Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

A Uniform Framework for Weighted Decision Diagrams and its Implementation

Jörn Ossowski, Christel Baier

Dresden University of Technology
Faculty of Computer Science
mail@jossowski.de, baier@tcs.inf.tu-dresden.de

Received: date / Revised version: date

Abstract. [1] This papers introduces a generic framework for OBDD variants with weighted edges. It covers many boolean and multi-valued OBDD-variants that have been studied in the literature and applied for the symbolic representation and manipulation of discrete functions. Our framework supports reasoning about reducedness and canonicity of new OBDD-variants and provides a platform for the implementation and comparison of OBDD-variants. Furthermore, we introduce a new multi-valued OBDD-variant, called normalized algebraic decision diagram, which supports min/max-operations and turns out to be very useful for, e.g., integer linear programming and model checking probabilistic systems. Finally, we explain the main features of an implementation of a newly developed BDD-package, called JINC, which relies on our generic notion of weighted decision diagrams, and realizes various synthesis algorithms, reordering techniques and transformation algorithms for boolean and multi-terminal OBDDs, without or with edge-attributes, and their zero-suppressed analogons.

1 Introduction

Since the introduction of Bryant's ordered binary decision diagrams (OBDDs) as a symbolic data structure for boolean functions [7,8], many application areas of OBDDs and their variants have been discovered, ranging from the verification of combinatorial and sequential circuits, model checking reactive systems, several CAD applications, matrix-vector computations, stochastic planning, counting problems, integer programming, and genetic programming, see e.g. [30,16,32,12,31,45].

Ordinary OBDDs and their boolean variants, such as shared OBDDs with negated edges [33], functional decision diagrams [25,4] or zero-suppressed BDDs [22,32], represent boolean functions. They rely on a compactification of binary decision trees, and thus provide a natural way for representing circuits and other graph-based structures. In the same way, multi-terminal variants of OBDDs provide a simple and natural representation for discrete functions with a non-boolean range, such as multiterminal BDDs (MTBDDs) [1,10,2,14], edge-valued BDDs (EVBDDs) [44] and their factored variant (FEVBDDs) [43], binary moment diagrams (BMDs) [9], K*BMDs [39], hybrid decision diagrams (HDD) [11] or multi-valued decision diagrams (MDD) [24]. Which BDD-variant yields the most compact representation and provides the most efficient algorithms for the synthesis and manipulation, depends on the concrete application area. Even often a combination of several boolean and multi-terminal BDD variants, together with appropriate transformation algorithms, is useful. For instance, probabilistic model checking relies on graph based algorithms to examine the topological structure of a model and on real-valued matrices to represent the probability matrix [18], and therefore requires a combination of ordinary BDDs with multi-valued DD variants [26].

The variants of OBDDs can be classified according to the expansion law and type of edge-attributes they use. All OBDD-variants have in common that an efficient implementation requires an appropriate notion of reducedness and criteria that ensure canonicity. The latter is crucial for checking equality and for the space efficiency. Several other concepts that have been originally developed for ordinary OBDDs, such as Bryant's apply algorithm for the synthesis of OBDDs or algorithms for improving the variable ordering, can be adapted for OBDD-variants. However, the implementation as well as formal correctness proofs of such adaptions are often somehow tedious, as they rely on copying known concepts. The

[1] Both authors are supported by the DFG-NWO project VOSS II. The second author is supported by the DFG-projects PROBPOR and SYANCO.

purpose of our paper is to work out the common principles that yield the basis for the reducedness, canonicity, synthesis and transformation algorithms of a wide range of OBDD-variants. More precisely, the contribution of this paper consists of three parts:

(1) a uniform framework to reason about OBDDs with edge-attributes,
(2) the introduction of a new real-valued OBDD-variant, called normalized algebraic decision diagrams (NADDs), which supports reasoning about function minima and maxima,
(3) the implementation of a BDD-package, called JINC [35], that relies on our general framework and supports the function representation and manipulation with ordinary OBDDs, OBDDs with negated edges, MTBDDs, EVBDDs, FEVBDDs, NADDs, and zero-suppressed (MT)BDDs.

Our general framework will focus on OBDD-variants that are based on the *Shannon expansion* stating that $f = (\neg z \wedge f|_{z=0}) \vee (z \vee f|_{z=1})$ for each variable of a boolean function f and its algebraic analogon $f = (1-z)*f|_{z=0} + z * f|_{z=1}$ for real-valued functions f. Here, $f|_{z=0}$ means the cofactor of f obtained by freezing value 0 for variable z. The Shannon expansion is inherent in the semantics of, e.g., ordinary OBDDs, OBDDs with negated edges, MTBDDs, EVBDDs and FEVBDDs. Our general notion of *weighted decision diagrams* (WDDs) covers all these OBDD-variants with edge attributes relying on the Shannon expansion. The attributes (weights) of the edges stand for transformations that have to be applied to the cofactors represented by the target state of the edges. The concept of WDDs, together with our considerations concerning the reducedness and canonicity of WDDs, provides a uniform and elegant way to reason about edge-weighted OBDDs, supports the introduction of new OBDD-variants and yields the basis for studying the efficiency and for the design of transformation algorithms. This concept has some parallels to Höreth's word-level graph manipulation package [21]. The motivation there was to combine different OBDD variants inside one package and to support transformations between them. [21] is focused on word level decision diagrams (WLDDs) which are decomposed by moments (BMDs and variants) while the focus of our approach is based on DD-variants with the Shannon expansion. Höreth does not provide a general framework to reason about canonicity. Factored DD-variants like FEVBDDs and NADDs which turned to be very useful for our purposes are not covered by [21].

[13] also discusses a generic framework for DD-variants. The main focus of this paper is the expression of several operations for multi-valued logic functions in finite semirings by means of the ITE and CASE operators. The considered DD-variants (e.g. Galois field DDs (GFDDs), Reed-Muller-Fourier DDs (RMFDDs) and Kronecker Galois Field DDs (KGFDDs)) do not apprear to be appropriate for our applications, which rely on complex matrix operations.

Normalized algebraic decision diagrams (NADDs) are special instances of WDDs for representing real-valued functions. Like FEVBDDs, they use affine mappings as edge attributes, but require that the maximum and minimum of any inner node is 1 and 0, respectively. This property makes NADDs comparably efficient for applications that require reasoning with extrema, the comparison of functions with thresholds, or composing functions via the minimum or maximum operator. Examples for such applications are symbolic algorithms for integer linear programming [40] or model checking probabilistic systems [17,5].

The last part of this paper summarizes the main features of JINC, a new and easy to use OBDD package that turned out to be comparably efficient and even outperforms other BDD-packages such as CUDD [41,42]. The implementation of JINC follows the principals of OWDDs. Benchmarks for integer linear programs based on JINC show the advantage of NADDs in this area.

Organization. Section 2 summarizes the main concepts of ordinary BDDs and their multi-valued variants. The concept of weighted decision diagrams will be discussed in Section 3, while Section 4 introduces normalized algebraic decision diagrams. Section 5 summarizes the main features of our BDD-package JINC. We finally report on some experimental studies (Section 6) and conclude in Section 7.

Parts of this paper are based on material presented at the Calculemus Workshop 2005 [34].

2 Preliminaries

Throughout the paper, we assume the reader to be familiar with the basic concepts of binary decision diagrams, which can be found e.g. in the text books [16,32,12,31, 45]. We just summarize here the main ideas and explain our notations that will be used in the further sections.

Variables, evaluations, $\mathbb{K}$-functions. Throughout the paper, $\mathcal{Z} = \{z_1, \ldots, z_n\}$ denotes a finite set of Boolean variables. An *evaluation* of $\mathcal{Z}$ is a function $\eta : \mathcal{Z} \to \{0,1\}$ that assigns a boolean value $\eta(z) \in \{0,1\}$ to each variable $z \in \mathcal{Z}$. We write $Eval(\mathcal{Z})$ for the set of all evaluations of $\mathcal{Z}$.

Let $\overline{\xi} = (\xi_1, \ldots, \xi_k) \in \{0,1\}^k$ be a boolean vector of length k and $\overline{z} = (z_{i_1}, \ldots, z_{i_k})$ a tuple of k pairwise distinct variables $z_{i_j} \in \mathcal{Z}$. Then, $\eta\left[\overline{z} = \overline{\xi}\right]$ denotes the evaluation $\mu \in Eval(\mathcal{Z})$ with $\mu(z_{i_j}) = \xi_j$ for $j = 1, \ldots, k$, and $\mu(z) = \eta(z)$ for $z \in \mathcal{Z} \setminus \overline{z}$. (Here and in the sequel, we identify variable tuples with the corresponding set of variables.) Similarly, if $\overline{z} = (z_{i_1}, \ldots, z_{i_n})$ is a permutation of the variables in $\mathcal{Z}$ and $\overline{\xi} = (\xi_{i_1}, \ldots, \xi_{i_n})$ then

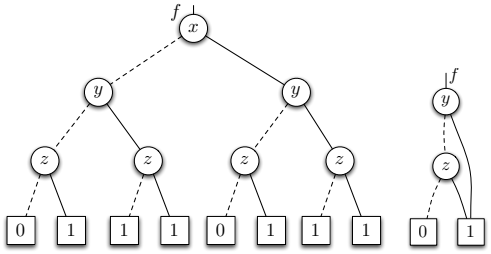


Fig. 1. Decision Tree and corresponding OBDD

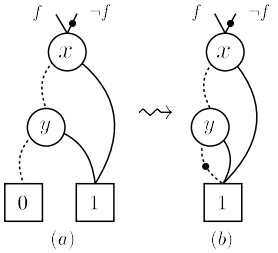


Fig. 2. The idea of negative edges

$[\overline{z} = \overline{\xi}]$ denotes the evaluation that assigns the boolean value ξ_j to variable z_{i_j} for $j = 1, \ldots, n$.

Let $\mathbb{K}$ be a nonempty set. A *$\mathbb{K}$-function* over $\mathcal{Z}$ is a function $f : \mathit{Eval}(\mathcal{Z}) \to \mathbb{K}$. The set of all $\mathbb{K}$-functions over $\mathcal{Z} = \{z_1, \ldots, z_n\}$ is denoted by $\mathbb{K}(\mathcal{Z})$ or $\mathbb{K}(z_1, \ldots, z_n)$. Given an evaluation $[\overline{z} = \overline{\xi}]$ for $\mathcal{Z}$, we often simply write $f(\overline{\xi})$ rather than $f([\overline{z} = \overline{\xi}])$.

Boolean (or switching) functions arise from $\mathbb{K} = \mathbb{B}$ where $\mathbb{B} = \{0, 1\}$. Thus, $\mathbb{B}(\mathcal{Z})$ denotes the set of all boolean functions $f : \mathit{Eval}(\mathcal{Z}) \to \{0, 1\}$. Another important special case is $\mathbb{K} = \mathbb{R}$ where $\mathbb{R}(\mathcal{Z})$ identifies the real-valued functions $f : \mathit{Eval}(\mathcal{Z}) \to \mathbb{R}$. Using appropriate binary encodings for the rows and columns, $\mathbb{R}$-functions can be used to represent real-valued matrices, graphs with real-valued edge labels, or probabilistic transition systems. For a tuple $\overline{z} = (z_{i_1}, \ldots, z_{i_k})$ of pairwise distinct variables in $\mathcal{Z}$, $\overline{\xi} = (\xi_1, \ldots, \xi_k)$ and $f \in \mathbb{K}(\mathcal{Z})$, the *cofactor* $f|_{\overline{z}=\overline{\xi}}$ is defined by $f|_{\overline{z}=\overline{\xi}}(\eta) = f(\eta[\overline{z} = \overline{\xi}])$. Variable z is called essential for f if $f|_{z=0} \neq f|_{z=1}$.

A *variable ordering* for $\mathcal{Z}$ is a permutation $\pi = (z_{i_1}, \ldots, z_{i_n})$ of the variables in $\mathcal{Z}$. Each variable ordering π induces a total order $<_\pi$ in the obvious way, i.e., $z_{i_j} <_\pi z_{i_k}$ iff $j < k$. Given $f \in \mathbb{K}(\mathcal{Z})$, the cofactors of the form $f|_{z_{i_1}=\xi_1,\ldots,z_{i_k}=\xi_k}$ where $z_{i_1}, \ldots, z_{i_k}$ are the first k variables of π are called π-consistent.

Binary Decision Diagrams. Bryant's ordered binary decision diagrams (OBDDs) [7] are data structures for boolean functions. They rely on a compactification of decision trees by collapsing subtrees representing the same function. A simple example is given in Figure 1.

Formally, an OBDD for $\mathcal{Z}$ and variable ordering $\pi = (z_{i_1}, \ldots, z_{i_n})$ for $\mathcal{Z}$ is a rooted directed acyclic graph $\mathcal{B}$ where each inner (non-terminal) node v is labeled with its variable name $\mathit{var}(v) \in \mathcal{Z}$ and has exactly two edges directed towards its two children $\mathit{succ}_0(v)$ and $\mathit{succ}_1(v)$, corresponding to the value assigned to variable $\mathit{var}(v)$. Each drain (terminal node) d is labeled with a value $\mathit{value}(d) \in \mathbb{B}$. Furthermore, it is required that on every path $v_0, \ldots, v_\ell$ from the root v_0 to a drain v_ℓ the order of the variables appearing on that path is consistent with π, i.e., $\mathit{var}(v_{i-1}) <_\pi \mathit{var}(v_i)$ for $1 \leq i < \ell$. In particular, OBDDs enjoy the read once property stating that on every path each variable appears at most once. In the pictures, we use solid lines for the 1-edges (i.e., edges from a node v to its 1-successor $\mathit{succ}_1(v)$) and a dashed line for 0-edges. Node v is called z-node if v is an inner node with $\mathit{var}(v) = z$.

The semantics of an OBDD $\mathcal{B}$ is the boolean function $f_\mathcal{B} = f_{v_0} \in \mathbb{B}(\mathcal{Z})$ associated with its root nodes v_0, where the functions f_v are defined by $f_d = \mathit{value}(d)$ for each drain d and $f_v = (\neg z \wedge f_{\mathit{succ}_0(v)}) \vee (z \wedge f_{\mathit{succ}_1(v)})$ for each z-node v. An OBDD $\mathcal{B}$ is called reduced if $f_v \neq f_w$ for all nodes v, w where $v \neq w$. Reduced π-OBDDs provide a universal data structure for boolean functions (i.e., for each $f \in \mathbb{B}(\mathcal{Z})$ there exists a reduced π-ROBDD $\mathcal{B}$ with $f = f_\mathcal{B}$) and enjoy the weak canonicity property stating that two reduced π-OBDDs for the same function f are isomorphic (i.e., equal up to the names of the nodes).

The most popular and efficient way to implement BDD-based algorithms is to use the concept of *shared OBDDs* [33] (SOBDDs), which are defined as reduced OBDDs, except that there might be more than one root node. Since SOBDD provide a canonical representations of boolean functions (in the sense that that the functions f_v, f_w of nodes v and w agree iff $v = w$), equivalence checking can be reduced to the comparison of nodes and performed in constant time. Boolean operators can be realized on SOBDDs by DFS- or BFS-based top-down traversal of the relevant subgraphs and – using appropriate cache- and hash-techniques – in time linear in the sizes of the corresponding sub-OBDDs. To support the negation operator, [33] introduced the concept of *negated edges*. The idea is simply to attach one bits per edge indicating whether the function of the target node or its negation is represented. Reducedness now means that for each pair (v, w) of nodes $v \neq w$ the functions f_v and f_w neither agree nor they are negations of each other. Figure 2 illustrates the idea of negative edges. The dot on a edge means that the function represented by the succeeding node has to be negated. Instead of changing the values bits of an incoming edge of node v can be altered to generate a representation of $\neg f_v$. The transformation of an OBDD into an equivalent reduced OBDD with negated edges is illustrated in Figure 2.

To ensure canonicity in SOBDDs with negated edges, some further restrictions are necessary. For instance, [33] requires that only 0-edges are augmented with complement bits, while the 1-edges cannot be negated.

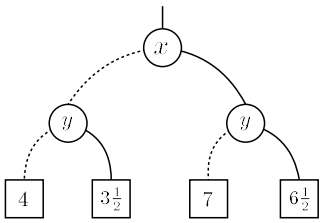


Fig. 3. MTBDD for $3x - \frac{1}{2}y + 4$

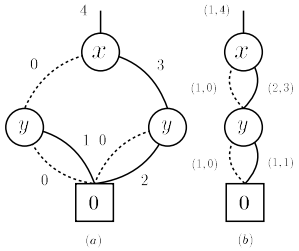


Fig. 4. (a) EVBDD and (b) FEVBDD of $3x + xy + y + 4$

Decision diagrams for multi-valued functions. To represent $\mathbb{K}$-functions where $|\mathbb{K}| \geq 3$, several variants of OBDDs have been proposed that assign values in $\mathbb{K}$ to the drains. The simplest multi-valued variant of OBDDs are multi-terminal binary decision diagrams (MTBDDs), also called algebraic decision diagrams [14, 15]. The syntax of MTBDDs is the same as for OBDDs, except that $value(d) \in \mathbb{K}$ for each drain d. Figure 3 shows the MTBDD-representation of the $\mathbb{R}$-function $3x - \frac{1}{2}y + 4$.

Variants of MTBDDs for representing real-valued functions are edge-valued binary decision diagrams (EVBDDs) [44] and their factored variant (FEVBDDs) [43]. The former attached additives weights to the edges. The meaning of an edge with the additive weight b is the transformation $f \mapsto f + b$. Thus, EVBDDs allow to collapse all cofactors that only differ by an additive constant. FEVBDDs use pairs consisting of an additive and a multiplicative weight as edge attributes. The meaning of an edge with the attribute (a, b) stands for the transformation $f \mapsto af + b$. Compared to MTBDDs, edge-valued BDDs yield the advantage that the operation $f \mapsto f + c$ for some constant c can be performed in constant time, while FEVBDD additionally support scalar multiplication. Furthermore, FEVBDDs provide a more compact representation than EVBDDs, which again are more efficient than MTBDDs. For both EVBDDs and FEVBDDs, some restrictions for the edge attributes are required to guarantee canonicity. E.g., one might require that only the 0-drain is used and that only the 1-edges are augmented with weights, while no weights are attached to the 0-edges [44,43]. For FEVBDDs, further restrictions are necessary to ensure the uniqueness of FEVBDD-representations. This will be discussed later. An example for an EVBDD and a FEVBDD is given in Figure 4.

The proofs for the soundness of such restrictions for (F)EVBDDs and other types of edge-attributes (to ensure universality and canonicity) rely on a series of cases and are somehow tedious. In the next section, we introduce a uniform framework to reason about the reducedness and canonicity of weighted decision diagrams. This frameworks covers all BDD-variants mentioned above and provides the basis to introduce new variants of multi-terminal BDDs with edge-attributes that rely on the Shannon expansion.

3 Weighted Decision Diagrams

Several techniques that have been originally proposed for OBDDs can be adapted for the attributed variants. This includes implementation techniques such as the use of unique and computed tables, algorithms to realize composition operators or for improving the variable ordering, and also mathematical reasoning about universality and canonicity. The purpose of this section is to provide a general framework for OBDD-variants which yields a uniform platform for theoretical considerations (proving universality and uniqueness, comparison of OBDD-variants, etc.) and implementation issues, such as synthesis algorithms or transformations between different OBDD-variants.

3.1 Syntax and semantics

We first present the general definition of ordered weighted decision diagrams, OWDDs for short. They rely on a fixed type of transformations that serve as weights for the edges. That is, the edges of an OWDD are augmented with bijections $\varphi : \mathbb{K} \to \mathbb{K}$. The idea is that any φ-labelled edge to (a node for) a function f stands for the function $\varphi \circ f$. To reason about reducedness and canonicity, certain restrictions are needed for the types of bijections that are used:

Notation 1 (Transformation-types, the sets Φ and Φ^c) *In the sequel, let $\mathcal{Z}$ be a finite set of boolean variables, $\mathbb{K}$ a set with at least two elements[2]. A transformation-type for K is a pair $\mathcal{T} = (\Phi, \Phi^c)$ consisting of a nonempty set Φ consisting of bijections $\mathbb{K} \to \mathbb{K}$ such that*

(1) Φ is closed under inversion and composition, i.e., if $\varphi, \psi \in \Phi$ then $\varphi^{-1} \in \Phi$ and $\varphi \circ \psi \in \Phi$. (In particular, Φ contains the identity id.)

(2) If $f \in \mathbb{K}(\mathcal{Z})$ is non-constant, $\varphi \in \Phi$ and $f = \varphi \circ f$ then $\varphi = id$.

[2] The requirement $\mathbb{K}$ to be a semi-ring as in [2] could be added, but it is irrelevant as long as we do not speak about operations on $\mathbb{K}$.

and a nonempty subset Φ^c of Φ such that condition (1) holds for Φ^c instead of Φ and

(2') if $f \in \mathbb{K}(\mathcal{Z})$ is constant and $\varphi \in \Phi^c$ such that $f = \varphi \circ f$ then $\varphi = id$.

□

Conditions (1) and (2) are important for the uniqueness of the function representation as we have:

Lemma 1. *If $\varphi, \psi \in \Phi$, $f \in \mathbb{K}(\mathcal{Z})$ is non-constant and $\varphi \circ f = \psi \circ f$ then $\varphi = \psi$.*

Proof. $\varphi \circ f = \psi \circ f$ yields $f = (\varphi^{-1} \circ \psi) \circ f$ and $\varphi^{-1} \circ \psi \in \Phi$ by (1). Hence, $\varphi^{-1} \circ \psi = id$ by (2). □

In some BDD-variants with weighted edges, the constant functions (represented by the incoming edges of the drains) require a special treatment as there might be several possibilities to transform a constant $c \in \mathbb{K}$ into another constant $c' \in \mathbb{K}$ via the bijections $\varphi \in \Phi$. This motivates the use of $\Phi^c \subseteq \Phi$. In fact, the analog statement of Lemma 1 holds for $\varphi, \psi \in \Phi^c$ and constant functions.

Definition 1 (Ordered Weighted decision diagram (OWDD)). Let $\mathcal{Z}, \mathbb{K}$ and $\mathcal{T} = (\Phi, \Phi^c)$ be as above and π a variable ordering for $\mathcal{Z}$. A π-OWDD for $(\mathcal{Z}, \mathbb{K}, \mathcal{T})$ is a rooted, binary branching, acyclic graph $\mathcal{B}$ with several additional information. For the inner nodes, we have

- a function var that assigns a variable $var(v) \in \mathcal{Z}$ to any inner node v,
- functions $v \mapsto succ_0(v)$ and $v \mapsto succ_1(v)$ that specify the successors of v,
- functions $v \mapsto \phi_0(v)$ and $v \mapsto \phi_1(v)$ that specify the transformations associated with the outgoing edges from v.

For the terminal nodes, we have a function $v \mapsto value(v) \in \mathbb{K}$. If v is an inner node and $\xi \in \{0,1\}$ such that $succ_\xi(v)$ is an inner node then we require that $var(v)$ occurs in π before $var(succ_\xi(v))$ and $\phi_\xi(v) \in \Phi$. If $succ_\xi(v)$ is a terminal node then we require $\phi_\xi(v) \in \Phi^c$.

The root of $\mathcal{B}$ is a pair $r = \langle \phi_r, v_r \rangle$ consisting of a function $\phi_r \in \Phi \cup \Phi^c$ and a node v_r from which all other nodes in $\mathcal{B}$ are reachable. As for the edges we require $\phi_r \in \Phi$ if v_r is an inner node and $\phi_r \in \Phi^c$ if v_r is terminal. (In this case, v_r is the only node in $\mathcal{B}$.) □

The semantics of a OWDD $\mathcal{B}$ is formalized by associating a function $f_v \in \mathbb{K}(\mathcal{Z})$ to any node in $\mathcal{B}$ and a function for the root r. Intuitively, an incoming edge of node v labelled with φ stands for the function $\varphi \circ f_v$.

Notation 2 (Semantics of OWDDs) Let $\mathcal{Z}, \mathbb{K}, \mathcal{T} = (\Phi, \Phi^c)$, π be as above and $\mathcal{B}$ an π-OWDD for $(\mathcal{Z}, \mathbb{K}, \mathcal{T})$. The function $f_\mathcal{B}$ for $\mathcal{B}$ is the function induced by its root $r = \langle \phi_r, v_r \rangle$ which is given by $f_\mathcal{B} = \phi_r \circ f_{v_r}$ where the function f_v for the nodes is defined in a bottom-up fashion. (The level of a z-node denotes the position of variable z in π. The nodes on the bottom-level are the terminal nodes.)

The terminal nodes (drains) represent constant functions as expected, i.e., $f_v = value(v)$ for any terminal node v. If v is a z-node then $f_v : Eval(\mathcal{Z}) \to \mathbb{K}$ is defined as follows. Let $\eta \in Eval(\mathcal{Z})$ and $\eta(z) = \xi \in \{0,1\}$ then $f_v(\eta) = \phi_\xi(v) \circ f_{succ_\xi(v)}(\eta)$. Thus, f_v is given by the function

$$f_v = ITE(z, \phi_1(v) \circ f_{succ_1(v)}, \phi_0(v) \circ f_{succ_0(v)})$$

where $ITE : \mathcal{B}(\mathcal{Z}) \times \mathbb{K}(\mathcal{Z}) \times \mathbb{K}(\mathcal{Z}) \to \mathbb{K}(\mathcal{Z})$ is the If-Then-Else-operator which is defined as follows. Let $\beta \in \mathcal{B}(\mathcal{Z})$ be a boolean function, $f_1, f_2 \in \mathbb{K}(\mathcal{Z})$ two $\mathbb{K}$-functions and $\eta \in Eval(\mathcal{Z})$ an evaluation. Then,

$$ITE(\beta, f_1, f_2)(\eta) = \begin{cases} f_1(\eta) : \text{if } \beta(\eta) = 1, \\ f_2(\eta) : \text{if } \beta(\eta) = 0. \end{cases}$$

□

That is, if $\mathbb{K} = \{0,1\}$ then we may write f_v as

$$f_v \;=\; (\neg z \wedge \phi_0(v) \circ f_{succ_0(v)}) \;\vee\; (z \wedge \phi_1(v) \circ f_{succ_1(v)})$$

and if $\mathbb{K} = \mathbb{R}$

$$f_v \;=\; (1-z) \cdot (\phi_0(v) \circ f_{succ_0(v)}) \;+\; z \cdot (\phi_1(v) \circ f_{succ_1(v)}).$$

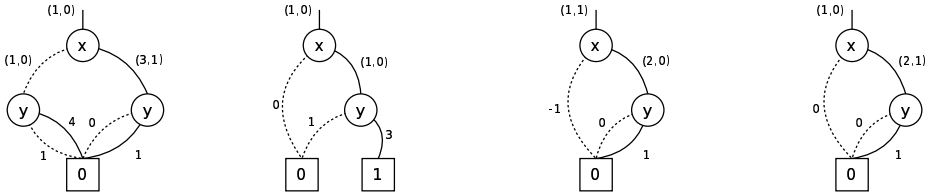
Before discussing the reducedness and canonicity for OWDDs, we observe that the notion of OWDDs covers several types of known BDD-variants.

For $\mathbb{K} = \{0,1\}$ our notion of OWDDs specializes to ordinary OBDDs [7], when dealing with the trivial transformation-type $\mathcal{T}_{\text{ID}} = (\Phi, \Phi^c)$ where $\Phi = \Phi^c = \{id\}$. Ordered BDDs with negated edges [33] are obtained by the transformation-type $\mathcal{T}_{\text{NEG}} = (\Phi, \Phi^c)$ where $\Phi = \Phi^c = \{id, \neg\}$.

For $\mathbb{K} = \mathbb{R}$ (or $\mathbb{K} = \mathbb{N}$ or any other semi-ring) MTBDDs [2,14] are obtained through $\mathcal{T}_{\text{ID}} = (\{id\}, \{id\})$, while edge-valued BDDs [44] arise from the transformation-type $\mathcal{T}_{\text{EV}} = (\Phi, \Phi^c)$ where $\Phi = \Phi^c = \{x \mapsto x + b : b \in \mathbb{K}\}$.

Factored edge-valued BDDs (FEVBDDs) arise as special instances of OWDDs by dealing with the affine transformations $x \mapsto ax + b$ where $a, b \in \mathbb{R}$ and $a \neq 0$. (Note that the requirement $a \neq 0$ is necessary, since the elements of Φ must be bijections.) That is, we deal with $\mathcal{T}_{\text{FEV}} = (\Phi, \Phi^c)$ where $\Phi = \{x \mapsto ax + b : a, b \in \mathbb{K}, a \neq 0\}$. Here, the incoming edges of the drains require a special treatment since given two constant functions c and c', there is no unique pair (a, b) such that $c' = ac + b$. For this reason, in $\mathcal{T}_{\text{FEV}}$, the transformation-set Φ^c for the constant functions is the set of functions $x \mapsto x + b$ where $b \in \mathbb{K}$.

Fig. 5 shows four FEVBDDs where we simply write (a, b) to denote the function $x \mapsto ax + b$. The OWDD in (1) represents the function $3y + 1$, while the OWDDs in (2.a), (2.b) and (2.c) represent the function $f = x(1 + 2y)$. The following example shows how the function f



(1) (2.a) (2.b) (2.c)

Fig. 5. Example for OWDDs with multiplicative and additive edge-weights

represented by the OWDD in Figure (2.a) can be calculated. The functions f_x and f_y for the x- and y-node are obtained by:

- $f_y = (1-y)\cdot(1+0) + y\cdot(3+1) = 3y+1$
- $f_x = (1-x)\cdot(0+0) + x\cdot(1\cdot f_y + 0) = x\cdot(3y+1)$

This yields $f = 1\cdot f_x + 0 = x\cdot(3y+1)$.

Remark 1. The semantics of OWDDs relies on the Shannon expansion. To treat other composition rules (e.g., the ones that are inherent in the meaning of zero-suppressed BDDs [32] or binary moment diagrams [9]), our framework has to be generalized by requiring that beside the transformation-set Φ we are given an appropriate composition operator Ω_π (which might depend on the variable ordering π) such that the function represented by a z-node v is given by

$$f_v = \Omega_\pi(z, \varphi_0(v)\circ f_{succ_0(v)}, \varphi_1(v)\circ f_{succ_1(v)}).$$

We did not use this (more general) approach, since the Shannon expansion is the simplest and most intuitive decomposition rule. However, the following techniques for OWDDs could also be adopted for the general approach. □

3.2 Reducedness

Since OWDDs rely on the Shannon expansion and since each $\mathbb{K}$-function f can be decomposed into $f = ITE(z, f|_{z=1}, f|_{z=0})$, it is clear that π-OWDDs provide a universal data structure for $\mathbb{K}$-functions. We simply may represent any cofactor $f|_{z_{i_1}=\xi_1,\ldots,z_{i_k}=\xi_k}$, that is consistent with the given variable ordering $\pi = (z_{i_1},\ldots,z_{i_n})$, by an OWDD-node and attach the trivial weight $\phi = id$ to each edge. (Recall that id belongs to Φ and Φ^c.) However, the so-obtained OWDDs contain many redundancies and cannot ensure a canonical representation. We now introduce the notion of reducedness of OWDDs which roughly means lack of redundancies, and then discuss how canonicity can be ensured.

Notation 3 (The equivalence $\equiv_\mathcal{T}$) Let $\mathcal{T} = (\Phi, \Phi^c)$ be a transformation-type (as in Notation 1). The equivalence $\equiv_\mathcal{T}$, or briefly $\equiv$, is the finest binary relation on $\mathbb{K}(\mathcal{Z})$ satisfying the following rules:

- If $f, g \in \mathbb{K}(\mathcal{Z})$ such that f is non-constant then $f \equiv_\mathcal{T} g$ if there exists $\varphi \in \Phi$ with $f = \varphi \circ g$.
- If $f, g \in \mathbb{K}(\mathcal{Z})$ such that f is constant then $f \equiv_\mathcal{T} g$ if there exists $\varphi \in \Phi^c$ with $f = \varphi \circ g$.

□

Note that $\equiv$ is in fact an equivalence. First, the symmetry of $\equiv$ is obtained by the observation that $f = \varphi\circ g$ for some $\varphi \in \Phi$ implies (i) $g = \varphi^{-1}\circ f$ and $\varphi^{-1} \in \Phi$ and (ii) g is constant iff so is f. In particular, $f \not\equiv g$ if f is constant while g is not. Furthermore, $f \equiv f$ (as $f = id \circ f$ and $id \in \Phi^c \subseteq \Phi$) and if $f \equiv g$, $g \equiv h$ then $f \equiv h$, as $f = \varphi\circ g$ and $g = \psi \circ h$ implies $f = (\varphi\circ\psi)\circ h$.

Moreover, we have $f \equiv g$ iff there there exists $\varphi, \psi \in \Phi$ ($\varphi, \psi \in \Phi^c$ if f and g are constant) such that $\varphi \circ f = \psi \circ g$, since then $f = (\varphi^{-1}\circ\psi)\circ g \equiv g$.

In the sequel, capitol letters $F, G, \ldots$ will be used for the equivalence classes of $\mathbb{K}(\mathcal{Z})$ under $\equiv$.

Definition 2 (Reduced OWDDs). A π-OWDD $\mathcal{B}$ is called reduced iff for all nodes v, w in $\mathcal{B}$ we have $v \neq w$ implies $f_v \not\equiv f_w$. □

Example 1. The two π-OWDDs with transformation type $\mathcal{T}_{\text{FEV}}$ shown in Fig. 5 (2.b) and (2.c) are reduced, while the ones in Fig. 5 (1) and (2.a) are not. In (2.a), the two sinks represent (constant) functions that can be transformed to each other via bijections in Φ^c. In (1), the two y-nodes represent (non-constant) functions that can be transformed to each other via bijections in Φ. □

With the definition of reducedness on OWDDs there is still freedom to choose the representatives in the $\equiv$-equivalence classes. Thus, reduced π-OWDDs for the same function need not to be isomorphic. (We use the notion "isomorphism" for π-OWDDs $\mathcal{B}$ and $\mathcal{C}$ in the sense that $\mathcal{B}$ and $\mathcal{C}$ agree up to renaming of the nodes.) Instead, as we will see in Theorem 4, reduced π-OWDDs are *weakly isomorphic* by which we mean that they agree when abstracting away from the names of the nodes and ignoring the weights for the edges and the root. In particular, weakly isomorphic OWDDs have the same size (number of nodes).

Lemma 2. *Let $f, g \in \mathbb{K}(\mathcal{Z})$ and $f \equiv g$. Then, f and g have the same essential variables and for all $z \in \mathcal{Z}$ we have $f|_{z=0} \equiv g|_{z=0}$ and $f|_{z=1} \equiv g|_{z=1}$.*

Proof. Let $\varphi \in \Phi$ (or $\varphi \in \Phi^c$, respectively) such that $f = \varphi \circ g$.

$$\begin{aligned} f|_{z=\xi}(\eta) &= f(\eta[z := \xi]) \\ &= \varphi(g(\eta[z := \xi])) \\ &= \varphi(g|_{z=\xi}(\eta)) \\ &= (\varphi \circ g|_{z=\xi})(\eta) \end{aligned}$$

Therefore $f|_{z=\xi} = \varphi \circ g|_{z=\xi} \equiv g|_{z=\xi}$. □

Lemma 2 yields that cofactors can be built for $\equiv$-equivalence classes. That is, if $F \subseteq \mathbb{K}(\mathcal{Z})$ and $\xi \in \{0,1\}$ then we may write $F|_{z=\xi}$ to denote the unique $\equiv$-equivalence class that contains the functions $f|_{z=\xi}$ for all $f \in F$. A similar notation $F|_{z_1=\xi_1,\ldots,z_k=\xi_k}$ is used if we consider cofactors for several variables.

Theorem 4 (Weak canonicity of reduced OWDDs). *Let π be a variable ordering and $\mathcal{B}, \mathcal{C}$ reduced π-OWDDs. Then: If $f_\mathcal{B} \equiv f_\mathcal{C}$ then $\mathcal{B}$ and $\mathcal{C}$ are weakly isomorphic. In particular, $|\mathcal{B}| = |\mathcal{C}|$.*

Proof. Let $\pi = (z_1, \ldots, z_n)$ be the variable ordering. As $f_\mathcal{B} \equiv f_\mathcal{C}$ we have $[f_\mathcal{B}]_\equiv = [f_\mathcal{C}]_\equiv$. Let $F_\mathcal{B} = [f_\mathcal{B}]_\equiv$. The proof establishes a 1-1-relationship between the cofactors[3] $F_\mathcal{B}|_{z_1=\xi_1,\ldots,z_k=\xi_k}$ and the nodes in $\mathcal{B}$ and $\mathcal{C}$ respectively. The equivalence of $f_\mathcal{B}$ and $f_\mathcal{C}$ together with Lemma 2 yields that we can just argument with $\mathcal{B}$ and get a result for all equivalent π-OWDDs.

We first observe that for each equivalence class $F = F_\mathcal{B}|_{z_1=\xi_1,\ldots,z_k=\xi_k}$ there is a node v in $\mathcal{B}$ such that f_v belongs to F. (such a node v is obtained by traversing $\mathcal{B}$ according to the assignment $[z_1 = \xi_1, \ldots, z_k = \xi_k]$), and vice versa, the function f_v of each node v belongs to $F_\mathcal{B}|_{z_1=\xi_1,\ldots,z_k=\xi_k}$ if v is reachable from the root node of $\mathcal{B}$ via the assignment $[z_1 = \xi_1, \ldots, z_k = \xi_k]$. As $\mathcal{B}$ is reduced, there is a bisjection between the nodes in $\mathcal{B}$ and the cofactors $F_\mathcal{B}|_{z_1=\xi_1,\ldots,z_k=\xi_k}$.

It remains to show that the edge-relation of $\mathcal{B}$ just depends on the equivalence classes of these cofactors. We prove this by induction on the number of levels. For the bottom level, $f_\mathcal{B}$ is constant and $\mathcal{B}$ consists of a root $r = \langle \phi, v \rangle$ where v is terminal node, representing a function in $[f_\mathcal{B}]_\equiv$ (as we have $f_\mathcal{B} = \phi \circ f_v$ and $\phi \in \Phi^c$).

In the step of induction, we consider v is an inner node in $\mathcal{B}$ such that $[f_v]_\equiv = F_\mathcal{B}|_{z_1=\xi_1,\ldots,z_k=\xi_k}$ then v is labelled with a variable z_ℓ where $\ell > k$ and z_ℓ is the first essential variable for f_v (and all functions in $[f_v]_\equiv$). Moreover, the function f_{v_0} for the 0-successor v_0 of v is in the equivalence class

$$[f_v]_\equiv\big|_{z_\ell=0} = F_\mathcal{B}|_{z_1=\xi_1,\ldots,z_k=\xi_k,\ldots,z_\ell=0}$$

for arbitrary assignments of the variables $z_{k+1}, \ldots, z_{\ell-1}$. A similar condition holds for the 1-successor of v. Hence, if we ignore the edge-weights then all reduced π-OWDDs for the same function have the same structure. □

[3] Note that some of these cofactors might agree.

Theorem 4 yields that for fixed transformation-type $\mathcal{T} = (\Phi, \Phi^c)$, variable ordering π and $\mathbb{K}$-function f, all reduced π-OWDDs for f have the same structure, and thus the same number of nodes. but possibly different weights on the edges. This observation yields a simple reduction algorithm that traverses the given (possibly not reduced) OWDD $\mathcal{B}$ in bottom-up manner and collapses redundant nodes on each level. We start with the level of the drains, and then successively treat the level of all z_{i_n}-nodes, then the level of all $z_{i_{n-1}}$-nodes, etc., where $\pi = (z_{i_1}, \ldots, z_{i_n})$. On each level, we work in two phases. Phase 1 realizes the so-called *elimination-rule* which collapses all drains with the same value (for the bottom level) and removes all inner nodes v such that

$$\langle \phi_0(v), succ_0(v) \rangle = \langle \phi_1(v), succ_1(v) \rangle$$

(for all other levels). More precisely, eliminating node v where $\langle \phi_0(v), succ_0(v) \rangle = \langle \phi_1(v), succ_1(v) \rangle$ means that we replace any edge $w \to v = succ_\xi(w)$ with $w \to succ_1(v)$ and assign the weight $\phi_\xi(w) \circ \phi_1(v)$ to this new edge. Phase 2 realizes the weighted analogon to the so-called *isomorphism rule*. It combines all nodes of the current level representing equivalent functions into a single node. For this, we select nodes $v_1, \ldots, v_k$ of the current level such that the induced functions f_{v_i} are pairwise not equivalent w.r.t. $\equiv$ and such that the function f_v of each other node of the current level is equivalent to one of the functions f_{v_i}. That is, $f_v = \varphi \circ f_{v_i}$ for some $\varphi \in \Phi$. (For the bottom level, we require $\varphi \in \Phi^c$.) We then redirect any edge $w \to v = succ_\xi(w)$ to $w \to v_i$ and redefine its weight by $succ_\xi(w) \circ \varphi$.

A further simple observation concerns the connection between OWDDs of different transformation-types. Let $\mathcal{T}_1 = (\Phi_1, \Phi_1^c)$ and $\mathcal{T}_2 = (\Phi_2, \Phi_2^c)$ be two transformation-types for the same set $\mathbb{K}$ such that $\Phi_1 \subseteq \Phi_2$ and $\Phi_1^c \subseteq \Phi_2^c$ and assume we are given a fixed variable ordering π and function $f \in \mathbb{K}(\mathcal{Z})$ that is represented by reduced π-OWDDs $\mathcal{B}_1$ over $(\mathbb{K}, \mathcal{Z}, \mathcal{T}_1)$ and $\mathcal{B}_2$ over $(\mathbb{K}, \mathcal{Z}, \mathcal{T}_2)$. Since $\equiv_{\mathcal{T}_1}$ is finer than $\equiv_{\mathcal{T}_2}$ and since there is a one-to-one-correspondence between the nodes in $\mathcal{B}_i$ and the $\equiv_{\mathcal{T}_i}$-equivalence classes of the π-consistent cofactors of f, we have $|\mathcal{B}_1| \geq |\mathcal{B}_2|$. A reduced π-OWDD for f over $(\mathbb{K}, \mathcal{Z}, \mathcal{T}_2)$ is obtained from $\mathcal{B}_1$ by applying the above reduction algorithm. Vice versa, a reduced π-OWDD for f over $(\mathbb{K}, \mathcal{Z}, \mathcal{T}_1)$ is obtained from $\mathcal{B}_2$ by traversing $\mathcal{B}_2$ levelwise in a bottom up manner. For the treatment of node v on the current level, we regard all incoming edges $w \to v = succ_\xi(w)$ and check whether $\phi_\xi(w) \circ f_v \equiv_{\mathcal{T}_1} f_v$. If not then we either generate a new or reuse an existing node u on v's level such that $\phi_\xi(w) \circ f_v \equiv_{\mathcal{T}_1} f_u$ and replace the edge $w \to v$ by $w \to u$. The weight of this new edge is the unique function $\varphi \in \Phi_1$ (resp. Φ_1^c if v and u are drains) such that $\phi_\xi(w) \circ f_v = \varphi \circ f_u$.

3.3 Canonicity

Although reduced OWDDs for the same $\mathbb{K}$-function, transformation-type and variable ordering have the same topological structure, they cannot ensure unique representations. To guarantee canonical representations, which is an essential aspect for all symbolic computations with BDD-variants, we have to choose unique representatives out of all equivalence classes w.r.t. $\equiv$. We formalize this by means of a *selection function* $\mathcal{S} : \mathbb{K}(\mathcal{Z})/\equiv \to \mathbb{K}(\mathcal{Z})$. For $f \in \mathbb{K}(\mathcal{Z})$, we simply write $\mathcal{S}(f)$ rather than $\mathcal{S}([f]_\equiv)$. Thus, we consider $\mathcal{S}$ as a function $\mathcal{S} : \mathbb{K}(\mathcal{Z}) \to \mathbb{K}(\mathcal{Z})$ such that $f \equiv \mathcal{S}(f)$ for all $f \in \mathbb{K}(\mathcal{Z})$.

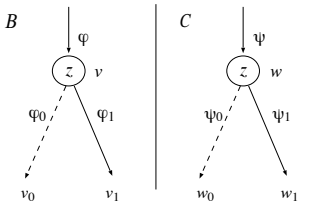
Definition 3 ($\mathcal{S}$-reduced OWDDs). Let $\mathcal{S}$ be a selection rule. A π-OWDD $\mathcal{B}$ is called $\mathcal{S}$-reduced if $\mathcal{B}$ is reduced and $f_v = \mathcal{S}(f_v)$ for all nodes v in $\mathcal{B}$. □

Theorem 5 (Canonicity of $\mathcal{S}$-reduced OWDDs). *Let $\mathcal{S}$ a selection function and let $\mathcal{B}$, $\mathcal{C}$ be $\mathcal{S}$-reduced π-OWDDs with $f_\mathcal{B} = f_\mathcal{C}$. Then, $\mathcal{B}$ and $\mathcal{C}$ are isomorphic.*

Proof. Our argument is by induction on $n = |\mathcal{Z}|$ (number of variables).

If $n = 0$ then $f_\mathcal{B} = f_\mathcal{C}$ is constant. Let c be the value of $f_\mathcal{B} = f_\mathcal{C}$ and $c' = \mathcal{S}(c)$. Then, the root of $\mathcal{B}$ and $\mathcal{C}$ consists of a terminal node labelled with c' together with the unique transformation $\varphi \in \Phi^c$ such that $\varphi(c') = c$. (The uniqueness of φ follows from the conditions for Φ^c.)

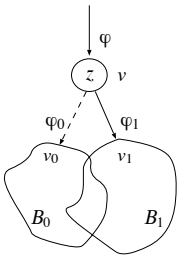
In the induction step, we assume that the root nodes of $\mathcal{B}$ and $\mathcal{C}$ are inner nodes, say z-nodes. Then, z is the first essential variable in $f_\mathcal{B} = f_\mathcal{C}$ according to the ordering π. Let $r_\mathcal{B} = \langle \varphi, v \rangle$ be the root of $\mathcal{B}$ and $r_\mathcal{C} = \langle \psi, w \rangle$ the root of $\mathcal{C}$.



Then, $\varphi \circ f_v = f_\mathcal{B} = f_\mathcal{C} = \psi \circ f_w$. Thus, $f_v \equiv f_w$. As $\mathcal{B}$ and $\mathcal{C}$ are $\mathcal{S}$-reduced we have $f_v = f_w$ and $\varphi = \psi$ (by condition (2) of Φ, cf. Notation 1). For $\xi \in \{0,1\}$, let $v_\xi = succ_\xi(v)$, $w_\xi = succ_\xi(w)$, $\phi_\xi(v) = \varphi_\xi$, $\phi_\xi(w) = \psi_\xi$. Then, $\varphi_\xi \circ f_{v_\xi} = f_v|_{z=\xi} = f_w|_{z=\xi} = \psi_\xi \circ f_{w_\xi}$. Hence, $f_{v_\xi} \equiv f_{w_\xi}$. Again, as $\mathcal{B}$ and $\mathcal{C}$ are $\mathcal{S}$-reduced, we get $f_{v_\xi} = f_{w_\xi}$ and $\varphi_\xi = \psi_\xi$ (by the conditions for Φ and Φ^c). By induction hypothesis, the sub-OWDD with root nodes v_ξ and w_ξ are isomorphic. Hence, $\mathcal{B}$ and $\mathcal{C}$ are isomorphic. □

Theorem 6 (Universality and uniqueness). *Let π be a variable ordering, $\mathcal{S}$ a selection function and $f : Eval(\mathcal{Z}) \to \mathbb{K}$. Then, there exists a unique $\mathcal{S}$-reduced π-OWDDs $\mathcal{B}$ with $f = f_\mathcal{B}$. (Uniqueness is up to isomorphism.)*

Proof. It remains to provide the proof for the existence of a $\mathcal{S}$-reduced π-OWDD for f. The construction is by induction on the number n of essential variables of f. If $n = 0$ then f is constant. Let $c = \mathcal{S}(f)$. There exists a $\varphi \in \Phi^c$ with $\varphi(c) = f$. Thus, we may use a OWDD consisting of the root $\langle \varphi, v \rangle$ where v is a terminal node v labelled with c. In the induction step, we assume that f is not constant.



Let z be the first essential variable of f according to π and let $g = \mathcal{S}(f)$ and $f = \varphi \circ g$ where $\varphi \in \Phi$. For $\xi \in \{0,1\}$, let $g_\xi = \mathcal{S}(g|_{z=\xi})$ and $g|_{z=\xi} = \varphi_\xi \circ g_\xi$ where $\varphi_\xi \in \Phi$ if $g|_{z=\xi}$ is not constant and $\varphi_\xi \in \Phi^c$ if $g|_{z=\xi}$ is constant. By induction hypothesis there exist $\mathcal{S}$-reduced π-OWDDs $\mathcal{B}_0$ and $\mathcal{B}_1$ for g_0 and g_1 respectively. We may assume w.l.o.g. that $\mathcal{B}_0$ and $\mathcal{B}_1$ share the same nodes for common cofactors. More precisely, we may assume that if w_0 is a node in $\mathcal{B}_0$ and w_1 a node in $\mathcal{B}_1$ such that $f_{w_0} = f_{w_1}$ then $w_0 = w_1$. (Otherwise the nodes in $\mathcal{B}_1$ can be renamed as there is an isomorphism between the sub-OWDDs with root nodes w_0 and w_1, cf. Theorem 5.) We then may compose $\mathcal{B}_0$, $\mathcal{B}_1$ to a $\mathcal{S}$-reduced π-OWDD for f as shown in the picture on the left. □

3.4 Examples

Clearly, since ordinary OBDDs and MTBDDs do not support other transformations than the identity, the equivalence classes are singletons, and hence, there is no choice for the selection function. I.e., we have to deal with $\mathcal{S}(f) = f$ for all $f \in \mathbb{K}(\mathcal{Z})$.

We now turn to a few examples of OWDDs with non-trivial weights. Although the chosen selection function does not affect the size of the obtained reduced OWDDs, crucial aspects for an efficient realization are that the selection function should be easy to implement and space-efficient. The former essentially means that the selection function should fit well with the standard bottom-up synthesis of OWDDs resulting from composition operators. Space-efficiency is often achieved by fixing the trivial weight *id* for a certain edge types (which then need not be represented explicitly).

OBDDs with negative edges. In the case of OBDDs with negative edges the transformation-sets $\Phi = \Phi^c = \{id, \neg\}$ are used. The induced equivalence $\equiv_{\text{NEG}}$ is the finest relation that identifies each boolean function f and its negation $\neg f$. Thus, for the selection function we have the freedom to choose one of them. Many implementations use the selection function that drop the 0-drain and attach a complement bit only to the 0-edges. This corresponds to the unique selection function such that $\mathcal{S}_{\text{NEG}}(f)(1,\ldots,1) = 1$ for all $f \in \mathcal{B}(\mathcal{Z})$.

EVBDDs. Edge-valued BDDs rely on the transformation-type $\mathcal{T}_{\text{EV}} = (\Phi, \Phi^c)$ where $\Phi = \Phi^c = \{x \mapsto x + b : b \in \mathbb{R}\}$. Thus, two $\mathbb{R}$-functions f, g are identified under the induced equivalence $\equiv_{\text{EV}}$ if and only if $f = g + b$ for some

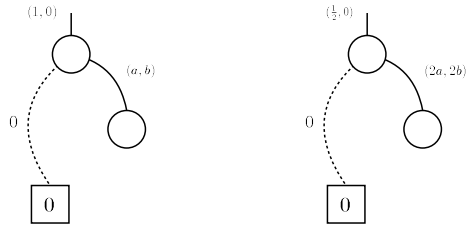


Fig. 6. Two different zero-successor-restricted FEVBDDs representing the same function

real number b. The standard way to ensure canonicity is to require that all 0-edges are equipped with the trivial transformation *id* (i.e., the additive weight 0) and that only the 0-drain is used. This restriction corresponds to the use of the unique selection function where $\mathcal{S}_{\text{EV}}(f)(0,\ldots,0) = 0$ for all $f \in \mathbb{K}(\mathcal{Z})$ holds.

FEVBDDs. In all examples above the transformation sets Φ and Φ^c were the same. This is different for factored EVBDDs that rely on affine transformations $x \mapsto ax+b$ for $(a,b) \in \mathbb{R}^2$, $a \neq 0$, for all non-constant functions $f \in \mathbb{R}(\mathcal{Z})$ and the translations $x \mapsto x+b$ (as in EVBDDs) for the constant functions. However, this distinction does not affect the induced equivalence $\equiv_{\text{FEV}}$ which is given by $f \equiv_{\text{FEV}} g$ iff there exists real numbers a, b with $a \neq 0$ and $f = ag + b$.

The so-called rational rule presented in [43][4] requires that the 0-drain represents all constant functions and that the identity function is assigned to all 0-edges. The corresponding selection function can be defined inductively. For the constant functions we deal with $\mathcal{S}_{\text{FEVrat}}(c) = 0$ for all $c \in \mathbb{R}$. Let us now assume that $f \in \mathbb{R}(\mathcal{Z})$ is non-constant and z the first essential variable of f (i.e., the first variable z in the given ordering π such that $f|_{z=0} \neq f|_{z=1}$.) If $f|_{z=0}$ is non-constant then $\mathcal{S}_{\text{FEVrat}}(f) = g$ where g is the unique function equivalent to f (w.r.t. $\equiv_{\text{FEVrat}}$) such that $g|_{z=0} = \mathcal{S}_{\text{FEV}}(g|_{z=0})$. It remains to consider the case that $f|_{z=0}$ is constant and z the first essential variable of f. Then, $\mathcal{S}_{\text{FEVrat}}(f) = g$ where g is the unique function with $f \equiv_{\text{FEV}} g$ such that $g|_{z=0} = \mathcal{S}_{\text{FEVrat}}(g|_{z=0})$ $g|_{z=1} = \mathcal{S}_{\text{FEVrat}}(g|_{z=1}) + b$ for some $b \in \mathbb{R}$. Figure 6 provides an example to illustrate why the distinction between the cases "$f|_{z=0}$ is constant" and "$f|_{z=0}$ is non-constant" is necessary. It shows two non-isomorphic reduced FEVBDDs with the 0-drain and where all 0-edges are equipped with the trivial transformation *id*.

3.5 Composition operators

The standard implementation techniques that have been proposed for ordinary OBDDs and realized in many BDD packages, can be adapted for OWDDs. This includes the concept of shared OWDDs that rely on a fixed variable ordering π selection function $\mathcal{S}$ and represent several $\mathbb{K}$-functions in a single $\mathcal{S}$-reduced graph. To keep the graph $\mathcal{S}$-reduced during the synthesis algorithms, we use a unique-table that contains full information of each node v by means of the tuple $(z, \langle \phi_1, succ_1(v)\rangle, \langle \phi_0(v), succ_0(v)\rangle$.

Let us sketch the main steps of synthesis algorithms that realize the construction of a sub-OWDD representing $f \mathbin{op} g$ where $f, g \in \mathbb{K}(\mathcal{Z})$ and op is a binary operator on $\mathbb{K}$, e.g., $+$ or $*$ for $K = \mathbb{R}$.[5] We assume that f and g are given as pairs $\langle \varphi, v\rangle$ and $\langle \psi, w\rangle$ consisting of transformations $\varphi, \psi \in \Phi$ and nodes v, w in the shared OWDD. That is, $f = \varphi \circ f_v$, $g = \psi \circ f_w$ and $\varphi \in \Phi^c$ if v is a drain, and the analogous condition for ψ. The rough idea of the OWDD-synthesis algorithm follows the same pattern as Bryant's APPLY-algorithm for OBDDs and relies on the fact that

$$\begin{aligned}(f \mathbin{op} g)|_{z=0} \;&=\; f|_{z=0} \mathbin{op} g|_{z=0}\\ &=\; (\varphi \circ \phi_0(v)) \circ f_{succ_0(v)} \;\; op \;\; (\psi \circ \psi_0(v)) \circ f_{succ_0(w)}\end{aligned}$$

and the analogous equation for the positive cofactor of $f \mathbin{op} g$. Algorithm 1 sketches the main ideas of the weighted APPLY-algorithms for OWDDs.

The arguments of $APPLY(\cdot)$ are two pairs $\langle \varphi, v\rangle$ and $\langle \psi, w\rangle$ consisting of transformations $\varphi, \psi \in \Phi$ and nodes v and w of a shared π-OWDD. (More precisely, we assume $\varphi \in \Phi^c$ if v is a drain, and similarly, $\psi \in \Phi^c$ if w represents a constant function.)

Subroutine *Find_or_Add*$(\cdot)$ serves to realize the weighted version of the OBDD-isomorphism rule. It takes as arguments a variable z and two pairs $\langle \varphi_\xi, v_\xi\rangle$, $\xi = 0, 1$, consisting of a transformation $\varphi_\xi \in \Phi$ and a node v_ξ at some level below the z-level[6] such that $\langle \varphi_0, v_0\rangle \neq \langle \varphi_1, v_1\rangle$. The purpose of *Find_or_Add*$(z, \langle \varphi_1, v_1\rangle, \langle \varphi_0, v_0\rangle)$ is to return a transformation-node-pair $\langle \varphi, v\rangle$ such that

$$\varphi \circ f_v = ITE(z, \langle \varphi_1, v_1\rangle, \langle \varphi_0, v_0\rangle).$$

The latter requires to check in the unique-table whether there exists a node v representing $\mathcal{S}(ITE(z, \langle \varphi_1, v_1\rangle, \langle \varphi_0, v_0\rangle))$. If not then such a node has to be created and inserted in the unique-table. In addition, the transformation $\varphi \in \Phi$ has to be computed.

Similarly, *Find_or_Add_Drain*(c) where $c \in \mathbb{K}$ means a subroutine that finds or adds a drain v with $value(v) \equiv c$ and returns the unique pair $\langle \varphi, v\rangle$ with $c = \varphi(value(v))$ and $\varphi \in \Phi^c$.

The assignment "$z := \min\{var(v), var(w)\}$" means that z is assigned to $var(v)$ if either both v and w are inner nodes and $var(v) \leq var(w)$ (according to the variable ordering π) or v is an inner node and w a drain.

[4] The gcd-rule for FEVBDDs with just integer-values is also presented.

[5] op is extended to an operator $op : \mathbb{K}(\mathcal{Z}) \to \mathbb{K}(\mathcal{Z})$ in the standard way. That is, for $f, g \in \mathbb{K}(\mathcal{Z})$, we put $(f \mathbin{op} g)(\eta) = f(\eta) \mathbin{op} g(\eta)$ for all $\eta \in Eval(\mathcal{Z})$.

[6] This means that v_ξ is either a drain or a z'-node for some variable z' with $z <_\pi z'$. Furthermore, $\varphi_\xi \in \Phi^c$ if v_ξ is a drain.

Similarly, if w is an inner node and v a drain then $z = var(w)$.

Algorithm 1 *APPLY*$(\langle\varphi, v\rangle, \langle\psi, w\rangle, op)$

```
(* input: a binary operation op : IK^2 → IK and          *)
(*        two functions f = φ ∘ f_v and g = ψ ∘ f_w      *)
(*        (given as pairs consisting of a transformation *)
(*         and a node of a shared OWDD)                  *)
(* output: a transformation-node-pair ⟨λ, u⟩ such        *)
(*         that f op g = λ ∘ f_u                         *)

if v and w are drains then
   c := φ(value(v)) op ψ(value(w));
   return Find_or_Add_Drain(c)
else
   z := min{var(v), var(w)};
   for ξ = 0, 1 do
      ⟨φ_ξ, v_ξ⟩ := { ⟨id, v⟩                : if z <_π var(v)
                    { ⟨φ_ξ(v), succ_ξ(v)⟩    : if z = var(v)
      ⟨ψ_ξ, w_ξ⟩ := { ⟨id, w⟩                : if z <_π var(w)
                    { ⟨φ_ξ(w), succ_ξ(w)⟩    : if z = var(w)
      ⟨λ_ξ, u_ξ⟩ := APPLY(⟨φ ∘ φ_ξ, v_ξ⟩, ⟨ψ ∘ ψ_ξ, w_ξ⟩, op)
   end for
   if ⟨λ_0, u_0⟩ = ⟨λ_1, u_1⟩ then
      return ⟨λ_1, u_1⟩
   else
      return Find_or_Add(z, ⟨λ_1, u_1⟩, ⟨λ_0, u_0⟩)
   end if
end if
```

For simplicity, we skipped several details, such as the use of a computed table to avoid multiple recursive calls with the same arguments. For the general case, we have to deal with a computed table where the entries contain the tuples $(\langle\varphi, v\rangle, \langle\psi, w\rangle, \langle\lambda, u\rangle)$ consisting of the input-arguments and the result. Since now for each node-pair (v, w) the worst-case number of possible recursive calls (and entries in the computed-table) for the nodes v and w is given by the product of the number of paths leading to nodes v and w, respectively, the APPLY-algorithm can cause an exponential blow-up. However, in concrete cases a simplified representation of the computed-table and reduced number of recursive calls might be possible. This, for instance, holds for EVBDDs and the summation operator $op = +$, where it suffices to deal with entries of the form (v, w, u) to denote that $f_u = f_v + f_w$.

Depending on op and the chosen transformation-type the APPLY-algorithm can also be simplified by adding more terminal cases. For instance, dealing with FEVBDDs and the multiplication operator, no recursive calls of the APPLY-algorithm are required if one of the input nodes is a drain.

4 Normalized Algebraic Decision Diagrams

We now present a new instance of OWDDs for representing real-valued functions $f \in \mathbb{R}(\mathcal{Z})$, called normalized algebraic decision diagrams (NADDs). NADDs rely on the transformation-type as FEVBDDs, i.e., the transformation-type $\mathcal{T}_{\text{FEV}} = (\Phi, \Phi^c)$ where $\Phi = \{x \mapsto ax + b : a, b \in \mathbb{R}, a \neq 0\}$ and $\Phi^c = \{x \mapsto x + b : b \in \mathbb{R}\}$, and thus, they represent all functions that can be transformed to each other via affine mappings by a single node. NADDs differ from standard FEVBDD with the rational or gcd-selection rule [43] by choosing representatives from each $\equiv_{\text{FEV}}$-equivalence class[7] that are normalized in the sense that the minima and maxima of the chosen representatives are 0 and 1, respectively. (This, of course, only applies for the non-constant functions.) Thus, the selection rule of NADDs $\mathcal{S}_{NADD}$ assigns to each function $f \in \mathbb{R}(\mathcal{Z})$ a function $g \in \mathbb{R}(\mathcal{Z})$ with $g \equiv_{\text{FEV}} f$ and $\min g = 0$, $\max g = 1$.

Definition 4 (Normalized real-valued functions). Let $g \in \mathbb{R}(\mathcal{Z})$. g is called normalized if $\min g = 0$ and $\max g = 1$. □

Lemma 3 (Normalization lemma). *If $f \in \mathbb{R}(\mathcal{Z})$ is non-constant then there are exactly two normalized functions g and h that are $\equiv_{FEV}$-equivalent to f. Furthermore, we have $h = 1 - g$.*

Proof. We first observe that if $f = ag+b$ where $a, b \in \mathbb{R}$, $a \neq 0$ then $f = (-a)(1 - g) + (b - a)$. Thus:

(i) if $f \equiv_{\text{FEV}} g$ then $f \equiv_{\text{FEV}} 1 - g$.

Let $m = \min f$ and $M = \max f$. Since f is non-constant we have $m < M$. We now define $g \in \mathbb{R}(\mathcal{Z})$ by

$$g = \tfrac{1}{M-m} f - \tfrac{m}{M-m}.$$

Then, $g \equiv_{\text{FEV}} f$ and $\min g = 0$, $\max g = 1$. This yields the existence of at least two normalized functions (namely g and $1 - g$) that are $\equiv_{\text{FEV}}$-equivalent to f.

It remains to show that there are no other normalized functions that are $\equiv_{\text{FEV}}$-equivalent to f. Let $h \in \mathbb{R}(\mathcal{Z})$ be normalized such that $f = ch+d$ where $d \in \mathbb{R}$ and $c \in \mathbb{R} \setminus \{0\}$. We may assume w.l.o.g. that $c > 0$. Otherwise we replace h with $1-h$ and obtain $f = (-c)(1-h)+(d-c)$. Since h is normalized and $c > 0$ we get $m = \min f = d$ and $M = \max f = c + d$. Thus, $c = M - m$ and

$$f = (M - m)h + m = (M - m)g + m$$

which yields $h = g$. □

The normalization lemma yields that the selection function for NADDs has to select one of the normalized representatives

$$g = \tfrac{1}{M-m} f - \tfrac{m}{M-m} \quad \text{or} \quad 1 - g = -\tfrac{1}{M-m} f + \tfrac{M}{M-m}$$

in f's equivalence class, where $M = \max f$ and $m = \min f$. Note that for exactly one of these normalized functions, the multiplicative weight is positive.

[7] Recall that $f \equiv_{\text{FEV}} g$ if there exists $a, b \in \mathbb{R}$ with $a \neq 0$ and $f = ag + b$.

In our implementation, we used the unique selection function $\mathcal{S}_{NADD}$ such that the multiplicative weight of each 0-edge leading to an inner node is positive, while for nodes where the 0-edge leads to a drain, the multiplicative weight of the 1-edge is required to be positive. More precisely, $\mathcal{S}_{NADD}(f) = 0$ for all constant functions f. That is, only the 0-drain is used. If z is the first essential variable of f (according to the given variable ordering π) and $f|_{z=0}$ is not constant then $\mathcal{S}_{NADD}(f)$ is the unique normalized function g with $f \equiv_{\text{FEV}} g$ such that $g|_{z=0} = a\mathcal{S}_{NADD}(g|_{z=0}) + b$ where $a > 0$. If z is the first essential variable of f (according to the given variable ordering π) and $f|_{z=0}$ is constant then $\mathcal{S}_{NADD}(f)$ is the unique normalized function g with $f \equiv_{\text{FEV}} g$ such that $g|_{z=1} = a\mathcal{S}_{NADD}(g|_{z=1}) + b$ where $a > 0$.

Besides supporting scalar multiplication and the addition of constants constants, NADDs enjoy the property that function minima and extrema can be derived from the NADD representation. Note that if function f is represented by the tuple $\langle (a, b), v\rangle$ if $a > 0$ where v is a NADD-node and a, b are reals then $\max f = a + b$ and $\min f = b$ if $a > 0$, while $\max f = b$ and $\min f = a + b$ if $a < 0$. Thus, with NADDs $\min f$ and $\max f$ can be computed in constant time. Note, for other selection functions for FEVBDDs, such as $\mathcal{S}_{\text{FEV}}$, the computation of $\min f$ or $\max f$ requires a traversal of the subgraph for f.

On the other hand, selection rules like $\mathcal{S}_{\text{FEV}}$ permit a more sparse representation of inner nodes by tuples consisting of a variable name, the two successor nodes and two real values a_1, b_1 that provide the labeling of the 1-edge. Since the 0-edges of a $\mathcal{S}_{\text{FEVrat}}$-reduced FEVBDD are labeled with the identity function no additional information for the 0-edges is required. At the first view, NADDs seem to require a less memory-efficient representation where for each z-node v four real numbers a_0, b_0, a_1, b_1 have to be stored (where a_0, b_0 denote that $f_v|_{z=0} = a_0 f_{succ_0(v)} + b_0$ and a_1, b_1 indicate that $f_v|_{z=1} = a_1 f_{succ_1(v)} + b_1$). However, it suffices to store three of the values a_0, b_0, a_1, b_1. This is due to the fact that the minimum of the four values $a_0 + b_0$, b_0, $a_1 + b_1$ and b_1 is 0, while their maximum is 1. Hence, rather than storing a_0, b_0, a_1, b_1 it suffices to remember two of these four parameters and the parameter combination that yield the extrema 0 and 1. For an example, let us assume that $a_0 = \frac{1}{2}$, $b_0 = \frac{1}{2}$, $a_1 = -\frac{2}{3}$ and $b_1 = \frac{2}{3}$. Then, the values b_0 and b_1 can be derived from the two values a_0 and a_1 and the information that $a_0 + b_0 = 1$ and $a_1 + b_1 = 0$.

Clearly, the internal representation of NADDs obtained by this observation is still less efficient than for FEVBDDs with the rational selection function (which just requires to store a_1, b_1). However, a further trick can be applied that equalizes the memory requirement for NADDs and FEVBDDs with the rational selection function. This trick that has been realized in our implementation (see Section 5) relies on the fact that on modern computer systems the beginning of the allocated memory is always aligned to word boundaries, and hence, the last bits of the given pointers are zero. As the memory required for a NADD-node is a multiple of a byte, the last bits of these pointers can be used to store the parameter combination that yield the extrema 0 and 1.

Our experimental studies showed that with this implementation trick the memory usage of NADDs equals those of FEVBDDs with the rational selection function, while the time effort to handle these bits is negligible. In fact, in the example of linear programs (discussed in more detail in the next section) the advantages of NADDs over FEVBDDs with other selection function prevail the drawback of the additional computation time for the normalization.

5 Implementation

In the sections above we introduced a generic framework to reason about different OBDD variants. We now summarize the main features of our OBDD library JINC that realizes several OBDD variants and realizes the generic framework of OWDDs by means of templetization. The current version of JINC supports reasoning with ordinary OBDDs, OBDDs with negated edges, MTBDDs, NADDs and FEVBDDs, and in addition the zero-surpressed variants of OBDDs and MTBDDs. (The latter do not fit in the OWDD-framework since they do not rely on the Shannon expansion. However, other expansion rules can be treated with similar concepts.) JINC is written in C++ and available via www.jossowski.de .

5.1 Concept

The key motivation of JINC was to convert the more general approach of OWDDs into an efficient OBDD package. Therefore, JINC uses well established concepts from already known OBDD packages and implementation theories like [13,20,12]. Some of JINCs enhancements are discussed below.

The most important data-structure provided by JINC is the unique table. This table ensures the freedom of redundancies. JINC makes extensive use of templetization. The nodes of an OBDD variant are implemented as an object. The implementation of this basic class enables the use of all data structures provided by JINC. The unique table template is usually hidden in the variable template and is only used directly when performance is essential.

With this concept the implementation of a new BDD variant into JINC is reduced to the implementation of the different selection function. Most of the commonly used data-structures can be used without any modification. This is possible because of the high templatization of JINC. The design concept of JINC is illustrated in

Figure 7. The internal API must be provided by every variant. The public API uses the internal API to provide high level operations. The template level provides data-structures and algorithms that are common for all variants. The dependencies between the different parts are also shown in Figure 7.

The best example for the concept (and the dependencies) are the reordering algorithms on different OBDD variants. The basic operation behind all reordering algorithms is the swap of two neighbored variables. The only difference between reordering algorithms on different OBDD variant relies on different swap implementations. JINC uses this observation and provides a reordering template. After implementing the swap function for a new variant all reordering methods can be used without any further work.

Other data-structures like computed tables, variable template, etc are also provided by JINC. These data-structures are very flexible and at the same time efficient. The variable template provides functions to alter the variable variable ordering. Variables can be inserted and deleted at any point of the variable ordering. The variable ordering template makes it possible to abstract away from the variable ordering.

Computed tables are implemented as self resizing hash tables with an arbitrary number of arguments. Every new algorithm can use computed tables to dramatically speed-up the computation.

JINC comes with some OBDD variants. For this paper NADDs were implemented in JINC. Although the generic formulation of the APPLY-algorithm could be used as a template for all OWDD-instances. However, for efficiency reasonings separate formulations for either OWDD-instance and composition operator are more appropriate since they allow to incorporate terminal cases that are special for the concrete OWDD-instance and composition operator under consideration. For example, when computing the minimum of two functions f and g with NADDs, then the computation can be aborted, e.g., whenever $\max f < \min g$. For this reason, JINC provides optimized versions of the generic APPLY algorithm for all OBDD variants mentioned above.

Like common BDD packages JINC uses a delayed garbage collection to increase the efficency of the computed tables. This is also done via reference count. The main difference to other BDD packages is the collect phase. Whenever a node reaches a reference count equal to zero JINC marks this node as not used. JINC does not traverse the subtree to identify the other dead nodes. This moves the expensive traverse phase to the already expensive garbage collection phase. The advantage of this approach is that in the cases of marking nodes as dead and reuse them later on, JINC is much faster. The garbage collection phase is not measurable slower than in other packages. The disadvantage of this method is that only the number of dead functions is counted and not the number of dead nodes. With the number of dead nodes the amount of unused but reserved memory can be used. Another difference is that dead nodes are not stored in a list. Every unique table (each variable has its own) counts the number of dead nodes. In the case of garbage collection only levels with dead nodes are considered. Inside the unique table a bit-map is used to identify dead nodes efficiently. With this approach very little memory is used compared to the list. The other advantage is that masking a bit is much easier than searching for a dead node in a list.

This is also an other example of the templatization approach from JINC. The unique table is responsible for all garbage collection tasks. All above-mentioned behaviour is hidden inside the unique table template.

5.2 API

Besides this concept of easy extension JINC provides a clean API to reduce the errors while developing and implementing symbolic algorithms. The best way to show how easy it is to use JINC is to show some example code.

```
ADDFunction x="x";
ADDFunction y="y";
ADDFunction f=3*x+y+6;
```

The example above creates the projection function on x and y. The function f is then defined as $3 \cdot x + y + 6$.

Another example is the symbolic reachability analysis. Let δ be a transition function, *org* the conjunction over all original variables x_i (this is the cube set representation of the set of all original variables) and $\pi = (x_1, x_1', \ldots, x_n, x_n')$ the variable ordering. Mind that the original variables x_i and the copy variables x_i' are interleaved arranged.

A reachablity analysis algorithm (starting in v) implemented in JINC could look like:

```
ADDFunction visited=v;
ADDFunction oldVisited;
while(visited!=oldVisited){
  oldVisited=visited;
  visited=visited | (δ&visited).exists(org).moveUp();
}
```

The moveUp() command moves up all nodes one level. With the interleaved variable ordering this has the same effect as renaming all copy variables to original variables. The initial phase of this algorithm sets the starting vertex v. The variable *oldVisited* is used to store the former result of the while-loop. Whenever *visited* equals *oldVisited* the algorithm has calculated all reachable vertices and thus reached a fix-point. The first command in the while-loop updates the set *oldVisited*. The second command calculates all vertices that are reachable from *visited* within one step and adds these to *visited*. In more detail:

- **(δ&visited)** calculates the transition relation limited to transitions starting in *visited*

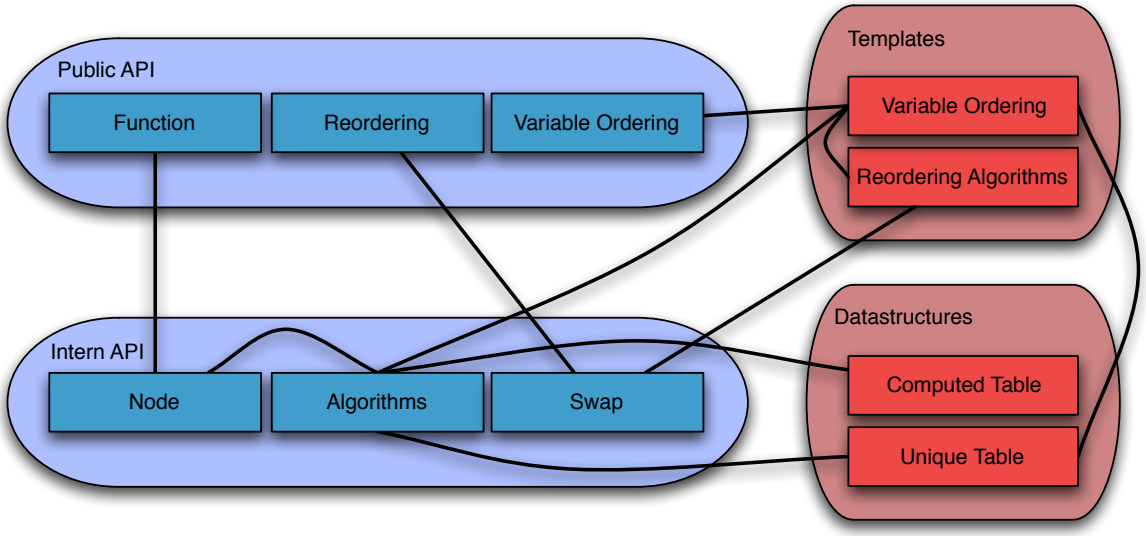


Fig. 7. The internal design of JINC

– **exists(org)** eliminates the starting vertices and thus results in the set of vertices that is reachable in one state (represented with copy variables)
– **moveUp()** renames the copy variables to original variables
– **visited=visited** — . . . unions the former calculated vertices with the newer ones

The two examples above show how easy it is to write BDD based algorithms.

6 Experimental results

6.1 $\{0,1\}$*-Integer Linear Programming*

In Section 4, NADDs that can calculate the minimum and maximum value of a function in constant time have been introduced. A well suited application for this kind of calculation is the integer linear programming. In this section we want to show that NADDs outperform other OBDD variants in this area. It is important to mention that for the construction of NADDs every node has to be normalized and thus a minimum and maximum calculation is already done. Therefore the used benchmarks are focused on building time, cofactor calculations and following maximum calculation. This kind of application is heavily used in probabilistic model checking [3,26].

Linear Programming means the optimization of a given (linear) function with constraints formatted by an inequality $Ax \leq b$ for some matrix A and vector b. Many practical problems can be solved with Integer Linear Programming which is a special instance of linear programs where the input variables are integer variables. In this section we express this integer variables with several Boolean variables so that OBDDs can be used.

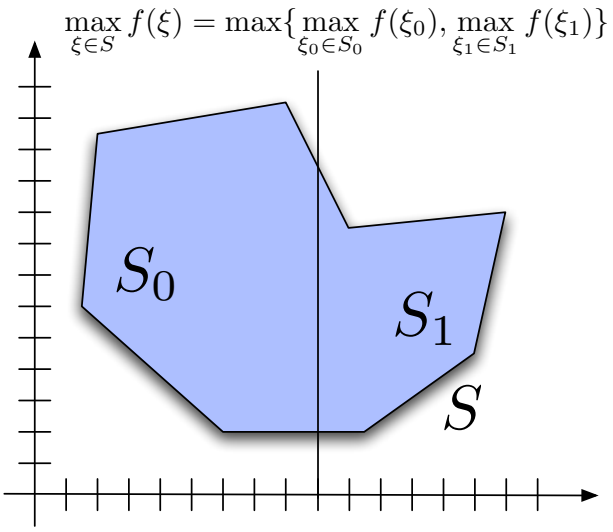


Fig. 8. Idea of the OBDD-based algorithm for integer linear programs

The set of feasible values for the variables can be expressed with regular OBDDs in a natural way. A basic algorithm based on ADDs would check all possible configurations and thus calculate the optimal value of a function with respect to the given constraints. Figure 8 illustrates the idea of an OBDD-based algorithm. In every step the set of feasible solutions S is splitted into $S|_{z=0} = S_0$ and $S|_{z=1} = S_1$ if z is an essential variable of S. The result of this ILP will be calculated recursively. Algorithm 2 shows the pseudo-code of this idea. The same algorithm could easily extended to fit the special property of NADDs. At each level it could be checked if the best solution so far is better than the best solution

that could be found in the remaining sub-function, i.e. $\max(f|_{z=i}) < ILP(S_{(1-i)}, f|_{z=(1-i)}), i \in \{0,1\}$.

Algorithm 2 Basic ILP Algorithm

Input: Goal function f and set of feasible solutions S
Output: Maximal value $x \in (\mathbb{R} \cup \{-\infty\})$ of f with respect to S

```
if f is constant then
  if S = 0 then
    return −∞
  else
    return value(f)
  end if
else
  z ← min{var(f), var(S)}
  w_0 ← ILP(f|_{z=0}, S|_{z=0})
  w_1 ← ILP(f|_{z=1}, S|_{z=1})
  return max{w_0, w_1}
end if
```

Algorithm 3 Modified ILP Algorithm

Input: Goal function f and set of feasible solutions S
Output: Maximal value $x \in (\mathbb{R} \cup \{-\infty\})$ of f with respect to S

```
if f is constant then
  if S = 0 then
    return −∞
  else
    return value(f)
  end if
else
  z ← min{var(f), var(S)}
  i ← random Element out of {0, 1}
  w_i ← ILP(f|_{z=i}, S|_{z=i})
  if max(f|_{z=(1−i)}) < w_i then
    return w_i
  end if
  w_{(1−i)} ← ILP(f|_{z=(1−i)}, S|_{z=(1−i)})
  return max{w_0, w_1}
end if
```

This modification is shown in Algorithm 3. The modified algorithm is well suited for NADDs because the check if the computation can be stopped can be done in constant time. We do not want to introduce any "smarter" algorithm to compute the maximum value of f with respect to given constraints. We just want to compare the properties of different variants in a given application. The details how to construct f and S will not be discussed here because this can be done in a straight way. Table 1 shows the comparison between MTBDDs and FEVBDDs with the rational selection function (henceforth called FEVBDDs) and NADDs with different $\{0,1\}$-ILP-problems taken from [29]. The construction times for FEVBDDs and NADDs are almost equal. FEVBDDs perform a little bit better in this kind of operation because the selection function is a bit easier to calculate. ADDs do not perform well because of its size.

The results show that in all benchmark $\{0,1\}$-ILPs NADDs outperform the other variants. In some $\{0,1\}$-ILPs ADDs are better than FEVBDDs and vice versa. An explanation for this is that in some cases the calculation of the unique representative for FEVBDDs is more time consuming than the bigger size of the ADDs.

6.2 CUDD *vs* JINC

To show the efficiency of JINC is implemented we want to compare the running time of JINC with the well-established BDD package CUDD. The comparison with CUDD has been chosen because other libraries like the ABCD package [6] and BDDNOW were developed with an other intention like parallelization or do not include ADDs. A comparison between CUDD and BuDDy [37] shows that the performance of both packages is almost the same. The OBDD package wld [19] also provides a generic platform for BDD-variants, but the main focus is on the Reed Muller decomposition and (variants of) binary moment diagrams (BMD, *BMDs, K*BMDs, etc.). The wld-package is appropriate for word-level verification of hardware systems, while we concentrate on the Shannon decomposition which appears to be more adequate for complex matrix operations, which, e.g., are required in the context of the quantitative analysis of probabilistic systems.

Kanban	CUDD	JINC
4	1	0.7
5	2.5	1.5
6	5.6	2.9
7	14	5
8	33	13
9	61	21

Table 2. Comparison between CUDD and JINC

We used a BDD-based state space generator and solver [27] for the Moebius performance evaluation tool to benchmark JINC with CUDD. As a benchmark we choose the production system Kanban in different sizes. Table 2 shows how well JINC performs in algebraic computations in comparison to CUDD. The comparison between different packages is always difficult so that we tried to eliminate troublesome effects like garbage collection and dynamic reordering. We set the parameters for both packages so that no garbage collection took place and we turned off the dynamic reordering.

For another comparison between JINC and CUDD we are using the probabilistic model checker Prism [38] (which uses CUDD for the symbolic calculation) and PROMOC [36] (which uses JINC). The leader election protocol from [23] is used as a benchmark.

Problem	Inputs	Constraints	ADD (Build)	ADD	FEVBDD (Build)	FEVBDD	NADD (Build)	NADD
p0033	33	16	0.06	0.34	0.09	0.26	0.10	0.10
stein27	27	118	2.35	36.32	2.29	2.05	2.41	0.46
bm23	27	20	83.22	80.92	92.38	102.04	94.04	7.79
lseu	89	28	254.47	–	271.72	–	278.93	25.7

Table 1. Results of ADDs, FEVBDDs and NADDs for different ILPs

N	K	ADD (PROMOC)	ADD (Prism)
4	2	0	0
4	4	1	0
4	6	2	2
4	8	3	6
4	10	7	19
4	12	8	42
5	2	0	0
5	4	3	2
5	6	19	11
5	8	48	335
6	2	1	0
6	4	11	12
6	6	175	206

Table 3. Calculation times to verify that a leader is elected with probability one

The leader election protocol is used to elect a leader out of N processors. In this case all processors are arranged in a synchronous ring.

This protocol uses the idea that every processor randomly chooses a number from 1 to K. All chosen numbers are passed around the ring. The processor with the highest chosen number will be the leader. If there are more than one processors with the same number, the election starts again.

Table 3 shows the times (in seconds) needed to verify that a leader is elected with probability one. Both tools use the same input language, the same variable ordering and have the same internal representation. In this benchmark it can be seen that JINC is again faster than CUDD in most cases. JINC has a better scalability than CUDD. CUDD initializes very fast and is therefore faster in smaller benchmarks.

7 Conclusion

In this paper we introduced a generic framework to reason about reducedness and canonicity of different OBDD variants. This framework supports the introduction of new OBDD variants and the study of the advantages and disadvantages of every variant. Every variant of FEVBDDs share the advantages for multiplication and addition. With the selection function for FEVBDDs it is possible to benefit from even more advantages. NADDs support the calculation of the maximum and minimum value of a function in constant. The disadvantage of NADDs (the additional number of parameters) could be reduced to some bits. The implementation of NADDs made it possible that FEVBDDs (with the rational rule) and NADDs are of equal size.

NADDs are well suited for minimum and maximum calculation. The advantages of NADDs over other variants (ADDs, FEVBDDs, ...) have been shown in a $\{0,1\}$-ILP benchmark. The OBDD library uses for this benchmark was JINC.

The design of JINC follows the principals of OWDDS and thus leads to a clean and fast OBDD library. First experiments showed that the use of JINC in context of probabilistic model checking seems to be very promising. In future work we will study in more detail which DD-variant is most adequate for the symbolic quantitative analysis of randomized protocols and other probabilistic systems.

JINC is already successfully used from other research teams (e.g.[28] in the application of data-mining).

References

1. R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proc. IEEE/ACM International Conference on CAD*, pages 188–191. IEEE Computer Society Press, 1993.
2. R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, 1997.
3. C. Baier. *On the Algorithmic Verification of Probabilistic Systems*. Habilitation, Universität Mannheim, 1998.
4. Bernd Becker, Rolf Drechsler, and Ralph Werchner. On the relation between BDDs and FDDs. *Information and Computation*, 123(2):185–197, 1995.
5. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. Foundations of Software Technology and Theoretical Computer Science (FST & TCS)*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513, 1995.
6. Armin Biere. Abcd is a compact bdd library. http://fmv.jku.at/abcd.
7. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35, 1986.

8. Randal E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
9. Randal E. Bryant and Yirng-An Chen. Verification of arithmetic circuits using binary moment diagrams. *International Journal on Software Tools for Technology Transfer*, 3(2):137–155, 2001.
10. E. Clarke, M. Fujita, P. McGeer, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *Proc. International Workshop on Logic Synthesis (IWLS)*, 1993.
11. E. M. Clarke, M. Fujita, and X. Zhao. Hybrid decision diagrams: overcoming the limitations of mtbdds and bmds. In *International Conference on Computer Aided Design*, pages 159–163. IEEE Computer Society Press, 1995.
12. R. Drechsler and B. Becker. *Binary Decision Diagrams: Theory and Implementation.* Kluwer Academic Publishers, 1998.
13. Rolf Drechsler, Dragan Jankovic, and Radomir S. Stankovic. Generic implementation of dd packages in mvl. In *EUROMICRO*, pages 1352–1359, 1999.
14. M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2-3):149–169, 1997.
15. Masahiro Fujita, Patrick C. McGeer, and Jerry Chih-Yuan Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997.
16. G. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms.* Kluwer Academic Publishers, 1996.
17. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
18. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
19. Marc Herbstritt. wld - a c++ library for decision diagrams. http://ira.informatik.uni-freiburg.de/software/wld.
20. Stefan Höreth. Implementation of a multiple-domain decision diagram package. In *Proceedings of the IFIP WG 10.5 International Conference on Correct Hardware Design and Verification Methods*, pages 185–202, London, UK, UK, 1997. Chapman & Hall, Ltd.
21. Stefan Höreth. A word-level graph manipulation package. *STTT*, 3(2):182–192, 2001.
22. Shin ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th international conference on Design automation (DAC)*, pages 272–277. ACM Press, 1993.
23. A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1), 1990.
24. T. Kam, T. Villa, R. K. Brayton, and A. L. SagiovanniVincentelli. Multi-valued decision diagrams: Theory and applications. *Multiple-Valued Logic: An International Journal*, 4(1-2):9–62, 1998.
25. U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *European Conf. on Design Automation*, pages 43–47, 1992.
26. M. Kwiatkowska. Model checking for probability and time: From theory to practice. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 351–360. IEEE Computer Society Press, 2003. Invited Paper.
27. K. Lampka and M. Siegle. Symbolic Activity-Local State Graph Generation in the Context of Moebius. In *Proc. of the Satelite Workshop on Stochastic Petri Nets and related Formalisms at the 30'th International Colloquium on Automata, Languages and Programming, Eindhoven, Netherlands*, June 28-29 2003.
28. Elsa Loekito and James Bailey. Are zero-suppressed binary decision diagrams good for mining frequent patterns in high dimensional datasets? In Peter Christen, Paul J. Kennedy, Jiuyong Li, Inna Kolyshkina, and Graham J. Williams, editors, *Sixth Australasian Data Mining Conference (AusDM 2007)*, volume 70 of *CRPIT*, pages 135–146, Gold Coast, Australia, 2007. ACS.
29. A. Martin, T. Achterberg, and T. Koch. Miplib - mixed integer problem library. http://miplib.zib.de.
30. K. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.
31. C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design.* Springer-Verlag, 1998.
32. S. Minato. *Binary decision diagrams and applications for VLSI CAD.* Kluwer Academic Publishers, 1996.
33. S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC)*, pages 52–57, 1990.
34. J. Ossowski and C. Baier. Symbolic reasoning with weighted and normalized decision diagrams. In *Proceedings of the Calculemus 2005 workshop of FM05*, 2005.
35. Joern Ossowski. Jinc is a fast and object-oriented obdd library. http://www.jossowski.de.
36. Joern Ossowski. Promoc a probabilistic model checker based on jinc. http://www.jossowski.de.
37. Buddy Obdd Package. A comparison study between the cudd and buddy. citeseer.ist.psu.edu/603059.html.
38. D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems.* PhD thesis, University of Birmingham, 2002.
39. R. Drechsler, B. Becker, and S. Ruppertz. K*BMDs: a new data structure for verification. In *IFI WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, Frankfurt, Germany, 1995.
40. A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1998.
41. F. Somenzi. Cudd: Cu decision diagram package release, 1998.
42. Fabio Somenzi. Efficient manipulation of decision diagrams. *STTT*, 3(2):171–181, 2001.
43. P. Tafertshofer and M. Pedram. Factored edge-valued binary decision diagrams. *Formal Methods in System Design*, 10(2-3):243–270, 1997.
44. S. Vrudhula, M. Pedram, and Y.-T. Lai. Edge-valued binary decision diagrams. *Representations of Discrete Functions*, pages 109–132, 1996.
45. I. Wegener. *Branching Programs and Binary Decision Diagrams. Theory and Applications.* Monographs on Discrete Mathematics and Applications, SIAM, 2000.