Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Zero-Suppressed Sentential Decision Diagrams

Masaaki Nishino[1] and **Norihito Yasuda**[2] and **Shin-ichi Minato**[2] and **Masaaki Nagata**[1]
[1]NTT Communication Science Laboratories, NTT Corporation
[2]Graduate School of Information Science and Technology, Hokkaido University
nishino.masaaki@lab.ntt.co.jp

Abstract

The Sentential Decision Diagram (SDD) is a prominent knowledge representation language that subsumes the Ordered Binary Decision Diagram (OBDD) as a strict subset. Like OBDDs, SDDs have canonical forms and support bottom-up operations for combining SDDs, but they are more succinct than OBDDs. In this paper we introduce an SDD variant, called the Zero-suppressed Sentential Decision Diagram (ZSDD). The key idea of ZSDD is to employ new trimming rules for obtaining a canonical form. As a result, ZSDD subsumes the Zero-suppressed Binary Decision Diagram (ZDD) as a strict subset. ZDDs are known for their effectiveness on representing sparse Boolean functions. Likewise, ZSDDs can be more succinct than SDDs when representing sparse Boolean functions. We propose several polytime bottom-up operations over ZSDDs, and a technique for reducing ZSDD size, while maintaining applicability to important queries. We also specify two distinct upper bounds on ZSDD sizes; one is derived from the treewidth of a CNF and the other from the size of a family of sets. Experiments show that ZSDDs are smaller than SDDs or ZDDs for a standard benchmark dataset.

Introduction

Knowledge compilation is the technique of compiling a Boolean function into a tractable representation. The Ordered Binary Decision Diagram (OBDD) (Bryant 1986) is one of the most popular representations and is used in various areas. OBDD is tractable, has canonical forms and supports polytime `Apply` operations for combining OBDDs.

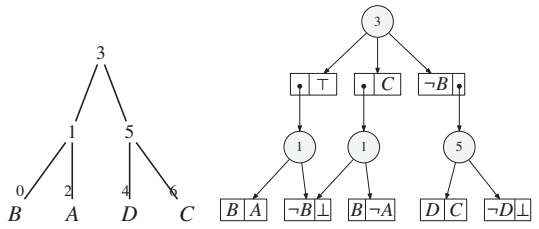
Following the success of OBDD, several variants of decision diagrams have been proposed (e.g., (Minato 1993; Bahar et al. 1997)). The Sentential Decision Diagram (SDD) (Darwiche 2011) is such a prominent variant of the OBDD. SDD subsumes the Ordered Binary Decision Diagram (OBDD) (Bryant 1986), and has canonical forms and supports polytime operations for combining SDDs. Moreover, SDDs have bounds on the size that are tighter than those of OBDDs. Because of these properties, SDDs are attracting much attention (Kisa et al. 2014; Chakraborty et al. 2014).

OBDD has many important variants other than SDD. *The Zero-suppressed Binary Decision Diagram* (Minato 1993) is one such variant. Same as OBDD, ZDD represents a Boolean function as a directed acyclic graph (DAG), but its reduction rules are different. As a result, ZDDs generally tend to be smaller than OBDDs when representing *sparse* Boolan functions. We say a Boolean function is sparse if it has a small number of models and each model has small number of variables whose value is 1.

In this paper, we introduce a new decision diagram called *the Zero-suppressed Sentential Decision Diagram (ZSDD)*. It is a variant of SDD and subsumes ZDD as a strict subset. This relationship is analogous to the SDD subsuming the OBDD. Like SDDs, ZSDDs have canonical form and support several polytime bottom-up operations for combining them. Just as ZDDs tend to be more succinct than OBDDs when representing sparse Boolean functions, ZSDDs tend to be more succinct than SDDs when representing sparse Boolean functions. Moreover, ZSDDs have tighter upper bounds on their size when they are used for representing CNFs compared with ZDDs; the size of a ZSDD representing a CNF is bounded by the treewidth of the CNF, while the size of a ZDD is bounded by the pathwidth.

In the following, after reviewing SDDs, we introduce ZSDDs and several bottom-up operations. Then we introduce implicit partitioning, a simple but powerful technique that makes ZSDDs much more succinct by suppressing unnecessary substructures. It reduces ZSDD size while maintaining applicability to several key types of queries. Since practically important queries such as model counting run in time linear to ZSDD size with implicit partitioning, it is attractive. We also give two upper bounds on ZSDD size with implicit partitioning; one is decided by the size of the family of sets interpretation of a Boolean function, while the other is decided by the treewidth of a CNF. We also give analyses on its applicability to several kinds of queries and transformations. Finally we conduct experiments to compare the size of ZSDD, SDD, and ZDD and show that ZSDD is more succinct than SDD or ZDDs with standard benchmark instances[1].

[1]Our sample software of ZSDD is available at https://github.com/nsnmsak/zsdd/.



(a) A vtree (b) An SDD

Figure 1: A vtree and an SDD representing $f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$

Technical Preliminaries

We first briefly show notations and some definitions used in the later sections. We use an upper case letter (e.g., X) to represent a variable and a lower case letter to represent its instantiation (e.g., x). We use a bold upper case letter (e.g., $\mathbf{X}$) to represent a set of variables, and use a bold lower case letter its instantiation (e.g., $\mathbf{x}$). Boolean function $f(\mathbf{X})$ maps instantiations of $\mathbf{X}$ to $true$ or $false$.

Since ZDDs and ZSDDs can be seen as condensed representations of *families of sets*, we introduce a family of sets interpretation of Boolean functions. A family of sets is a collection of subsets of a given base set S. For example, given base set $S = \{A, B, C, D\}$, then $\{\{A, B\}, \{B, C\}, \{C, D\}\}$ is an example of a family of sets over S. Every family of sets can be represented as a Boolean function. In particular, it can be straightforwardly represented in disjunctive normal form (DNF), where each subset contained in the family of sets corresponds to each term. The above example corresponds to Boolean function $f(A, B, C, D) = (A \wedge B \wedge \neg C \wedge \neg D) \vee (\neg A \wedge B \wedge C \wedge \neg D) \vee (\neg A \wedge \neg B \wedge C \wedge D)$. Let $\emptyset$ be an empty family; it corresponds to Boolean function $f(\mathbf{X}) = false$. The family of sets $\{\emptyset\}$ corresponds to Boolean function $f(\mathbf{X}) = \bigwedge_{i=1}^{|\mathbf{X}|} \neg x_i$. We use $\mathfrak{p}$ to represent the family of all subsets of the base set, and use $\bar{f}$ to represent the complement family of f, defined as $\bar{f} = \{a \mid a \in \mathfrak{p} \text{ and } a \notin f\}$. Every model of a Boolean function representing a family of sets has one-to-one correspondence to a subset in the family, and the number of variables taking value 1 in a model equals the size of the corresponding subset. Therefore, small families of sets tend to be sparse Boolean functions.

Sentential Decision Diagrams

The Sentential Decision Diagram (SDD) (Darwiche 2011) is a data structure that represents a Boolean function as a DAG. Like BDD, it has canonical form and supports bottom-up operations. SDD and OBDD mainly differ in how they decompose Boolean functions. OBDD decomposes a Boolean function by using *Shannon decomposition*, while SDD uses a new decomposition rule called $(\mathbf{X}, \mathbf{Y})$-*decomposition*, which is a generalization of Shannon decomposition. Let f be a Boolean function, and $\mathbf{X}$, $\mathbf{Y}$ be groups of variables that compose a partition of the variables of f. It follows that the function can be decomposed as

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

where $p_i(\mathbf{X})$, $s_i(\mathbf{Y})$ are subfunctions whose variables are $\mathbf{X}$ and $\mathbf{Y}$, respectively. We call $p_1, \ldots, p_n$ *primes* and $s_1, \ldots, s_n$ *subs*. If $p_i \wedge p_j = false$ for all $i \neq j$, $\bigvee_{i=1}^{n} p_i = true$, and $p_i \neq \bot$ for all i, then we say the decomposition is *an* $(\mathbf{X}, \mathbf{Y})$*-partition*, and denote it as $\{(p_1, s_1), \ldots, (p_n, s_n)\}$. Moreover, if $s_i \neq s_j$ for all $i \neq j$ is satisfied, then we say the $(\mathbf{X}, \mathbf{Y})$-partition is *compressed*.

An SDD represents a Boolean function by recursively applying $(\mathbf{X}, \mathbf{Y})$-partitions, where the order of partitions are determined by *a vtree*. A vtree is a binary tree whose leafs correspond to variables. We show a vtree example in Fig. 1 (a). The vtree root represents the partition of variables into two groups: variables that appear in the left subtree and those that appear in the right subtree. In this figure, root node $v = 3$ represents $(\mathbf{X}, \mathbf{Y})$-partition where $\mathbf{X} = \{A, B\}$ and $\mathbf{Y} = \{C, D\}$. Similarly, node $v = 1$ represents a partition where $\mathbf{X} = \{B\}$ and $\mathbf{Y} = \{A\}$. In this way, every non-leaf vtree node represents a partitioning. We use v^l, v^r to represent the left and the right child vtree node of v, respectively.

An SDD is defined as follows, where we use $\langle \cdot \rangle$ to denote a mapping from SDDs into Boolean functions.

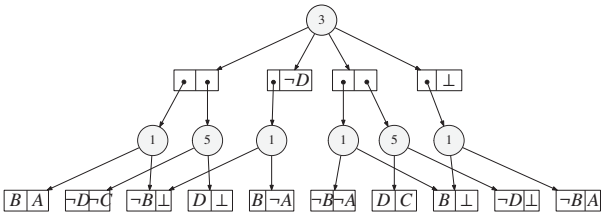
Definition 1. *α is an SDD that respects vtree v iff:*

- $\alpha = \top$ *or* $\alpha = \bot$.
  Semantics: $\langle \top \rangle = true$ *and* $\langle \bot \rangle = false$
- $\alpha = X$ *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 internal, $p_1, \ldots, p_n$ are SDDs that respect subtrees of v^l, $s_1, \ldots, s_n$ are SDDs that respect subtrees of v^r, and $\langle p_1 \rangle, \ldots, \langle p_n \rangle$ is a partition.*
  Semantics: $\langle \alpha \rangle = \bigvee_{i=1}^{n} \langle p_i \rangle \wedge \langle s_i \rangle$.

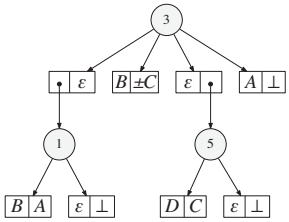
The size of SDD α, denoted $|\alpha|$, is obtained by summing the sizes of all its decompositions.

A constant or literal SDD is called *terminal*. Otherwise, it is called a *decomposition*. Fig. 1(b) is an example of an SDD that represents the Boolean function $f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$ given the vtree in Fig. 1(a). We represent $(\mathbf{X}, \mathbf{Y})$-partition as a circle node, and call it a *decision* SDD node. A decision node has child nodes, and a child node is represented as paired boxes $\boxed{p}\boxed{s}$. These child nodes are called *elements*, and the left box of an element corresponds to prime p, while the right box corresponds to sub s. Primes or subs are either a terminal SDD or a pointer to another decomposition SDD. The top-level decision node has three elements with primes representing $A \wedge B$, $\neg A \wedge B$,$\neg B$, and subs representing $true$, C, $C \wedge D$.

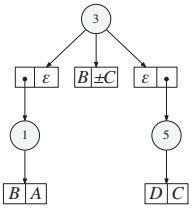
Two canonical SDDs were introduced in (Darwiche 2011). We use $\alpha = \beta$ to mean that SDDs α and β are syntactically equal, and $\alpha \equiv \beta$ to mean that they correspond to the same Boolean function, i.e., $\langle \alpha \rangle = \langle \beta \rangle$. We say a class of SDDs is canonical when the condition $\alpha = \beta$ iff $\alpha = \beta$ is satisfied for all SDDs α and β in the class. The two canonical forms are *the compressed and trimmed SDD*, and *the compressed, lightly-trimmed, and normalized SDD*. We say an SDD is trimmed if it does not have decompositions of the



(a) An SDD



(b) A trimmed ZSDD



(c) A ZSDD with implicit partitioning

Figure 2: An SDD and ZSDDs that respect vtree in Fig. 1(a), representing $\{\{A, B\}, \{B\}, \{B, C\}, \{C, D\}\}$.

form $\{(\top, \alpha)\}$ and $\{(\alpha, \top), (\neg\alpha, \bot)\}$. We say an SDD is lightly trimmed if it does not contain decompositions of the form $\{(\top, \top)\}$ and $\{(\top, \bot)\}$. We say an SDD is normalized if for all decompositions that respect vtree node v, its primes respect vtree node v^l and its subs respect vtree node v^r.

Canonical SDDs support bottom-up construction by using the `Apply` function, which takes two SDDs α, β and binary operation $\circ$, and returns a new SDD that represents a Boolean function that corresponds to $\langle\alpha\rangle \circ \langle\beta\rangle$. The `Apply` function runs in $O(|\alpha||\beta|)$ time if the resulting SDD is assumed to be uncompressed. As is proved in (Van den Broeck and Darwiche 2015), performing `Apply` operations to compressed SDDs results in an exponentially larger compressed SDD if the vtree is unchanged. However Van den Broeck and Darwiche (2015) report experimental evaluations showing that compressed SDDs become much smaller than non-compressed SDDs.

Zero-suppressed Sentential Decision Diagrams

We introduce a variant of SDD, which we name the *Zero-suppressed SDD (ZSDD)* since it subsumes the Zero-suppressed Binary Decision Diagram (ZDD) (Minato 1993) as a strict subset, the same as SDDs subsuming OBDDs as a strict subset. ZDD is an OBDD variant that also represents a Boolean function as a DAG, but it tends to be more succinct in representing sparse Boolean functions. Due to this feature, ZDDs are used in various fields such as frequent pattern mining (Minato, Uno, and Arimura 2008; Loekito and Bailey 2006), and are implemented in several standard OBDD manipulation software packages like CuDD (Somenzi 2012).

ZSDD shares a lot with SDD; it is based on the $(\mathbf{X}, \mathbf{Y})$-partition of a Boolean function, and partitions of variables are determined by a vtree. However, ZSDDs differ in the kind of terminal ZSDDs and their interpretation of trimmed subgraphs. In the following, we regard ZSDD as a representation of a family of sets, since this interpretation simplifies its explanation.

Definition 2. *α is a ZSDD that respects vtree v iff:*

- *$\alpha = \varepsilon$ or $\alpha = \bot$.*
  Semantics: $\langle\varepsilon\rangle = \{\emptyset\}$ and $\langle\bot\rangle = \emptyset$
- *$\alpha = X$ or $\alpha = \pm X$ and v is a leaf with variable X.*
  Semantics: $\langle X\rangle = \{\{X\}\}$ and $\langle\pm X\rangle = \{\{X\}, \emptyset\}$.
- *$\alpha = \{(p_1, s_1), \ldots, (p_n, s_n)\}$, v is internal, $p_1, \ldots, p_n$ are ZSDDs that respect subtrees of v^l, $s_1, \ldots, s_n$ are ZSDDs that respect subtrees of v^r, and $\langle p_1\rangle, \ldots, \langle p_n\rangle$ is a partition.*
  Semantics: $\langle\alpha\rangle = \bigcup_{i=1}^{n} \langle p_i\rangle \sqcup \langle s_i\rangle$.

We use terminal constant symbols ε and $\bot$. ε is the constant ZSDD that corresponds to the family of sets $\{\emptyset\}$, it corresponds to a Boolean function that returns 1 if all variables are 0. $\bot$ is the same as in SDD, and $\langle\bot\rangle = \emptyset = \mathit{false}$. Operation $\cup$ is the union operation. Operation $\sqcup$ is the join operation. The join $f \sqcup g$ of two families of sets, f and g, is defined as $f \sqcup g = \{a \cup b \mid a \in f \text{ and } b \in g\}$. Fig. 2(a), (b) shows examples of SDD and ZSDD, both represent sets of family $\{\{A, B\}, \{B\}, \{B, C\}, \{C, D\}\}$, while respecting the same vtree shown in Fig. 1(a). We can see that SDD size is 16 and ZSDD size is 8.

Next we will introduce two canonical ZSDDs, namely *compressed and trimmed* ZSDDs, and *compressed, lightly-trimmed and normalized* ZSDDs. We then prove their canonicity.

Definition 3. *We say a ZSDD is* trimmed *if it does not have decompositions of the form $\{(\varepsilon, \alpha), (\bar{\varepsilon}, \bot)\}$, $\{(\alpha, \varepsilon), (\bar{\alpha}, \bot)\}$, and $\{(\mathfrak{p}, \bot)\}$. We say a ZSDD is* lightly-trimmed *if it does not have decompositions of the form $\{(\varepsilon, \varepsilon), (\bar{\varepsilon}, \bot)\}$ and $\{(\mathfrak{p}, \bot)\}$.*

Definition 4. *We say a ZSDD α is* normalized *if for all decompositions in α respecting vtree node v, its primes respect vtree node v^l and subs respect vtree node v^r*

Fig. 2 (b) is a compressed and trimmed ZSDD, and Fig.3 is a compressed, lightly-trimmed, and normalized ZDDD representing the same family of sets. These definitions of canonical forms mirror that of SDD. The main difference is the trimming rule: ZSDDs remove a decision node if the corresponding Boolean function returns *true* only when all the variables in either subtrees v^l or v^r are assigned 0, while SDDs remove a decision node if the corresponding Boolean function does not depend on the variables in either of subtree v^l or v^r. This difference makes ZSDDs a more effective representation for sparse Boolean functions, since sparse Boolean functions have models that assign zero to many variables. The families of sets $\{\{A, B\}, \{B\}, \{B, C\}, \{C, D\}\}$ is a sparse Boolean function since it has four models while there are $2^4 = 16$ possible assignments, and each model has at most two variables that take value 1. This is why the ZSDD in Fig. 2(b) is smaller than the SDD in Fig. 2(a).

We should note that compressed and trimmed ZSDDs are not always smaller than compressed and trimmed SDDs. For

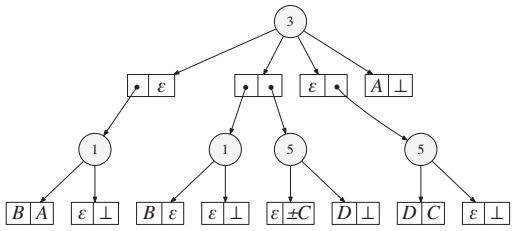


Figure 3: The normalized and lightly-trimmed ZSDD representing $\{\{A, B\}, \{B\}, \{B, C\}, \{C, D\}\}$ respects the vtree in Fig. 1(a).

Table 1: Operations supported with ZSDDs.

Operation	Description	Definition
$f \cap g$	intersection	$\{a \mid a \in f \text{ and } a \in g\}$
$f \cup g$	union	$\{a \mid a \in f \text{ or } a \in g\}$
$f - g$	difference	$\{a \mid a \in f \text{ and } a \notin g\}$
$\texttt{Change}(f, X)$	change	$\{\texttt{Change}(a, X) \mid a \in f\}$
$f \sqcup g$	orthogonal join	$\{a \cup b \mid a \in f \text{ and } b \in g\}$

example, m-ary Boolean function $f = \{\{X_1\}\}$ is represented by size 1 ZSDDs respecting any vtree, while the sizes of compressed and trimmed SDDs representing the same function are $O(m)$. On the other hand, m-ary Boolean formula $g = x_1$ can be represented by size 1 SDDs, while the sizes of ZSDDs representing g are $O(m)$.

Proposition 5. *If α and β are compressed and trimmed ZSDDs, then $\alpha \equiv \beta$ iff $\alpha = \beta$.*

The proof is presented in the appendix, and it is similar to that advanced for SDDs (Darwiche 2011), but different in how to guarantee the uniqueness of the vtree node a compressed and trimmed ZSDD respects.

`Apply` operations with ZSDDs

Like SDDs, ZSDDs support `Apply` operations, and can be constructed in a bottom-up manner. We introduce the operations in Tab. 1. By combining these five operations, we can construct ZSDDs that represent arbitrary families of sets. The first three operations in the table corresponds to binary operations over two Boolean functions. For example, the intersection $f \cap g$ of two families of sets can be obtained by applying Boolean operation $\wedge$ to the pair of Boolean formulas f, g. Therefore, these operations can be performed as the `Apply` operation that is almost equal to that for SDDs. We show the procedure in Algorithm 1. Here we assume α and β are compressed, lightly-trimmed and normalized ZSDDs. Operations for compressed and trimmed ZSDDs are almost the same, but functions $\texttt{Expand}(\gamma)$ and $\texttt{UniqueD}(\gamma)$ work differently.

The running time of `Apply` for compressed, lightly-trimmed, and normalized ZSDDs is $O(|\alpha||\beta|)$, which is the same with canonical SDDs. However, the running time with compressed and trimmed ZSDDs may be larger than $O(|\alpha||\beta|)$. This is because we need to compute $\bar{\varepsilon}$ on $\texttt{Expand}(\gamma)$. In the case of SDDs, the size of the canonical SDD representing the negation is the same as that of the original SDD. On the other hand, the ZSDD representing the negation of a Boolean function may be $O(m)$ times larger than the original ZSDD in the worst case, where m is the number of variables. However, it still remains polytime algorithm. Moreover, we can avoid the need to explicitly handle negation by using the technique called implicit partition. We show details in the next section. When we consider compressing SDDs after `Apply` operation, the size of the resulting ZSDD may increase exponentially. This is the same result as for the case of SDDs (Van den Broeck and Darwiche 2015).

Algorithm 1 $\texttt{Apply}(\alpha, \beta, \circ)$. α, β are compressed, lightly-trimmed, and normalized ZSDDs, and $\circ$ is a binary operation for families of sets.

$\texttt{Expand}(\gamma)$ returns $\{(\varepsilon, \varepsilon), (\bar{\varepsilon}, \bot)\}$ if $\gamma = \varepsilon$; $\{(\mathfrak{p}, \bot)\}$ if $\gamma = \bot$; else γ. $\texttt{UniqueD}(\gamma)$ returns ε if $\gamma = \{(\varepsilon, \varepsilon), (\bar{\varepsilon}, \bot)\}$; $\bot$ if $\gamma = \{(\mathfrak{p}, \bot)\}$; else the unique ZSDD with elements γ.

if α and β are constants or literals **then**
  return $\alpha \circ \beta$
else if $\texttt{Cache}(\alpha, \beta, \circ) \neq \texttt{nil}$ **then**
  return $\texttt{Cache}(\alpha, \beta, \circ)$
else
  $\gamma \leftarrow \{\}$
  for all elements (p_i, s_i) in $\texttt{Expand}(\alpha)$ **do**
    for all elements (q_j, r_j) in $\texttt{Expand}(\beta)$ **do**
      $p \leftarrow \texttt{Apply}(p_i, q_j, \cap)$
      if p is consistent **then**
        $s \leftarrow \texttt{Apply}(s_i, r_j, \circ)$
        add element (p, s) to γ
return $\texttt{Cache}(\alpha, \beta, \circ) \leftarrow \texttt{UniqueD}(\gamma)$

Algorithm 2 $\texttt{Change}(\alpha, X)$

if α is constants or literals **then**
  return ε if $\alpha = X$; X if $\alpha = \varepsilon$; else α
else if $\texttt{Cache}(\alpha, X, \texttt{Change}) \neq \texttt{nil}$ **then**
  return $\texttt{Cache}(\alpha, X, \texttt{Change})$
else
  $\gamma \leftarrow \{\}$
  for all elements (p_i, s_i) in $\texttt{Expand}(\alpha)$ **do**
    if X is contained in the left vtree v^l **then**
      $p \leftarrow \texttt{Change}(p_i, X)$, $s \leftarrow s_i$
    else
      $p \leftarrow p_i$, $s \leftarrow \texttt{Change}(s_i, X)$
    add element (p, s) to γ
return $\texttt{Cache}(\alpha, X, \texttt{Change}) \leftarrow \texttt{UniqueD}(\gamma)$

The next operation is $\texttt{Change}(\alpha, X)$, which is introduced in (Minato 1993) as a basic operation required for constructing ZDDs that represents arbitrary families of sets. It takes a ZSDD and variable X, and returns a family of sets whose item a that satisfies $X \in a$ is replaced by $a - \{X\}$ and whose item b that satisfies $X \notin b$ is replaced by $b \cup \{X\}$. For example, $\texttt{Change}(\alpha, C)$ where $\langle\alpha\rangle = \{\{A, B, C\}, \{A\}, \emptyset\}$ returns the family of sets $\{\{A, B\}, \{A, C\}, \{C\}\}$. Algorithm 2 shows the procedure of `Change` as applied to a compressed and normalized ZSDD. The process is similar to `Apply`, but some operations are different. A useful fact is that `Change` operation

runs in $O(|\alpha|)$ time even for canonical ZSDDs since we do not need to perform the compress operation. We can construct any ZSDD by combining Change and the above three set operations.

Another useful operation is orthogonal join. Join $f \sqcup g$ of two families, f and g, is defined as $f \sqcup g = \{a \cup b \mid a \in f \text{ and } b \in g\}$. Though it is not a simple binary operation, ZDD supports the polytime join operation (Knuth 2011; Minato 1994). Currently, we cannot find a polytime join operation for ZSDDs, however, we can perform polytime join $f \sqcup g$ on canonical ZSDDs if we can assume that f and g are orthogonal, i.e., $a \cap b = \emptyset$ for all $a \in f$, $b \in g$. For example, given $f = \{\{A, B\}, \{B\}\}$ and $g = \{\{C\}, \emptyset\}$, f and g are orthogonal and their orthogonal join $f \sqcup g = \{\{A, B\}, \{A, B, C\}, \{B\}, \{B, C\}\}$. The orthogonal join algorithm is similar to `Apply` in Alg. 1, and is implemented by substituting $p \leftarrow \texttt{Apply}(p_i, q_j, \cap)$ (line 9) with $\texttt{Apply}(p_i, q_j, \sqcup)$. Orthogonal join has the following property. The proof is given in the appendix.

Proposition 6. *Orthogonal join runs in $O(|\alpha||\beta|)$ with any canonical ZSDDs α and β.*

Implicit Partition

Here we introduce a technique called *implicit partition* defined as follows.

Definition 7. *We say canonical ZSDD α employs* implicit partitioning *if none of the partitions contained in α have an element of the form $(\beta, \bot)$.*

Implicit partition implicitly represents an element whose sub is $\bot$. Consider the compressed $(\mathbf{X}, \mathbf{Y})$-partition $\{(p_1, s_1), \ldots, (p_{n-1}, s_{n-1}), (p_n, \bot)\}$. Since it satisfies $\cup_{i=1}^{n} \langle p_i \rangle = \mathfrak{p}$, we can recover p_n from other elements using $\langle p_n \rangle = \mathfrak{p} - \bigcup_{i=1}^{n-1} \langle p_i \rangle$[2]. Hence, removing elements whose sub is $\bot$ does not drop any information about the Boolean function. Figure 2(c) shows a ZSDD that employs implicit partitioning. Compared with the ZSDD in Fig. 2(b), we can see that elements with $\bot$ are removed, which yields a more succinct representation.

Implicit partition makes ZSDDs more succinct representation for sparse Boolean functions. If we represent Boolean function f as a compressed $(\mathbf{X}, \mathbf{Y})$-partition $\{(p_1, s_1), \ldots, (p_{n-1}, s_{n-1}), (p_n, \bot)\}$, then p_n in element $(p_n, \bot)$ tends to be not sparse, i.e., it has relatively many models and each model has many variables whose value is 1. This is because p_n represents a Boolean function over $\mathbf{X}$ that returns *true* for $\mathbf{x}$ that makes $f = \mathit{false}$ for any assignment on $\mathbf{Y}$. If f is a sparse Boolean function, there tend to be many such assignments, and every assignment tends to have many variables whose value is 1. Thus implicitly representing p_n can reduce ZSDD size.

Implicit partition can reduce ZSDD size, and achieve $O(|\alpha||\beta|)$ `Apply` operations with some operators including $\cap$ and $\sqcup$. Furthermore, it supports all the polytime queries that the original ZSDD supports. We see the supported queries in the next section. In the following, we show some important properties of implicit partitioning. First we show that the canonicity is preserved when we employ implicit partitioning.

Proposition 8. *Let α, β be compressed and trimmed ZSDDs with implicit partitioning. Then $\alpha = \beta$ iff $\alpha \equiv \beta$*

Clearly, the size of a canonical ZSDD with implicit partitioning is always smaller than the same one without implicit partitioning. Furthermore, we can give an upper bound on the size of a ZSDD with implicit partitioning that is derived from the size of the family of sets.

Proposition 9. *If f is a family of sets, then the size of a compressed and trimmed ZSDD that exploits implicit partitioning and respects any vtree is always smaller than $\sum_{s \in f} |s|$,*

Proof is given in the appendix. This upper bound is common with ZDD, and it also motivates us to use ZSDDs to represent sparse Boolean functions.

Implicit partitioning requires additional computations on `Apply` operations since it uses $(p_n, \bot)$ at Alg. 1 line 7 and 8. Since $(p_n, \bot)$ does not appear in a ZSDD with implicit partitioning, we have to compute $(p_n, \bot)$ from other elements $(p_1, s_1), \ldots, (p_{n-1}, s_{n-1})$ before executing the procedures in line 9 to 12. However, for operations that satisfy $\texttt{Apply}(\bot, \alpha, \circ) = \bot$ for any α, we can skip this computation since $s = \bot$ at line 11 if $s_i = \bot$ or $r_i = \bot$, and the resulting element $(p, s = \bot)$ is removed from the ZSDD with implicit partition. Operations $\cap$ and $\sqcup$ satisfy this condition. Since $\texttt{Change}(\bot, X) = \bot$ for any variable X, `Change` also can be applied in polytime with ZSDDs that use implicit partitioning.

Properties of ZSDDs

We first show the relationships between a trimmed ZSDD and a trimmed ZSDD with implicit partitioning.

Proposition 10. *There exists a class of Boolean functions f_m and corresponding vtrees T_m, over m variables, such that f_m has a trimmed ZSDD with implicit partitioning of size $O(m^2)$ with regard to vtree T_m, yet the trimmed ZSDD of function f_m with regard to vtree T_m has size $\Omega(2^m)$.*

The proof is given in the appendix. We show that there exists partition $\{(p_1, s_1), \ldots, (p_{n-1}, s_{n-1}), (p_n, \bot)\}$ where the size of p_i is $O(m)$ and the size of p_n is $\Omega(2^m)$. This result suggests that ZSDDs with implicit partitioning can be much smaller than vanilla ZSDDs.

Next, we show the queries and transformations supported by canonical ZSDDs and ZSDDs with implicit partitioning. Details of the queries are found in (Darwiche and Marquis 2002).

Proposition 11. *The results in Table 2 hold.*

The proof is given in the appendix. Since ZSDDs are similar to SDDs, they can handle the same kinds of queries in polytime. An important fact is that employing implicit partitioning does not change the kinds of queries supported. Model counting is one of the most important queries since it is used in various applications such as probabilistic inference (Chavira and Darwiche 2008) and statistical relational learning (Fierens et al. 2015). Since we can count models in

[2]This corresponds to Boolean formula $\langle p_n \rangle = \neg \left(\bigvee_{i=1}^{n-1} \langle p_i \rangle \right)$.

Table 2: Analysis of supported queries for SDD, ZSDD and ZSDD with implicit partition (Z+I). Definitions of queries are in (Darwiche and Marquis 2002). We use $\surd$ if it can answer the query in polytime.

Notation	Query	SDD	ZSDD	Z+I
CO	consistency	$\surd$	$\surd$	$\surd$
VA	validity	$\surd$	$\surd$	$\surd$
CE	clausal entailment	$\surd$	$\surd$	$\surd$
IM	implicant check	$\surd$	$\surd$	$\surd$
EQ	equivalence check	$\surd$	$\surd$	$\surd$
CT	model counting	$\surd$	$\surd$	$\surd$
SE	sentential entailment	$\surd$	$\surd$	$\surd$
ME	model enumeration	$\surd$	$\surd$	$\surd$

Table 3: Analysis of supported transformations for SDD, ZSDD and ZSDD with implicit partitioning. Six columns indicates the form of output SDDs and ZSDDs: first three colums are uncompressed SDDs(S), ZSDDS (Z), ZSDDs with implicit partition (Z+I), and remainig columns are compressed and trimmed SDDs, (S(C)), ZSDDs (Z(C)), and ZSDDs with implicit partition (Z+I(C)). $\surd$ indicates that there exists a polytime algorithm, $\bullet$ indicates that such algorithm is proven to be impossible, and ? indicates our ignorence about the property. The results for S and S(C) are shown in (Van den Broeck and Darwiche 2015).

Notation	Transformation	S	Z	Z+I	S(C)	Z(C)	Z+I(C)
CH	change	?	$\surd$	$\surd$	?	$\surd$	$\surd$
$\cap\mathbf{C}$	intersection	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
$\cap\mathbf{BC}$	bounded intersection	$\surd$	$\surd$	$\surd$	$\bullet$	$\bullet$	$\bullet$
$\cup\mathbf{C}$	union	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
$\cup\mathbf{BC}$	bounded union	$\surd$	$\surd$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
$\sqcup\mathbf{OC}$	orthogonal join	?	$\surd$	$\surd$	?	$\surd$	$\surd$
$\bar{\mathbf{C}}$	complement	$\surd$	$\surd$	$\bullet$	$\surd$	$\surd$	$\bullet$
CO	conditioning	$\surd$	$\surd$	$\surd$	$\bullet$	$\bullet$	$\bullet$
FO	forgetting	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
SFO	singleton forgetting	$\surd$	$\surd$	$\bullet$	$\bullet$	$\bullet$	$\bullet$

time linear with ZSDD size if we employ implicit partitioning, implicit partitioning is useful.

Next we see the transformations supported in ZSDDs with implicit partitioning. Since typical transformations used in families of set interpretation are different from the standard transformations, we consider following transformations: change (**CH**), intersection ($\cap\mathbf{C}$), bounded intersection ($\cap\mathbf{BC}$), union ($\cup\mathbf{C}$), bounded union ($\cup\mathbf{BC}$), orthogonal join ($\sqcup\mathbf{OC}$), and complement ($\bar{\mathbf{C}}$). Intersection (union) is the intersection between multiple ZSDDs, and bounded intersection is the intersection of two ZSDDs. We also assume orthogonal join is performed between two ZSDDs. We also consider the transformations for Boolean function shown in (Darwiche and Marquis 2002): conditioning (**CD**), forgetting (**FO**), and singleton forgetting (**SFO**)[3].

Proposition 12. *The results in Table 3 hold.*

As for transformations, ZSDDs and implicitly partitioned ZSDDs show different results. This is due to the exponentially large difference in their sizes. Employing implicit partitioning may result in exponentially large representations after transformations $\cup\mathbf{BC}$ and $\bar{\mathbf{C}}$, however, the resulting ZSDD with implicit partitioning is still smaller than the equivalent vanilla ZSDD.

The next important property of ZSDD is its relationship with ZDD.

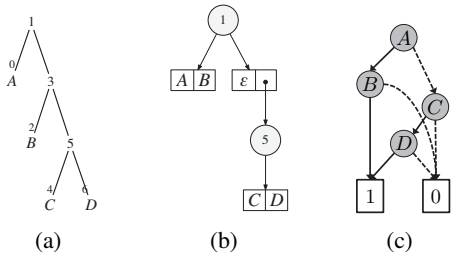


Figure 4: Example of (a) right-linear vtree, (b) ZSDD with implicit partition, and (c) ZDD. The ZSDD and the ZDD represent the family of sets $\{\{A, B\}, \{C, D\}\}$. The ZSDD respects the vtree in (a), and the ZDD employs the variable order A, B, C, D.

Property 13. *A compressed and trimmed ZSDD respecting a right-linear vtree corresponds to a ZDD that is based on the total variable order induced from the vtree. Every decomposition in the ZSDD corresponds to a decision node in the ZDD, and every decision node in the ZDD corresponds to a decomposition or literal in the ZSDD.*

We say a vtree is *right-linear* if each left-child of an internal node is a leaf. This property is analogous to the correspondence of SDDs to OBDDs when they respect right-linear vtrees. We show examples of a ZDD and a ZSDD that respect a right-linear vtree in Fig. 4.

Finally, we give a theoretical upper bound on ZSDD size when representing CNFs. Darwiche (2011) showed that the size of an SDD representing a CNF with m variables is bounded by $O(m2^w)$, where w is the treewidth of the CNF. Compressed, lightly trimmed and normalized ZSDDs have the same upper bound on size. Darwiche (2011) proved the bound by showing that the number of decision nodes respecting a vtree node is bounded by 2^w, and we can construct a SDD wherein every decision node has at most two elements. In the case of ZSDDs, we can use the same bound on the number of decision nodes, and we can construct a ZSDD for which every decision node has at most two elements. Hence it has the same bounds.

Evaluation

We compared the sizes of ZSDDs, SDDs, and ZSDDs that used right-linear vtrees. ZSDDs employing right-linear vtree corresponds to ZDDs. We used compressed and trimmed ZSDDs and those with implicit partitioning. We used the SDD package[4] to construct compressed and trimmed SDDs. The vtrees for ZSDDs and SDDs were determined by the dynamic reordering algorithm (Choi and Darwiche 2013) im-

[3] We omit results for $\wedge\mathbf{C}$, $\wedge\mathbf{BC}$, $\vee\mathbf{C}$, $\vee\mathbf{BC}$, and $\neg\mathbf{C}$, since they are equivalent to families of sets transformations $\cap\mathbf{C}$, $\cap\mathbf{BC}$, $\cup\mathbf{C}$, $\cup\mathbf{BC}$, and $\bar{\mathbf{C}}$, respectively.

[4] http://reasoning.cs.ucla.edu/sdd/

plemented in the SDD package. We used a balanced vtree as the initial vtree for reordering. Right-linear vtrees of ZSDDs are induced from the results of the reordering algorithm. Here we say an order is induced from a vtree if the left-right traversal of the vtree gives the visit order of variables (Xue, Choi, and Darwiche 2012). As the dataset, we used LGSynth89 benchmark dataset. We omitted datasets for which the reordering algorithm did not finish within 24 hours.

We show the results from the LGSynth89 benchmark dataset in Tab. 4. Here ZSDD, Z+I(C), SDD corresponds to compressed and trimmed ZSDDs, compressed and trimmed ZSDDs with implicit partitioning, and compressed and trimmed SDDs, respectively. Although the vtrees are optimized for SDDs, ZSDD and Z+I(C) are always smaller than both SDD and ZSDD with right linear vtrees. This would be because the CNFs in the dataset have smaller number of models, thus they are relatively sparse Boolean function. Since ZSDDs also support query functions such as model counting in polytime, ZSDDs can be used as small alternatives of SDDs. Z+I(C) always has smaller size than right-linear Z+I(C). Employing a linear order sometimes increases ZSDD size by more than ten times.

Table 4: Results of experiments on LGSynth89 dataset. ZSDD are compressed and trimmed ZSDDs, Z+I(C) are compressed and trimmed ZSDDs employing implicit partitioning, SDD are compressed and trimmed SDDs, and ZSDD linear and Z+I(C) linear uses the right-linear vtrees induced from vtrees used in other settings.

CNF	ZSDD	Z+I(C)	SDD	ZSDD linear	Z+I(C) linear
9symml	17467	5965	19278	36134	18578
alu2	10717	3271	13060	17722	9384
apex7	8832	3252	9905	232993	149648
b1	87	34	169	76	45
b9	8054	2876	8743	75572	43438
C17	82	43	104	68	47
C432	9009	3294	9634	62586	35320
c8	21191	9249	22359	207863	127795
cc	1350	424	1668	2062	1280
cht	3068	1013	4044	2357	1609
cm138a	349	125	489	360	197
cm150a	1185	417	1508	1797	1200
cm151a	611	197	758	782	566
cm152a	163	113	177	154	129
cm162a	875	318	1121	1060	669
cm163a	773	226	967	605	391
cm42a	348	121	468	333	181
cm82a	127	54	235	94	60
cm85a	914	324	1167	1064	630
cmb	946	378	1128	1622	1077
comp	1981	597	2502	1914	1172
count	2632	887	3177	3256	1998
cu	1412	517	1736	1558	989
decod	230	83	403	122	84
example2	10155	3892	11229	56746	36406
f51m	2998	1124	4350	4644	2568
frg1	100890	61609	104219	228894	156936
lal	10685	4236	11486	204556	110063
ldd	1237	239	2063	474	273
majority	71	45	126	61	47
mux	1355	475	1761	2065	1528
my_adder	2356	487	3109	1686	1046
parity	326	106	525	310	212
pcle	796	269	968	585	400
pcler8	1215	387	1423	1132	771
pm1	2422	1016	2770	6819	4090
sct	6995	2627	7880	37138	22059
tcon	415	156	583	277	183
term1	132677	76811	137150	3890084	2045598
ttt2	26192	8621	28451	326007	183673
unreg	4188	1563	4593	8348	5500
x2	582	176	825	412	273
x4	33251	13789	35729	1136754	716757
z4ml	1422	443	2325	1008	610

Conclusion

We proposed ZSDD, a variant of SDD. ZSDDs subsume ZDDs as a strict subset, and also support useful bottom-up operations that can construct any family of sets by combining operations. ZSDDs are generally more succinct than SDDs if representing sparse Boolean functions. The implicit partitioning technique is simple, but practical as it contributes much to reducing the size of ZSDDs. We also showed supported queries and transformations, and the results of experiments suggest that ZSDDs can be used as a more compact alternatives to SDDs and ZDDs.

References

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1997. Algebric decision diagrams and their applications. *Formal methods in system design* 10(2-3):171–206.

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

Chakraborty, S.; Fremont, D. J.; Meel, K. S.; Seshia, S. A.; and Vardi, M. Y. 2014. Distribution-aware sampling and weighted model counting for SAT. In *AAAI*, 1722–1730.

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172(6):772–799.

Choi, A., and Darwiche, A. 2013. Dynamic minimization of sentential decision diagrams. In *AAAI*, 187–194.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17(1):229–264.

Darwiche, A. 2001. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics* 11(1-2):11–34.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI*, 819–826.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2015. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming* 15(03):358–401.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014. Probabilistic sentential decision diagrams. In *KR*, 558–567.

Knuth, D. E. 2011. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley.

Loekito, E., and Bailey, J. 2006. Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams. In *KDD*, 307–316.

Minato, S.; Uno, T.; and Arimura, H. 2008. LCM over ZBDDs: Fast generation of very large-scale frequent itemsets using a compact graph-based representation. In *PAKDD*, 234–246.

Minato, S. 1993. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *DAC*, 272–277.

Minato, S. 1994. Calculation of unate cube set algebra using zero-suppressed BDDs. In *DAC*, 420–424.

Pipatsrisawat, K., and Darwiche, A. 2008. New compilation languages based on structured decomposability. In *AAAI*, 517–522.

Somenzi, F. 2012. CUDD: CU decision diagram package. http://vlsi.colorado.edu/~fabio/CUDD/.

Van den Broeck, G., and Darwiche, A. 2015. On the role of canonicity in knowledge compilation. In *AAAI*, 1641–1648.

Xue, Y.; Choi, A.; and Darwiche, A. 2012. Basing decisions on sentences in decision diagrams. In *AAAI*, 842–849.

Appendix

Proof of Proposition 5 We first introduce key concepts for proving the canonicity of ZSDDs.

Definition 14. *Let variable X* supports *Boolean function $f(\mathbf{X})$ if there is an assignment $\mathbf{x}$ where $x = 1$ and $f(\mathbf{x}) = 1$.*

Definition 15. *We say function $f(\mathbf{X})$* inherently supports *vtree node v if f is not trivial and if v is the deepest vtree node that includes all variables that support f.*

To prove the canonicity, we state the following lemmas.

Lemma 16. *A non-trivial function essentially supports exactly one vtree node.*

Proof. If there are two different vtree nodes, u and v, that include all variables that Boolean function f supports, then either u or v must be the ancestor of the other. Thus we can select exactly one deepest vtree node. □

The following lemma assumes ZSDDs to be compressed and trimmed, but it can also be applied to normalized ZSDDs.

Lemma 17. *Let α be a compressed and trimmed ZSDD. If $\alpha \equiv \{\emptyset\}$, then $\alpha = \varepsilon$. If $\alpha \equiv \emptyset$, then $\alpha = \bot$. Otherwise, there is a unique vtree node, v, that function $\langle\alpha\rangle$ inherently supports.*

Proof. Suppose that $\alpha \equiv \{\emptyset\}$ and that α respects vtree node v. If v is a leaf node, then the only ZSDD that can represent $\{\emptyset\}$ is ε. If v is an internal node, then the only partition that corresponds to $\{\emptyset\}$ is $\{(\{\emptyset\}, \{\emptyset\}), (\{\bar{\emptyset}\}, \emptyset)\}$. By applying trimming rules, this partition is replaced by a ZSDD that respects a left or right descendant vtree node and represents $\{\emptyset\}$. Hence $\{\emptyset\}$ is represented as terminal ZSDD ε. Similarly, the only ZSDD that can represent $\emptyset$ is $\bot$. From lemma 16, if a Boolean function is not trivial, there is only one vtree node that the function inherently supports. □

The Proposition 5 can be proved by using the above lemmas.

Proof. If $\alpha = \beta$, then $\alpha \equiv \beta$ by definition. Suppose $\alpha \equiv \beta$ and let $f = \langle\alpha\rangle = \langle\beta\rangle$. If $f = \{\emptyset\}$, then $\alpha = \beta = \varepsilon$. If $f = \emptyset$, then $\alpha = \beta = \bot$. If f is not trivial, then it must inherently support unique vtree node v. Suppose node v is a leaf, then only terminal ZSDDs respect leaf nodes, and α and β must be X or $\pm X$ unless f is trivial. This means that $\alpha = \beta$ if $\alpha = \beta$. Suppose that vtree node v is an internal node, and the proposition holds for subtrees v^l and v^r. Since a function has exactly one compressed $(\mathbf{X}, \mathbf{Y})$-partition (Darwiche 2011), $\alpha \equiv \beta$ means partitions $\alpha = \{(p_1, s_1), \ldots, (p_n, s_n)\}$ and $\beta = \{(q_1, r_1), \ldots, (p_k, r_k)\}$ are identical. From the assumption, $p_i = q_i$ and $s_i = r_i$ for all i, and thus $\alpha = \beta$. □

Proof of Proposition 6 We assume that α and β are both decomposition ZSDDs and represented as partitions $\alpha = \{(p_1, s_1), \ldots, (p_n, s_n)\}$ and $\beta = \{(q_1, r_1), \ldots, (q_k, r_k)\}$. Then the set of new primes consists of $p_i \sqcup q_j$, where $i = 1, \ldots, n$ and $j = 1, \ldots, k$. This set of primes forms a partition. Similarly, subs $s_i \sqcup r_j$ are compressed. Hence the algorithm for taking orthogonal join does not require compression, and it runs in time $O(|\alpha||\beta|)$. □

Proof of Proposition 8 Even if we employ implicit partitioning, trivial ZSDDs ε, $\bot$ are the only ones that represent trivial functions. Moreover, a ZSDD inherently supports a unique vtree node, and partitions are unique. Hence $\alpha \equiv \beta$ means $\alpha = \beta$. □

Proof of Proposition 9

Lemma 18. *If a family of sets f is not an empty family $\emptyset$, then compressed and trimmed ZSDDs employing implicit partitioning do not contain $\bot$.*

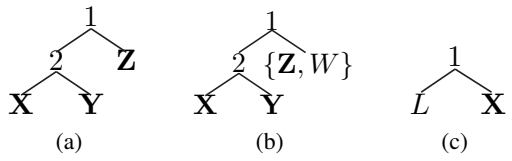
This lemma is obvious from the definition of implicit partitioning. A set contained in a family of sets corresponds to a tree structure contained in the ZSDD. The tree is made by first selecting the root decision node, then recursively selecting one child element for each decision node and selecting both prime and sub at each element. For example, in the ZSDD of Fig. 2(c), set $\{A, B\}$ corresponds to the leftmost path from the top decision node. Such subtrees representing sets satisfy the following condition.

Lemma 19. *If a ZSDD is compressed, trimmed, and employs implicit partitioning, then set s contained in the ZSDD corresponds to a tree, and the tree has at most $|s| - 1$ decision nodes whose size is 1.*

Proof. A decision node that has one element means both the prime and the sub of the element must contain at least one variable, otherwise prime or sub equal constants and the decision node is deleted from the ZSDD. If the tree has more than $|s| - 1$ such nodes, it means the set must contain more than $|s|$ items. Thus the tree contains at most $|s| - 1$ size 1 decision nodes. □

Lemma 20. *Given a compressed and trimmed ZSDD with implicit partitioning that represents family of sets f, then the difference between the sum of the sizes of decompositions, E, and the number of decision nodes in the ZSDD, D, satisfies $E - D + 1 \leq |f|$.*

Proof. Since there is no $\bot$ in the ZSDD with implicit partitioning, every sub-function f' that corresponds to a decision node must have at least one assignment that makes $f' = true$. If a decision node has n elements, it means that there are at least n such assignments, i.e., there are at least n sets contained in f. If a decision node has n elements and another node has m elements, it means that at least $n + m - 1$ sets are contained in f. In this way, if the total number of



(a) (b) (c)

Figure 5: vtrees used in proofs

decision nodes is D and the sum of decompositions is E, it means that f contains at least $E - D + 1$ sets. □

We can prove the proposition using these lemmas.

Proof. We consider the largest ZSDD that represents a family of sets f. Its size corresponds to solving the following optimization problem of maximizing E under constraints $E - D + 1 \leq |f|$ and $D \leq \sum_{s \in f} (|s| - 1)$. The solution of this optimization problem is $E = \sum_{s \in f} |s|$. □

Proof of Proposition 10 We prove this proposition by showing an example function. Function $f^a(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) = \left[\bigvee_{i=1}^{m} (\bigwedge_{j=1}^{i-1} \neg Y_j) \wedge Y_i \wedge \neg X_i \wedge Z_i\right] \vee [\bigwedge_{i=1}^{m} \neg Y_i]$ has $3m$ variables. Consider the vtree in the Fig. 5(a), where the variable orders over $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ are arbitrary. The compressed $(\mathbf{XY}, \mathbf{Z})$-partition of the function that respects the root vtree node is

$$\bigcup_{i=1}^{m} \left\{ \left(\bigvee_{j=1}^{i-1} \neg Y_j \wedge Y_i \wedge \neg X_i, Z_i \right) \right\}$$
$$\cup \left\{ \left(\bigwedge_{i=1}^{m} \neg Y_i, \top \right) \right\} \cup \left\{ \left(\bigvee_{i=1}^{m} \left(\bigwedge_{j=1}^{i-1} \neg Y_j \right) \wedge Y_i \wedge X_i, \bot \right) \right\}.$$

Following the result of (Van den Broeck and Darwiche 2015), each prime and sub in the first and second group can be represented by ZSDDs with $O(m)$ nodes. On the other hand, the prime of the last group results in a ZSDD with $\Omega(2^m)$ nodes. Since implicit partitioning ignores the last element, the size of a ZSDD with implicit partitioning is $O(m^2)$, otherwise $\Omega(2^m)$. □

Proof of Proposition 11 **CT** can be performed on ZSDDs with an algorithm that is very close to that for d-DNNFs (Darwiche 2001). Since the model count for any element of the form $(\alpha, \bot)$ is always zero, we can also count the number of models in linear time with the size of ZSDDs with implicit partitioning. Polytime **ME** can be performed by a bottom-up process similar to **CT**.

SE can be performed in the procedure shown in (Pipatsrisawat and Darwiche 2008), i.e., taking the intersection between two ZSDDs and then counting models. Since intersection can be performed in polytime even when exploiting implicit partitioning, both ZSDD and Z+I support **SE** in polytime, which means that ZSDD and Z+I support **EQ**, **CO**, **VA**, **IM**, and **CE**, since these queries can be answered by using **SE**. □

Proof of Proposition 12 We first consider uncompressed cases. Alg. 1 can perform $\cap\mathbf{BC}$ and $\cup\mathbf{BC}$ in polytime for vanilla ZSDDs. When we employ implicit partitioning, $\cap\mathbf{BC}$ can be computed in polytime, but $\cup\mathbf{BC}$ can result in exponentially large uncompressed ZSDDs. Consider function f^a used above and $g(\mathbf{X}, \mathbf{Y}, \mathbf{Z}, W) = W$, both of which respect the vtree in Fig. 5 (b). Since the union operation $\cup$ for families of sets corresponds to the Boolean operation $\vee$, we compute the union by $f^a \vee g$. The $(\mathbf{XY}, \mathbf{Z}W)$-partition of $f^a \vee g$ is

$$\bigcup_{i=1}^{m} \left\{ \left(\bigvee_{j=1}^{i-1} \neg Y_j \wedge Y_i \wedge \neg X_i, Z_i \vee W \right) \right\}$$
$$\cup \left\{ \left(\bigwedge_{i=1}^{m} \neg Y_i, \top \right) \right\} \cup \left\{ \left(\bigvee_{i=1}^{m} \left(\bigwedge_{j=1}^{i-1} \neg Y_j \right) \wedge Y_i \wedge X_i, W \right) \right\}.$$

Since the prime in the third group is a ZSDD whose size is $\Omega(2^m)$, $\cup\mathbf{BC}$ is not polytime for Z+I and Z+I(C).

Complement $\bar{\mathbf{C}}$ of a compressed and trimmed ZSDD is at most km times larger than the original ZSDD, where m is the number of vtree nodes, and k is a constant. This bound derives from the fact that the size of a compressed, lightly trimmed, and normalized ZSDD is at most km times larger than that of a compressed and trimmed ZSDD, and the size of the complement of a normalized ZSDD is almost equal to that of the original ZSDD. Thus it is polytime for uncompressed ZSDDs. Furthermore, as shown in (Darwiche 2011), $\bar{\mathbf{C}}$ of compressed partition $\{(p_1, s_1), \ldots, (p_n, s_n)\}$ is compressed partition $\{(p_1, \neg s_1), \ldots, (p_n, \neg s_n)\}$. Therefore, $\bar{\mathbf{C}}$ is also polytime for compressed ZSDDs. On the other hand, if a Z+I represents f^a in the proof of Prop.10, then its complement is exponentially large. Therefore $\bar{\mathbf{C}}$ is not polytime for Z+I and Z+I(C).

The proof for $\sqcup\mathbf{OC}$ follows from Proposition 6. The result is also true for Z+I and Z+I(C). $\mathbf{CH}$ can be performed using Alg. 2, and the obtained result is also syntactically a ZSDD since the primes in the partitions of the obtained ZSDD consistent, exhaustive and mutally exclusive. Furthermore, since the subs are compressed after `Change`, $\mathbf{CH}$ is polytime for compressed ZSDD.

$\mathbf{CD}$ can be performed with a recursive procedure similar to `Change`, which differs from `Change` in the procedure for terminal ZSDDs (Alg. 2 line 1, 2). It is a polytime operation for uncompressed ZSDD, and also for Z+I since conditioning $\bot \mid \ell = \bot$ for any literal ℓ and it does not change the size of Z+I.

Since support for $\mathbf{SFO}$ follows from the support for $\mathbf{CD}$ and $\cup\mathbf{BC}$, ZSDD supports $\mathbf{SFO}$. Given any two Z+I β and γ that respect the same vtree T. We add a new varaible L to vtree T to make a vtree shown in Fig. 5 (c). Given Z+I $alpha = (L \wedge \beta) \vee (\neg L \wedge \gamma)$ that respects the vtree, and we forget L from α results in $\beta \vee \gamma$, and it becomes exponentially larger than α when $\beta \equiv f^a$ and $\gamma \equiv g$, thus Z+I does not support $\mathbf{SFO}$.

The negative results for $\mathbf{FO}$, $\cup\mathbf{C}$ and $\cap\mathbf{C}$ follows from the results for ZDDs. The proofs for negative results for $\cup\mathbf{BC}$, $\cup\mathbf{BC}$, $\mathbf{CD}$ and $\mathbf{SFO}$ whose outputs are compressed are identical to that shown in (Van den Broeck and Darwiche 2015). □