Noname manuscript No.
(will be inserted by the editor)

CESRBDDs: binary decision diagrams with complemented edges and edge-specified reductions

Junaid Babar · Gianfranco Ciardo · Andrew Miner

Received: date / Accepted: date

Abstract We introduce CESRBDDs, a form of binary decision diagrams (BDDs) that can exploit reduction opportunities beyond those allowed by reduced ordered BDDs (elimination of redundant nodes), zero-suppressed BDDs (elimination of "high-zero" nodes), and recent proposals merging the two (chained or tagged BDDs). CESRBDDs also incorporate complemented edges, thus never store both the encoding of a function and of its complement. We prove that CESRBDDs are canonical and show how their storage requirements and computational efficiency compare very favorably against previous alternatives, both theoretically and experimentally, using an extensive set of benchmarks. Another advantage of CESRBDDs over chained or tagged BDDs is that their nodes only require one byte to store reduction and complement information.

Keywords binary decision diagrams · canonical forms · complemented edges

1 Introduction

Boolean decision diagrams (BDDs) are widely employed for a variety of applications, due to their ability to compactly encode many boolean functions of L input boolean variables, their appealing graphical representation as an $(L+1)$-level directed acyclic-graph, and the efficiency of their manipulation operations, whose complexity depends on the size of the argument BDDs, not on the size 2^L of the function's domain. Furthermore, most BDD types are canonical: for a fixed ordering of the input variables, each function has a unique BDD encoding, allowing for a fast $O(1)$ equality test.

Compactness and canonicity are achieved through careful rules for eliminating nodes. All canonical BDDs 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 levels (i.e., bypass input variables), and how to eliminate nodes and replace them with long edges. The two most popular forms of decision diagrams, reduced ordered BDDs [5] (usually abbreviated as ROBDDs or simply BDDs in the literature, but we will call them FBDDs to avoid confusion) and zero-suppressed BDDs [18] (usually abbreviated as ZDDs, but we will call them ZBDDs), use different reduction rules. Some applications are more suitable for FBDDs while others are more suitable for ZBDDs, 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 yet, there are applications where *both* rules would be useful.

Recently, "tagged BDDs" (TBDDs) [8] and "chain reduced BDDs" (CBDDs) or "chain reduced ZDDs" (CZDDs) [6] were introduced to combine the reduction rules, thus the advantages, of FBDDs and ZBDDs. At TACAS 2019, we presented *edge-specified reduction BDDs* (ESRBDDs) [4], where a two-bit code specifies the reduction used along each edge: fully-reduced, zero-

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

J. Babar
Trusted Systems Group, Collins Aerospace, USA
E-mail: junaid.babar@collins.com

G. Ciardo
Computer Science Department, Iowa State University, USA
E-mail: ciardo@iastate.edu

A. Miner
Computer Science Department, Iowa State University, USA
E-mail: asminer@iastate.edu

suppressed, or "one-suppressed" (alternative to zero-suppressed, naturally suggested by symmetry considerations). The present work greatly extends this idea to further symmetric reductions that can be easily defined and interpreted along long edges, adds the well-known idea of complemented edges [1], and proves that the resulting definition of CESRBDDs is still nevertheless canonical. We believe CESRBDDs are not only conceptually simpler than previous attempts to integrate multiple reductions, but they also provide the foundations for further extensions, as additional reduction rules could be possibly integrated with small overhead, since r bits are enough to specify up to 2^r possible reduction rules. Finally, unlike TBDDs, CBDDs, and CZDDs, CESRBDDs treat all reduction rules equally: they do not prioritize one rule over another.

The paper is organized as follows. Section 2 recalls definitions of various classes of BDDs and attempts to integrate FBDDs and ZBDDs. Section 3 formally defines unreduced and reduced CESRBDDs, and describes their reduction algorithm. Section 4 proves that (reduced) CESRBDDs are canonical. Section 5 describes the Apply operation for CESRBDDs and studies its complexity. Sections 6 and 7 offer a theoretical and a experimental comparison of CESRBDDs with previous forms of BDDs. Section 8 contains our conclusions and plans for future work in this direction.

2 Classes of BDDs and their properties

This section surveys the variants of BDDs relevant to our work. Since we consider ordered BDDs only, we first define what it means for a BDD to be ordered.

Definition 1 (ordered BDD). An L-variable *ordered BDD* is an edge-labeled directed acyclic graph where the terminal nodes are **0** and **1**, at level 0, while each non-terminal node p belongs to a level $p.lvl = k \in \{1, ..., L\}$ (corresponding to the domain of boolean variable x_k) and has an edge to a 0-child, $p[0]$, and an edge to a 1-child, $p[1]$, satisfying $k > p[0].lvl$ and $k > p[1].lvl$ (this is the *ordered property*). The 0-child is also known as the "low" or "L" child, and the 1-child as the "high" or "H" child; graphically, the corresponding edges are shown using a dashed or solid line, respectively. □

BDDs encode boolean functions of boolean variables, but we observe that, without giving a meaning to *long edges* (from a node p at level k to a node q at level $h < k - 1$, skipping levels $k - 1, ..., h + 1$), the function encoded by a BDD node is not well defined. Each of the following classes of (ordered) BDDs, thus, restricts or gives meaning to these long edges, uniquely defining the function encoded by a node or edge.

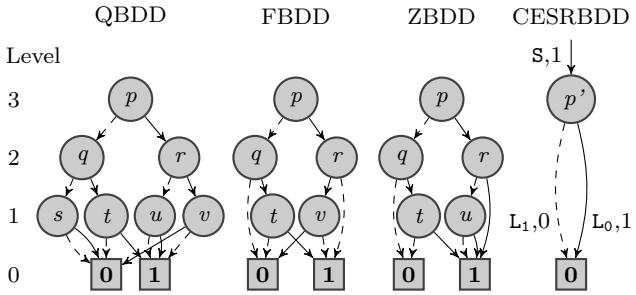


Fig. 1 BDDs encoding $(\neg x_3 \wedge x_2 \wedge x_1) \vee (x_3 \wedge \neg(x_2 \wedge x_1))$.

In particular, Fig. 1 shows the graphical representation of the same function using the BDD classes defined in the next three sections: QBDDs, FBDDs, and ZBDDs, as well as the equivalent CESRBDD, which we introduce in Sect. 3. In all cases, the dashed edge leaving a node at level k corresponds to the case $x_k = 0$, the solid edge to the case $x_k = 1$. Also, the functions encoded by the various nodes are as follows:

At level 1: $F_s = 0, \quad F_t = x_1, \quad F_u = 1, \quad F_v = \neg x_1.$
At level 2: $F_q = x_1 \wedge x_2, \quad F_r = \neg(x_1 \wedge x_2).$
At level 3: $F_p = (\neg x_3 \wedge x_2 \wedge x_1) \vee (x_3 \wedge \neg(x_2 \wedge x_1)).$

In the literature, a BDD is defined as encoding either a single function (in which case it has a single *root* and all its nodes are reachable from this root) or a set of functions (in which case each such function corresponds to a root and all nodes are reachable from at least one root, this is sometimes called a BDD forest). The former view is adequate when studying the size of the BDD encoding a given function. The latter view reflects common practice, since all BDD libraries support sharing of BDD nodes among different functions and most BDD applications require to encode multiple functions; it is also essential when studying the size of a BDD encoding a given *set* of functions. Here, we take the latter view and explicitly assume a set of *functions of interest*, so that the BDD contains all and only the set $\mathcal{N}$ of nodes needed to encode these functions. Also, for consistency with our CESRBDDs, which define the function encoded by an edge, we assume that each BDD variant has an associated set $\mathcal{R}$ of *root edges*, conceptually originating at level $L + 1$, each encoding a function of interest of the form $\mathbb{B}^L \to \mathbb{B}$. If we say "the function encoded by BDD e", we imply that $\mathcal{R} = \{e\}$.

2.1 Quasi-reduced BDDs

We first define quasi-reduced BDDs [12] because, even if they were not the first form of canonical BDDs to be defined, they are the simplest, while all the other forms can be seen as ways to increase their compactness by eliminating some of their nodes.

Definition 2 (QBDD). An L-variable ordered BDD is a *quasi-reduced BDD* (QBDD) if it has no *duplicates*, i.e., no nodes p and q at level $k > 0$ satisfy $p[0] = q[0]$ and $p[1] = q[1]$ (this is required of all the BDD forms we consider), and no long edges, i.e., if node p is at level $k > 0$, $p[0].lvl = p[1].lvl = k - 1$. Then, the function $F_p : \mathbb{B}^k \to \mathbb{B}$ encoded by a QBDD node p at level k (where we let $\mathbb{B} = \{0, 1\}$) is recursively defined as:

$$F_p(i_1,...,i_k) = \begin{cases} F_{p[i_k]}(i_1,...,i_{k-1}) & \text{if } p.lvl > 0 \\ p & \text{if } p \text{ is node } \mathbf{0} \text{ or } \mathbf{1}. \end{cases} \square$$

In QBDDs, root edges point to nodes at level L.

2.2 Fully reduced BDDs (Bryant's BDDs)

The idea of BDDs was introduced 60 years ago [14], but their usefulness and widespread acceptance was not realized until Bryant's 1986 seminal paper [5], which introduced fundamental efficiency improvements and formally proved several desirable properties. Since then, the term "BDD" has become synonymous with Bryant's "reduced ordered BDD", but we use "fully-reduced BDD" and let "BDD" denote any generic variant.

Definition 3 (FBDD). An L-variable ordered BDD is a *fully-reduced BDD* (FBDD) if it has no *duplicates* and no *redundant* nodes, i.e., no non-terminal node p satisfying $p[0] = p[1]$. The function $F_p : \mathbb{B}^L \to \mathbb{B}$ encoded by an FBDD node p at level k is recursively defined as:

$$F_p(i_1, ..., i_L) = \begin{cases} F_{p[i_k]}(i_1, ..., i_L) & \text{if } p.lvl > 0 \\ p & \text{if } p \text{ is node } \mathbf{0} \text{ or } \mathbf{1}. \end{cases} \square$$

In FBDDs, root edges point to nodes at arbitrary levels, and the meaning of a long edge is that skipped variables are *don't-cares*. In Fig. 1, nodes s and u at level 1 are redundant, thus not in the FBDD, as they encode the constant functions 0 and 1, which are independent of x_1. This reduction is effective when it is often the case that the value of the encoded function does not change if we flip the value of some variable, as this corresponds to a redundant QBDD node, not stored in an FBDD.

2.3 Zero-suppressed decision diagrams

The *zero-suppressed BDDs* [18] offer an alternative interpretation of long edges.

Definition 4 (ZBDD). An L-variable ordered BDD is a *zero-suppressed BDD* (ZBDD) if it has no *duplicates* and no *high-zero* nodes, i.e., no non-terminal node p satisfies $p[1] = \mathbf{0}$. The function $F_p : \mathbb{B}^l \to \mathbb{B}$ encoded by a ZBDD node p at level k must be recursively defined w.r.t. a level $l \geq k$: $F_p^l(i_1, ..., i_l) =$

$$\begin{cases} \mathbf{0} & \text{if } l > k \wedge \exists h \in \{k+1, ..., l\},\ i_h = 1 \\ F_p^k(i_1, ..., i_k) & \text{if } l > k \wedge \forall h \in \{k+1, ..., l\},\ i_h = 0 \\ F_{p[i_k]}^{k-1}(i_1, ..., i_{k-1}) & \text{if } l = k > 0 \\ p & \text{if } l = k = 0. \end{cases} \square$$

In ZBDDs, root edges point to nodes at arbitrary levels, and the meaning of a long edge is that the function has value 0 if any skipped variable has value 1. In Fig. 1, nodes s and v at level 1 are high-zero, thus not in the ZBDD (node s is also redundant, since $s[0] = s[1]$). This reduction is effective when the encoded function is sparse, i.e., it tends to be 0 when some variable has value 1, as this corresponds to a high-zero QBDD node, not stored in a ZBDD.

2.4 Canonicity and efficiency

A fundamental property enjoyed by QBDDs, FBDDs, and ZBDDs is that, given a *variable order* (mapping of the L boolean variables describing the domain to levels of the decision diagram), any function $f : \mathbb{B}^L \to \mathbb{B}$ has a *unique* representation in each of them. If all nodes are stored using a *unique table* to avoid duplicates, then we can compare two functions for equality in $O(1)$ time: they are equal iff they are encoded by the same node.

This *canonicity* has profound implications not only for memory efficiency, since multiple nodes encoding the same function are never stored, but also for time efficiency. This is because all decision diagram operations recurse down by levels and, thanks to an *operation cache*, computing $f = f' \odot f''$, where f' and f'' are encoded as decision diagram nodes and $\odot$ is an element-wise boolean operation such as $\wedge$ or $\vee$, requires at most $O(||f'|| \cdot ||f''||)$ time, where $||g||$ is the number of nodes in the decision diagram encoding function g. Canonicity ensures that $||g||$ is minimal (for the given variable order and the particular variant of decision diagram), since no duplicate nodes are possible.

Obviously, a QBDD encoding a function is always at least as large as the FBDD or ZBDD encoding the same function. However, whether it is better to use FBDDs or ZBDDs on any given application largely depends on whether the QBDD representation for the functions to be encoded would contain more redundant nodes or more high-zero nodes. The difference in size can be up to a factor of $L/2$, but the best choice between FBDDs and ZBDDs may not be clear *a priori*; indeed, it may even change for different computational phases or different functions of the application at hand.

Most importantly, using FBDDs or ZBDDs means forgoing the reductions that would be achieved by the other. This has prompted researchers to explore canonical BDD extensions that provide both types of savings. An early proposal associates a fixed reduction to each level [9]: an edge skipping over levels h and $h-1$, for example, is interpreted with the reduction associated with level h for x_h and then the reduction associated with level $h-1$ for x_{h-1}. However, forcing the same reduction interpretation for all edges when skipping a given level provides little flexibility and requires deciding *a priori* the reduction for each level (thus, globally, picking one of the possible 2^L choices, assuming only two reductions, FBDD-type or ZBDD-type, are considered at each level), thus we do not consider this approach any further. In the following, we describe instead two recent proposals that integrate FBDD-type and ZBDD-type reductions in a completely flexible way.

2.5 Chaining nodes or tagging edges

In 2018, Bryant introduced *chain-reduced BDDs* (CBDDs) and *chain-reduced ZDDs* (CZDDs) [6], by allowing FBDD nodes to encode chains of high-zero nodes, or ZBDD nodes to encode chains of redundant nodes, respectively. In the CBDD variant, each node specifies two levels, the first one indicating the level at which a chain of high-zero nodes would have started (analogous to the level of an ordinary node), the second one indicating where the chain would have ended, so that edges leaving a CBDD node have the same semantic as FBDD edges. In the CZDD variant, the meaning of edges is as in ZBDDs and nodes encode chains of redundant nodes.

In either case, a node stores an additional level. Worse, the size of the CBDD and CZDD encoding can differ: there are functions whose CZDD encoding is twice as large as the CBDD encoding, and others whose CBDD encoding is three times as large as the CZDD encoding. Thus, even if CBDDs and CZDDs exploit both types of reductions, choosing between them is still important, and conceivably even more difficult (although less critical) than choosing between FBDDs and ZBDDs.

In 2017, Van Dijk introduced the *tagged BDDs* (TBDDs) [8], which associate a level tag to each edge. Again, two versions can be defined: one where FBDD reductions are implied from the source node to the tag level, and ZBDD reductions after that tag until the level above the target node; the other where ZBDD reductions are implied first and FBDD reductions second.

In either case, each node needs to store three levels: its own level plus one additional level for the tag of each of its two outgoing edges. Furthermore, again, the two versions can result in different tagged BDD sizes, and it is not easy to choose between them *a priori*.

2.6 Complemented edges

Orthogonally to the type of allowed reductions, further node savings can be achieved by allowing complemented edges, introduced in 1978 [1]. A *complement bit* is associated to each BDD edge; if the bit is 1, the value of the function F_q encoded by node q to which the edge points must be complemented. This may avoid the need to store the node q' encoding the complement $\neg F_q$ of function F_q; without complemented edges, the subDAGs rooted at q and q' would coincide, except that terminal **0** and **1** are swapped. Complemented edges can potentially save up to 50% of the nodes.

However, these savings are realized only if canonicity is enforced, and rules to include complemented edges so that BDDs remain canonical were defined only ten years later [11,15]. One way is to require that the complement bit of the 0-edge of each node be set to 0, i.e., only the 1-edge needs to store a complement bit, and to allow only the terminal node **0** (terminal node **1** is represented by a complemented edge to node **0**). Then, at any level k, only one of q encoding F_q or q' encoding $\neg F_q$ may be present, specifically, the one having value 0 when all its arguments have value 0.

Complemented edges not only save nodes (thus memory) but also reduce runtime of BDD operations, both because having fewer nodes implies more operation cache hits and because computing the BDD encoding the elementwise negation of a function f requires $O(1)$ time instead of $O(||f||)$ time.

3 CESRBDD definition

The proposals to combine FBDD and ZBDD reductions in the previous section have two disadvantages. First, nodes require substantially more memory: QBDD nodes need p bytes for each child pointer and for the "next" node pointer (needed because the *unique table*, a hash table storing all active nodes, uses a linked list to manage collisions), and FBDD or ZBDD nodes need an additional l bytes to store the node's level, for a total of $3p + l$ bytes. CBDD or CZDD nodes need $3p + 2l$ bytes, since they store an additional level per node, while TBDD nodes need $3p + 3l$ bytes, since they store an additional level for each of their two edges. If both a pointer and a level require 4 bytes for storage, CBDDs, CZDDs, and TBDDs represent a 25%, 25%, and 50% increase over FBDDs (or ZBDDs), respectively. Second, when tackling a particular problem, choosing be-

tween FBDDs, ZBDDs, CBDDs, CZDDs, or one of the two TBDD versions is non-trivial even for experts. The number of nodes can vary by a factor of 2 or 3, or even a factor of L in either direction for FBDDs vs. ZBDDs. The tradeoff is complicated because variants resulting in fewer nodes tend to require larger nodes.

Of course, the number of levels is unlikely to reach 2^{32} in practice, making a 4-byte integer a reasonable choice to store a level, but the trend toward larger addressable memory means that 8-byte pointers are becoming the norm, and this changes the relative overhead. Nevertheless, the approach we discuss next has by far the smallest memory overhead, just a few additional bits per edge, while offering more reduction opportunities. In [4], we presented *edge-specified reduction BDDs*, where a code associated to each long edge specifies whether to interpret it as fully-reduced (X, don't care), zero-suppressed ($\mathtt{H_0}$, high-zero), or "one-suppressed" ($\mathtt{L_0}$, low-zero, a new reduction analogous to zero-suppressed). This paper extends the idea by allowing further reductions and by incorporating complemented edges [1], while still maintaining canonicity.

3.1 Unreduced CESRBDDs

Definition 5 An L-variable *complemented edge-specified reduction BDD* (CESRBDD) is an L-variable ordered BDD where each edge has an associated *reduction rule* (S if the edge is short, one of X, $\mathtt{L_0}$, $\mathtt{H_0}$, $\mathtt{L_1}$, or $\mathtt{H_1}$ if the edge is long), and a *complement flag* in $\mathbb{B}$. For $v \in \mathbb{B}$, we write $p[v] = \langle \rho,c,q \rangle$ to indicate that the v-edge of p points to q, has reduction rule ρ, and complement flag c; we also write $p[v].rule = \rho$, $p[v].comp = c$, and $p[v].node = q$. The function encoded by an edge $\langle \rho,c,p \rangle$ with respect to level n, with $L \geq n \geq p.lvl$, is $F^n_{\langle \rho,c,p \rangle} =$

$$\begin{cases} \text{if } p.lvl = n \wedge n = 0, & c \oplus p \\ \text{if } p.lvl = n \wedge n > 0, & x_n \ ?\ c \oplus F^{n-1}_{p[1]} \ : \ c \oplus F^{n-1}_{p[0]} \\ \text{if } p.lvl < n \wedge \rho = \mathtt{X}, & F^{n-1}_{\langle \rho,c,p \rangle} \\ \text{if } p.lvl < n \wedge \rho = \mathtt{L_t}, & x_n \ ?\ F^{n-1}_{\langle \rho,c,p \rangle} \ : \ \mathtt{t} \\ \text{if } p.lvl < n \wedge \rho = \mathtt{H_t}, & x_n \ ?\ \mathtt{t} \ : \ F^{n-1}_{\langle \rho,c,p \rangle} \end{cases}$$

where $\oplus$ denotes "exclusive-or", "$a?b{:}c$" is b if a is true, else it is c, and $F^n_{\langle \rho,c,p \rangle}$ means $F^n_{\langle \rho,c,p \rangle}(x_1, ..., x_n)$. □

Thus, $F^n_{\langle \rho,c,p \rangle}$ is independent of ρ if $p.lvl = n$. An edge with rule X means that the skipped variables are irrelevant along that path, it corresponds to a path of redundant nodes in a QBDD (FBDD semantic). An edge with rule $\mathtt{H_0}$ means that the function has value 0 if any skipped variable has value 1, it corresponds to a path of high-zero nodes in a QBDD (ZBDD semantic). An edge with rule $\mathtt{L_0}$ means that the function has value 0 if any skipped variable has value 0, it corresponds to a path of low-zero nodes in a QBDD. An edge with rule $\mathtt{H_1}$ means that the function has value 1 if any skipped variable has value 1, it corresponds to a path of high-one nodes in a QBDD. Finally, an edge with rule $\mathtt{L_1}$ means that the function has value 1 if any skipped variable has value 0, it corresponds to a path of low-one nodes in a QBDD. In addition, if the complement flag c is set to one, the value of the function encoded by the destination node q must be complemented. Thus, CESRBDD root edges are of the form $\langle \rho,c,p \rangle$, where $p.lvl \leq L$ and $\rho \neq \mathtt{S}$ if $p.lvl < L$, so that $F^L_{\langle \rho,c,p \rangle} : \mathbb{B}^L \to \mathbb{B}$.

As we see next, CESRBDDs eliminate not only redundant nodes (as FBDDs do) and high-zero nodes (as ZBDDs do), but also other natural variants of high-zero nodes: low-zero, high-one, and low-one nodes. They do so by explicitly specifying a reduction rule to be applied to each long edge. Fig. 2 illustrates the five patterns, where the *pattern* node to be eliminated is p (from now on, we write $\langle \mathtt{S}|\rho,c,q \rangle$ to mean either $\langle \mathtt{S},c,q \rangle$ or $\langle \rho,c,q \rangle$, and use $\langle \mathtt{S}|\mathtt{X},1,\mathbf{0} \rangle$ instead of the equivalent $\langle \mathtt{S}|\mathtt{X},0,\mathbf{1} \rangle$).

Definition 6 A CESRBDD node p at level $n > 0$ is:

- *redundant*, if $p[0] = p[1] = \langle \mathtt{S}|\mathtt{X},c,q \rangle$; it is equivalent to an X-reduced edge, since $F^n_{\langle \rho,0,p \rangle} = F^n_{\langle \mathtt{X},c,q \rangle}$ for any ρ, given that $p.lvl = n$, see Fig. 2(a).
- *low-*$\mathtt{t}$, if $p[0] = \langle \mathtt{S}|\mathtt{X},\mathtt{t},\mathbf{0} \rangle$ and $p[1] = \langle \mathtt{S}|\mathtt{L_t},c,q \rangle$, for $\mathtt{t} \in \mathbb{B}$; it is equivalent to an $\mathtt{L_t}$-reduced edge, since $F^n_{\langle \rho,0,p \rangle} = F^n_{\langle \mathtt{L_t},c,q \rangle}$ for any ρ, see Fig. 2(b,c).
- *high-*$\mathtt{t}$, if $p[0] = \langle \mathtt{S}|\mathtt{H_t},c,q \rangle$ and $p[1] = \langle \mathtt{S}|\mathtt{X},\mathtt{t},\mathbf{0} \rangle$, for $\mathtt{t} \in \mathbb{B}$; it is equivalent to an $\mathtt{H_t}$-reduced edge, since $F^n_{\langle \rho,0,p \rangle} = F^n_{\langle \mathtt{H_t},c,q \rangle}$ for any ρ, see Fig. 2(d,e).

3.2 Complementing a function

Complemented edges give the ability to encode $\neg f$ by encoding f in a node q, and pointing to q with an edge whose complement flag is set to 1. In CESRBDDs, this extends to the meaning of the reduction rules attached to the edges leaving p, thus we define the complements of reduction rules and edges, and their properties.

Definition 7 Given a reduction rule ρ, its complement $\neg\rho$, is defined as follows:

$$\begin{array}{lll} \neg\mathtt{S} = \mathtt{S} & \neg\mathtt{L_0} = \mathtt{L_1} & \neg\mathtt{H_0} = \mathtt{H_1} \\ \neg\mathtt{X} = \mathtt{X} & \neg\mathtt{L_1} = \mathtt{L_0} & \neg\mathtt{H_1} = \mathtt{H_0}. \end{array}$$ □

Definition 8 Given an edge $\langle \rho,c,q \rangle$, its complement is

$$\neg\langle \rho,c,q \rangle \ = \ \langle \neg\rho,\neg c,q \rangle.$$ □

The following theorem proves that complementing an edge complements the function it encodes.

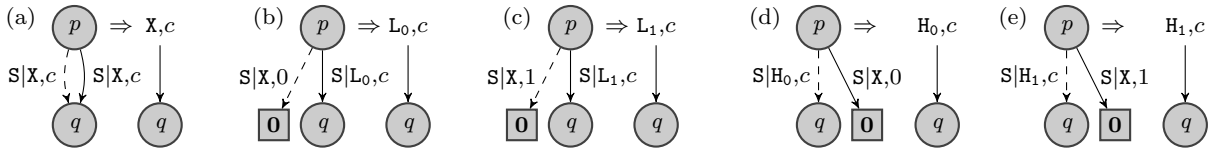


Fig. 2 Patterns of nodes not allowed in canonical CESRBDDs: redundant, low-zero, low-one, high-zero, and high-one.

Theorem 1 For any edge $\langle\rho,c,q\rangle$ and level $n \geq q.lvl$, $F^n_{\neg\langle\rho,c,q\rangle} = \neg F^n_{\langle\rho,c,q\rangle}$.

Proof: By induction on n. In the base case, $n = 0$,

$$F^0_{\neg\langle\rho,c,q\rangle} \;=\; F^0_{\langle\neg\rho,\neg c,q\rangle} \;=\; \neg c \oplus q \;=\; \neg F^0_{\langle\rho,c,q\rangle}.$$

For the inductive step, we assume the theorem holds for $n = m$, and prove it holds for $n = m + 1$, by splitting the proof according to the definition of F.

If $q.lvl = n$,

$$\begin{aligned} F^n_{\langle\neg\rho,\neg c,q\rangle} &= x_n \;?\; \neg c \oplus F^{n-1}_{q[1]} : \neg c \oplus F^{n-1}_{q[0]} \\ &= \neg\Big(x_n \;?\; c \oplus F^{n-1}_{q[1]} : c \oplus F^{n-1}_{q[0]}\Big) = \; \neg F^n_{\langle\rho,c,q\rangle}. \end{aligned}$$

If $q.lvl < n$ and $\rho = \mathtt{X}$,

$$F^n_{\langle\neg\mathtt{X},\neg c,q\rangle} = F^{n-1}_{\langle\neg\mathtt{X},\neg c,q\rangle} \;=\; \neg F^{n-1}_{\langle\mathtt{X},c,q\rangle} \;=\; \neg F^n_{\langle\mathtt{X},c,q\rangle}.$$

If $q.lvl < n$ and $\rho = \mathtt{L_t}$,

$$\begin{aligned} F^n_{\langle\neg\mathtt{L_t},\neg c,q\rangle} &= F^n_{\langle\mathtt{L}_{\neg\mathtt{t}},\neg c,q\rangle} \;=\; x_n \;?\; F^{n-1}_{\langle\neg\mathtt{L_t},\neg c,q\rangle} \;:\; \neg\mathtt{t} \\ &= x_n \;?\; \neg F^{n-1}_{\langle\mathtt{L_t},c,q\rangle} \;:\; \neg\mathtt{t} \;=\; \neg F^n_{\langle\mathtt{L_t},c,q\rangle}. \end{aligned}$$

If $q.lvl < n$ and $\rho = \mathtt{H_t}$,

$$\begin{aligned} F^n_{\langle\neg\mathtt{H_t},\neg c,q\rangle} &= F^n_{\langle\mathtt{H}_{\neg\mathtt{t}},\neg c,q\rangle} \;=\; x_n \;?\; \neg\mathtt{t} \;:\; F^{n-1}_{\langle\neg\mathtt{H_t},\neg c,q\rangle} \\ &= x_n \;?\; \neg\mathtt{t} \;:\; \neg F^{n-1}_{\langle\mathtt{H_t},c,q\rangle} \;=\; \neg F^n_{\langle\mathtt{H_t},c,q\rangle}. \end{aligned}$$

□

3.3 Reduced CESRBDDs

To force CESRBDDs to be canonical, we reduce them by requiring, as usual, that all possible node eliminations be performed; however, we must impose further restrictions to fix a single representation when multiple equivalent possibilities exist.

Definition 9 A CESRBDD is *reduced* iff:

1. It contains no duplicates (as defined in Def. 2).
2. No edge points to terminal node **1**.
3. For any non-terminal node p, $p[0].comp = 0$.
4. For any edge $\langle\rho,0,\mathbf{0}\rangle$, ρ must be $\mathtt{X}$, $\mathtt{L_1}$, or $\mathtt{H_1}$. For any edge $\langle\rho,1,\mathbf{0}\rangle$, ρ must be $\mathtt{X}$, $\mathtt{L_0}$, or $\mathtt{H_0}$. Moreover, edges $\langle\mathtt{L_1},0,\mathbf{0}\rangle$ and $\langle\mathtt{L_0},1,\mathbf{0}\rangle$ cannot originate from nodes at level 2 or be root edges when $L = 1$.
5. There are no pattern nodes, i.e., no redundant, high-zero, high-one, low-zero, or low-one nodes.
6. There is no node p at level 2 with $p[0] = \langle\mathtt{X},0,\mathbf{0}\rangle$ and $p[1] = \langle\mathtt{H_1},0,\mathbf{0}\rangle$. Such a node would encode $x_1 \wedge x_2$, so we call it a "level-two-and" node. □

We now give a rationale for Definition 9. First of all, note that requirement (3) implicitly forbids low-one nodes, and requirement (2) forbids representing them with a complemented 0-edge to terminal **1**, so their mention in requirement (5) is redundant, we just include it for clarity and symmetry. The reason for requirement (1) is obvious. Requirement (2) forces the use of an edge to terminal **0** with complement flag set to 1 instead of an equivalent edge to terminal **1**, which can then be eliminated. Requirement (3) implies that a CESRBDD may not contain two nodes encoding a function and its complement; it also implies that only 1-edges need to store a complement flag. Requirements (2) and (3) are standard for complemented-edge BDDs.

Requirement (4) arbitrarily resolves several cases where different edges can encode the same function, see Fig. 3, where we observe right away that patterns involving node p violate requirement (5). The constant function 0 is represented with edge $\langle\mathtt{X},0,\mathbf{0}\rangle$, instead of one of the equivalent edges $\langle\mathtt{L_0},0,\mathbf{0}\rangle$ and $\langle\mathtt{H_0},0,\mathbf{0}\rangle$; the constant function 1 is represented with edge $\langle\mathtt{X},1,\mathbf{0}\rangle$, instead of one of the equivalent edges $\langle\mathtt{L_1},1,\mathbf{0}\rangle$ and $\langle\mathtt{H_1},1,\mathbf{0}\rangle$ (the figure lists the possible functions for $L = 1$, but this encoding of constant functions holds for any L). Then, observe that, for $n \in \{1,...,L\}$, $F^n_{\langle\mathtt{L_1},0,\mathbf{0}\rangle} = \bigvee_{1\leq k\leq n} \neg x_k$, $F^n_{\langle\mathtt{L_0},1,\mathbf{0}\rangle} = \bigwedge_{1\leq k\leq n} x_k$, $F^n_{\langle\mathtt{H_0},1,\mathbf{0}\rangle} = \bigwedge_{1\leq k\leq n} \neg x_k$, and $F^n_{\langle\mathtt{H_1},0,\mathbf{0}\rangle} = \bigvee_{1\leq k\leq n} x_k$. However, disjunction and conjunction coincide if $n = 1$, i.e., $F^1_{\langle\mathtt{L_0},1,\mathbf{0}\rangle} = F^1_{\langle\mathtt{H_1},0,\mathbf{0}\rangle} = x_1$ and $F^1_{\langle\mathtt{L_1},0,\mathbf{0}\rangle} = F^1_{\langle\mathtt{H_0},1,\mathbf{0}\rangle} = \neg x_1$. Thus, requirement (4) arbitrarily forbids edges $\langle\mathtt{L_1},0,\mathbf{0}\rangle$ and $\langle\mathtt{L_0},1,\mathbf{0}\rangle$ to originate from nodes at level 2 or be used as root edges when $L = 1$, since in either case they would be skipping only one variable, x_1. In other words, the function of one variable $f_1 = x_1$ is uniquely represented by edge $\langle\mathtt{H_1},0,\mathbf{0}\rangle$, and the function of one variable $f_2 = \neg x_1$ is uniquely represented by edge $\langle\mathtt{H_0},1,\mathbf{0}\rangle$.

Requirement (5) is analogous to the prohibition of redundant nodes in FBDDs or high-zero nodes in ZBDDs: it seeks to maximize the length of CESRBDD edges and is key to achieve canonicity.

Finally, we need requirement (6) because node p at the bottom right of Fig. 4 encodes $x_1 \wedge x_2$, thus for

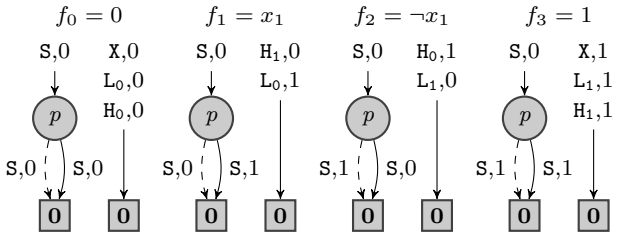


Fig. 3 CESRBDDs encoding all functions of one variable.

x_2	x_1	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

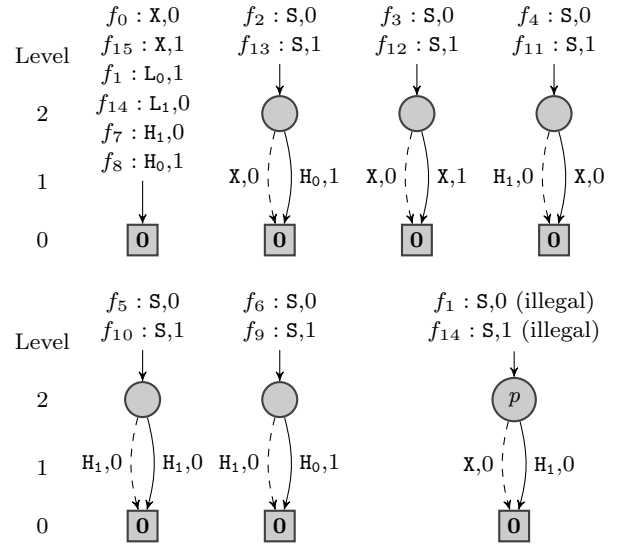


Fig. 4 CESRBDDs encoding all functions of two variables. The last encoding is illegal because p is a level-two-and node.

$n \geq 3$, $F^n_{\langle \mathtt{L}_0,0,p\rangle} = F^n_{\langle \mathtt{L}_0,1,\mathbf{0}\rangle} = \bigwedge_{1\leq k\leq n} x_k$, destroying canonicity. If p is at a level n above 2, the problem does not arise, since then p encodes $x_n \wedge \bigvee_{1\leq k\leq n-1} x_k$, a function that cannot be encoded by a single CESRBDD edge to terminal **0**. Note that a similar issue would arise if p had edges $p[0] = \langle \mathtt{X},1,\mathbf{0}\rangle$ and $p[1] = \langle \mathtt{H}_0,1,\mathbf{0}\rangle$, since, then, $F^n_{\langle \mathtt{L}_1,0,p\rangle} = F^n_{\langle \mathtt{L}_1,0,\mathbf{0}\rangle} = \neg \bigwedge_{1\leq k\leq n} x_k$, but such a node is already disallowed by requirement (3).

In Fig. 1, the CESRBDD encoding requires a single nonterminal node, p. We will show in Sec. 4 that CESRBDDs have no nodes at level 1, thus nodes s, t, u, and v are obviously absent. Then, nodes q and r at level 2 are also absent, as the functions they encode, $x_1 \wedge x_2$ and its negation, are represented by edges $\langle L_0,1,\mathbf{0}\rangle$ and $\langle L_1,0,\mathbf{0}\rangle$ originating from a node at level 3 (see f_1 and f_{14} in Fig. 4). However, the resulting node p would have $p[0] = \langle L_0,1,\mathbf{0}\rangle$ and $p[1] = \langle L_1,0,\mathbf{0}\rangle$, violating requirement (3); thus we complement both edges and use a root edge $\langle \mathtt{S},1,p'\rangle$ instead. Of course, the CESRBDD does not have terminal node **1** either but, this is just a consequence of using complemented edges. For fairness we observe that, if we allowed complemented edges for QBDDs, nodes **1**, r, u, and v would not be present (they encode the negation of the function encoded by **0**, q, s, and v, respectively); analogously, in the FBDD, nodes **1**, r, and v would not be present. As for ZBDDs, we have not found any reference allowing complemented edge. However, *0-element edges* have been proposed for ZBDDs [7]; if employed in this case, the resulting encoding would have only terminal **0** and four nonterminal nodes, one fewer than ordinary ZBDDs, but one more than FBDDs with complemented edges.

3.4 Reduction algorithm

Unreduced CESRBDDs are universal, i.e., can encode any function of the form $\mathbb{B}^L \to \mathbb{B}$, since QBDDs (or FBDDs, ZBDDs) are CESRBDDs, and are universal. However, it is not obvious that reduced CESRBDDs are universal. We prove they are by providing an algorithm that, given an unreduced CESRBDD, returns a reduced CESRBDD encoding the same function. Such a reduction algorithm can thus be used to transform a QBDD, FBDD, or ZBDD into a reduced CESRBDD.

Algorithm 1 shows procedure ReduceEdge, which takes a level n and an edge e' in a possibly unreduced CESRBDD, and builds an equivalent edge e in a reduced CESRBDD such that $F^n_{e'} = F^n_e$. It uses two helpers: procedure ReduceNode takes a node p, recursively reduces its children, and returns an equivalent edge (in case p must be eliminated); and procedure MergeEdge takes a level n, a reduction rule ρ, and an edge e, and "applies ρ to e" by building an edge e' such that $F^n_{e'} = F^n_{\langle \rho,0,p\rangle}$ for any p satisfying $F^{p.lvl}_{\langle \mathtt{S},0,p\rangle} = F^{p.lvl}_e$. To reduce a CESRBDD, we call ReduceEdge(L, e) for each root edge $e \in \mathcal{R}$. The following theorem proves the correctness of these procedures.

Theorem 2 Given level n and edge $\langle \rho,c,p\rangle$ s.t. $n \geq p.lvl$ and $\rho = \mathtt{S}$ if $n = p.lvl$, the call ReduceEdge$(n, \langle \rho,c,p\rangle)$ returns a reduced CESRBDD e' s.t. $F^n_{\langle \rho,c,p\rangle} = F^n_{e'}$ and the call ReduceNode(p) returns a reduced CESRBDD e' s.t. $F^{p.lvl}_{\langle \mathtt{S},0,p\rangle} = F^{p.lvl}_{e'}$.

Proof: The proof is by induction on n. For the base case $n = 0$, p must be a terminal node and $\rho = \mathtt{S}$. Then, the call ReduceNode(p) returns $e' = \langle \mathtt{S},0,\mathbf{0}\rangle$ if $p = \mathbf{0}$, and trivially $F^0_{\langle \mathtt{S},0,\mathbf{0}\rangle} = F^0_{\langle \mathtt{S},0,\mathbf{0}\rangle}$, or $e' = \langle \mathtt{S},1,\mathbf{0}\rangle$ if $p = \mathbf{1}$, and $F^0_{\langle \mathtt{S},0,\mathbf{1}\rangle} = F^0_{\langle \mathtt{S},1,\mathbf{0}\rangle}$ (line 6); in either case e' is reduced. The call ReduceEdge$(0, \langle \mathtt{S},c,p\rangle)$ calls ReduceNode(p), sets $e' = \langle \mathtt{S},c',\mathbf{0}\rangle$, where $c' = 0$ if

Algorithm 1 ReduceEdge($n, \langle \rho,c,p \rangle$) reduces edge $\langle \rho,c,p \rangle$ w.r.t level n; requires $n \geq p.lvl$ and $\rho = \mathtt{S}$ if $n = p.lvl$.

1: **procedure** ReduceEdge(level n, edge $\langle \rho,c,p \rangle$) ▷ return edge e s.t. $F^n_{\langle \rho,c,p \rangle} = F^n_e$ and e is reduced
2: $e \leftarrow$ ReduceNode(p);
3: **if** $c = 1$ **then** $e \leftarrow \neg e$; ▷ push down the complement flag c to the edge returned by ReduceNode
4: **return** MergeEdge(n, ρ, e);
5: **procedure** ReduceNode(node p) ▷ return edge e s.t. $F^{p.lvl}_{\langle \mathtt{S},0,p \rangle} = F^{p.lvl}_e$ and e is reduced
6: **if** $p.lvl = 0$ **then return** $\langle \mathtt{S},(p = \mathbf{1})\,?\,1:0,\mathbf{0} \rangle$; ▷ if called on $\mathbf{1}$, return $\langle \mathtt{S},1,\mathbf{0} \rangle$; if called on $\mathbf{0}$, return $\langle \mathtt{S},0,\mathbf{0} \rangle$
7: **if** cached [ReduceNode(p) : e] **then return** e; ▷ return cached result if available
8: $p' \leftarrow$ new node at level $p.lvl$;
9: $p'[0] \leftarrow$ ReduceEdge($p.lvl - 1, p[0]$); $p'[1] \leftarrow$ ReduceEdge($p.lvl - 1, p[1]$); ▷ recursively reduce p's children
10: $c \leftarrow p'[0].comp$;
11: **if** $c = 1$ **then** { $p'[0] \leftarrow \neg p'[0]$; $p'[1] \leftarrow \neg p'[1]$; } ▷ ensure that $p'[0].comp = 0$ by complementing both children
12: **if** p' is a redundant node **then** { $e \leftarrow p'[0]$; $e.rule = \mathtt{X}$; } ▷ set $e.rule$ to $\mathtt{X}$, in case $p'[0].rule$ is $\mathtt{S}$ and not $\mathtt{X}$
13: **else if** p' is a high-$\mathtt{t}$ node **then** { $e \leftarrow p'[0]$; $e.rule \leftarrow \mathtt{H_t}$; } ▷ set $e.rule$ to $\mathtt{H_t}$, in case $p'[0].rule$ is $\mathtt{S}$ and not $\mathtt{H_t}$
14: **else if** p' is a low-zero node **then** { $e \leftarrow p'[1]$; $e.rule \leftarrow \mathtt{L_0}$; } ▷ set $e.rule$ to $\mathtt{L_0}$, in case $p'[1].rule$ is $\mathtt{S}$ and not $\mathtt{L_0}$
15: **else if** p' is a level-two-and node **then** $e \leftarrow \langle \mathtt{L_0},1,\mathbf{0} \rangle$; ▷ replace p' with the equivalent $\langle \mathtt{L_0},1,\mathbf{0} \rangle$ edge
16: **else if** p' duplicates an existing node q **then** $e \leftarrow \langle \mathtt{S},0,q \rangle$; ▷ discard p' and use q, at the same level, instead
17: **else** $e \leftarrow \langle \mathtt{S},0,p' \rangle$; ▷ node p' is still at the level of p
18: **if** $c = 1$ **then** $e \leftarrow \neg e$; ▷ if we complemented the children, push up the complement flag c
19: cache [ReduceNode(p) : e];
20: **return** e; ▷ since low-zero is tested after high-$\mathtt{t}$, it returns $\langle \mathtt{H_0},1,\mathbf{0} \rangle$, not $\langle \mathtt{L_1},0,\mathbf{0} \rangle$, if $p.lvl = 1$ and $F^1_{\langle \mathtt{S},0,p \rangle} = \neg x_1$
21: **procedure** MergeEdge(level n, rule ρ, edge e) ▷ return e' s.t. $F^n_{e'} = F^n_{\langle \rho,0,p \rangle}$, for any p satisfying $F^{p.lvl}_{\langle \mathtt{S},0,p \rangle} = F^{p.lvl}_e$
22: **if** $e.rule = \mathtt{S}$ **then** $e.rule \leftarrow \rho$; ▷ ρ and $\mathtt{S}$ can be merged into ρ
23: **if** $p.lvl = 1 \wedge e = \langle \mathtt{H_0},1,\mathbf{0} \rangle \wedge \rho = \mathtt{L_1}$ **then return** $\langle \mathtt{L_1},0,\mathbf{0} \rangle$; ▷ $\mathtt{H_0}$ and $\mathtt{L_1}$ can be merged, $F^n_{\langle \rho,c,p \rangle} = \bigvee_{k=1}^n \neg x_k$
24: **if** $p.lvl = 1 \wedge e = \langle \mathtt{H_1},0,\mathbf{0} \rangle \wedge \rho = \mathtt{L_0}$ **then return** $\langle \mathtt{L_0},1,\mathbf{0} \rangle$; ▷ $\mathtt{H_1}$ and $\mathtt{L_0}$ can be merged, $F^n_{\langle \rho,c,p \rangle} = \bigwedge_{k=1}^n x_k$
25: **if** $e = \langle \mathtt{X},0,\mathbf{0} \rangle \wedge \rho \in \{\mathtt{S},\mathtt{X},\mathtt{L_0},\mathtt{H_0}\}$ **then return** $\langle \mathtt{X},0,\mathbf{0} \rangle$; ▷ $F^n_{\langle \rho,c,p \rangle} = 0$
26: **if** $e = \langle \mathtt{X},1,\mathbf{0} \rangle \wedge \rho \in \{\mathtt{S},\mathtt{X},\mathtt{L_1},\mathtt{H_1}\}$ **then return** $\langle \mathtt{X},1,\mathbf{0} \rangle$; ▷ $F^n_{\langle \rho,c,p \rangle} = 1$
27: **if** $(n = 1) \wedge (e = \langle \mathtt{L_t},\neg\mathtt{t},\mathbf{0} \rangle)$ **then return** $\langle \mathtt{H_{\neg t}},\mathtt{t},\mathbf{0} \rangle$; ▷ enforce requirement 4
28: **if** $\rho = e.rule \vee \rho = \mathtt{S}$ **then return** e; ▷ ρ and $e.rule$ are already compatible
29: $q \leftarrow$ new node at level $p.lvl + 1$; ▷ incompatible rules, create a new node immediately above p
30: **if** $\rho = \mathtt{X}$ **then** { $q[0] \leftarrow e$; $q[1] \leftarrow e$; }
31: **else if** $\rho = \mathtt{L_t}$ **then** { $q[0] \leftarrow \langle \mathtt{X},\mathtt{t},\mathbf{0} \rangle$; $q[1] \leftarrow e$; }
32: **else if** $\rho = \mathtt{H_t}$ **then** { $q[1] \leftarrow \langle \mathtt{X},\mathtt{t},\mathbf{0} \rangle$; $q[0] \leftarrow e$; }
33: $e.comp \leftarrow q[0].comp$;
34: **if** $e.comp = 1$ **then** { $q[0] \leftarrow \neg q[0]$; $q[1] \leftarrow \neg q[1]$; } ▷ ensure that $q[0].comp = 0$ by complementing both children
35: **if** $n = q.lvl$ **then** $e.rule \leftarrow \mathtt{S}$; **else** $e.rule \leftarrow \rho$;
36: **if** q duplicates existing node q' **then** $e.node \leftarrow q'$; **else** $e.node \leftarrow q$; ▷ discard q and use duplicate q' if present
37: **return** e;

$c = 0$ and $p = \mathbf{0}$ or $c = 1$ and $p = \mathbf{1}$, otherwise $c' = 1$ (line 3), and returns e'; obviously, $F^0_{\langle \mathtt{S},c,p \rangle} = F^0_{e'}$ and, again, the returned edge is reduced.

Next, assume the theorem holds up to a given n for any edge $\langle \rho,c,p \rangle$ with $p.lvl \leq n$, and show it holds for $n+1$.

If $p.lvl = m \leq n+1$, the call ReduceNode(p) sets $p'[v] \leftarrow$ ReduceEdge($m-1, p[v]$), for $v \in \mathbb{B}$, by inductive hypothesis reduced and s.t. $F^{m-1}_{p[v]} = F^{m-1}_{p'[v]}$. Thus, $F^m_{\langle \mathtt{S},0,p \rangle} = F^m_{\langle \mathtt{S},0,p' \rangle}$, but we need to ensure that $\langle \mathtt{S},0,p' \rangle$ is reduced. First of all, if $p'[0].comp = 1$, we remember this fact by setting $c = 1$ and complement the outgoing edges of p' (line 11). Then, the next five lines check whether node p' should be deleted because it is redundant, high-$\mathtt{t}$, or low-zero (it cannot be low-one, since $p'[0].comp = 0$), in which case we set edge e' according to Fig. 2, or it is a level-two-and node, in which case we set e' to $\langle \mathtt{L_0},1,\mathbf{0} \rangle$, or it duplicates a node q, in which case we set e' to $\langle \mathtt{S},0,q \rangle$. Finally, we complement edge e' if c had been set to 1 (line 18) and return it, as it is now reduced and still satisfies $F^m_{\langle \mathtt{S},0,p \rangle} = F^m_{e'}$.

Consider now the call ReduceEdge($n+1, \langle \rho,c,p \rangle$) when $p.lvl = m \leq n+1$, which first sets e' to ReduceNode(p), just shown to return a reduced edge s.t. $F^m_{\langle \mathtt{S},0,p \rangle} = F^m_{e'}$, then it complements e' if $c = 1$ (line 3), and calls MergeEdge($n+1, \rho, e'$) which combines ρ with e' as follows. First, MergeEdge sets the reduction rule of e' to ρ if it was $\mathtt{S}$, since $\mathtt{S}$ combined with any ρ is just ρ (line 22), and checks for some special cases (lines 23–27) where the returned edge e' points to $\mathbf{0}$, is obviously reduced, and satisfies $F^{n+1}_{\langle \rho,c,p \rangle} = F^{n+1}_{e'}$ in ReduceEdge. Then, if rules ρ (from above) and $e'.rule$ (from below) are compatible, it simply returns the adjusted edge.

Otherwise, for rules that are incompatible, i.e., are different and neither is S, it creates a new node q at level $m+1$, sets its outgoing edges, enforces $q[0].comp = 0$ by complementing its children if needed, and checks whether q duplicates an existing node. The logic for the creation of node q is as follows: when ρ is incompatible with $e'.rule$, the edge from above is "shortened by one level", so that it points to q instead of p, but q must still encode the logic of reduction ρ by appropriately setting its outgoing edges to e' or $\langle \mathtt{X},\mathtt{t},\mathbf{0}\rangle$. For example, Fig. 5 shows the case where ReduceNode(p) returns $e' = \langle \mathtt{X},0,u\rangle$, which is incompatible with all the rules of the four edges pointing to p in the unreduced CESRBDD. Edge $\langle \mathtt{L_0},0,p\rangle$ becomes $\langle \mathtt{L_0},0,q_a\rangle$ (or $\langle \mathtt{S},0,q_a\rangle$ if it is short), where $q_a[0] = \langle \mathtt{X},0,\mathbf{0}\rangle$ (since $f_a^4 = 0$ if $x_4 = 0$) and $q[1] = \langle \mathtt{X},0,u\rangle$ (since $f_a^4 = f^2_{\langle \mathtt{S},0,u\rangle}$ if $x_4 = 1$. All 4×5 incompatible pairs (each of the five non-S values for $e'.rule$ is incompatible with four non-S rules) are treated in similar way, and in no instance the new node q violates req. 5 of Def. 9. Thus, in all cases, the edge returned by MergeEdge (and therefore by ReduceEdge) is reduced, and edge e' returned by ReduceEdge satisfies $F^{n+1}_{\langle \rho,c,p\rangle} = F^{n+1}_{e'}$. □

We proved that reduced CESRBDDs are universal: given any $f : \mathbb{B}^L \to \mathbb{B}$, there is a reduced CESRBDD $\langle \rho,c,p\rangle$ such that $F^L_{\langle \rho,c,p\rangle} = f$. However, this does not imply that CESRBDDs are canonical, i.e., if $F^L_e = F^L_{e'}$, then $e = e'$. Theorem 4 in the next section proves this fundamental result. Thus, from now on we limit our discussion to reduced CESRBDDs and "CESRBDD e" will mean "the unique reduced CESRBDD encoding a particular function F^L_e".

We now address the size of reduced CESRBDDs, since a reduced CESRBDD is not necessarily the smallest CESRBDD encoding a set of functions, as shown in Fig. 5, where $F^3_{\langle \mathtt{S},0,p\rangle} = F^2_{\langle \mathtt{S},0,q\rangle} = x_2 \wedge \neg x_1$. The unreduced CESRBDD on the left encodes the four functions

$$a = \quad x_4 \wedge x_2 \wedge \neg x_1 \qquad b = \neg x_4 \vee (x_2 \wedge \neg x_1)$$
$$c = \neg x_4 \wedge x_2 \wedge \neg x_1 \qquad d = \quad x_4 \vee (x_2 \wedge \neg x_1)$$

and so does the reduced CESRBDD on the right, where redundant node p is replaced by nodes p_a, p_b, p_c, and p_d, each created by ReduceEdge when it finds out that the edge $\langle \mathtt{X},0,q\rangle$ returned by ReduceNode(p) is incompatible with the rules $\mathtt{L_0}$, $\mathtt{L_1}$, $\mathtt{H_0}$, and $\mathtt{H_1}$ on the four edges pointing to p. Reducing the CESRBDD increases the number of nonterminal nodes from two to five, although to be fair the four root edges should be somehow accounted for. Alternatively, we can consider only CESRBDDs encoding a single function, by adding to either CESRBDD a node r at level 6 and its two children at level 5, collectively having edges $\langle \mathtt{L_0},0,p\rangle$, $\langle \mathtt{L_1},0,p\rangle$, $\langle \mathtt{H_0},0,p\rangle$, and $\langle \mathtt{H_1},0,p\rangle$. The unreduced and reduced CESRBDD with root edge $\langle \mathtt{S},0,r\rangle$ would then have five and eight nonterminal nodes, respectively.

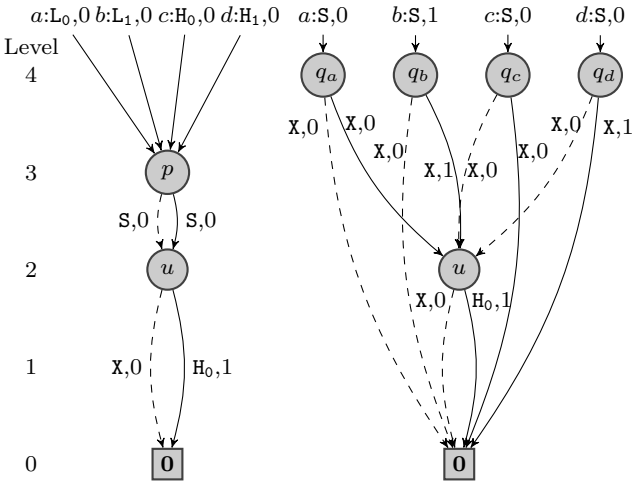


Fig. 5 Unreduced and equivalent reduced CESRBDD.

This lack of minimality stems from creating nodes at level $p.lvl+1$, if p is redundant, low-t, or high-t, and has incoming edges with incompatible rules. While this may at first seem undesirable, we stress that it can only happen when multiple reduction rules (beyond S) are present in the unreduced CESRBDD, since S is compatible with any rule and each rule is compatible with itself. In particular, an FBDD only uses rules S and X and a ZBDD only uses rules S and $\mathtt{H_0}$, so their rule pairs are always compatible and a reduced CESRBDD has never more nodes than the equivalent FBDD or ZBDD (even ignoring the effect of complemented edges, which can of course further reduce the size of the CESRBDD). This is a special case of the results we present next, in particular Corollary 1.

Definition 10 The *incoming rule set* for node $q \in \mathcal{N}$ is the set of rules on (root or ordinary) edges into q, $\mathcal{I}(q) = \{\rho : \exists e, e.node = q \wedge e.rule = \rho \wedge (e \in \mathcal{R} \vee \exists p \in \mathcal{N}, v \in \mathbb{B}, p[v] = e)\}$. The *cost* of node $q \in \mathcal{N}$ is $C(q) = 1$ if $\mathcal{I}(q) = \{\mathtt{S}\}$ and $|\mathcal{I}(q) \setminus \{\mathtt{S}\}|$ otherwise. The *total cost* of a CESRBDD with root edges $\mathcal{R}$ and nodes $\mathcal{N}$ is the sum of its nodes' costs, $C(\mathcal{R},\mathcal{N}) = \sum_{q\in\mathcal{N}} C(q)$. □

Theorem 3 Given a CESRBDD with root edges $\mathcal{R}$ and nodes $\mathcal{N}$, the reduced CESRBDD with root edges $\mathcal{R}'$ and nodes $\mathcal{N}'$ encoding the same functions, i.e., for each $e \in \mathcal{R}$ there is an $e' \in \mathcal{R}'$ s.t. $F^L_e = F^{\prime L}_{e'}$, satisfies $C(\mathcal{R},\mathcal{N}) \geq C(\mathcal{R}',\mathcal{N}')$.

Proof: We prove the theorem by considering how the transformations performed by the reduction algorithm cannot increase the total cost.

If ReduceNode or ReduceEdge eliminate a node p because it duplicates a node q, p's incoming edges are

redirected to q. Let q' denote q after the incoming edges of p have been redirected to it. Then $\mathcal{I}(q') = \mathcal{I}(q) \cup \mathcal{I}(p)$, and it follows that $C(q') \leq C(q) + C(p)$.

If ReduceNode eliminates a node p because it is redundant, low-`t`, or high-`t`, it returns an edge e' with rule $\rho' \neq \mathtt{S}$. Then, ReduceEdge can merge e' with all incoming edges to p having compatible rule $\rho \in \{\mathtt{S}, \rho'\}$, but must instead create a new node p_ρ (line 29) for each rule ρ in $\mathcal{I}(p) \setminus \{\mathtt{S}, \rho'\}$. This new node will have $\mathcal{I}(p_\rho) \subseteq \{\mathtt{S}, \rho\}$, thus, $C(p_\rho) = 1$, and it follows that $C(p) \geq \sum_{\rho \in \mathcal{I}(p) \setminus \{\mathtt{S}, \rho'\}} C(p_\rho)$.

The only other cases where edges or nodes are modified in a way that can affect the total cost are the check for level-two-and nodes (line 15) and for special cases involving an edge to **0** (lines 23–27), and none of this increases the total cost. □

Corollary 1 Since the minimum node cost is one, if every node $q \in \mathcal{N}$ satisfies $|\mathcal{I}(q) \setminus \{\mathtt{S}\}| \leq 1$, the reduction algorithm cannot increase the total number of nodes.

4 Canonicity of CESRBDDs

We first show that reduced *CESRBDDs have no nodes at level 1*, since all four possible nodes at level 1 (see Fig. 3, where node p is at level 1) correspond to forbidden patterns and are replaced by long edges, using requirement (4) to resolve equivalences. As a consequence, all edges to terminal **0** must be long.

Lemma 1 A reduced CESRBDD has no level-1 nodes.

Proof: Consider all possible nodes q with $q.lvl = 1$. Due to req. 3 in Def. 9, we must have $q[0] = \langle \mathtt{S},0,0 \rangle$, while $q[1] = \langle \mathtt{S},c,0 \rangle$. If $c = 0$, then q is a redundant node. If $c = 1$, then q is both a low-zero and a high-one node. Either way, req. 5 in Def. 9 disallows node q. □

The next theorem proves that CESRBDDs are canonical, i.e., once a variable order is defined, each function $f : \mathbb{B}^L \to \mathbb{B}$ has a unique CESRBDD encoding.

Theorem 4 In a reduced CESRBDD, for any integer $n \geq 0$, for any two edges $e = \langle \rho,c,q \rangle$, $e' = \langle \rho',c',q' \rangle$ with $q.lvl \leq n$ and $q'.lvl \leq n$: if $F^n_e = F^n_{e'}$ then (1) $c = c'$ and $q = q'$ and, (2) if $q.lvl < n$ then $\rho = \rho'$.

Proof: The proof is by induction on n. We prove the base cases, $n = 0, 1, 2$, by enumeration.

For $n = 0$, we have $q = q' = \mathbf{0}$, and $\mathbf{0}.lvl = n$, thus we only need to prove that $c = c'$ if $F^0_e = F^0_{e'}$. This is immediate since $F^0_e = c$ and $F^0_{e'} = c'$, two constant functions.

For $n = 1$, we already discussed how the four possible functions of one variable, $f_0 = 0$, $f_1 = x_1$, $f_2 = \neg x_1$, and $f_3 = 1$ are uniquely encoded by edges $\langle \mathtt{X},0,\mathbf{0} \rangle$, $\langle \mathtt{H}_1,0,\mathbf{0} \rangle$, $\langle \mathtt{H}_0,1,\mathbf{0} \rangle$, and $\langle \mathtt{X},1,\mathbf{0} \rangle$, respectively (see Fig 3).

For $n = 2$, all 16 possible functions F^2_e are encoded in Fig. 4, and we now prove that these are the *only* reduced representations for these functions. Consider all possible non-terminal nodes q at level 2. From Lemma 1 and reqs. 2 and 4 in Def. 9, we have that $q[v]$ can be either $\langle \mathtt{X},\mathtt{t},\mathbf{0} \rangle$, $\langle \mathtt{H}_0,1,\mathbf{0} \rangle$, or $\langle \mathtt{H}_1,0,\mathbf{0} \rangle$. From req. 3 in Def. 9, we know $q[0].comp = 0$, leaving two possibilities for $q[0]$. If $q[0] = \langle \mathtt{X},0,\mathbf{0} \rangle$, then $q[1] \neq \langle \mathtt{X},0,\mathbf{0} \rangle$, because otherwise q would be a redundant node, and $q[1] \neq \langle \mathtt{H}_1,0,\mathbf{0} \rangle$, because otherwise q would be a level-two-and node (the illegal pattern in the bottom right of Fig. 4). This leaves two possibilities for $q[1]$, corresponding to functions f_2 and f_3, or their complements f_{13} and f_{12}, in Fig. 4. If instead $q[0] = \langle \mathtt{H}_1,0,\mathbf{0} \rangle$, then $q[1] \neq \langle \mathtt{X},1,\mathbf{0} \rangle$, because otherwise q would be a high-one node. This leaves three possibilities for $q[1]$, corresponding to functions f_4, f_5, and f_6, or their complements f_{11}, f_{10}, and f_9 in Fig. 4. Finally, considering the possibility of edges directly to terminal node **0**: from req. 4 in Def. 9, the only possible edges are $\langle \mathtt{X},0,\mathbf{0} \rangle$, $\langle \mathtt{X},1,\mathbf{0} \rangle$, $\langle \mathtt{L}_0,1,\mathbf{0} \rangle$, $\langle \mathtt{L}_1,0,\mathbf{0} \rangle$, $\langle \mathtt{H}_0,1,\mathbf{0} \rangle$, and $\langle \mathtt{H}_1,0,\mathbf{0} \rangle$, corresponding respectively to f_0, f_{15}, f_1, f_{14}, f_8, and f_7 in the left portion of Fig. 4.

Now, assuming the theorem holds for all $m < n$, we will prove it holds for n.

First, assume $q.lvl = q'.lvl = n$. It follows that, for $v \in \mathbb{B}$,

$$c \oplus F^{n-1}_{q[v]} = c' \oplus F^{n-1}_{q'[v]}.$$

Let $q[v] = \langle \rho,c_v,q_v \rangle$ and $q'[v] = \langle \rho'_v,c'_v,q'_v \rangle$. If $c \neq c'$, then

$$F^{n-1}_{\langle \rho_0,c_0,q_0 \rangle} = 1 \oplus F^{n-1}_{\langle \rho'_0,c'_0,q'_0 \rangle} = F^{n-1}_{\langle \neg\rho'_0,\neg c'_0,q'_0 \rangle},$$

the second equality due to Theorem 1. Considering the first and last term above, we can conclude by inductive hypothesis that $c_0 = \neg c'_0$, but this implies either $q[0].comp = c_0 \neq 0$ or $q'[0].comp = c'_0 \neq 0$, contradicting req. 3 in Def. 9. Thus, we must have $c = c'$ and, for $v \in \mathbb{B}$,

$$F^{n-1}_{\langle \rho_v,c_v,q_v \rangle} = F^{n-1}_{\langle \rho'_v,c'_v,q'_v \rangle}.$$

By inductive hypothesis, $c_v = c'_v$ and $q_v = q'_v$, and either $q_v.lvl = n - 1$, in which case both $q[v]$ and $q'[v]$ are short edges, implying $\rho_v = \rho'_v = \mathtt{S}$, or $q_v.lvl < n - 1$ and $\rho_v = \rho'_v$ by inductive hypothesis. In either case, $q[v] = q'[v]$ for $v \in \mathbb{B}$; as the CESRBDD is reduced, it contains no duplicate nodes (req. 1 in Def. 9), thus $q = q'$ and the property holds.

Next, suppose $q.lvl < n$ and $q'.lvl < n$. If $\rho = \rho'$, for all cases of ρ in the definition of F, we can conclude that $F^{n-1}_{\langle \rho,c,q \rangle} = F^{n-1}_{\langle \rho',c',q' \rangle}$. Then, by inductive hypothesis, $c = c'$ and $q = q'$, and the property holds. We now show by contradiction that $\rho \neq \rho'$ is impossible, by examining all possible cases for $\rho \neq \rho'$.

1. $\rho = \mathtt{X}$: If $\rho' = \mathtt{L_t}$ or $\rho' = \mathtt{H_t}$ for some $\mathtt{t}$, we conclude that $F_e^{n-1} = F_{e'}^{n-1} = \mathtt{t}$. By inductive hypothesis, $e = e' = \langle \mathtt{X},\mathtt{t},\mathbf{0}\rangle$, implying $\rho' = \mathtt{X}$, a contradiction.
2. $\rho = \mathtt{L_t}$ for a $\mathtt{t} \in \mathbb{B}$: If $\rho' = \mathtt{L}_{\neg\mathtt{t}}$, we conclude $\mathtt{t} = \neg\mathtt{t}$, a contradiction. If $\rho' = \mathtt{H_t}$, we conclude $F_e^{n-1} = F_{e'}^{n-1} = \mathtt{t}$, and $e = e' = \langle \mathtt{X},\mathtt{t},\mathbf{0}\rangle$, implying $\rho' = \mathtt{X}$, a contradiction. If $\rho' = \mathtt{H}_{\neg\mathtt{t}}$, we conclude $F_e^{n-1} = \neg\mathtt{t}$ and $F_{e'}^{n-1} = \mathtt{t}$, and, by inductive hypothesis, $e' = \langle \mathtt{X},\mathtt{t},\mathbf{0}\rangle$, implying $\rho' = \mathtt{X}$, a contradiction.
3. $\rho = \mathtt{H_t}$ for a $\mathtt{t} \in \mathbb{B}$: If $\rho' = \mathtt{H}_{\neg\mathtt{t}}$, we conclude $\mathtt{t} = \neg\mathtt{t}$, a contradiction.

The remaining cases are symmetric.

Finally, we show that $q.lvl = n > q'.lvl$ is impossible, by contradiction. Consider the possible cases for ρ':

1. $\rho' = \mathtt{X}$: Then, we must have

$$c \oplus F_{q[0]}^{n-1} = c \oplus F_{q[1]}^{n-1} = F_{\langle \mathtt{X},c',q'\rangle}^{n-1}.$$

   If $c = 0$, we have

$$F_{q[0]}^{n-1} = F_{q[1]}^{n-1} = F_{\langle \mathtt{X},c',q'\rangle}^{n-1}$$

   and if $c = 1$, we have (from Theorem 1)

$$F_{q[0]}^{n-1} = F_{q[1]}^{n-1} = F_{\neg\langle \mathtt{X},c',q'\rangle}^{n-1} = F_{\langle \mathtt{X},\neg c',q'\rangle}^{n-1}.$$

   For $v \in \mathbb{B}$, inductive hypothesis implies $q[v].node = q'$ and $q[v].comp = \neg c'$. Then, if $q'.lvl = n-1$, we have $q[v].rule = \mathtt{S}$; otherwise, by inductive hypothesis, $q[v].rule = \mathtt{X}$. Either way, node q is redundant, contradicting req. 5 in Def. 9.
2. $\rho' = \mathtt{L_t}$: Then, we must have

$$c \oplus F_{q[1]}^{n-1} = F_{\langle \mathtt{L_t},c',q'\rangle}^{n-1} \quad \text{and} \quad c \oplus F_{q[0]}^{n-1} = \mathtt{t}.$$

   From $F_{q[0]}^{n-1} = c \oplus \mathtt{t}$ and inductive hypothesis, we have $q[0] = \langle \mathtt{X},c \oplus \mathtt{t},\mathbf{0}\rangle$. If $c = 0$, then $F_{q[1]}^{n-1} = F_{\langle \mathtt{L_t},c',q'\rangle}^{n-1}$ and, by inductive hypothesis, we have $q[1].comp = c'$, $q[1].node = q'$, and either $q'.lvl = n-1$, thus $q[1].rule = \mathtt{S}$, or $q'.lvl < n-1$, thus $q[1].rule = \mathtt{L_t}$, by inductive hypothesis. Since $q[0] = \langle \mathtt{X},\mathtt{t},\mathbf{0}\rangle$, we have that q is a low-$\mathtt{t}$ node, contradicting req. 5 in Def. 9. If $c = 1$, we have $F_{q[1]}^{n-1} = F_{\neg\langle \mathtt{L_t},c',q'\rangle}^{n-1}$ and, by inductive hypothesis, $q[1].comp = \neg c'$, $q[1].node = q'$, and either $q'.lvl = n-1$, thus $q[1].rule = \mathtt{S}$, or $q'.lvl < n-1$, thus $q[1].rule = \neg\mathtt{L_t} = \mathtt{L}_{\neg\mathtt{t}}$, by inductive hypothesis. Since $q[0] = \langle \mathtt{X},\neg\mathtt{t},\mathbf{0}\rangle$, we have that q is a low-$\neg\mathtt{t}$ node, contradicting req. 5 in Def. 9.
3. $\rho' = \mathtt{H_t}$: Then, we must have

$$c \oplus F_{q[0]}^{n-1} = F_{\langle \mathtt{H_t},c',q'\rangle}^{n-1} \quad \text{and} \quad c \oplus F_{q[1]}^{n-1} = \mathtt{t}.$$

   From $F_{q[1]}^{n-1} = c \oplus \mathtt{t}$ and inductive hypothesis, we have $q[1] = \langle \mathtt{X},c \oplus \mathtt{t},\mathbf{0}\rangle$. If $c = 0$, we have $F_{q[0]}^{n-1} = F_{\langle \mathtt{H_t},c',q'\rangle}^{n-1}$ and, by inductive hypothesis, we have $q[0].comp = c'$, $q[0].node = q'$, and either $q'.lvl = n-1$, thus $q[0].rule = \mathtt{S}$, or $q'.lvl < n-1$, thus $q[0].rule = \mathtt{H_t}$, by inductive hypothesis. Since $q[1] = \langle \mathtt{X},\mathtt{t},\mathbf{0}\rangle$, we have that q is a high-$\mathtt{t}$ node, contradicting req. 5 in Def. 9. If $c = 1$, we have $F_{q[0]}^{n-1} = F_{\neg\langle \mathtt{H_t},c',q'\rangle}^{n-1}$ and, by inductive hypothesis, $q[0].comp = \neg c'$, $q[0].node = q'$, and either $q'.lvl = n-1$, thus $q[0].rule = \mathtt{S}$, or $q'.lvl < n-1$, thus $q[0].rule = \neg\mathtt{H_t} = \mathtt{H}_{\neg\mathtt{t}}$, by inductive hypothesis. Since $q[1] = \langle \mathtt{X},\neg\mathtt{t},\mathbf{0}\rangle$, we have that q is a high-$\neg\mathtt{t}$ node, contradicting req. 5 in Def. 9. □

5 The *Apply* operation for CESRBDDs

The *Apply* operation is the main algorithm to manipulate decision diagrams [5]. Its binary form computes a boolean elementwise operation $\odot$ over the functions encoded by two decision diagrams.

Algorithm 2 shows ApplyF, for FBDDs, which recursively applies the operation to the children of the argument nodes. Due to the semantics of long FBDD edges, ApplyF benefits when both operands skip over levels. For a given BDD type T, let $\Psi_{a\odot b}^{T}$ be the number of unique recursive calls (duplicate calls cost little, thanks to memoization) issued by Apply$T(\odot, L, a, b)$, where a and b are functions $\mathbb{B}^L \to \mathbb{B}$ encoded as BDDs $T(a)$ and $T(b)$ of type T, and $|G|$ is the number of nodes in graph G (for some T, the level is not required, thus L is not passed as a parameter). Then [5], for any $\odot$,

$$\Psi_{a\odot b}^{\mathrm{FBDD}} \le |\mathrm{FBDD}(a)| \cdot |\mathrm{FBDD}(b)|.$$

Algorithm 3 shows ApplyZ, the analogous operation for ZBDDs. It works similarly, but in general it must recurse one level at a time; only some operators can use algorithm ApplyZ0, which skips levels. Specifically, $\odot$ must satisfy $0 \odot 0 = 0$, so that applying $\odot$ to long edges pointing to p_0 and p_1 returns a long edge to $p_0 \odot p_1$. When $0 \odot 0 = 1$, as for NOR, we must either use ApplyZ or negate the result of the complementary operation, i.e., compute OR (with ApplyZ0, which can skip levels) then complement the result (which introduces a node at each skipped level). Then,

$$\begin{array}{ll} \Psi_{a\odot b}^{\mathrm{ZBDD}} \le |\mathrm{ZBDD}(a)| \cdot |\mathrm{ZBDD}(b)| & \text{if } 0 \odot 0 = 0, \\ \Psi_{a\odot b}^{\mathrm{ZBDD}} \le |\mathrm{ZBDD}(a)| \cdot |\mathrm{ZBDD}(b)| \cdot L & \text{if } 0 \odot 0 \neq 0. \end{array}$$

Algorithm 4 shows ApplyE, for ESRBDDs (we omit MergeEdgeE, the ESRBDD version of MergeEdge in Algorithm 1). It requires a level n in addition to the operands ESRBDD(a) and ESRBDD(b), giving [3]

$$\Psi_{a\odot b}^{\mathrm{ESRBDD}} \le |\mathrm{ESRBDD}(a)| \cdot |\mathrm{ESRBDD}(b)| \cdot 2L.$$

In reality, this time complexity is actually competitive, since it can be shown [3] that, for any operation $\odot$,

$$\Psi_{a\odot b}^{\text{ESRBDD}} \leq 2 \cdot \min\{\Psi_{a\odot b}^{\text{FBDD}}, \Psi_{a\odot b}^{\text{ZBDD}}\}.$$

Algorithm 5 shows ApplyC, for CESRBDDs, which also requires a level n in addition to the two operand edges CESRBDD(e) and CESRBDD(f). After checking whether $n = 0$ (line 2), it determines whether the reduction rules of e and f are compatible (lines 4–6), in which case it sets the reduction rule of the result accordingly instead of the default `S`. Then, it checks whether e or f point to terminal $\mathbf{0}$, since it may then be possible, given the specific semantic of operator $\odot$ and of the involved long edges, to determine that the result is 0, 1, e, $\neg e$, f, or $\neg f$ without further recursion (lines 7–18); note that analogous checks are possible in ApplyF, ApplyZ, ApplyZ0, and ApplyE, but are omitted in their pseudocodes. If none of the above terminal cases applies, ApplyC chooses the level m at which the cache should be searched for a previously computed result, and a new node q created in case that search is unsuccessful: m remains set to n if the reduction rules of e and f were incompatible, otherwise it is changed to $\max\{e.node.lvl, f.node.lvl\}$, as the computation can then skip levels from n to $m + 1$ included (line 20). Then, it recursively calls itself on the 0-edges and the 1-edges (lines 25–26) to build node q, which it reduces (using procedure ReduceNode from Algorithm 1) to an edge g, and caches the result. The returned edge is the result of calling MergeEdge, also from Algorithm 1, on ρ and g; this is needed in case $n > m$.

To bound the number of recursive calls issued by ApplyC, consider that, when called on two edges $\langle\rho,c,p\rangle$ and $\langle\rho',c',p'\rangle$ it must recurse down one level at a time in the worst case (if ρ and ρ' are not compatible), rather than skip all the levels from n to $\max\{p.lvl, p'.lvl\}$. The number of edges in a CESRBDD is limited to twice the number of nodes and, for each pair of nodes from the two argument CESRBDDs, ApplyC may only recursively call itself on their respective 0-edges and on their respective 1-edges; also, calls to ApplyC are cached and can be reused regardless of whether they were due to 0-edge or 1-edge recursion. Thus, if we let $\mathcal{A}_{\mathtt{t}}$ and $\mathcal{B}_{\mathtt{t}}$ be the set of `t`-edges in CESRBDD(a) and CESRBDD(b), for $\mathtt{t} \in \mathbb{B}$ (including the two root edges as, e.g., 0 edges), and if we let $\lambda(e)$ be the length of edge e, we obtain

$$\Psi_{a\odot b}^{\text{CESRBDD}} \leq \sum_{(e,e')\in\mathcal{A}_0\times\mathcal{B}_0\cup\mathcal{A}_1\times\mathcal{B}_1} \max\{\lambda(e), \lambda(e')\},$$

which in turn provides the bound

$$\Psi_{a\odot b}^{\text{CESRBDD}} \leq |\text{CESRBDD}(a)| \cdot |\text{CESRBDD}(b)| \cdot 2L,$$

Algorithm 2 Apply for FBDDs.

```
 1: procedure ApplyF(op ⊙, node p, q)
 2:     if p.lvl = 0 ∧ q.lvl = 0 then return p ⊙ q;
 3:     if cached [⊙, p, q : r] then return r;
 4:     m ← Max(p.lvl, q.lvl);    r ← new node at level m;
 5:     r[0] ← ApplyF(⊙,CofacF(m,p,0),CofacF(m,q,0));
 6:     r[1] ← ApplyF(⊙,CofacF(m,p,1),CofacF(m,q,1));
 7:     r ← ReduceF(r);                          ▷ FBDD reduce
 8:     cache [⊙, p, q : r];
 9:     return r;
10: procedure CofacF(level n, node p, bool i)   ▷ n ≥ p.lvl
11:     if n = p.lvl then return p[i]; ▷ x_n = i cofactor is p[i]
12:     return p;                          ▷ else use FBDD semantic
```

Algorithm 3 Apply for ZBDDs.

```
 1: procedure ApplyZ(op ⊙, level n, node p, q)
 2:     if p.lvl = 0 ∧ q.lvl = 0 then return p ⊙ q;
 3:     if cached [⊙, n, p, q : r] then return r;
 4:     r ← new node at level n;      n' ← n − 1;
 5:     r[0] ← ApplyZ(⊙,n',CofacZ(n,p,0),CofacZ(n,q,0));
 6:     r[1] ← ApplyZ(⊙,n',CofacZ(n,p,1),CofacZ(n,q,1));
 7:     r ← ReduceZ(r);                          ▷ ZBDD reduce
 8:     cache [⊙, n, p, q : r];
 9:     return r;
10: procedure ApplyZ0(op ⊙, node p, q)
11:     if p.lvl = 0 ∧ q.lvl = 0 then return p ⊙ q;
12:     if cached [⊙, p, q : r] then return r;
13:     m ← Max(p.lvl, q.lvl);              ▷ skip down to level m
14:     r ← new node at level m;
15:     r[0] ← ApplyZ0(⊙,CofacZ(m,p,0),CofacZ(m,q,0));
16:     r[1] ← ApplyZ0(⊙,CofacZ(m,p,1),CofacZ(m,q,1));
17:     r ← ReduceZ(r);                          ▷ ZBDD reduce
18:     cache [⊙, p, q : r];
19:     return r;
20: procedure CofacZ(level n, node p, bool i)   ▷ n ≥ p.lvl
21:     if n = p.lvl then return p[i]  ▷ x_n = i cofactor is p[i]
22:     return (i) ? 0 : p;                ▷ else use ZBDD semantic
```

Algorithm 4 Apply for ESRBDDs.

```
 1: procedure ApplyE(op ⊙, level n, edge ⟨α,a⟩, ⟨β,b⟩)
 2:     if n = 0 then return ⟨S,a ⊙ b⟩;
 3:     if cached [⊙, n, ⟨α,a⟩, ⟨β,b⟩ : ⟨ρ,r⟩] then return ⟨ρ,p⟩;
 4:     r ← new node at level n;
 5:     for all i ∈ {0, 1} do
 6:         ⟨α',a'⟩ ← CofacE(n, ⟨α,a⟩, i);
 7:         ⟨β',b'⟩ ← CofacE(n, ⟨β,b⟩, i);
 8:         n' ← n − 1;      m ← Max(a'.lvl, b'.lvl);
 9:         if m < n' ∧ α' = β' ∧ (α' = X ∨ 0 ⊙ 0 = 0) then
10:             r[i] ← ApplyE(⊙, m, ⟨α',a'⟩, ⟨β',b'⟩);
11:             r[i] ← MergeEdgeE(n', α', r[i]);
12:         else          ▷ cannot skip levels n',n'−1,...,m+1
13:             r[i] ← ApplyE(⊙, n', ⟨α',a'⟩, ⟨β',b'⟩);
14:     ⟨ρ,r⟩ ← ReduceEdgeE(n,⟨S,r⟩);   ▷ ESRBDD reduce
15:     cache [⊙, n, ⟨α,a⟩, ⟨β,b⟩ : ⟨ρ,r⟩];
16:     return ⟨ρ,r⟩;
17: procedure CofacE(level n, edge ⟨ρ,p⟩, bool i)
18:     if n = p.lvl then return p[i]; ▷ x_n = i cofactor is p[i]
19:     if (i = 0 ∧ ρ = L0) ∨ (i = 1 ∧ ρ = H0) then ⟨ρ,p⟩ ← ⟨X,0⟩;
20:     if n = p.lvl + 1 then ρ ← S;
21:     return ⟨ρ,p⟩;          ▷ else use ESRBDD semantic
```

Algorithm 5 Apply algorithm for CESRBDDs.

```
1: procedure ApplyC(op ⊙, level n, edge e, edge f)   ▷ return edge g s.t. F^n_g = F^n_e ⊙ F^n_f; all edges are reduced
2:     if n = 0 then return ⟨S,e.comp ⊙ f.comp,0⟩;   ▷ apply ⊙ to the only terminal, 0, appropriately complemented
3:     ρ ← S;  m ← n;
4:     if e.rule = X ∧ f.rule = X then ρ ← X;   ▷ can avoid computation on skipped levels
5:     else if e.rule = L_t ∧ f.rule = L_u then ρ ← L_{t⊙u};   ▷ can avoid computation on skipped levels
6:     else if e.rule = H_t ∧ f.rule = H_u then ρ ← H_{t⊙u};   ▷ can avoid computation on skipped levels
7:     if e.node = 0 ∧ (e.rule = X ∨ ρ ≠ S) then
8:         if e.rule = X then ρ_f ← f.rule else ρ_f ← ρ
9:         if e.comp⊙0=0∧e.comp⊙1=0 then return ⟨X,0,0⟩;   ▷ return 0 regardless of f
10:        if e.comp⊙0=0∧e.comp⊙1=1 then return MergeEdge(n,ρ_f,f);   ▷ return f regardless of whether F^n_e=0 or 1
11:        if e.comp⊙0=1∧e.comp⊙1=0 then return ¬MergeEdge(n,ρ_f,f);   ▷ return ¬f regardless of whether F^n_e=0 or 1
12:        if e.comp⊙0=1∧e.comp⊙1=1 then return ⟨X,1,0⟩;   ▷ return 1 regardless of f
13:    if f.node = 0 ∧ (f.rule = X ∨ ρ ≠ S) then
14:        if f.rule = X then ρ_e ← e.rule else ρ_e ← ρ
15:        if 0⊙f.comp=0∧1⊙f.comp=0 then return ⟨X,0,0⟩;   ▷ return 0 regardless of e
16:        if 0⊙f.comp=0∧1⊙f.comp=1 then return MergeEdge(n,ρ_e,e);   ▷ return e regardless of whether F^n_f=0 or 1
17:        if 0⊙f.comp=1∧1⊙f.comp=0 then return ¬MergeEdge(n,ρ_e,e);   ▷ return ¬e regardless of whether F^n_f=0 or 1
18:        if 0⊙f.comp=1∧1⊙f.comp=1 then return ⟨X,1,0⟩;   ▷ return 1 regardless of e
19:    if ρ ≠ S then   ▷ we reach this test if none of the special cases allowing early return applied
20:        m ← Max(e.node.lvl, f.node.lvl);   ▷ if ρ ≠ S, we are avoiding computation on levels n,...,m+1
21:        if e.node.lvl = m then e.rule ← S;
22:        if f.node.lvl = m then f.rule ← S;
23:    if cached [⊙,m,e,f : g] then return MergeEdge(n,ρ,g);   ▷ if m < n, it equals Max(e.node.lvl, f.node.lvl)
24:    q ← new node at level m;   ▷ recursively build node q encoding ApplyC(⊙,m,e,f)
25:    q[0] ← ApplyC(⊙,m−1,CofacC(m,e,0), CofacC(m,f,0));
26:    q[1] ← ApplyC(⊙,m−1,CofacC(m,e,1), CofacC(m,f,1));
27:    g ← ReduceNode(q);
28:    cache [⊙,m,e,f : g];
29:    return MergeEdge(n,ρ,g);   ▷ merge ρ and g in case n > m, i.e., we avoided computation on levels n,...,m+1
30: procedure CofacC(level n, edge ⟨ρ,c,p⟩, i ∈ 𝔹)   ▷ return an edge e s.t. F^{n−1}_e = F^n_{⟨ρ,c,p⟩}(x_n = i)
31:    if n = p.lvl then return c = 0 ? p[i] : ¬p[i];   ▷ when p is at level n, return the i-edge, complemented if c = 1
32:    if (i=0 ∧ ρ=L_t) ∨ (i=1 ∧ ρ=H_t) then return ⟨X,t,0⟩;   ▷ p is irrelevant in this case, return the constant function t
33:    if n = p.lvl + 1 then ρ ← S;   ▷ make sure ρ = S if the edge is short
34:    return ⟨ρ,c,p⟩;
```

as $|\mathcal{A}_0\times\mathcal{B}_0\cup\mathcal{A}_1\times\mathcal{B}_1|\leq 2|\text{CESRBDD}(a)|\cdot|\text{CESRBDD}(b)|$, and the length of any edge is at most L.

Of course, the above bounds fail to reflect the fact that the presence of many long edges with different rules means that the equivalent FBDD or ZBDD would have many more nodes, just like the fact that the presence of edges with both a 0 and a 1 complement flag pointing to the same node p means that the equivalent FBDD or ZBDD would also contain a node p' encoding the complement of p. In other words, $|\text{CESRBDD}(a)|$ and $|\text{CESRBDD}(b)|$ are smaller than $|\text{FBDD}(a)|$ and $|\text{FBDD}(b)|$ or $|\text{ZBDD}(a)|$ and $|\text{ZBDD}(b)|$ in these cases.

As for all BDD variants, specific properties of $\odot$ may enable a faster Apply algorithm. The most common is *commutativity*: if $x\odot y = y\odot x$ for all x, y (for booleans, this simply requires $0\odot 1 = 1\odot 0$). FBDDs only need to cache either $r = p\odot q$ or $r = q\odot p$ (a choice usually based on the operands' identifiers, e.g., their memory address). For CESRBDDs, we can cache the result of ApplyC($\odot$,m,e,f) and avoid computing ApplyC($\odot$,m,f,e), based on some total order on edges.

6 Comparing CESRBDDs with other variants

Section 5 studied the complexity of Apply for various BDD types. Now, we compare CESRBDDs against other BDD variants in terms of both number of nodes and computational cost to encode a given function.

6.1 Graph sizes

We compare the relative size, in number of nodes, for QBDDs, FBDDs, ceFBDDs (FBDDs with complemented edges), ZBDDs, CBDDs, CZDDs, TBDDs, ESRBDDs (our earlier version of edge-specified reduction BDDs, without complemented edges and without rules $\mathtt{L_1}$ and $\mathtt{H_1}$ [4]), and CESRBDDs, when encoding the same function. Each entry in Table 1 provides a lower bound

on the worst-case relative size increase when converting from BDD type T_1 (row) to BDD type T_2 (column). More formally, it is the bound for "number of nodes a T_2 requires to encode f" divided by "number of nodes a T_1 requires to encode f" for all functions f of L boolean variables, including the terminal nodes **0** and/or **1**, or just **0** for ceFBDDs and CESRBDDs. Each superscript provides an example whose encoding achieves the bound (possibly in the limit, as L grows). Trivially, entry $[T_1, T_2]$ is always at least 1, since the constant function 0 of 0 variables requires just one node, **0**, in any BDD representation.

First, we discuss converting other BDDs into CESRBDDs, to obtain column CESRBDD in Table 1. FBDDs, ceFBDDs, or ZBDDs can be transformed into (possibly unreduced) CESRBDDs by annotating their edges. For FBDDs and ceFBDDs, we annotate uncomplemented edges with S,0 (if short) or X,0 (if long), complemented edges with S,1 or X,1. For ZBDDs, we annotate edges with S,0 or $\mathtt{H_0}$,0. The resulting CESRBDD is isomorphic to the original FBDD, ceFBDD, or ZBDD. Then, Algorithm 1 produces a reduced CESRBDD which, because of Corollary 1, cannot contain more nodes than the unreduced CESRBDD, thus no more nodes than the FBDD, ceFBDD, or ZBDD.

The conversion from a chained BDD to a (possibly unreduced) CESRBDD is shown in Fig. 6(a,b). For each chain node $x_k : x_h$ with $x_k > x_h$, create a "top node" with variable x_k, and a "bottom node" with variable x_h and incoming edge(s) only from its corresponding top node. For a CBDD, the top node is a high-zero node, with a 0-edge labeled $\mathtt{H_0}$,0 to its bottom node and a 1-edge to **0**. All top nodes and non-chained nodes have incoming edges labeled with S,0 or X,0. For a CZDD, the top node is a redundant node, with both edges labeled X,0 to its bottom node. All top nodes and non-chained nodes have incoming edges labeled with S,0 or $\mathtt{H_0}$,0. In the worst case, every node in the CBDD/CZDD is a chain node, and the resulting CESRBDD has twice as many nodes. Again, Algorithm 1 produces a reduced CESRBDD, and Corollary 1 ensures that this reduction does not increase the number of nodes.

In a TBDD, each edge can be characterized based on its tag, as either short, X only, $\mathtt{H_0}$ only, or both X and $\mathtt{H_0}$. Short edges become S,0 edges in the corresponding CESRBDD, and edges that are X or $\mathtt{H_0}$ only become X,0 or $\mathtt{H_0}$,0 edges. Edges that are both X and $\mathtt{H_0}$ require the insertion of a node at the level where the reduction rule changes, as shown in Fig. 6(c) for TBDDs where edges use FBDD reduction first, ZBDD reduction second. The worst case for the conversion occurs when *every* edge requires such a node. Then, since every TBDD node has two outgoing edges, the resulting unreduced CESRBDD will have three times as many nodes as the TBDD. Each node in this CESRBDD corresponds to either a TBDD node, in which case all of its incoming edges have rule $\mathtt{H_0}$, or a TBDD edge, in which case it has a single incoming edge with rule X. Again, Corollary 1 ensures that using Algorithm 1 to obtain a reduced CESRBDD does not increase the number of nodes. There is another possibility where Corollary 1 does not apply: if an edge in the TBDD is X only, its destination node could have incoming edges with rules $\mathtt{H_0}$ and X. During reduction, this could result in a net gain of one node, but this X-only edge would not have introduced a node along the edge to handle the rule change; thus, the worst case remains one more node per TBDD edge.

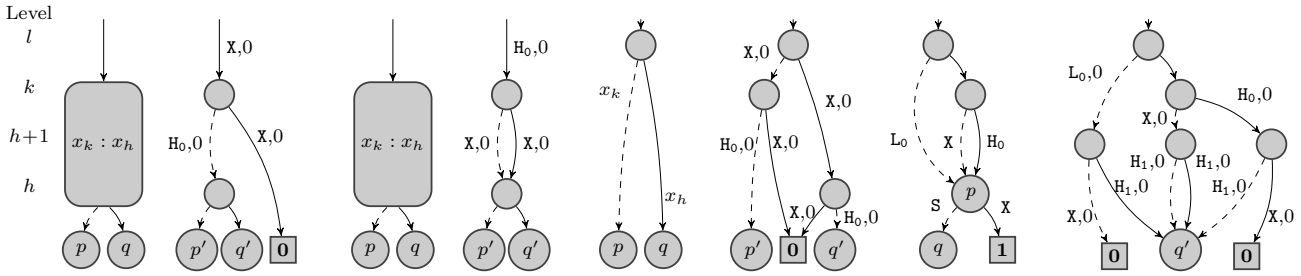
Finally, Fig. 6(d) shows the worst-case conversion from our earlier ESRBDDs into CESRBDDs. Again, we can redraw the ESRBDD as an unreduced CESRBDD by simply adding 0 for the complement flag on each edge. We cannot apply Corollary 1, because nodes could have incoming edges with a variety of rules. Consider a node p in the reduced ESRBDD corresponding to a pattern that must be eliminated in CESRBDDs but not in ESRBDDs, e.g., a high-one node. In the worst case, p has incoming edges labeled with all non-S ESRBDD rules: $\mathtt{L_0}$, $\mathtt{H_0}$, and X. Algorithm 1 eliminates node p, but creates three nodes at the level above p. However, this cannot occur *everywhere*, as nodes above p must exist as sources for p's incoming edges. If there are two nodes above p to produce these edges, three nodes in the ESRBDD are replaced by five nodes in the reduced CESRBDD. Furthermore, we can chain several instances of this pattern together using p's outgoing edge. If we repeat the pattern R times, the input ESRBDD has $3R+2$ nodes, and the resulting CESRBDD may have at most $5R+1$ nodes, for a ratio of $(5R+1)/(3R+2)$. Thus, the worst-case ratio is bounded by 5/3.

The worst-case ratios in the other direction, i.e., converting from a CESRBDD into one of the other forms, can all be realized considering a CESRBDD with a single long edge $\langle \mathtt{H_1}, 0, \mathbf{0} \rangle$, since none of the other forms can exploit the $\mathtt{H_1}$ reduction rule. QBDDs, ZBDDs, and CZDDs result in a root node at level L, and two sequences of nodes at levels $L-1$,...,1, one of high-one nodes ending in **0**, the other of redundant nodes ending in **1**, for a total of $2L+1$ nodes. FBDDs, CBDDs, and TBDDs can avoid the sequence of $L-1$ redundant nodes, thus requiring $L+2$ nodes, while ceFBDDs are also able to avoid terminal **1**, thus requiring $L+1$ nodes. ESRBDDs also require just $L+1$ nodes instead of $L+2$ nodes, but for a different reason: the node at level 1 is both a high-one and a low-zero node, thus

Table 1 Lower bound on the worst-case relative size increase (number of nodes) when converting one BDD type into another.

from ↓ to →	QBDD	FBDD	ceFBDD	ZBDD	CBDD	CZDD	TBDD	ESRBDD	CESRBDD
QBDD	—	1	1	1	1	1	1	1	1
FBDD	$L{+}1^{\dagger}$	—	1	[13] $L{+}1^{\dagger}$	[6] 1	[6] $2^{\dagger}$	[4] 1	[4] 1	1
ceFBDD	$L{+}1^{\dagger}$	[17] $2^{\star}$	—	?	?	?	?	$2^{\star}$	1
ZBDD	$2L{+}1^{\ddagger}$	[13] $L{+}2^{\ddagger}$	?	—	[6] $3^{\ddagger}$	[6] 1	[4] 1	[4] 1	1
CBDD	$L{+}1^{\dagger}$	?	?	?	—	[6] $2^{\dagger}$	?	[4] $2^{\heartsuit}$	$2^{\heartsuit}$
CZDD	$2L{+}1^{\ddagger}$	?	?	?	[6] $3^{\ddagger}$	—	?	[4] $2^{\heartsuit}$	$2^{\heartsuit}$
TBDD	$2L{+}1^{\ddagger}$	?	?	?	?	?	—	[4] $3^{\heartsuit}$	$3^{\heartsuit}$
ESRBDD	$2L{+}1^{\ddagger}$	$L{+}2^{\ddagger}$	?	$L{+}1^{\dagger}$	$L{+}2^{\S}$	$L{+}2^{\S}$	$L{+}2^{\S}$	—	$5/3^{\heartsuit}$
CESRBDD	$2L{+}1^{\natural}$	$L{+}2^{\natural}$	$L{+}1^{\natural}$	$2L{+}1^{\natural}$	$L{+}2^{\natural}$	$2L{+}1^{\natural}$	$L{+}2^{\natural}$	$L{+}1^{\natural}$	—

†: 1. ‡: $\bigwedge_{k=1}^{L} \neg x_k$. §: $\bigwedge_{k=1}^{L} x_k$. ♮: $\bigvee_{k=1}^{L} x_k$. ⋆: $x_L \wedge \bigotimes_{k=1}^{L-1} x_k \vee \neg x_L \wedge \neg \bigotimes_{k=1}^{L-1} x_k$. ♡: Fig. 6



(a) CBDD→CESRBDD (b) CZDD→CESRBDD (c) TBDD→CESRBDD (d) ESRBDD→CESRBDD

Fig. 6 Worst-case conversions from CBDD, CZDD, TBDD, and ESRBDD to CESRBDDs.

it is eliminated. This gives us the CESRBDD row in Table 1.

6.2 *Apply* comparisons

As a way to understand the (worst case) relative runtime costs of using CESRBDDs instead of FBDDs or ZBDDs, we now study the complexity of ApplyC not in terms of the size of the CESRBDDs encoding its arguments, but relative to the complexity of ApplyF and ApplyZ, when computing on the same functions.

Theorem 5 Given two functions $a, b : \mathbb{B}^L \to \mathbb{B}$,

$$\Psi_{a \odot b}^{\text{CESRBDD}} \leq (1 + M) \cdot \Psi_{a \odot b}^{\text{FBDD}}$$

where M is the fraction of pairs of nodes in recursive calls to ApplyF that, when converted into CESRBDD edges, require the creation of a node in MergeEdge.

Proof: First, since any FBDD can be viewed as a (possibly unreduced) CESRBDD where the only edge rules are S and X and the complement flags are all 0, the corresponding CESRBDD can be obtained using Algorithm 1. Calling ReduceEdge($\langle$S,0,$p\rangle$) returns either $\langle$S,c',$p'\rangle$ with $p'.lvl = p.lvl$ (if p is not a pattern node) or an edge $\langle\rho,c',p'\rangle$ with $p'.lvl < p.lvl$ and $\rho \in \{\mathtt{L_0},\mathtt{L_1},\mathtt{H_0},\mathtt{H_1}\}$. Calling ReduceEdge($\langle$X,0,$p\rangle$) returns either $\langle$X,c',$p'\rangle$ with $p'.lvl = p.lvl$ (if p is not a pattern node) or an edge $\langle$S|X,c',$p'\rangle$ where node p' is a new node at level $p.lvl + 1$ created by MergeEdge.

Now, consider each unique recursive call generated by the top-level call ApplyF$(\odot, a, b)$ to see how many corresponding recursive calls are generated by ApplyC. Given FBDD edges $\langle$S|X,0,$p\rangle$ and $\langle$S|X,0,$q\rangle$, nine possibilities exist for the corresponding CESRBDD edges:

1. p and q are not pattern nodes. The CESRBDD edges are $\langle$S|X,c'_p,$p'\rangle$ and $\langle$S|X,c'_q,$q'\rangle$ with $p'.lvl = p.lvl$ and $q'.lvl = q.lvl$. This produces one ApplyC call, corresponding to the ApplyF call.
2. p is a pattern node with a short edge, q is not a pattern node. The CESRBDD edges are $\langle\rho,c'_p,p'\rangle$ and $\langle$S|X,c'_q,$q'\rangle$ with $p'.lvl < p.lvl$ and $q'.lvl = q.lvl$. This produces one ApplyC call, corresponding to the ApplyF call.
3. p is a pattern node with a long edge, q is not a pattern node. The CESRBDD edges are $\langle$S|X,c'_p,$p'\rangle$ and $\langle$S|X,c'_q,$q'\rangle$ with $p'.lvl = 1 + p.lvl$ and $q'.lvl = q.lvl$. Node p' built by MergeEdge has outgoing edges $p'[0] = p'[1] = \langle\rho_r,c'_r,r'\rangle$ for some node r' with $r'.lvl < p.lvl$. Then, the single ApplyF call corresponds to two ApplyC calls: the "original" call for the pair of edges, which in turn issues two identical calls (thus, only one requiring computation) for the pair of edges $\langle\rho_r,c'_r,r'\rangle$, $\langle$S|X,c'_q,$q'\rangle$.

4. p is not a pattern node, q is a pattern node with a short edge. This is symmetric to case 2.
5. p and q are pattern nodes with short edges. The CESRBDD edges are $\langle \rho_p, c'_p, p' \rangle$ and $\langle \rho_q, c'_q, q' \rangle$ with $p'.lvl < p.lvl$ and $q'.lvl < q.lvl$. The ApplyF call corresponds to *at most* one ApplyC call: if rules ρ_p and ρ_q are compatible ($\rho_p = \mathtt{L_t}$ and $\rho_q = \mathtt{L_u}$ or $\rho_p = \mathtt{H_t}$ and $\rho_q = \mathtt{H_u}$) and the edges start at a higher level, there is no ApplyC call corresponding to p, q.
6. p is a pattern node with a long edge, q is a pattern node with a short edge. The CESRBDD edges are $\langle \mathtt{S|X}, c'_p, p' \rangle$ and $\langle \rho_q, c'_q, q' \rangle$ with $p'.lvl = 1 + p.lvl$ and $q'.lvl < q.lvl$. Node p' built by MergeEdge has outgoing edges $p'[0] = p'[1] = \langle \rho_r, c'_r, r' \rangle$ for some node r' with $r'.lvl < p.lvl$. The single ApplyF call corresponds to two ApplyC calls: the "original" call for the pair of edges, which in turn issues two identical calls (again, only one requiring computation) for the pair of edges $\langle \rho_r, c'_r, r' \rangle$ and $\langle \rho_q, c'_q, q' \rangle$.
7. p is not a pattern node, q is a pattern node with a long edge. This is symmetric to case 3.
8. p is a pattern node with a short edge, q is a pattern node with a long edge. This is symmetric to case 6.
9. p and q are pattern nodes with long edges. The CESRBDD edges are $\langle \mathtt{S|X}, c'_p, p' \rangle$ and $\langle \rho_q, c'_q, q' \rangle$ with $p'.lvl = 1 + p.lvl$ and $q'.lvl = 1 + q.lvl$. Nodes p' and q' built by MergeEdge have outgoing edges $p'[0] = p'[1] = \langle \rho_r, c'_r, r' \rangle$ and $q'[0] = q'[1] = \langle \rho_s, c'_s, s' \rangle$, for some nodes r' and s' with $r'.lvl < p.lvl$ and $s'.lvl < q.lvl$. The single ApplyF call corresponds to two ApplyC calls: the "original" call for the pair of edges, which in turn issues two identical calls (again, only one requiring computation) for the pair of edges $\langle \rho_r, c'_r, r' \rangle$ and $\langle \rho_s, c'_s, s' \rangle$.

Since the total number of unique calls to ApplyF is $\Psi^{\text{FBDD}}_{a \odot b}$ and since each recursive call for pair p, q corresponds to a second recursive call in ApplyC only when the conversion of p or q requires MergeEdge to build an additional node, the theorem follows. □

As we include terminal nodes in the list of pairs, M is less than one, which gives the following corollary.

Corollary 2 *Given two functions* $a, b : \mathbb{B}^L \to \mathbb{B}$,

$$\Psi^{CESRBDD}_{a \odot b} < 2 \cdot \Psi^{FBDD}_{a \odot b}.$$

The same result holds for ceFBDDs as well, if we assume that both Apply algorithms take equal advantage of properties of complement with respect to $\odot$.

Theorem 6 Given two functions $a, b : \mathbb{B}^L \to \mathbb{B}$,

$$\Psi^{\text{CESRBDD}}_{a \odot b} \leq (1 + 2M) \cdot \Psi^{\text{ZBDD}}_{a \odot b},$$

where M is the fraction of pairs of nodes in recursive calls to ApplyZ that, when converted into CESRBDD edges, require the creation of a node in MergeEdge.

Proof: (Sketch) The proof is similar to that of Theorem 5. However, in cases 3, 6, 7, and 9, the nodes built by MergeEdge will in general have *different* outgoing edges, with the high edge always equal to $\langle \mathtt{X}, 0, \mathbf{0} \rangle$ (corresponding to the ZBDD semantic of high-zero nodes at skipped levels). Thus, the original ApplyZ call could produce three ApplyC calls, although we note that one of them (with edge $\langle \mathtt{X}, 0, \mathbf{0} \rangle$) is a terminal case. □

Corollary 3 *Given two functions* $a, b : \mathbb{B}^L \to \mathbb{B}$,

$$\Psi^{CESRBDD}_{a \odot b} < 3 \cdot \Psi^{ZBDD}_{a \odot b}.$$

We now consider comparisons in the other direction. CESRBDDs can collapse chains of FBDD pattern nodes into a single edge (such as $\mathtt{H_1}$). As shown in Table 1, in the worst case, an FBDD, ceBDD, or ZBDD can have $O(L)$ times as many nodes as the CESRBDD encoding the same function. From the complexity results discussed earlier, we obtain the bounds

$$\Psi^{\text{FBDD}}_{a \odot b} \leq O(L^2) \cdot \Psi^{\text{ApplyC}}_{a \odot b},$$
$$\Psi^{\text{ZBDD}}_{a \odot b} \leq O(L^2) \cdot \Psi^{\text{ApplyC}}_{a \odot b}.$$

However, as we show below, these bounds are not tight. While a single recursive call to ApplyC with compatible long edges, such as ApplyC($\odot$,n,$\langle \mathtt{H_1}, c, p \rangle$,$\langle \mathtt{H_1}, c', p' \rangle$), corresponds to multiple recursive calls in ApplyF, those calls are always along a pair of chains of FBDD nodes, not between all possible nodes in the chains.

Theorem 7 Given two functions $a, b : \mathbb{B}^L \to \mathbb{B}$,

$$\Psi^{\text{FBDD}}_{a \odot b} \leq (1 + E) \cdot \Psi^{\text{CESRBDD}}_{a \odot b},$$

where E is the number of levels skipped by the longest non-$\mathtt{X}$ edge in the CESRBDD encodings of a and b.

Proof: Consider the worst case for an ApplyC($\odot$,n,e,f) call and focus on the value of ρ after executing line 6. If $\rho = \mathtt{L_t}$ or $\rho = \mathtt{H_t}$, the computation can jump to level $m = \text{Max}(e.node.lvl, f.node.lvl)$. Since FBDDs cannot exploit this reduction rule, the corresponding FBDDs contain a chain of low-zero or low-one nodes (if $\rho = \mathtt{L_t}$) or a chain of high-zero or high-one nodes (if $\rho = \mathtt{H_t}$). ApplyF thus requires two additional recursive calls for levels $n, ..., m + 1$, one of those calls always involving two terminal nodes. Counting only the non-trivial calls, ApplyF requires one additional recursive call for each level skipped by both e and f. The theorem follows. □

Proofs similar to that of Theorem 7 hold for ceFBDDs and ZBDDs, leading to the following bounds.

Corollary 4 *Given two functions* $a, b : \mathbb{B}^L \to \mathbb{B}$,

$$\Psi^{FBDD}_{a \odot b} \leq L \cdot \Psi^{CESRBDD}_{a \odot b},$$
$$\Psi^{ceFBDD}_{a \odot b} \leq L \cdot \Psi^{CESRBDD}_{a \odot b},$$
$$\Psi^{ZBDD}_{a \odot b} \leq L \cdot \Psi^{CESRBDD}_{a \odot b}.$$

We observe that the above bounds are tight, as they are realized, for example, when comparing the single call ApplyC($\odot, L, \langle \mathtt{H}_1,0,\mathbf{0}\rangle, \langle \mathtt{H}_0,1,\mathbf{0}\rangle$) with the L equivalent calls required for FBDDs, ceFBDDs, or ZBDDs.

7 Experimental results

We compare QBDDs (our implementation allows them to have long edges to $\mathbf{0}$), FBDDs, ceFBDDs, ZBDDs, CBDDs, CZDDs, TBDDs, ESRBDDs, and CESRBDDs with respect to the size of their encoding and the performance of their Apply operation, using three sets of benchmarks. The first two are those used in [6] and represent general textual information and digital logic functions, respectively; the third is typical in state space analysis of concurrent systems. Since [4] compares ESRBDDs to FBDDs, ZBDDs, CBDDs, CZDDs, and TBDDs, we focus our comparison on CESRBDDs, but include ceFBDDs to help assess the effect of complemented edges vs. that of our edge-specified reduction.

The experiments to compare the size of the BDD encodings first built the FBDD encoding, used *sifting* [19] to determine a *good enough* variable order, converted the FBDD to each of the other BDD types, and finally counted the number of nodes in the resulting encoding. The experiments comparing the Apply operation for different BDD types measured the time needed to build the final encoding, using the same variable order. We considered models for which the FBDD encoding could be built in an hour using at most 16 GB of RAM, on a RedHat Linux virtual machine with 16 cores (at clock speed 3.2 GHz) and 384 GB RAM.

7.1 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, we pad the shorter ones with the ASCII symbol `NULL`). We use the encoding schemes in [6]: *one-hot* and *binary*. Thus, each dictionary generates four benchmarks, one for each choice of encoding and alphabet.

We consider two dictionaries. The first contains 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 is a set of passwords from SecLists [16] (non-ASCII characters are replaced by `NULL`), containing 999,999 passwords of length between 1 and 39, with characters taken from 91 symbols including `NULL`.

Table 2 reports the number of nodes required to store each dictionary, for each encoding and alphabet choice; the best result on each row is in boldface. Except for QBDDs, FBDDs, and ceFBDDs, the one-hot encoding results in fewer nodes, demonstrating the effectiveness of the zero-suppressed reduction to encode sparse data. CESRBDDs have the fewest nodes, regardless of encoding and alphabet, and ESRBDDs are second best. For binary encodings, CESRBDDs use between 32 and 52% fewer nodes than the next best BDD type (excluding ESRBDDs); for one-hot encodings, they use between 19 and 26% fewer nodes than the next best BDD type. ZBDDs, CZDDs, TBDDs, and ESRBDDs use the same number of nodes for one-hot encodings, likely because (a) there are no redundant nodes and (b) 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 can only 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_1w' is a word iff w_2w' is also a word, for all w'. In the English word list, "Hlidhskjalf" and its alternate spelling "Hlithskjalf" is one such rare instance (no w' can continue either of them to form an additional word).

7.2 Combinational Circuits

FBDDs are often used to synthesize and verify digital circuits. We select a set of combinational circuits from the LGSynth'91 benchmarks [20] and, for each one, we build a BDD encoding all its output logic functions.

Table 2 reports the number of nodes to encode all outputs of each circuit. In contrast to the dictionaries, these benchmarks stress the ability to eliminate redundant nodes. Thus, QBDDs and ZBDDs have the worst performance while the rest perform relatively well. CESRBDDs are the best encoding followed closely by ceFBDDs, indicating that the ability to use complement edges makes a significant difference for these experiments with CESRBDDs and ceFBDDs using between 15 and 25% fewer nodes than TBDDs and ESRBDDs (the best BDD types without complemented edges).

The BDD encoding of a combinational circuit can be built *solely* using the Apply operation, thus this is a good benchmark to compare the performance of the Apply operations for different BDD types. Table 3 reports the time needed to encode all outputs of each combinational circuit using Apply. Once again, the experiments indicate that BDD types able to eliminate

Table 2 Experimental results (number of nodes) for two dictionary benchmark and one combinational circuit benchmark.

Word List	**QBDD**	**FBDD**	**ceFBDD**	**ZBDD**	**CBDD**	**CZDD**	**TBDD**	**ESR**	**CESR**
Binary Compact	1,120,437	1,120,250	965,217	657,969	971,387	657,969	657,902	484,765	**378,439**
Binary Full	1,285,501	1,285,285	1,062,292	851,555	1,153,438	851,554	851,479	520,576	**408,278**
One-hot Compact	9,739,638	9,739,638	8,283,652	311,227	656,649	311,227	311,227	311,227	**230,294**
One-hot Full	22,775,492	22,775,492	19,131,968	311,227	656,712	311,227	311,227	311,227	**230,294**
Password List	**QBDD**	**FBDD**	**ceFBDD**	**ZBDD**	**CBDD**	**CZDD**	**TBDD**	**ESR**	**CESR**
Binary Compact	5,705,516	5,704,777	4,910,044	2,960,478	4,542,925	2,960,465	2,960,209	2,399,272	**1,992,612**
Binary Full	5,649,626	5,648,670	4,872,795	3,532,847	4,960,446	3,532,816	3,532,467	2,410,589	**2,001,316**
One-hot Compact	72,858,088	72,858,088	61,201,172	1,486,430	3,055,784	1,486,430	1,486,430	1,486,430	**1,192,739**
One-hot Full	101,737,047	101,737,047	86,438,081	1,486,430	3,056,067	1,486,430	1,486,430	1,486,430	**1,192,739**
Circuit	**QBDD**	**FBDD**	**ceFBDD**	**ZBDD**	**CBDD**	**CZDD**	**TBDD**	**ESR**	**CESR**
C432	2,675	1,506	1,219	2,658	1,506	2,189	1,494	1,498	**1,162**
C499	29,483	28,769	23,978	29,316	28,769	28,749	28,610	28,428	**21,861**
C880	15,048	6,496	5,261	15,044	6,496	9,640	6,496	6,491	**4,881**
C1355	85,694	75,498	62,418	85,439	75,498	77,976	75,243	74,757	**59,698**
C1908	18,456	16,210	13,180	17,859	16,174	16,047	15,687	15,685	**12,501**
C2670	74,940	15,662	12,686	74,468	15,658	21,012	15,539	15,601	**11,872**
C3540	152,523	51,878	41,749	150,539	51,778	64,563	50,871	51,146	**40,992**
C5315	26,011	3,793	3,053	25,785	3,784	4,749	3,716	3,742	**2,998**

Table 3 Experimental results (runtime in seconds) for Apply operations on the combinational circuit benchmark.

Circuit	**QBDD**	**FBDD**	**ceFBDD**	**ZBDD**	**CBDD**	**CZDD**	**TBDD**	**ESR**	**CESR**
C432	28	11	9	29	10	21	10	10	**7**
C499	173	161	131	182	154	148	144	142	**118**
C880	98	42	19	110	38	81	41	37	**16**
C1355	1819	1210	591	1799	1167	1191	1187	1172	**552**
C1908	137	88	32	130	83	74	75	71	**30**
C2670	780	109	43	801	91	174	88	89	**39**
C3540	2712	418	**198**	2829	371	703	321	339	199
C5315	661	61	**24**	670	52	69	54	49	25

redundant nodes and use complemented edges are at a significant advantage: CESRBDDs and ceFBDDs are clearly the best. Complement edges in particular play an important role: for instance, the FBDD encoding of C1355 uses only 20% more nodes than the ceFBDD encoding, but their Apply requires 104% more time than for ceFBDDs. A similar observation holds comparing ESRBDDs and CESRBDDs. Indeed, these circuits use many complement operations; this, combined with the ability of CESRBDDs and ceFBDDs to compute the complement in constant time, likely explains the significant gains in speed. However, the performance of CESRBDDs is superior to that of ceFBDDs in all but the last two circuits, where ceFBDDs are slightly better.

7.3 Safe Petri Nets

Decision diagrams are frequently used in symbolic model checking to encode sets of states. We selected 37 *safe* Petri nets (every place contains at most one token, so can be mapped to a boolean variable) from the 2018 Model Checking Contest `https://mcc.lip6.fr/2018/`. Most of these models have parameters affecting their size and complexity, yielding $N = 103$ model instances.

Providing detailed results for all the model instances would require excessive space, so Table 4 reports results for only a subset of models. A summary of the results over all model instances is given in Table 5, using the geometric mean [10] as score for each BDD type T. Letting N be the number of model instances, $\mathcal{N}_T(n)$ the set of nodes needed to encode the state space of instance n using BDD type T, and $\mathcal{N}_{min}(n)$ the minimum $\mathcal{N}_T(n)$ over all BDD types T, the node score is

$$score(T) = \sqrt[N]{\prod_{n=1}^{N} \frac{|\mathcal{N}_T(n)|}{|\mathcal{N}_{min}(n)|}}.$$

We use an analogous formula for the runtime score. For nodes, CESRBDDs have the smallest possible score, 1, indicating that they are the smallest for each model instance. The next best by a clear margin are ESRBDDs, while TBDDs, CZDDs, and ZBDDs are in the next group, indicating that eliminating zero-suppressed nodes is crucial for these models. QBDDs, FBDDs, and ceFBDDs fared poorly by a large margin. This is likely

Table 4 Number of nodes for a subset of the safe Petri net benchmarks.

Model	QBDD	FBDD	ceFBDD	ZBDD	CBDD	CZDD	TBDD	ESR	CESR
AutoFlight-06a	520,755	520,755	473,229	207,409	356,729	207,409	207,409	178,855	**143,912**
Dekker-015	1,191,942	1,191,942	1,007,263	504,726	844,466	504,726	504,726	403,801	**287,192**
DES-01b	709,303	709,303	623,925	246,647	442,610	246,647	246,647	217,325	**159,097**
DiscoveryGPU-14a	80,865	75,682	61,302	43,016	75,571	43,016	39,689	40,953	**32,401**
DLCround-04a	19,865	19,865	17,282	15,140	19,633	15,140	15,140	6,885	**4,912**
FlexibleBarrier-12a	19,614	19,614	17,890	10,081	19,548	10,081	10,081	9,763	**6,830**
IBM319-none	16,235	16,166	13,917	826	1,913	826	826	816	**671**
IBM5964-none	31,291	31,291	28,716	2,013	4,624	2,013	2,013	2,013	**1,485**
LamportFastMutEx-4	507,897	507,897	459,275	122,487	252,361	122,487	122,487	119,111	**86,529**
MAPKbis-5320	79,037	79,037	70,236	43,568	69,057	43,568	43,568	34,813	**31,870**
NeoElection-3	414,962	414,962	371,113	15,519	34,860	15,519	15,519	15,507	**12,438**
NQueens-08	3,698,534	3,698,534	3,301,182	1,443,628	2,295,689	1,443,628	1,443,628	1,069,242	**788,054**
ParamProductionCell-4	94,752	94,752	85,332	27,158	50,588	27,158	27,158	26,631	**21,386**
Peterson-2	11,433	11,433	9,962	3,250	6,660	3,250	3,250	3,136	**2,741**
Philosophers-000010	83,216	83,216	75,415	28,257	57,349	28,257	28,257	26,974	**20,059**
Raft-03	24,502	24,472	22,063	12,280	23,040	12,280	12,260	11,635	**9,436**
Railroad-010	2,109,610	2,109,610	1,819,204	554,541	1,096,122	554,541	554,541	516,121	**408,872**
Referendum-0020	343,676	343,676	320,517	194,607	339,552	194,607	194,607	184,789	**130,503**
Allocation-R020C002	5,532,167	5,532,167	5,001,952	2,826,856	4,554,792	2,826,856	2,826,856	2,167,111	**1,735,824**
Mutex-r0020w0010	553,073	553,073	498,698	358,580	502,831	358,580	358,580	195,377	**152,693**
SafeBus-03	12,056	12,056	10,933	3,534	6,967	3,534	3,534	3,334	**3,046**
SharedMemory-000010	40,225	40,225	36,830	12,229	21,980	12,229	12,229	10,032	**7,835**
SimpleLoadBal-10	503,777	503,777	443,359	191,460	376,896	191,460	191,460	182,403	**143,649**

Table 5 Final scores for model categories.

Models	QBDD	FBDD	ceFBDD	ZBDD	CBDD	CZDD	TBDD	ESR	CESR
Dictionaries	14.052	14.050	11.866	1.604	2.771	1.604	1.604	1.314	1.000
Password Lists	14.325	14.324	12.221	1.420	2.467	1.420	1.420	1.225	1.000
Circuits (Size)	2.832	1.294	1.053	2.805	1.293	1.566	1.278	1.279	1.000
Circuits (Time)	6.704	2.197	1.099	6.891	1.989	2.871	1.936	1.885	1.006
Safe Petri Nets	4.815	4.800	4.269	1.485	2.722	1.485	1.480	1.282	1.000

because all or most of the places in these Petri nets are covered by linear invariants and, as observed in [2], this precludes the presence of redundant nodes.

Table 5 also summarizes node scores for the word dictionaries and password lists, as well as node and runtime scores for combinational circuits benchmarks, using geometric mean. Clearly, CESRBDDs are the only BDD type consistently close to the ideal score of 1.

7.4 Memory considerations: node size

So far, we have compared BDD types based on how many nodes they require. However, the actual memory consumption depends also on the size of the respective nodes. Assuming a pointer-based implementation, all BDDs store two child pointers. In addition, FBDDs and ZBDDs store a level, CBDDs and CZDDs store two levels, TBDDs store three levels, ESRBDDs store a level and two edge rules, and CESRBDDs store a level, two edge rules, and a complement bit per node.

Even with their richer set of reductions, CESRBDDs require just six more bits per node compared to FBDDs and ZBDDs: one for the 1-edge complement flag (since only 1-edges can be complemented in a reduced CESRBDD), five to encode the $5\times5=25\leq32=2^5$ possible combinations of reduction rules for the two outgoing edges (each can be `X`, $\mathtt{L}_0$, $\mathtt{H}_0$, $\mathtt{L}_1$, or $\mathtt{H}_1$; reduction rule `S` needs no explicit code, an edge is short iff it points to a node exactly one level below). ESRBDDs need four more bits per node compared to FBDDs and ZBDDs [4].

QBDD nodes are the smallest (typically requiring 64 or 128 bits, when 32–bit or 64–bit pointers are used, respectively), but are often not as compact as FBDDs or ZBDDs. Thus, Table 6 reports the *additional* cost required for each node type, compared to FBDDs and ZBDDs, when level integers are stored using 16 bits (as suggested by [6]), 20 bits (as suggested by [8]), or 32 bits. Clearly, CESRBDDs and ESRBDDs are more memory efficient than CBDDs, CZDDs, and TBDDs.

Table 6 Overhead (bits per node) w.r.t FBDDs/ZBDDs.

Lvl bits	ceFBDD	CBDD	CZDD	TBDD	ESR	CESR
16	+1	+16	+16	+32	+4	+6
20	+1	+20	+20	+40	+4	+6
32	+1	+32	+32	+64	+4	+6

8 Conclusions

ESRBDDs have been shown to be a simple and efficient generalization of previous attempts at combining reduction rules. CESRBDDs extend ESRBDDs by adding complemented edges as well as the ability to eliminate further types of nodes, specifically those corresponding to "low-one" and "high-one" patterns, while retaining the desirable property of canonicity. Like ESRBDDs, CESRBDDs are not biased towards any particular reduction rule, and thus relieve users from the burden of prioritizing reduction rules based on a specific application. Furthermore, CESRBDD nodes are more compact than those in CBDDs, CZDDs, and TBDDs.

We have given theoretical and empirical evidence of the efficiency of CESRBDDs in building and manipulating boolean functions compared to alternative BDD variants. Specifically, we have shown that CESRBDDs may be used in lieu of FBDDs, ceFBDDs, and ZBDDs without fear of a performance penalty, and without some of their drawbacks. We have also shown that CESRBDDs outperform previous efforts at integrating multiple reduction rules, specifically, CBDDs, CZDDs, TBDDs, and ESRBDDs.

Our future efforts will be directed towards improving the Apply operation by exploiting the presence of complementary reductions rules, towards adding further reduction rules (the cost of having n rules is just $\log_2 n$ bits per edge), and towards exploring a framework where users can choose just a subset of reduction rules for their application, while retaining canonicity.

References

1. Akers, S.B.: Functional testing using binary decision diagrams. In: Proc. 8th Int. Symp. on Fault-Tolerant Computing, pp. 75–82 (1978)
2. Amparore, E., Donatelli, S., Ciardo, G., Miner, A.: i_{Rank}: a variable order metric for DEDS subject to linear invariants. In: T. Vojnar, L. Zhang (eds.) Proc. TACAS, LNCS 11428, pp. 285–302. Springer, Prague, Czech Republic (2019). DOI 10.1007/978-3-030-17465-1_16
3. Babar, J.: Decision diagrams: Extensions and applications to reachability analysis. Ph.D. thesis, Iowa State University (2019)
4. Babar, J., Jiang, C., Ciardo, G., Miner, A.: Binary decision diagrams with edge-specified reductions. In: T. Vojnar, L. Zhang (eds.) Proc. TACAS, LNCS 11428, pp. 303–318. Springer, Prague, Czech Republic (2019). DOI 10.1007/978-3-030-17465-1_17
5. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comp. **35**(8), 677–691 (1986)
6. Bryant, R.E.: Chain reduction for binary and zero-suppressed decision diagrams. In: D. Beyer, M. Huisman (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 81–98. Springer, Cham (2018)
7. Denzumi, S., Kawahara, J., Tsuda, K., Arimura, H., Minato, S., Sadakane, K.: Densezdd: A compact and fast index for families of sets. Algorithms **11** (2018)
8. 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 '17, pp. 108–115. FMCAD Inc, Austin, TX (2017). URL `http://dl.acm.org/citation.cfm?id=3168451.3168478`
9. Drechsler, R., Becker, B.: Ordered Kronecker functional decision diagrams – a data structure for representation and manipulation of boolean functions. Trans. Comp.-Aided Des. Integ. Cir. Sys. **17**(10), 965–973 (2006). DOI 10.1109/43.728917. URL `https://doi.org/10.1109/43.728917`
10. 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). DOI 10.1145/5666.5673. URL `http://doi.acm.org/10.1145/5666.5673`
11. Karplus, K.: Representing boolean functions with if-then-else DAGs. Tech. rep., University of California at Santa Cruz, Santa Cruz, CA, USA (1988)
12. Kimura, S., Clarke, E.M.: A parallel algorithm for constructing binary decision diagrams. In: Proc. Int. Conf. on Computer Design (ICCD), pp. 220–223. IEEE Comp. Soc. Press (1990)
13. Knuth, D.E.: The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part I. Addison-Wesley (2011)
14. Lee, C.Y.: Representation of switching circuits by binary-decision programs. Bell Syst. Techn. J. **38**(4), 985–999 (1959)
15. Madre, J.C., Billon, J.P.: Proving circuit correctness using formal comparison between expected and extracted behaviour. In: Proc. 25th ACM/IEEE Design Automation Conference, DAC '88, pp. 205–210. IEEE Computer Society Press, Los Alamitos, CA, USA (1988)
16. Miessler, D., Haddix, J.: SecLists. `https://github.com/danielmiessler/SecLists`
17. Minato, S.: Binary Decision Diagrams and Their Applications for VLSI CAD. Ph.D. thesis, Kyoto University (1995)
18. Minato, S.: Zero-suppressed BDDs and their applications. Software Tools for Technology Transfer **3**, 156–170 (2001)
19. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: Int. Conference on CAD, pp. 139–144 (1993)
20. Yang, S.: Logic synthesis and optimization benchmarks user guide: version 3.0. Microelectronics Center of North Carolina (MCNC) (1991)