eta_0 \leftarrow$ `Apply`(false, γ, $\circ$) {returns γ, $\gamma^\star$, $\top$ or $\bot$}
10: $\alpha \leftarrow \{(\beta, \eta_1), (\beta^\star, \eta_0)\}$
11: $\eta_1^\star \leftarrow$ negation of η_1 {returns γ, $\gamma^\star$, $\top$ or $\bot$}
12: $\eta_0^\star \leftarrow$ negation of η_0 {returns γ, $\gamma^\star$, $\top$ or $\bot$}
13: $\alpha^\star \leftarrow \{(\beta, \eta_1^\star), (\beta^\star, \eta_0^\star)\}$
14: **return** $(\alpha, \alpha^\star)$

The second states that applying a Boolean operator $\circ$ to a constant and function f yields either a constant, the function f, or its negation $\neg f$ (see Lines 8 & 9 of Algorithm 1).[5]

It should be clear that Algorithm 1 has a linear complexity in the number of circuit inputs as it adds at most two SDD nodes for each recursive call, each of which has two elements. Since the function $f_n(\mathbf{X}, \mathbf{Y})$ identified earlier has a tree-structured circuit, it must then have an SDD of linear size when using the vtree corresponding to its structure. Figure 7 depicts such a vtree for $n = 4$, together with the corresponding SDD. Note that this vtree dissects the order $\pi = \langle X_4, \ldots, X_1, Y_1, \ldots, Y_4 \rangle$. We then have the following corollary, which proves the second part of Theorem 1.

Corollary 3 *There is a normalized and compressed SDD of function f_n, of size $O(n)$, corresponding to a dissection of the order $\pi = \langle X_n, \ldots, X_1, Y_1, \ldots, Y_n \rangle$.*

We have thus shown that dissecting an OBDD into an SDD can lead to an exponential reduction in size, suggesting that the ability to branch on sets of variables (sentences) in SDDs may be a powerful one.

Figure 5 depicts a right-linear vtree corresponding to the order $\pi = \langle X_4, \ldots, X_1, Y_1, \ldots, Y_4 \rangle$ and Figure 6 depicts the corresponding SDD, which also corresponds to an OBDD in this case. Hence, the SDD in Figure 7(b) can be viewed as a dissection of the one in Figure 6.

Finally, we remark that we have identified a class of Boolean functions, where certain variable orders lead to exponentially large OBDDs, but where certain dissections (respecting the same variable order) lead to SDDs of only linear

[5]To show that $\mathsf{true} \circ f \in \{\mathsf{true}, \mathsf{false}, f, \neg f\}$, let $1 \circ 1 = a$ and $1 \circ 0 = b$. If $a = b$, then $\mathsf{true} \circ f$ is a trivial function. If $a = 1$ and $b = 0$, then $\mathsf{true} \circ f = f$. Otherwise, $a = 0$, $b = 1$ and $\mathsf{true} \circ f = \neg f$. A similar argument can be used to show that $\mathsf{false} \circ f \in \{\mathsf{true}, \mathsf{false}, f, \neg f\}$.

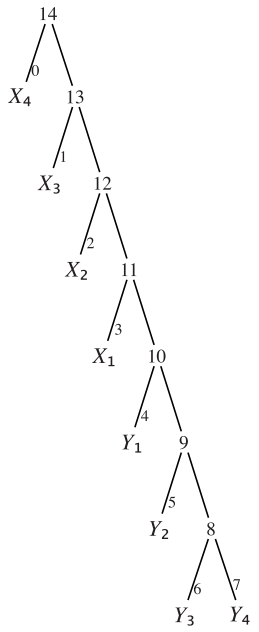


Figure 5: A right-linear vtree.

size. In the example we introduced in this section, there are in fact variable orders that lead to OBDDs (and SDDs) of polynomial size, for example, $\pi = \langle Y_n, X_n, \ldots, Y_1, X_1 \rangle$. However, the fact that dissections can obtain exponential reductions in size for a given variable order, has some interesting practical implications. In particular, it suggests that dynamically searching for good dissections may be a promising direction to pursue. Sifting algorithms, for example, which are based on swapping neighboring variables in a total variable order, have been particularly effective for dynamic variable re-ordering (Rudell 1993). Dissection introduces a new dimension that would allow SDDs to navigate around barriers that could be faced when only navigating variable orders using OBDDs.

On the Left-Right Order of Vtrees

In this section, we consider the following question: Can switching the left and right children of a vtree node lead to an exponential change in the size of the corresponding SDD? The answer is affirmative as we show next.

Consider the following function:

$$f_n(X_1, \ldots, X_n, Y_1, \ldots, Y_n) = \bigvee_{i=1}^{n} \left[\bigwedge_{j=1}^{i-1} \neg X_j \right] \wedge X_i \wedge Y_i.$$

This function has a compressed $(\mathbf{X}, \mathbf{Y})$-partition of size n, with the i^{th} prime being $p_i = \left[\bigwedge_{j=1}^{i-1} \neg X_j\right] \wedge X_i$ and the i^{th} sub being $s_i = Y_i$. Yet, the compressed $(\mathbf{Y}, \mathbf{X})$-partition of function f_n is of size 2^n. To see this, consider an instantiation $\mathbf{y}$ of variables $\mathbf{Y}$. The sub-function $f_n|\mathbf{y}$ corresponds to a disjunction of a set of primes that are unique to instantiation $\mathbf{y}$. Primes are mutually exclusive, so the disjunction

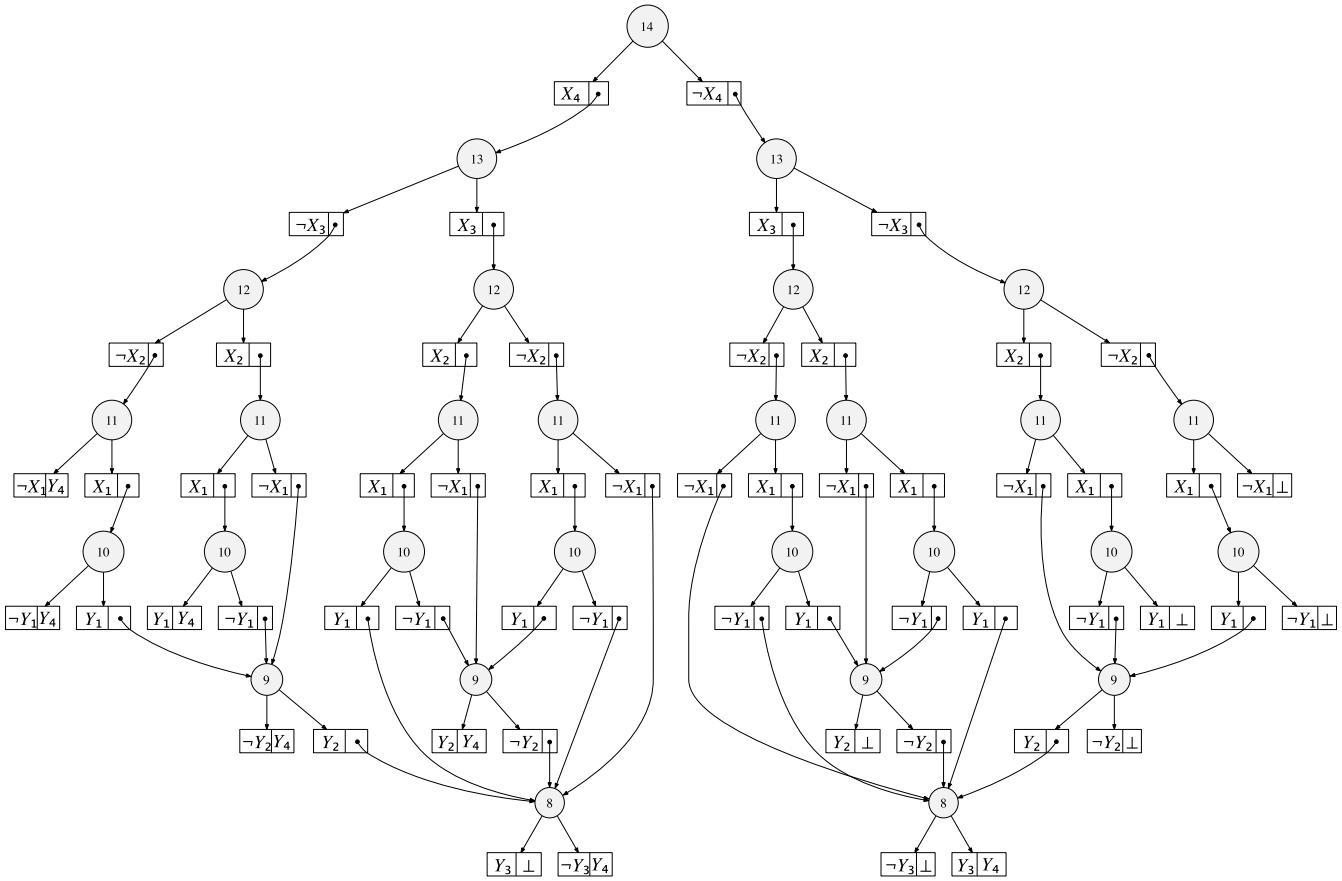


Figure 6: An SDD for the right-linear vtree in Figure 5. The SDD corresponds to an OBDD and is equivalent to the SDD in Figure 7(b).

itself is also unique to instantiation $\mathbf{y}$. Thus, there are 2^n distinct sub-functions of the form $f_n|\mathbf{y}$. Further, instantiations $\mathbf{y}$ are the primes of the unique, compressed $(\mathbf{Y}, \mathbf{X})$-partition of function f_n, which must have size 2^n.

There is an SDD of size $O(n)$ for the function f_n if it uses a vtree with root v where: (1) variables $X_1, \ldots, X_n$ appear in the left subtree and variables $Y_1, \ldots, Y_n$ appear in the right subtree, and (2) the left subtree is right linear for order $X_n, \ldots, X_1$ and the right subtree is right linear for order $Y_1, \ldots, Y_n$. If we now switch the left and right subtrees of root v, the corresponding SDD must have size $\Omega(2^n)$ as it must include a $(\mathbf{Y}, \mathbf{X})$-partition of function f_n. Thus, the left-right order of a vtree can lead to an exponential difference in the size of corresponding SDD.

In the remainder of this section, we show a more general result on the relationship between the compressed $(\mathbf{X}, \mathbf{Y})$-partition of a function $f(\mathbf{X}, \mathbf{Y})$ and its $(\mathbf{X}, \mathbf{Y})$-decompositions, which implies our result above. Central to this general result is the notion of a basis.

The Basis of a Set of Boolean Functions

Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be a set of Boolean functions, and let $\mathcal{F}^*$ denote the closure of $\mathcal{F}$ with respect to Boolean operators $\circ$, i.e., the set $\mathcal{F}^*$ is the smallest set where $\mathcal{F} \subseteq \mathcal{F}^*$ and where $f, g \in \mathcal{F}^*$ implies $f \circ g \in \mathcal{F}^*$, for all Boolean operators $\circ$. Moreover, define the *basis* $\mathcal{G}$ of a set $\mathcal{F}$ to be the set of non-false functions $g \in \mathcal{F}^*$ such that for all $g' \in \mathcal{F}^*$, if $g' \models g$ then $g' = g$. That is, the basis $\mathcal{G}$ of a set $\mathcal{F}$ is the set of minimal non-false functions in the closure $\mathcal{F}^*$ under the partial ordering $\models$.

First, we characterize the properties of a basis.

Theorem 3 *A set of Boolean functions $\mathcal{G} = \{g_1, \ldots, g_m\}$ is the basis of a set $\mathcal{F} = \{f_1, \ldots, f_n\}$ if and only if the following conditions hold:*

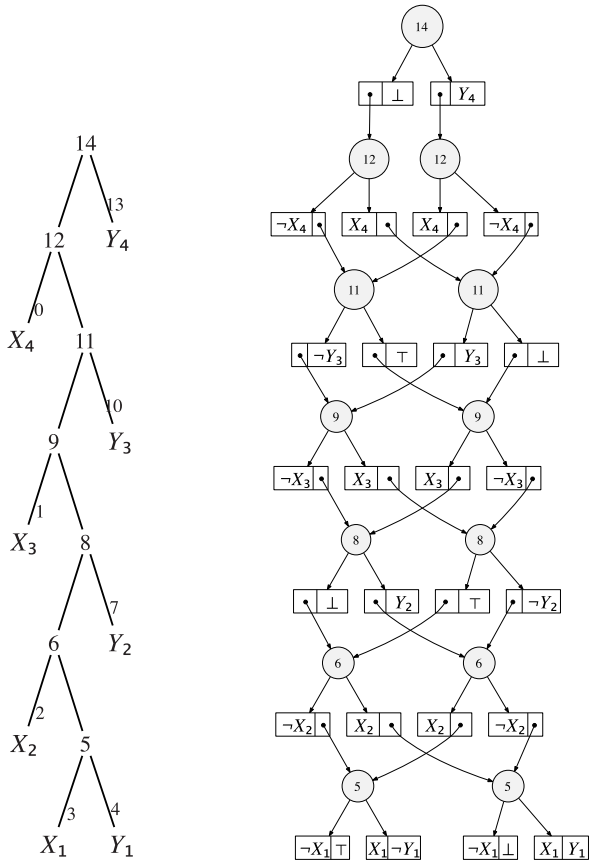
(a) For all $g_i \in \mathcal{G}$, we have $g_i \neq$ false.
(b) For all $g_i \in \mathcal{G}$ and $f_j \in \mathcal{F}$, either $g_i \models f_j$ or $g_i \models \neg f_j$.
(c) For all $g_i \in \mathcal{G}$, all $\mathcal{F}_i = \{f_j \in \mathcal{F} \mid g_i \models f_j\}$ are distinct.
(d) $g_1 \vee \cdots \vee g_m =$ true and $g_i \wedge g_j =$ false for all $i \neq j$.

The proof of this theorem is delegated to the Appendix.

Decompositions and Partitions

We now have the following interesting theorem.

Theorem 4 *Let $\{(g_1(\mathbf{X}), h_1(\mathbf{Y})), \ldots, (g_n(\mathbf{X}), h_n(\mathbf{Y}))\}$ be an $(\mathbf{X}, \mathbf{Y})$-decomposition of a function $f(\mathbf{X}, \mathbf{Y})$, and let*



(a) vtree (b) SDD

Figure 7: A vtree and its corresponding SDD.

$\mathcal{P}$ be the basis of the functions $g_1, \ldots, g_n$. Then $\mathcal{P}$ are the primes of an $(\mathbf{X}, \mathbf{Y})$-partition of function f. Moreover, if the functions $h_1, \ldots, h_n$ are mutually exclusive, then $\mathcal{P}$ are the primes of the unique, compressed $(\mathbf{X}, \mathbf{Y})$-partition of function f.

The proof of this theorem is also delegated to the Appendix. The importance of this theorem is that it allows one to use knowledge about the bases of Boolean functions to derive results about the sizes of $(\mathbf{X}, \mathbf{Y})$-partitions. For example, we will show next that the basis of a set of Boolean functions can be exponentially smaller or larger than the number of such functions. We will then show that this implies the concrete result we showed earlier in the section: The size of the compressed $(\mathbf{X}, \mathbf{Y})$-partition of a function $f(\mathbf{X}, \mathbf{Y})$ can be exponentially different than the size of its compressed $(\mathbf{Y}, \mathbf{X})$-partition.

Consider first the set $\mathcal{S}$ of all Boolean functions over variables $X_1, \ldots, X_n$. Note that there are 2^{2^n} such Boolean functions. The closure of set $\mathcal{S}$ under Boolean operators is the set $\mathcal{S}$ itself. Hence, the basis of set $\mathcal{S}$ corresponds to the set of all 2^n instantiations over variables $X_1, \ldots, X_n$. This is an example where the size of the basis of Boolean functions $\mathcal{S}$ is exponentially smaller than the size of the set $\mathcal{S}$.

Consider now the set $\mathcal{S}$ of Boolean functions $f_1, \ldots, f_n$ over variables $X_1, \ldots, X_n$, where $f_i = X_i$. The closure of $\mathcal{S}$ under Boolean operators $\circ$ is the set of all Boolean functions over variables $X_1, \ldots, X_n$. Thus, the basis of set $\mathcal{S}$ is the set of Boolean functions corresponding to the 2^n instantiations of variables $X_1, \ldots, X_n$. This is an example where the size of the basis of Boolean functions $\mathcal{S}$ is exponentially larger than the size of the set $\mathcal{S}$. This observation and Theorem 4 implies the result we showed earlier in the section, in which we showed a function $f(\mathbf{X}, \mathbf{Y})$ whose compressed $(\mathbf{X}, \mathbf{Y})$-partition and compressed $(\mathbf{Y}, \mathbf{X})$-partition have sizes that differ exponentially.

Conclusion

We considered in this paper the size of a decision diagram from the viewpoint of basing decisions on sentences (i.e., sets of variables), as in SDDs, in contrast to basing decisions on literals (i.e., single variables), as in OBDDs. We first identified a class of Boolean functions where, for a given variable ordering, there is a dissection of that ordering that results in an SDD that is exponentially smaller than the corresponding OBDD. In the process, we provided a general algorithm for constructing compact SDDs from tree-structured circuits. We further identified a fundamental property of the decompositions that underlie SDDs, which we used to show how switching children in a vtree can also lead to exponential differences in the size of an SDD.

Acknowledgments

This work has been partially supported by NSF grant #IIS-0916161.

Proofs

Proof of Theorem 3. Note that conjoin and complement are sufficient to induce the closure $\mathcal{F}^*$. We first show that Conditions (a–d) are necessary for a set $\mathcal{G} = \{g_1, \ldots, g_m\}$ to be the basis of set $\mathcal{F} = \{f_1, \ldots, f_n\}$.

(a) By definition of a basis.

(b) Suppose that Condition (b) does not hold, i.e., $g_i \not\models f_j$ and $g_i \not\models \neg f_j$. This implies that $g_i \wedge f_j \neq \mathsf{false}$. and $g_i \wedge \neg f_j \neq \mathsf{false}$. Moreover, function $g_i \wedge f_j$ is in the closure $\mathcal{F}^\star$. However, $g_i \wedge f_j \models g_i$, which implies that g_i is not a basis function, which is a contradiction.

(c) Suppose g_i has the corresponding set $\mathcal{F}_i$. We want to show that g_i is equivalent to the function

$$\gamma = \Big(\bigwedge_{f \in \mathcal{F}_i} f \Big) \wedge \Big(\bigwedge_{f \in \mathcal{F} \setminus \mathcal{F}_i} \neg f \Big).$$

From Condition (b), we know that either $g_i \models f$ or $g_i \models \neg f$ for each $f \in \mathcal{F}$. Thus, if we show that $g_i = \gamma$, we know that each $\mathcal{F}_i$ is distinct, since each g_i is distinct. By definition, $g_i \models f$ for all $f \in \mathcal{F}_i$, and thus $g_i \models \neg f$ for all $f \in \mathcal{F} \setminus \mathcal{F}_i$. Moreover, we have that $g_i \models \gamma$ and further that $\gamma \neq \mathsf{false}$ since $g_i \neq \mathsf{false}$. If we conjoin to γ any function in $\mathcal{F}$, we either get back γ or false (by

construction of γ). Note that any function in the closure $\mathcal{F}^*$ can be represented as a disjunction of terms

$$\Big(\bigwedge_{f\in\mathcal{H}} f\Big) \wedge \Big(\bigwedge_{f\in\mathcal{F}\setminus\mathcal{H}} \neg f\Big).$$

for some $\mathcal{H} \subseteq \mathcal{F}$. Thus, if we conjoin to γ any function in the closure $\mathcal{F}^*$, we also get back either γ or false. Since $\gamma \in \mathcal{F}^*$, function γ must be a basis function, and thus $\gamma = g_i$ (since $g_i \models \gamma$).

(d) First, if $\gamma = g_1 \vee \cdots \vee g_m \neq \mathsf{true}$, then $\neg\gamma \neq \mathsf{false}$. Moreover, for all $g_i \in \mathcal{G}$, $g_i \not\models \neg\gamma$. Since $\neg\gamma \in \mathcal{F}^*$, function $\neg\gamma$ should have been a basis function, which is a contradiction. Next, if there are $g_i, g_j \in \mathcal{G}$ where $\gamma = g_i \wedge g_j \neq \mathsf{false}$, then $\gamma \models g_i$ and $\gamma \models g_j$, which by the definition of a basis, implies that $\gamma = g_i = g_j$, which is a contradiction.

Next, we show that Conditions (a–d) are sufficient for a set $\mathcal{G}$ to be the basis of a set $\mathcal{F}$. For each function $g_i \in \mathcal{G}$, let $\mathcal{F}_i = \{f_j \in \mathcal{F} \mid g_i \models f_j\}$, and let

$$\gamma_i = \Big(\bigwedge_{f\in\mathcal{F}_i} f\Big) \wedge \Big(\bigwedge_{f\in\mathcal{F}\setminus\mathcal{F}_i} \neg f\Big),$$

which is in the closure $\mathcal{F}^*$. First, by Condition (b), since $g_i \models f$ for all $f \in \mathcal{F}_i$, we know that $g_i \models \neg f$ for all $f \in \mathcal{F} \setminus \mathcal{F}_i$, and further that $g_i \models \gamma_i$. Since $g_i \neq \mathsf{false}$ by Condition (a), we also know that $\gamma_i \neq \mathsf{false}$, since $g_i \models \gamma_i$. Next, by Condition (c), for any g_i and g_j for $i \neq j$, there is a function $f \in \mathcal{F}$ where $f \in \mathcal{F}_i$ and $f \notin \mathcal{F}_j$ (or vice versa). Thus, $\gamma_i \models f$ and $\gamma_j \models \neg f$ (or vice versa), and so $\gamma_i \wedge \gamma_j = \mathsf{false}$ for all $i \neq j$. Third, by Condition (d), all $g_i \in \mathcal{G}$ are mutually exclusive and exhaustive. Since all γ_i are mutually exclusive, and since $g_i \models \gamma_i$, the functions γ_i are also exhaustive. Hence, $g_i = \gamma_i$ for all i. Fourth, if we conjoin to $g_i = \gamma_i$ any function in $\mathcal{F}^*$, we either get back $g_i = \gamma_i$ or false. As $g_i = \gamma_i \in \mathcal{F}^*$, function g_i must be a basis function of $\mathcal{F}$. Finally, since all $g_i \in \mathcal{G}$ are mutually exclusive and exhaustive, the set $\mathcal{G}$ must be a basis of $\mathcal{F}$. □

Proof of Theorem 4. Let $\mathcal{P}$ be the basis of the functions $g_1, \ldots, g_n$. Since the basis forms a partition (mutually exclusive, exhaustive, and all $g_j \neq \mathsf{false}$), we just need to show that for each $p_i \in \mathcal{P}$ and instantiations $\mathbf{x}$ and $\mathbf{x}^\star$ where $\mathbf{x} \models p_i$ and $\mathbf{x}^\star \models p_i$, we must have $f|\mathbf{x} = f|\mathbf{x}^\star$. Suppose $\mathbf{x} \models p_i$ and $\mathbf{x}^\star \models p_i$. By Condition (b) of Theorem 3, either $p_i \models g_j$ or $p_i \models \neg g_j$. Hence, instantiations $\mathbf{x}$ and $\mathbf{x}^\star$ imply the same set of functions g_i implied by p_i, leading to

$$f|\mathbf{x} = f|\mathbf{x}^\star = \bigvee_{p_i\models g_j} h_j,$$

which is the sub of prime p_i.

Suppose now that functions h_j are mutually exclusive. Consider any two instantiations $\mathbf{x}$ and $\mathbf{x}^\star$ such that $\mathbf{x} \models p_i$ and $\mathbf{x}^\star \models p_j$ where $i \neq j$. By Condition (c) of Theorem 3, primes p_i and p_j imply different sets g_k. Hence,

$$f|\mathbf{x} = \bigvee_{p_i\models g_j} h_j \neq \bigvee_{p_j\models g_j} h_j = f|\mathbf{x}^\star$$

since functions h_j are mutually exclusive. Hence, the subs of primes p_i and p_j are distinct and the $(\mathbf{X}, \mathbf{Y})$-partition is compressed. □

References

Bodlaender, H. L. 1998. A partial *k*-arboretum of graphs with bounded treewidth. *Theoretical Computer Science* 209(1-2):1–45.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35:677–691.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 819–826.

Ferrara, A.; Pan, G.; and Vardi, M. Y. 2005. Treewidth in verification: Local vs. global. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, 489–503.

Huang, J., and Darwiche, A. 2004. Using DPLL for efficient OBDD construction. In *The Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 157–172.

McMillan, K. L. 1994. Hierarchical representations of discrete functions, with application to model checking. In *Computer Aided Verification (CAV)*, 41–54.

Pipatsrisawat, K., and Darwiche, A. 2008. New compilation languages based on structured decomposability. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI)*, 517–522.

Pipatsrisawat, K., and Darwiche, A. 2010. A lower bound on the size of decomposable negation normal form. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, 345–350.

Prasad, M. R.; Chong, P.; and Keutzer, K. 1999. Why is ATPG easy? In *Design Automation Conference (DAC)*, 22–28.

Rudell, R. 1993. Dynamic variable ordering for Ordered Binary Decision Diagrams. In *ICCAD*, 42–47.

Sieling, D., and Wegener, I. 1993. NC-algorithms for operations on binary decision diagrams. *Parallel Processing Letters* 3:3–12.

Wegener, I. 2000. *Branching Programs and Binary Decision Diagrams*. Society for Industrial and Applied Mathematics (SIAM).