Binary Decision Diagrams with Edge-Specified Reductions

Junaid Babar(✉), Chuan Jiang, Gianfranco Ciardo, and Andrew Miner

Department of Computer Science, Iowa State University, Ames, IA 50011, USA
{junaid,cjiang,ciardo,asminer}@iastate.edu

Abstract. Various versions of binary decision diagrams (BDDs) have been proposed in the past, differing in the reduction rule needed to give meaning to edges skipping levels. The most widely adopted, fully-reduced BDDs and zero-suppressed BDDs, excel at encoding different types of boolean functions (if the function contains subfunctions independent of one or more underlying variables, or it tends to have value zero when one of its arguments is nonzero, respectively). Recently, new classes of BDDs have been proposed that, at the cost of some additional complexity and larger memory requirements per node, exploit both cases. We introduce a new type of BDD that we believe is conceptually simpler, has small memory requirements in terms of node size, tends to result in fewer nodes, and can easily be further extended with additional reduction rules. We present a formal definition, prove canonicity, and provide experimental results to support our efficiency claims.

1 Introduction

Decision diagrams (DDs) have been widely adopted for a variety of applications. This is due to their often compact, graph-based representations of functions over boolean variables, along with operations to manipulate those boolean functions based on the sizes of the graph representations, rather than the size of the domain of the function. Most DD types are canonical for boolean functions: for a fixed ordering of the function variables, each function has a unique (modulo graph isomorphism) DD representation, or encoding.

Compactness, and canonicity, is achieved through careful rules for eliminating nodes. All canonical DDs eliminate nodes that duplicate information: if nodes p and q encode the same function, one of them is discarded. Additional compactness comes from a reduction rule (or rules) that specifies both how to interpret "long" edges that skip over function variables, and how to eliminate nodes and replace them with long edges. Two popular forms of decision diagrams, Binary Decision Diagrams (BDDs) [1] and Zero-suppressed binary Decision Diagrams (ZDDs) [8], use different reduction rules. Some applications are more suitable for BDDs while others are more suitable for ZDDs, depending on which of the two reductions can be applied to a greater number of nodes. Unfortunately, it is not always easy to know, *a priori*, which reduction rule is best for a particular application. Worse, there are applications where *both* rules are useful.

T. Vojnar and L. Zhang (Eds.): TACAS 2019, Part II, LNCS 11428, pp. 303–318, 2019.
https://doi.org/10.1007/978-3-030-17465-1_17

Recently, Tagged BDDs (TBDDs) [10] and Chain-reduced BDDs (CBDDs) or ZDDs (CZDDs) [2] have been introduced to combine the reduction rules of BDDs and ZDDs. We introduce a new type of BDD, called *Edge Specified Reduction* BDDs (ESRBDDs), that we believe is conceptually simpler and has smaller node storage requirements than TBDDs, CBDDs, and CZDDs, while still exploiting the BDD and ZDD reduction rules. Additionally, ESRBDDs are flexible in that additional reduction rules may be added with low cost. Finally, unlike TBDDs, CBDDs, and CZDDs, ESRBDDs treat the BDD and ZDD reduction rules equally: there is no need to prioritize one rule over another.

The paper is organized as follows. Section 2 recalls definitions for BDDs and ZDDs and describes related work. Section 3 formally defines ESRBDDs, gives their reduction algorithm, proves that they are a canonical form, and compares them with related DDs. Section 4 gives detailed experimental results to show how the various DDs compare in practice. Section 5 provides conclusions.

2 Related Decision Diagrams

We focus on various types of DDs that have been proposed to efficiently encode boolean functions of boolean variables, and briefly recall DDs relevant to our work. For consistency in notation, all DD types we present encode functions of the form $f : \mathbb{B}^L \to \mathbb{B}$ and have L levels, with level L at the top.

The first and most widely-known type is the *reduced-ordered binary decision diagrams* (BDDs) [1]. A BDD is a directed acyclic graph where the two terminal nodes $\mathbf{0}$ and $\mathbf{1}$ are at level 0, we write $lvl(\mathbf{0}) = lvl(\mathbf{1}) = 0$, while each nonterminal node p belongs to a level $lvl(p) \in \{1, ..., L\}$ and has two outgoing edges, $p[0]$ and $p[1]$, pointing to nodes at lower levels (this is the "ordered" property). The "reduced" property instead forbids both *duplicate* nodes (p and q are duplicates if $lvl(p) = lvl(q)$, $p[0] = q[0]$, and $p[1] = q[1]$), and *redundant* nodes (p is redundant if $p[0] = p[1]$). The function F_p encoded by BDD node p is defined as

$$F_p(x_{1:L}) = \begin{cases} F_{p[x_{lvl(p)}]}(x_{1:L}) & lvl(p) > 0 \\ p & lvl(p) = 0, \end{cases}$$

where $(x_{1:L})$ is a shorthand for the boolean tuple $(x_1, ..., x_L)$.

Another widely-used type is the *zero-suppressed binary decision diagrams* (ZDDs) [8], which differ from BDDs only in that they forbid *high-zero* nodes (node p is high-zero if $p[1] = \mathbf{0}$) instead of redundant nodes. The function encoded by ZDD node p is defined with respect to a level $n \geq m = lvl(p)$, as

$$F_p^n(x_{1:n}) = \begin{cases} \mathbf{0} & n > m \wedge \exists i, m < i \leq n, x_i = 1 \\ F_p^m(x_{1:m}) & n > m \wedge \forall i, m < i \leq n, x_i = 0 \\ F_{p[x_m]}^{m-1}(x_{1:m-1}) & n = m > 0 \\ p & n = m = 0. \end{cases}$$

Both BDDs and ZDDs are *canonical*: any function $f : \mathbb{B}^L \to \mathbb{B}$ has a unique node p encoding it, an essential property guaranteeing *time* efficiency. Just as

important is their *memory* efficiency, i.e., the number of nodes required to encode a given function. In this respect, BDDs and ZDDs are particularly suited to different situations. BDDs require fewer nodes if there are many "don't cares", i.e., it often happens that $F_p(x_{1:L}) = F_p(y_{1:L})$ when $x_{1:L}$ and $y_{1:L}$ differ in one position, as this corresponds to redundant nodes, not stored in BDDs. ZDDs require fewer nodes if the function tends to have value 0 when many arguments have value 1 as this corresponds to high-zero nodes, not stored in ZDDs.

Quasi-reduced BDDs (QBDDs) [5] are also canonical: they are just like BDDs (or ZDDs) except they only forbid duplicate nodes. QBDD edges connect nodes on adjacent levels. Since edges are not allowed to *skip* levels, nodes do not need to store level information, and redundant and high-zero nodes cannot be eliminated. A useful variation is to eliminate only redundant (or high-zero) nodes whose children are $\mathbf{0}$, and thus allow long edges directly to $\mathbf{0}$. In either case, QBDDs require at least as many nodes as BDDs and ZDDs to encode a given function, so they provide an upper bound on both the BDD and the ZDD sizes.

Various decision diagrams have been proposed to combine the characteristics of BDDs and ZDDs and exploit the reduction potential of both. *Tagged binary decision diagrams* (TBDDs) [10] associate a level tag to each edge. BDD reductions are implied along the edge from the level of the node to the level of the tag, and ZDD reductions are implied from the level of the tag to the level of the node pointed to by the edge. Alternatively, TBDDs can apply reductions in the reverse order along an edge: ZDD reductions first and BDD reductions second. Either reduction order can be used in TBDDs, but a TBDD can only use one of them, i.e., they cannot both be used in the same TBDD.

Chain-reduced BDDs (CBDDs) and *chain-reduced ZDDs* (CZDDs) [2] augment BDDs and ZDDs by using nodes to encode chains of high-zero nodes and redundant nodes, respectively. Each node specifies two levels, the first level indicating where the chain starts (similar to the level of an ordinary BDD or ZDD node), and the second, additional, level indicating where the chain ends.

Finally, *ordered Kronecker functional decision diagrams* [3] allow multiple decomposition types (Shannon, positive Davio, and negative Davio), enabling both BDD and ZDD reductions. However, each level has a fixed decomposition type, thus this approach is less flexible, potentially less efficient, and hindered by the need to know which decomposition will perform best for each level.

3 ESRBDDs

Definition 1. An L-level *(ordered) edge-specified reduction* binary decision diagram (ESRBDD) is a directed acyclic graph where the two *terminal* nodes $\mathbf{0}$ and $\mathbf{1}$ are at level 0, $lvl(\mathbf{0}) = lvl(\mathbf{1}) = 0$, while each *nonterminal* node p belongs to a level $lvl(p) \in \{1, ..., L\}$ and has two outgoing edges, $p[0]$ and $p[1]$, pointing to nodes at lower levels. An edge is a pair $e = \langle e.rule, e.node \rangle$, where $e.rule$ is a *reduction rule* in $\{\mathtt{S}, \mathtt{L_0}, \mathtt{H_0}, \mathtt{X}\}$ and $e.node$ is the node to which edge e points. For $i \in \{0, 1\}$, if $lvl(p[i].node) = lvl(p) - 1$, we say that $p[i]$ is a *short* edge and require that $p[i].rule = \mathtt{S}$. If instead $lvl(p[i].node) < lvl(p) - 1$, the only other

possibility, we say that $p[i]$ is a *long* edge, since it "skips over" one or more levels, and require that $p[i].rule \in \{\mathtt{H_0}, \mathtt{L_0}, \mathtt{X}\}$.

The reduction rule on an edge specifies its meaning when skipping levels, thus it is just S for short edges while, for long edges, the rules $\mathtt{H_0}$, $\mathtt{L_0}$, and X correspond to the "zero-suppressed" rule of [8], the "one-suppressed" rule (a new rule analogous to the zero-suppressed, as we shall see), and the "fully-reduced" rule of [1], respectively. To make this more precise, we recursively define the boolean function $F^n_{\langle\kappa,p\rangle} : \mathbb{B}^n \to \mathbb{B}$ encoded by an ESRBDD edge $\langle\kappa,p\rangle$ with respect to a level $n \in \{0, ..., L\}$, subject to $lvl(p) \leq n$, as

$$F^n_{\langle\kappa,p\rangle}(x_{1:n}) = \begin{cases} \text{if } lvl(p) = n = 0 & p \\ \text{if } lvl(p) = n > 0 & (x_n)\ ?\ F^{n-1}_{p[1]}(x_{1:n-1}) \ :\ F^{n-1}_{p[0]}(x_{1:n-1}) \\ \text{if } lvl(p) < n, \kappa = \mathtt{X}, & (x_n)\ ?\ F^{n-1}_{\langle\kappa,p\rangle}(x_{1:n-1}) \ :\ F^{n-1}_{\langle\kappa,p\rangle}(x_{1:n-1}) \\ \text{if } lvl(p) < n, \kappa = \mathtt{H_0}, & (x_n)\ ?\ \mathbf{0} \ :\ F^{n-1}_{\langle\kappa,p\rangle}(x_{1:n-1}) \\ \text{if } lvl(p) < n, \kappa = \mathtt{L_0}, & (x_n)\ ?\ F^{n-1}_{\langle\kappa,p\rangle}(x_{1:n-1}) \ :\ \mathbf{0}, \end{cases}$$

where the if-then-else operator $(x_n)?f_1{:}f_0$ is a shorthand for $(\neg x_n \wedge f_0) \vee (x_n \wedge f_1)$.

We defined an ESRBDD as a directed acyclic graph, so it can potentially have multiple *roots* (nodes with no incoming edges). However, since our focus is on the size of the DD encoding a given function, we assume from now on that our ESRBDDs have a single root node $p^\star$, pointed to by a *dangling edge* with rule $\kappa^\star$. We denote the set of all nodes reachable from $p^\star$ (and therefore all nodes in the ESRBDD) as $Nodes(p^\star)$. The dangling edge $\langle\kappa^\star,p^\star\rangle$ encodes the function $F^L_{\langle\kappa^\star,p^\star\rangle}$, which is independent of $\kappa^\star$ only if $lvl(p^\star) = L$, in which case we require $\kappa^\star = \mathtt{S}$, while we require $\kappa^\star \in \{\mathtt{L_0}, \mathtt{H_0}, \mathtt{X}\}$ if $lvl(p^\star) < L$. Finally, we will informally say "ESRBDD $\langle\kappa^\star,p^\star\rangle$" to refer to the entire graph below (and including) dangling edge $\langle\kappa^\star,p^\star\rangle$.

Before introducing reduced ESRBDDs and showing they are canonical, we need some terminology. We say that an ESRBDD nonterminal node q:

- *duplicates* node p if $lvl(p) = lvl(q)$, $p[0] = q[0]$, and $p[1] = q[1]$,
- is *redundant* if $q[0] = q[1] = \langle\kappa,p\rangle$, with $\kappa \in \{\mathtt{S}, \mathtt{X}\}$,
- is *high-zero* if $q[0].rule \in \{\mathtt{S}, \mathtt{H_0}\}$, $q[1].rule \in \{\mathtt{S}, \mathtt{X}\}$, and $q[1].node = \mathbf{0}$,
- is *low-zero* if $q[0].rule \in \{\mathtt{S}, \mathtt{X}\}$, $q[0].node = \mathbf{0}$, and $q[1].rule \in \{\mathtt{S}, \mathtt{L_0}\}$.

Note that BDDs [1] can be viewed as ESRBDDs where the edge labels are restricted to $\{\mathtt{S}, \mathtt{X}\}$, and a reduced BDD corresponds to an ESRBDD with no duplicate nodes and no redundant nodes. Similarly, ZDDs [8] can be viewed as ESRBDDs where edge labels are restricted to $\{\mathtt{S}, \mathtt{H_0}\}$, and a reduced ZDD corresponds to an ESRBDD with no duplicate nodes and no high-zero nodes. Also, we note that there is no corresponding definition in the existing literature for the version of ESRBDDs where the edge labels are restricted to $\{\mathtt{S}, \mathtt{L_0}\}$.

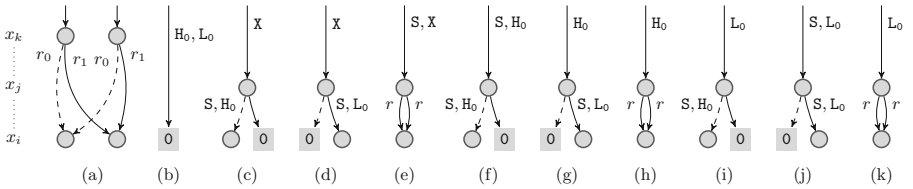

Fig. 1. Patterns not allowed in RESRBDDs

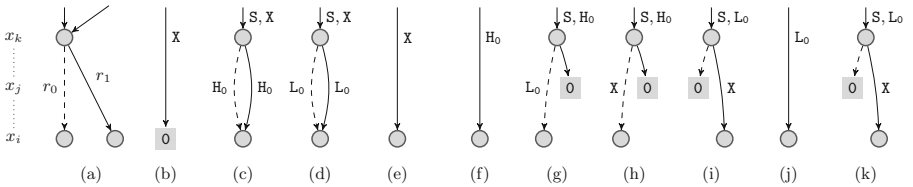

Fig. 2. Replacement rules for patterns in Fig. 1

Definition 2. An ESRBDD is *reduced* if the following restrictions hold:

- **R1**. There are no duplicate nodes.
- **R2**. There are no redundant nodes.
- **R3**. There are no high-zero nodes.
- **R4**. There are no low-zero nodes.
- **R5**. For any edge $e = \langle \kappa, \mathbf{0} \rangle$, $\kappa \in \{\mathtt{S}, \mathtt{X}\}$.

The last restriction disallows edges $\langle \mathtt{H}_0, \mathbf{0} \rangle$ and $\langle \mathtt{L}_0, \mathbf{0} \rangle$ in the reduced ESRBDD. This is because $F^n_{\langle \mathtt{H}_0, \mathbf{0} \rangle} \equiv F^n_{\langle \mathtt{L}_0, \mathbf{0} \rangle} \equiv F^n_{\langle \mathtt{X}, \mathbf{0} \rangle} \equiv \mathbf{0}$, and since we want to enforce canonicity in the reduced ESRBDD, we have *arbitrarily* chosen $\langle \mathtt{X}, \mathbf{0} \rangle$ as the unique representation for such long edges.

3.1 Reducing an ESRBDD

An ESRBDD can be converted into a reduced ESRBDD using Algorithm 1. The algorithm first replaces any edges $\langle \mathtt{H}_0, \mathbf{0} \rangle$ or $\langle \mathtt{L}_0, \mathbf{0} \rangle$ with $\langle \mathtt{X}, \mathbf{0} \rangle$, to satisfy restriction R5. Then, it repeatedly chooses a high-zero, low-zero, redundant, or duplicate node q and eliminates it. If node q duplicates node p, then it redirects all incoming edges from q to p (line 7). Otherwise, q is a high-zero, low-zero, or redundant node, and lines 9–14 find a node d' with $lvl(d') < lvl(q) = n - 1$, and a rule $\kappa' \in \{\mathtt{X}, \mathtt{H}_0, \mathtt{L}_0\}$ such that $F^n_{\langle \mathtt{S}, q \rangle}(x_{1:n}) = F^n_{\langle \kappa', d' \rangle}(x_{1:n})$. Note that a short edge to node q becomes a long edge to node d' because $lvl(d') < lvl(q)$. For the special case of $d' = \mathbf{0}$, *any* edge to q is equivalent to edge $\langle \mathtt{X}, \mathbf{0} \rangle$, so the algorithm replaces those edges (line 16).

When $d' \neq \mathbf{0}$, we have $F^n_{\langle \mathtt{S}, q \rangle}(x_{1:n}) = F^n_{\langle \kappa', d' \rangle}(x_{1:n})$ for $n = lvl(q) + 1$, and these edges are replaced in line 18. It follows that $F^n_{\langle \kappa', q \rangle}(x_{1:n}) = F^n_{\langle \kappa', d' \rangle}(x_{1:n})$ for $n > lvl(q)+1$; these replacements are made in line 19. For rules $\kappa \in \{\mathtt{X}, \mathtt{H}_0, \mathtt{L}_0\}$

Algorithm 1. Reduce an ESRBDD

```
 1: procedure REDUCE(ESRBDD ⟨κ*,p*⟩)
 2:     V ← Nodes(p*);
 3:     ∀κ ∈ {H0, L0}, replace all ⟨κ,0⟩ edges with ⟨X,0⟩;
 4:     while V contains a high-zero, low-zero, redundant, or duplicate node do
 5:         Choose a high-zero, low-zero, redundant, or duplicate node q ∈ V;
 6:         if q duplicates p then
 7:             ∀κ ∈ {S, X, H0, L0}, replace all ⟨κ,q⟩ edges with ⟨κ,p⟩;
 8:         else
 9:             if q is a redundant node then
10:                 κ′ ← X;    d′ ← q[1].node;
11:             else if q is a high-zero node then
12:                 κ′ ← H0;    d′ ← q[0].node;
13:             else if q is a low-zero node then
14:                 κ′ ← L0;    d′ ← q[1].node;
15:             if d′ = 0 then
16:                 ∀κ ∈ {S, X, H0, L0}, replace all ⟨κ,q⟩ edges with ⟨X,0⟩;
17:             else
18:                 Replace all ⟨S,q⟩ edges with ⟨κ′,d′⟩;
19:                 Replace all ⟨κ′,q⟩ edges with ⟨κ′,d′⟩;
20:                 for all rules κ ∈ {X, H0, L0} \ {κ′}, such that an edge ⟨κ,q⟩ exists do
21:                     Create node q′ at level lvl(q) + 1 and add q′ to V;
22:                     if κ = X then
23:                         q′[0] ← ⟨κ′,d′⟩;    q′[1] ← ⟨κ′,d′⟩;
24:                     else if κ = H0 then
25:                         q′[0] ← ⟨κ′,d′⟩;    q′[1] ← ⟨X,0⟩;
26:                     else if κ = L0 then
27:                         q′[0] ← ⟨X,0⟩;    q′[1] ← ⟨κ′,d′⟩;
28:                     Replace all ⟨κ,q⟩ edges with ⟨κ,q′⟩ or ⟨S,q′⟩;
29:         Remove q from V;
```

with $\kappa \neq \kappa'$, we cannot replace $\langle \kappa,q \rangle$ with a single long edge to node d', because the edge needs different reduction rules: the κ rule is needed above level $lvl(q)$, and the κ' rule is needed from level $lvl(q)$ down. So lines 21–27 of the algorithm create a new node q' at level $lvl(q) + 1$, of the appropriate shape such that $F^n_{\langle \kappa,q \rangle}(x_{1:n}) = F^n_{\langle \mathtt{S},q' \rangle}(x_{1:n})$ for $n = lvl(q') + 1$. It then follows that $F^n_{\langle \kappa,q \rangle}(x_{1:n}) = F^n_{\langle \kappa,q' \rangle}(x_{1:n})$ for $n > lvl(q') + 1$. These replacements are made in line 28, where the replacement $\langle \kappa,q' \rangle$ is used for long edges, and $\langle \mathtt{S},q' \rangle$ is used for short edges.

In the above discussion, any edge that is replaced by the algorithm encodes the same function as its replacement, giving us the following lemma.

Lemma 1. In Algorithm 1, each edge replacement preserves the function encoded by the ESRBDD $\langle \kappa^\star,p^\star \rangle$.

It remains to show that the algorithm always terminates.

Lemma 2. Algorithm 1 terminates in $\mathcal{O}(|Nodes(p^\star)|)$ steps.

Proof: The proof is based on the observation that, at every iteration of the algorithm, a node q is chosen to be processed (line 5), at most two nodes are created at level $lvl(q)+1$ (line 21), and node q is removed (line 29). These new nodes (q' on line 21), by construction, satisfy one of the following patterns:

- $q'[0] = q'[1] = \langle \kappa', d' \rangle$, where $d' \neq \mathbf{0}$, and $\kappa' \in \{\mathtt{H}_0, \mathtt{L}_0\}$,
- $q'[0] = \langle \mathtt{X}, \mathbf{0} \rangle$, and $q'[1] = \langle \kappa', d' \rangle$, where $d' \neq \mathbf{0}$, and $\kappa' \in \{\mathtt{X}, \mathtt{H}_0\}$,
- $q'[0] = \langle \kappa', d' \rangle$, and $q'[1] = \langle \mathtt{X}, \mathbf{0} \rangle$, where $d' \neq \mathbf{0}$, and $\kappa' \in \{\mathtt{X}, \mathtt{L}_0\}$.

These nodes are neither redundant, high-zero, nor low-zero, but they could be duplicates. Since the elimination of duplicate nodes (line 7) does not create new nodes, the two nodes created at $lvl(q)+1$ result in at most two additional iterations of the algorithm. Therefore, for every node in the original ESRBDD, the algorithm iterates at most three times. □

Theorem 1. Algorithm 1 converts ESRBDD $\langle \kappa^\star, p^\star \rangle$ to an equivalent reduced ESRBDD in $\mathcal{O}(|Nodes(p^\star)|)$ steps.

Proof: Lemma 2 establishes that Algorithm 1 terminates in $\mathcal{O}(|Nodes(p^\star)|)$ steps. Based on the condition of the while loop, when the loop terminates, we know that the ESRBDD contains no high-zero, low-zero, redundant, or duplicate nodes. From line 3 and the fact that the algorithm never adds an edge of the form $\langle \mathtt{H}_0, \mathbf{0} \rangle$ or $\langle \mathtt{L}_0, \mathbf{0} \rangle$, we conclude that when Algorithm 1 terminates, any edge to terminal node $\mathbf{0}$ must have edge rule $\mathtt{S}$ or $\mathtt{X}$. Therefore, when the Algorithm terminates, the ESRBDD is reduced. Lemma 1 establishes that Algorithm 1 produces an equivalent (in terms of encoded function) ESRBDD. □

While we have established that Algorithm 1 always terminates and produces a reduced ESRBDD, we have not yet established that the Algorithm produces the *same* reduced ESRBDD, regardless of the order in which nodes are chosen in line 5. This is guaranteed by the canonicity property, discussed next. Additionally, we note here that, unlike most other decision diagrams (including BDDs, ZDDs, CBDDs, CZDDs, and TDDs), a reduced ESRBDD is not necessarily a minimum size ESRBDD encoding of a function, even for a fixed variable order, as elimination of some node q during the reduction could trigger the creation of two new nodes. An example of this is shown in Fig. 3, where redundant node q is eliminated. Edges $\langle \mathtt{S}, q \rangle$ and $\langle \mathtt{X}, q \rangle$ can be simply redirected as $\langle \mathtt{X}, p \rangle$, but the $\langle \mathtt{H}_0, q \rangle$ and $\langle \mathtt{L}_0, q \rangle$ edges require the creation of two new nodes $q_{\mathtt{H}_0}$ and $q_{\mathtt{L}_0}$.

While the "chaotic" non-deterministic reduction procedure in Algorithm 1 is handy in proving termination under the most general conditions, in practice we utilize a deterministic depth-first version of this algorithm that reduces a node only after having reduced its children.

3.2 Canonicity of Reduced ESRBDDs

We are now ready to discuss the *canonicity* of reduced ESRBDDs, i.e., to show that a function has a unique encoding as a reduced ESRBDD. In the following, we say that functions $F^n_{\langle \kappa, p \rangle}$ and $F^n_{\langle \kappa', p' \rangle}$ are *equivalent*, written $F^n_{\langle \kappa, p \rangle} \equiv F^n_{\langle \kappa', p' \rangle}$, if $F^n_{\langle \kappa, p \rangle}(x_{1:n}) = F^n_{\langle \kappa', p' \rangle}(x_{1:n})$ for all possible inputs $(x_{1:n}) \in \mathbb{B}^n$.

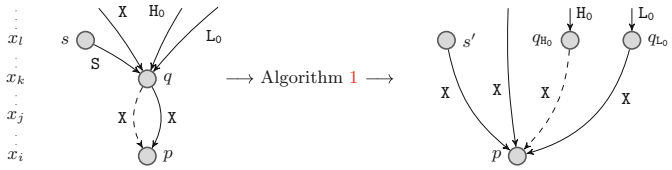

Fig. 3. A worst-case example where elimination of node q creates two nodes.

Theorem 2. In a reduced ESRBDD, for any $n \in \mathbb{N}$, for any two edges $e = \langle \kappa, p \rangle$, $e' = \langle \kappa', p' \rangle$ with $lvl(p) \leq n$, $lvl(p') \leq n$, if $F_e^n \equiv F_{e'}^n$ then (1) $p = p'$, and (2) if $lvl(p) < n$ then $\kappa = \kappa'$.

Proof: The proof is by induction on n. For the base case, we use $n = 0$ and from the definition of F we have $F_e^0 \equiv F_{e'}^0 \;\rightarrow\; p = p'$.

Now, suppose the theorem holds for $n = m$, where $m \geq 0$, we will prove it holds for $n = m + 1$. Regardless of $\langle \kappa, p \rangle$, we have

$$F^n_{\langle \kappa, p \rangle}(x_{1:n}) = (x_n)?f_1(x_{1:n-1}):f_0(x_{1:n-1})$$

for some functions f_0 and f_1. Similarly, we have

$$F^n_{\langle \kappa', p' \rangle}(x_{1:n}) = (x_n)?f'_1(x_{1:n-1}):f'_0(x_{1:n-1}).$$

It follows that $F^n_{\langle \kappa, p \rangle} \equiv F^n_{\langle \kappa', p' \rangle}$ if and only if $f_0 \equiv f'_0$ and $f_1 \equiv f'_1$.
First, suppose $lvl(p) = n$ and $lvl(p') = n$. From the definition of F, it follows that $F^{n-1}_{p[0]} \equiv F^{n-1}_{p'[0]}$ and $F^{n-1}_{p[1]} \equiv F^{n-1}_{p'[1]}$. By inductive hypothesis, $p[0].node = p'[0].node$ and $p[1].node = p'[1].node$. If $lvl(p[0].node) < n-1$, then by inductive hypothesis, $p[0] = p'[0]$; otherwise, $lvl(p[0].node) = n - 1$ and we must have $p[0].rule = \mathtt{S}$ and $p'[0].rule = \mathtt{S}$, thus $p[0] = p'[0]$. By a similar argument, it follows that $p[1] = p'[1]$. We therefore have either that $p = p'$ and the theorem holds, or that p duplicates p', which is impossible because of restriction R1.
Next, suppose $lvl(p) < n$ and $lvl(p') < n$. If $\kappa = \kappa'$, then in all cases for F we conclude that $F^{n-1}_{\langle \kappa, p \rangle} \equiv F^{n-1}_{\langle \kappa', p' \rangle}$ and by inductive hypothesis we have that $p = p'$, so the theorem holds. We now show that $\kappa \neq \kappa'$ is impossible, by contradiction. Consider the possible cases for $\kappa \neq \kappa'$:

1. $\kappa = \mathtt{X}$: If $\kappa' = \mathtt{L_0}$ or $\kappa' = \mathtt{H_0}$, from the definition of F we conclude that $F^{n-1}_{\langle \kappa, p \rangle} \equiv F^{n-1}_{\langle \kappa', p' \rangle}$ and that $F^{n-1}_{\langle \kappa, p \rangle} \equiv \mathbf{0}$.
2. $\kappa = \mathtt{L_0}$: If $\kappa' = \mathtt{H_0}$, from the definition of F we conclude that $F^{n-1}_{\langle \kappa, p \rangle} \equiv \mathbf{0}$ and $F^{n-1}_{\langle \kappa', p' \rangle} \equiv \mathbf{0}$.
3. The remaining cases are symmetric.

In all cases, we conclude that $F^{n-1}_{\langle \kappa, p \rangle} \equiv \mathbf{0}$ and $F^{n-1}_{\langle \kappa', p' \rangle} \equiv \mathbf{0}$. By the inductive hypothesis, we have that $p = \mathbf{0}$ and $p' = \mathbf{0}$. According to R5, if $p = \mathbf{0}$ then κ cannot be $\mathtt{L_0}$ or $\mathtt{H_0}$. But this implies $\kappa = \mathtt{X}$ and $\kappa' = \mathtt{X}$, contradicting our assumption that $\kappa \neq \kappa'$.

Finally, suppose $lvl(p) = n$ and $lvl(p') < n$ (the case $lvl(p) < n$ and $lvl(p') = n$ is symmetric). We show that this is impossible, by contradiction. Consider the possible cases for κ':

1. $\kappa' = \texttt{X}$: From the definition of F, we must have $F^{n-1}_{p[0]} \equiv F^{n-1}_{\langle \kappa', p' \rangle}$ and $F^{n-1}_{p[1]} \equiv F^{n-1}_{\langle \kappa', p' \rangle}$. By the inductive hypothesis, we conclude that $p[0].node = p'$ and $p[1].node = p'$. If $lvl(p') = n-1$, then we have $p[0] = p[1] = \langle \texttt{S}, p' \rangle$; otherwise, we have $lvl(p') < n-1$ and by inductive hypothesis, $p[0] = p[1] = \langle \kappa', p' \rangle = \langle \texttt{X}, p' \rangle$. Either way, node p is redundant, and from R2 we have a contradiction.
2. $\kappa' = \texttt{H}_0$: From the definition of F, we must have $F^{n-1}_{p[0]} \equiv F^{n-1}_{\langle \kappa', p' \rangle}$ and $F^{n-1}_{p[1]} \equiv \mathbf{0}$. By the inductive hypothesis, we conclude that $p[0].node = p'$ and $p[1].node = \mathbf{0}$. If $lvl(p') = n-1$, then we have $p[0] = \langle \texttt{S}, p' \rangle$; otherwise, we have $lvl(p') < n-1$ and by inductive hypothesis, $p[0] = \langle \kappa', p' \rangle = \langle \texttt{H}_0, p' \rangle$. Either way, node p is high-zero, and from R3 we have a contradiction.
3. $\kappa' = \texttt{L}_0$: From the definition of F, we must have $F^{n-1}_{p[0]} \equiv \mathbf{0}$ and $F^{n-1}_{p[1]} \equiv F^{n-1}_{\langle \kappa', p' \rangle}$. By the inductive hypothesis, we conclude that $p[0].node = \mathbf{0}$ and $p[1].node = p'$. If $lvl(p') = n-1$, then we have $p[1] = \langle \texttt{S}, p' \rangle$; otherwise, we have $lvl(p') < n-1$ and by inductive hypothesis, $p[1] = \langle \kappa', p' \rangle = \langle \texttt{L}_0, p' \rangle$. Either way, node p is low-zero, and from R4 we have a contradiction. □

The canonicity result establishes that, regardless of how a ESRBDD is constructed for a given function, the resulting reduced ESRBDD is guaranteed to be unique (assuming a given variable order). Thus, we can determine in constant time whether two functions encoded as reduced ESRBDDs are equivalent (as is already the case for reduced ordered BDDs and ZDDs). From now on, unless otherwise specified, we assume that all ESRBDDs are reduced.

3.3 Comparing ESRBDDs to Other Types of Decision Diagrams

For the remainder of the paper, we consider the relative size of the different types of DD based on the interpretation of long edges, namely, BDDs, ZDDs, CBDDs, CZDDs, TBDDs, and ESRBDDs. We also consider ESRBDDs without the $\texttt{L}_0$ edge label, denoted ESRBDD$-\texttt{L}_0$. These are summarized in Table 1, some entries (comparisons between BDDs, ZDDs, CBDDs, and CZDDs) are known from prior work [2,6], some entries are discussed below, and some entries are unknown. Entry $[T_1, T_2]$ describes the worst-case increase in the number of nodes, as a multiplicative factor, More formally, it is the bound for "number of nodes required to encode f using T_2" divided by "number of nodes required to encode f using T_1" for all functions f over L boolean variables. Note that the node counts always include both terminal nodes. A factor of 1 indicates that type T_1 cannot require fewer nodes than type T_2.

First, we discuss how an arbitrary BDD can be converted into a TBDD or ESRBDD, and fill in the BDD row in Table 1. To build a TBDD from a BDD, every edge to a non-terminal node p in the BDD is annotated with the level tag $lvl(p)$. By definition, any such annotated edge in a TBDD implies BDD

Table 1. Worst-case relative increase when converting one DD type into another.

	BDD	ZDD	CBDD	CZDD	TBDD	ESR−$\mathtt{L_0}$	ESR
BDD →	—	$L/2$ [6]	1 [2]	2 [2]	1	1	1
ZDD →	$L/2$ [6]	—	3 [2]	1 [2]	1	1	1
CBDD →	?	?	—	2 [2]	?	2	2
CZDD →	?	?	3 [2]	—	?	2	2
TBDD →	?	?	?	?	—	3	3
ESRBDD−$\mathtt{L_0}$ →	$L/2$	$L/2$	3	2	1	—	3/2
ESRBDD →	$2L/3$	$2L/3$	$L/2$	$L/2$	$L/2$	$L/2$	—

reductions for the skipped levels. A TBDD thus constructed is no larger than the BDD, and may be further reduced (since it could contain high-zero nodes) by applying the TBDD reduction described in [10]. Similarly, we can annotate long edges in the BDD with $\mathtt{X}$ (Fig. 4(a)), and short edges with $\mathtt{S}$, to obtain an unreduced ESRBDD. We then apply Algorithm 1. We now show that this will not increase the ESRBDD size, and thus the resulting ESRBDD cannot be larger than the original BDD.

Lemma 3. Suppose we have an unreduced ESRBDD where, for every node q, there exists a rule $\kappa \in \{\mathtt{X}, \mathtt{H_0}, \mathtt{L_0}\}$ such that every edge to q is either $\langle \mathtt{S}, q \rangle$ or $\langle \kappa, q \rangle$. Then reducing the ESRBDD will not increase the number of nodes.

Proof: Apply Algorithm 1 and in line 5, always choose a node at the lowest level. Then, when a node q is chosen, all incoming edges to q will be labeled either with $\mathtt{S}$ or with κ. The $\langle \mathtt{S}, q \rangle$ edges will not cause any node to be created. The $\langle \kappa, q \rangle$ edges will cause at most one node to be created. But then node q is removed. Thus, the overall number of nodes cannot increase. □

It is also easy to convert a ZDD into a TBDD or ESRBDD. To obtain a TBDD, annotate every edge from non-terminal node p with the level tag $lvl(p)$, so that ZDD reductions are used for all the edges; then reduce the TBDD. To obtain an ESRBDD, annotate long edges in the ZDD with $\mathtt{H_0}$, see Fig. 4(b), and short edges with $\mathtt{S}$, and apply Algorithm 1.

The conversion from a chained DD to an unreduced ESRBDD is illustrated in Fig. 4(c) and (d). For each chain node $x_k : x_i$ with $x_k > x_i$, create a "top node" with variable x_k, and a "bottom node" with variable x_i, that is only pointed to by its corresponding top node. In a CBDD, the top node will be a high-zero node, and all top nodes and non-chained nodes will have incoming edges labeled with $\mathtt{X}$ or $\mathtt{S}$. In a CZDD, the top node will be a redundant node, and all top nodes and non-chained nodes will have incoming edges labeled with $\mathtt{H_0}$ or $\mathtt{S}$. At worst, the unreduced ESRBDD has twice the nodes of the original CBDD or CZDD and, from Lemma 3, reducing this ESRBDD does not increase its size.

In a TBDD, each edge can be characterized as short, purely $\mathtt{X}$, purely $\mathtt{H_0}$, or partly $\mathtt{X}$ and partly $\mathtt{H_0}$. To convert into an ESRBDD, the short edges are labeled with $\mathtt{S}$, the purely $\mathtt{X}$ edges are labeled with $\mathtt{X}$, the purely $\mathtt{H_0}$ edges are labeled

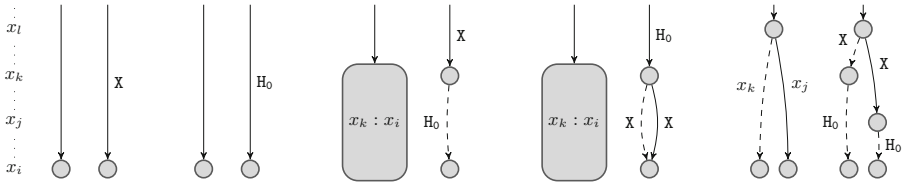

Fig. 4. Converting to ESRBDDs.

with $\mathtt{H_0}$. Edges that are partly $\mathtt{X}$ and partly $\mathtt{H_0}$ require the addition of a node at the level where the reduction rule changes, as shown in Fig. 4(e). The worst case occurs when *every* edge requires such a node. Then, since every TBDD node has two outgoing edges, the resulting unreduced ESRBDD will have triple the number of nodes. Since all of the introduced nodes have incoming $\mathtt{X}$ edges, and all other nodes have incoming $\mathtt{S}$ or $\mathtt{H_0}$ edges, from Lemma 3 this ESRBDD will not increase in size when it is reduced. We note here that, if there are some purely $\mathtt{X}$ edges in the TBDD, then Lemma 3 no longer applies; however, the number of nodes that would be added during reduction is no more than the number of nodes saved by not having to introduce a node on the purely $\mathtt{X}$ edges.

We now consider converting from ESRBDDs into the other DD types. In the case where $\mathtt{L_0}$ edges are not allowed (row ESRBDD$-\mathtt{L_0}$ in Table 1), the worst case BDD is from ESRBDD $\langle \mathtt{H_0}, \mathbf{1} \rangle$ and the worst case ZDD is from ESRBDD $\langle \mathtt{X}, \mathbf{1} \rangle$. In both cases, the ESRBDD has 2 nodes, while the resulting BDD/ZDD has $L + 2$ nodes, giving ratios of $L/2 + o(L)$, similar to the discussion in [6, p. 250]. The example ZDD in [2], which produces a CBDD with three times as many nodes, can be converted into an ESRBDD of the same size. Similarly, the example BDD in [2], which produces a CZDD with twice as many nodes, can be converted into an ESRBDD of the same size. Any ESRBDD without $\mathtt{L_0}$ edges can be converted into a TBDD by labeling $\mathtt{X}$ edges with a level tag such that the $\mathtt{X}$ rule is always applied, and labelling $\mathtt{H_0}$ edges with a level tag such that the $\mathtt{H_0}$ rule is always applied. Therefore, the TBDD cannot be larger than the ESRBDD. An ESRBDD$-\mathtt{L_0}$ can be converted into an ESRBDD by running Algorithm 1 to eliminate any low-zero nodes. For each low-zero node that is eliminated, we could have an incoming $\mathtt{X}$ and $\mathtt{H_0}$ edge, causing the creation of two nodes. Suppose we eliminate n low-zero nodes that cause creation of two nodes. Then, because each low-zero node must have 2 incoming edges, we must have $2n$ incoming edges to these nodes. Above, we must have at least $2n - 1$ nodes to produce these edges. We could then "stack" such a pattern m times. This gives an ESRBDD with $m(n + 2n - 1) + 2 = m(3n - 1) + 2$ nodes, and a reduced ESRBDD with $m(2n + 2n - 1) + 2 = m(4n - 1) + 2$ nodes. The upper bound of this ratio is 3/2, which occurs when $n = 1$ and m goes to infinity.

For the case of ESRBDDs with all types of edges (row ESRBDD in Table 1), the $\mathtt{L_0}$ edge allows us to build different worst cases. Consider an ESRBDD $\langle \mathtt{S}, p \rangle$ where $lvl(p) = L$, $p[0] = \langle \mathtt{H_0}, \mathbf{1} \rangle$, and $p[1] = \langle \mathtt{L_0}, \mathbf{1} \rangle$. This ESRBDD has 3 nodes.

Table 2. Numbers of nodes for dictionary benchmarks.

Word List		QBDD	BDD	CBDD	ZDD	CZDD	TBDD	ESR
Binary	Compact	1,120,437	1,120,250	971,387	657,969	657,969	657,902	**484,765**
	Full	1,285,501	1,285,285	1,153,438	851,555	851,554	851,479	**520,576**
One-hot	Compact	9,739,638	9,739,638	656,649	**311,227**	**311,227**	**311,227**	**311,227**
	Full	22,775,492	22,775,492	656,712	**311,227**	**311,227**	**311,227**	**311,227**

Password List		QBDD	BDD	CBDD	ZDD	CZDD	TBDD	ESR
Binary	Compact	5,705,516	5,704,777	4,542,925	2,960,478	2,960,465	2,960,209	**2,399,272**
	Full	5,649,626	5,648,670	4,960,446	3,532,847	3,532,816	3,532,467	**2,410,589**
One-hot	Compact	72,858,088	72,858,088	3,055,784	**1,486,430**	**1,486,430**	**1,486,430**	**1,486,430**
	Full	101,737,047	101,737,047	3,056,067	**1,486,430**	**1,486,430**	**1,486,430**	**1,486,430**

Because BDDs cannot exploit $\mathtt{H}_0$ or $\mathtt{L}_0$ edges, this will produce a BDD with $2(L-1)+3 = 2L+1$ nodes, giving a worst-case ratio of $2L/3$. The ZDD worst-case is similar, using instead $p[0] = \langle \mathtt{X}, \mathbf{1} \rangle$. Finally, for DD types that can exploit both $\mathtt{X}$ and $\mathtt{H}_0$ edges, the ESRBDD $\langle \mathtt{L}_0, \mathbf{1} \rangle$ corresponds to the worst case: the CBDD, CZDD, TBDD, and ESRBDD$-\mathtt{L}_0$ will all require $L+2$ nodes.

4 Experimental Results

We compare the performance of QBDDs (with long edges to **0**), BDDs, ZDDs, CBDDs, CZDDs, TBDDs, and ESRBDDs on three sets of benchmarks. The first two benchmarks are similar to those used in [2], and are representative of general textual information and digital logic functions, respectively. The third benchmark is typical in state space analysis of concurrent systems.

4.1 Dictionaries

A dictionary can be encoded as an indicator function over the set of strings of a given length from either the `compact` alphabet consisting of the distinct symbols found in the dictionary plus `NULL`, or the `full` alphabet of all 128 ASCII characters (to ensure that all encoded strings have the same length, shorter ones are padded with the ASCII symbol `NULL`). We use the encoding schemes described in [2]: *one-hot* and *binary*. Therefore, each dictionary generates four benchmarks, one for each choice of encoding and alphabet.

We compare the different DD types on two dictionaries. The first one is the English words in file `/usr/share/dict/words` under MacOS, containing 235,886 words with lengths ranging from 1 to 24. Its compact alphabet contains lower and upper case letters plus hyphen and `NULL` (54 in total). The second one is a set of passwords from SecLists [7] (non-ASCII characters are replaced with `NULL`), containing 999,999 passwords with lengths ranging from 1 to 39. Its compact alphabet consists of 91 symbols including `NULL`.

Table 3. Numbers of nodes for combinational circuit benchmarks.

Circuit	QBDD	BDD	CBDD	ZDD	CZDD	TBDD	ESR
C432	2,675	1,506	1,506	2,658	2,189	**1,494**	1,498
C499	29,483	28,769	28,769	29,316	28,749	28,610	**28,428**
C880	15,048	6,496	6,496	15,044	9,640	6,496	**6,491**
C1355	85,694	75,498	75,498	85,439	77,976	75,243	**74,757**
C1908	18,456	16,210	16,174	17,859	16,047	15,687	**15,685**
C2670	74,940	15,662	15,658	74,468	21,012	**15,539**	15,601
C3540	152,523	51,878	51,778	150,539	64,563	**50,871**	51,146
C5315	26,011	3,793	3,784	25,785	4,749	**3,716**	3,742

Table 2 reports the number of nodes required to store each dictionary, according to different encodings and alphabets (the best result on each row is in boldface). Except for QBDDs and BDDs, the one-hot encoding results in fewer nodes, demonstrating the effectiveness of the zero-suppressed idea when encoding large, sparse data. Among the DD types we consider, ESRBDDs have the fewest nodes, regardless of encoding and alphabet. For binary encodings, ESRBDDs use 19%–39% fewer nodes than TBDDs, the second best choice. With one-hot encodings, ZDDs, CZDDs, TBDDs, and ESRBDDs tie for best because (a) there are no redundant nodes and (b) any low-zero nodes that are eliminated do not cause an overall decrease in the number nodes in the ESRBDDs. Indeed, redundant nodes are rare even with binary encodings, as they arise when two words w_1 and w_2 not only have bit patterns that differ in a position, but they also share all their possible continuations, i.e., $w_1 w'$ is a word if and only if $w_2 w'$ is also a word, for all w'. In the English word list, "Hlidhskjalf" and its alternate spelling "Hlithskjalf" is one such rare instance (note that no w' can continue either of them to form an additional word).

4.2 Combinational Circuits

BDDs are commonly used to synthesize and verify digital circuits. We select a set of combinational circuits from the LGSynth'91 benchmarks [11] and, for each circuit, we build a DD encoding all its output logic functions. For each circuit, the variable order is determined using Sifting [9] while building the BDD.

Table 3 reports the number of nodes needed to encode all outputs of each circuit. In contrast to the dictionaries, these benchmarks place importance on the ability to eliminate redundant nodes. Thus, QBDDs and ZDDs have the worst performance. TBDDs and ESRBDDs are always the two best representations, and the difference between them is less than 0.7%.

4.3 Safe Petri Nets

Decision diagrams are frequently used in symbolic model checking to represent sets of states. We have selected a set of 37 *safe* Petri nets from the 2018 Model Checking Contest https://mcc.lip6.fr/2018/. A Petri net is safe if each one of its places can contain at most one token—each place can, therefore, be mapped

Table 4. Final scores for the safe Petri net benchmarks.

QBDD	BDD	CBDD	ZDD	CZDD	TBDD	ESR
3.108	2.971	2.038	1.215	1.167	1.160	**1.001**

Table 5. Number of nodes for a subset of the safe Petri net benchmarks.

Model	QBDD	BDD	CBDD	ZDD	CZDD	TBDD	ESR
DiscoveryGPU-PT-14a	80,865	75,682	75,571	43,016	43,016	**39,689**	40,953
BusinessProcesses-PT-04	282,787	282,787	130,825	67,228	67,228	67,228	**66,983**
Referendum-PT-0020	343,676	343,676	339,552	194,607	194,607	194,607	**184,789**
NeoElection-PT-3	414,962	414,962	34,860	15,519	15,519	15,519	**15,507**
SimpleLoadBal-PT-10	503,777	503,777	376,896	191,460	191,460	191,460	**182,403**
LamportFastMutEx-PT-4	507,897	507,897	252,361	122,487	122,487	122,487	**119,111**
AutoFlight-PT-06a	520,755	520,755	356,729	207,409	207,409	207,409	**178,855**
RwMutex-PT-r0020w0010	553,073	553,073	502,831	358,580	358,580	358,580	**195,377**
DES-PT-01b	709,303	709,303	442,610	246,647	246,647	246,647	**217,325**
Dekker-PT-015	1,191,942	1,191,942	844,466	504,726	504,726	504,726	**403,801**
Railroad-PT-010	2,109,610	2,109,610	1,096,122	554,541	554,541	554,541	**516,121**
NQueens-PT-08	3,698,534	3,698,534	2,295,689	1,443,628	1,443,628	1,443,628	**1,069,242**
ResAllocation-PT-R020C002	5,532,167	5,532,167	4,554,792	2,826,856	2,826,856	2,826,856	**2,167,111**

directly to a boolean variable. Most of these models have scaling parameters that affect their size and complexity, yielding $N = 103$ model instances.

Providing detailed results for all the model instances would require excessive space, so to summarize over all model instances, Table 4 shows a score for each DD type i. The score is the geometric mean [4]:

$$score(i) = \sqrt[N]{\prod_{n=1}^{N} \frac{T_i(n)}{T_{min}(n)}}$$

where N is the total number of model instances, $T_i(n)$ is the number of nodes needed to represent the state space of instance n using DD type i, and $T_{min}(n)$ is the smallest number of nodes needed to represent the state space of instance n by any of the DD types we consider. ESRBDDs have by far the smallest overall score, barely larger than 1, indicating that they are either the smallest or slightly larger than the smallest for each model instance.

Table 5 shows $T_i(n)$ for model instances n that required more than 250,000 nodes in the QBDD representation. For parameterized models that had multiple model instances satisfying this criterion, we present data for only the largest such model instance. We have also included the results for *DiscoveryGPU*—the only model where ESRBDDs were not the best (they were a close second).

4.4 Memory Considerations: The Size of Nodes

So far, we have compared DD types based on how many nodes they require. However, the actual memory consumption also depends on the size of the respective nodes. All of these DDs store two child pointers. In addition, BDDs and ZDDs

Table 6. Overhead of node sizes (bits per node) as compared to QBDD nodes.

Level bits	BDD	ZDD	CBDD	CZDD	TBDD	ESR−$\mathtt{L_0}$	ESR
16	+16	+16	+32	+32	+48	+18	+20
20	+20	+20	+40	+40	+60	+22	+24
32	+32	+32	+64	+64	+96	+34	+36

store a level, CBDDs and CZDDs store two levels, TBDDs store three levels, while ESRBDDs store a level and two edge rules. Since all short edges must be labeled by S, it is only necessary to label the long edges, and this requires $\log_2 n$ bits per edge if there are n non-S reduction rules. Without $\mathtt{L_0}$ edges, a single bit distinguishes $\mathtt{H_0}$ from X; otherwise, two bits are required for rules $\{\mathtt{H_0}, \mathtt{L_0}, \mathtt{X}\}$. QBDD nodes are therefore the smallest (typically requiring 64 or 128 bits, when 32–bit or 64–bit pointers are used, respectively) and Table 6 indicates the *additional* cost required for each node type, when the level integers are stored using 16 bits (as suggested by [2]), 20 bits (as suggested by [10]), and 32 bits.

ESRBDDs are clearly more memory efficient than CBDDs, CZDDs and TBDDs. There are a few instances in our experiments where TBDDs use marginally fewer nodes than ESRBDDs (less than 3.2% fewer nodes in every such instance), but not enough to overcome their per-node memory overhead.

5 Conclusions

We have shown that ESRBDDs are a simple, yet efficient, generalization of previous attempts at combining reduction rules. Unlike previous efforts, they are not biased towards any particular reduction rule and therefore eliminate the need for the user to prioritize the reduction rules. They also provide a framework for further generalizations through additional reduction rules—for example, "high-one" and "low-one", the duals of "low-zero" and "high-zero" respectively.

ESRBDDs allow users to select a subset of reduction rules that suit their needs, and make it possible to integrate domain-specific reduction rules (a common phenomenon) with a subset of existing ones. ESRBDD nodes are also more compact than all previous such efforts, and new reduction rules can be added at a small cost—$\log_2 n$ bits per edge, where n is the number of reduction rules. Our future efforts will be directed towards adapting BDD manipulation operations (such as *Apply*) to work with the reduction rules in ESRBDDs, and towards including complement edges and other reduction rules, such as "high-one", "low-one", or "identity" reductions, while maintaining canonicity.

Acknowledgments. This work was supported in part by National Science Foundation grant ACI-1642397.

References

1. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. Comp. **35**(8), 677–691 (1986)
2. Bryant, R.E.: Chain reduction for binary and zero-suppressed decision diagrams. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10805, pp. 81–98. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89960-2_5
3. Drechsler, R., Becker, B.: Ordered Kronecker functional decision diagrams – a data structure for representation and manipulation of Boolean functions. Trans. Comput. Aided Des. Integr. Circ. Syst. **17**(10), 965–973 (2006)
4. Fleming, P.J., Wallace, J.J.: How not to lie with statistics: the correct way to summarize benchmark results. Commun. ACM **29**(3), 218–221 (1986)
5. Kimura, S., Clarke, E.M.: A parallel algorithm for constructing binary decision diagrams. In: Proceedings of International Conference on Computer Design (ICCD), pp. 220–223. IEEE Computer Society Press, September 1990
6. Knuth, D.E.: The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part I. Addison-Wesley, Boston (2011)
7. Miessler, D., Haddix, J.: SecLists. https://github.com/danielmiessler/SecLists
8. Minato, S.-I.: Zero-suppressed BDDs and their applications. Softw. Tools Technol. Transf. **3**, 156–170 (2001)
9. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: International Conference on CAD, pp. 139–144, November 1993
10. van Dijk, T., Wille, R., Meolic, R.: Tagged BDDs: combining reduction rules from different decision diagram types. In: Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design, FMCAD 2017, Austin, TX, pp. 108–115. FMCAD Inc. (2017)
11. Yang, S.: Logic synthesis and optimization benchmarks user guide: version 3.0. Microelectronics Center of North Carolina (MCNC) (1991)