Compiling CSPs: A Complexity Map of (Non-Deterministic) Multivalued Decision Diagrams

Jérôme Amilhastre
Cameleon Software
Le Galilée, 185 rue Galilée
F-31670 Labège, France
`jamilhastre@cameleon-software.com`

Hélène Fargier
IRIT
Université Paul Sabatier
F-31062 Toulouse Cedex 9
France
`fargier@irit.fr`

Alexandre Niveau
CRIL
Université d'Artois
F-62307 Lens Cedex
France
`niveau@cril.fr`

Cédric Pralet
ONERA—The
French Aerospace Lab
F-31055, Toulouse
France
`cpralet@onera.fr`

***Abstract*—Constraint Satisfaction Problems (CSPs) offer a powerful framework for representing a great variety of problems. The difficulty is that most of the requests associated with CSPs are NP-hard. As these requests must be addressed online, Multivalued Decision Diagrams (MDDs) have been proposed as a way to compile CSPs.**

In the present paper, we draw a compilation map of MDDs, in the spirit of the `NNF` compilation map, analyzing MDDs according to their succinctness and to their polytime transformations and queries. Deterministic ordered MDDs are a generalization of ordered binary decision diagrams to non-Boolean domains: unsurprisingly, they have similar capabilities. More interestingly, our study puts forward the interest of *non-deterministic* ordered MDDs: when restricted to Boolean domains, this fragment captures `OBDD` and `DNF` as proper subsets and has performances close to those of `DNNF`. The comparison to classical, deterministic MDDs shows that relaxing the determinism requirement leads to an increase in succinctness and allows more transformations to be satisfied in polytime (typically, the disjunctive ones). Experiments on random problems confirm the gain in succinctness.[1]

I. Introduction

Constraint Satisfaction Problems (CSPs) offer a powerful framework for representing a great variety of problems. Different kinds of requests can be posted on a CSP, such as the classical extraction of a solution, but also enforcement of strong consistency for the domains, dynamic addition of new constraints, solutions counting, and even combinations of these requests. For instance, the interactive solving of a configuration problem amounts to a sequence of unary constraints additions, while maintaining strong consistency.

Most of these requests are NP-hard; however, they must sometimes be addressed online. A possible way of solving this contradiction consists in representing the set of solutions of the CSP as a Multivalued Decision Diagram [1], [2], [3], that is, as a graph whose nodes are labeled by variables and whose edges represent assignments of the variables. In such diagrams, each path from the root to the sink represents a solution of the CSP. They allow several operations, like those previously cited, to be achieved in time polynomial w.r.t. the size of the diagram. This size can theoretically be exponentially higher than the original CSP's, but it remains low in many applications. Indeed, as they are graphs, MDDs can take advantage of the (conditional) interchangeability of values and save space by merging identical subproblems. As a matter of fact, decision diagrams have been used in various contexts, e.g., in product configuration [4], in recommender systems [5], or, in their original Boolean form, in planning [6], [7] and diagnosis [8]. Up to our knowledge, these applications always consider *deterministic* MDDs, that is, MDDs in which edges going out of a node have mutually exclusive labels. This implies that a given solution is represented once, and only once, in the diagram. However, this assumption is probably not compulsory for many operations, like the aforementioned ones. *Non-deterministic* structures could thus be appealing, depending on what is lost and gained when assuming or relaxing the assumption of determinism.

To evaluate the interest of determinism in such data structures, we propose to draw a compilation map of MDDs, in the spirit of the `NNF` Knowledge Compilation (KC) map [9]. Such a map provides a way to identify the most succinct language supporting in polytime the operations needed for a given application. In this purpose, we conducted a general complexity analysis on MDDs with respect to a variety of requests, coming either from reasoning-oriented problems or from CSP (decision-oriented) applications.

The next section presents the `MDD` framework and its sublanguages, such as deterministic or ordered MDDs. In Section III, we study the case of Boolean domains in order to picture `MDD` in the `NNF` KC map: we show that, beyond `dOMDD`, which somehow corresponds to `OBDD`, the `MDD` family includes the non-deterministic decision diagrams language (`OMDD`) that generalizes both the `OBDD` and `DNF` languages. Section IV is devoted to the `MDD` KC map, including a succinctness analysis of the different subclasses of the `MDD` family, as well as the complexity analysis of many queries and transformations. Section V then presents our first experimental results about the relative succinctness of deterministic and non-deterministic MDDs. Last, important proofs are gathered in the appendix.

II. Multivalued Decision Diagrams

A Constraint Satisfaction Problem $P = \langle X, C \rangle$ consists of a set of variables $X = \{x_1, \ldots, x_n\}$ having a finite domain

[1]This work is partially funded by the ANR project "Business Recommendation for Configurable Products" (BR4CP), ANR-11-BS02-008

of values, and a set of constraints C that specify allowed combinations of values for subsets of variables. For $y \in X$, $\mathrm{Dom}(y)$ denotes the domain of y. For $Y = \{y_1, \dots, y_k\} \subseteq X$, $\mathrm{Dom}(Y)$ denotes $\mathrm{Dom}(y_1) \times \cdots \times \mathrm{Dom}(y_k)$, and $\vec{y}$ denotes a Y*-assignment* of variables from Y, i.e., $\vec{y} \in \mathrm{Dom}(Y)$. When $Z \cap Y = \emptyset$, $\vec{z} \, . \, \vec{y}$ is the *concatenation* of $\vec{z}$ and $\vec{y}$. Last, $\vec{y}_{|y_i}$ denotes the value assigned to y_i in $\vec{y}$.

The set of solutions $\mathrm{Sol}(P)$ of a CSP $P = \langle X, C \rangle$ is the set of elements in $\mathrm{Dom}(X)$ that satisfy all constraints in C. $\mathrm{Sol}(P)$ can be represented as a multivalued decision diagram on X [1], [3].

Definition 1 (Multivalued Decision Diagrams)**.** A *Multivalued Decision Diagram* (MDD) on a set of finite domain variables X is a directed acyclic graph $\phi = \langle \mathcal{N}, \mathcal{E} \rangle$ where $\mathcal{N}$ is a set of nodes containing at most one root (denoted $\mathrm{Root}(\phi)$) and at most one leaf (the *sink*, denoted $\mathrm{Sink}(\phi)$).

Except for the sink, each node $N \in \mathcal{N}$ is labeled by a variable $\mathrm{Var}(N)$ in X, and each edge $E \in \mathcal{E}$ going out of N is labeled by a value $\mathrm{Lbl}(E)$ in the domain of $\mathrm{Var}(N)$.[2]

The size of ϕ, denoted $\|\phi\|$, is equal to its number of nodes and edges, plus the cardinalities of the domains in X.

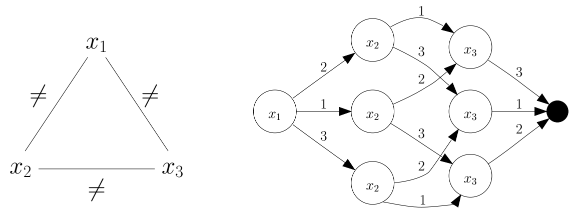

Fig. 1. The "Alldifferent" coloring problem on a complete graph (3 variables, 3 colors) and a dOMDD representing its set of solutions ($x_1 < x_2 < x_3$).

Each solution of P is thus represented by a path from the root of a MDD to its sink (see e.g. Figure 1). To check whether an assignment $\vec{x} = \langle a_1, \dots, a_n \rangle \in \mathrm{Dom}(X)$ belongs to $\mathrm{Sol}(P)$, one only has to traverse the MDD from the root to the sink and, at every node N labeled with variable x_i, to follow an edge labeled with a_i. If such a path exists, then $\vec{x}$ belongs to $\mathrm{Sol}(P)$. Otherwise, if the sink cannot be reached, then $\vec{x}$ is not a solution of P.

Beyond the CSP framework, an MDD can represent any subset of $\mathrm{Dom}(X)$—or equivalently, any Boolean function over $\mathrm{Dom}(X)$.

Definition 2 (Semantics of MDDs)**.** An MDD ϕ on X represents a function $\mathrm{I}(\phi)$ from $\mathrm{Dom}(X)$ to $\{\top, \bot\}$, called the *interpretation* of ϕ and defined as follows: for every X-assignment $\vec{x}$, $\mathrm{I}(\phi)(\vec{x}) = \top$ if and only if there exists a path p from the root to the sink of ϕ such that for each edge $E = \langle N, N', a \rangle$ along p, $\vec{x}_{|\,\mathrm{Var}(N)} = a$.

[2]We also denote $\mathrm{Out}(N)$ (resp. $\mathrm{In}(N)$) the set of outgoing (resp. incoming) edges of N. An edge $E \in \mathcal{E}$ is often denoted by a triple $\langle N, N', a \rangle$ of its source node N denoted $\mathrm{Src}(E)$, its destination node N' denoted $\mathrm{Dest}(E)$, and its associated value $a = \mathrm{Lbl}(E)$.

We say that $\vec{x}$ is a model of ϕ whenever $\mathrm{I}(\phi)(\vec{x}) = \top$; $\mathrm{Mod}(\phi)$ denotes the set of models of ϕ. An MDD ϕ is said to be *equivalent* to another MDD ψ (denoted $\phi \equiv \psi$) iff $\mathrm{Mod}(\phi) = \mathrm{Mod}(\psi)$.

Existing proposals referring to decisions diagrams as a way to compile CSP assume that they are deterministic and ordered [1], [3].

Definition 3 (Deterministic MDD)**.** A node N in an MDD is *deterministic* iff values labeling edges going out of N are pairwise distinct. A *deterministic MDD* (dMDD) is an MDD containing only deterministic nodes.

Definition 4 (Ordered MDD)**.** Let $<$ be a total order over X. An MDD is said to be *ordered w.r.t.* $<$ iff, for every couple of nodes $\langle N, M \rangle$ such that N an ancestor of M, $\mathrm{Var}(N) < \mathrm{Var}(M)$ holds.

In the following sections, we investigate the influence of the assumption of determinism on (i) the relative succinctness of compiled forms, and (ii) the performances of the compiled form with respect to different requests. To do that, six languages are considered.

Definition 5 (The MDD family)**.**

- MDD is the language[3] of all Multivalued Decision Diagrams over X;
- dMDD is the language of all deterministic MDDs;
- $\mathtt{OMDD}_<$ is the language of MDDs ordered w.r.t. $<$ (these MDDs may be non-deterministic);
- OMDD is the union of all $\mathtt{OMDD}_<$ languages;
- $\mathtt{dOMDD}_<$ is the language of all deterministic MDDs that are ordered w.r.t. $<$;
- dOMDD is the union of all the $\mathtt{dOMDD}_<$ languages.

It obviously holds that $\mathtt{dOMDD}_< \subseteq \mathtt{dOMDD} \subseteq \mathtt{dMDD} \subseteq \mathtt{MDD}$, that $\mathtt{dOMDD}_< \subseteq \mathtt{OMDD}_< \subseteq \mathtt{OMDD} \subseteq \mathtt{MDD}$, and that $\mathtt{dOMDD} \subseteq \mathtt{OMDD}$.

Similarly to binary decision diagrams, we suppose that MDDs are in reduced form, that is, (i) isomorphic nodes (labeled by the same variable and pointing to the same children with the same labels) have been merged, (ii) redundant nodes (having a unique child and one outgoing edge per value in the variable's domain) have been skipped, and (iii) nodes with no successor (except for the sink) have been removed. Assuming that MDDs are reduced is harmless because reduction can be done in time polynomial w.r.t. the size of the diagram.

Proposition 6 (Reduction of MDD and its sub-languages)**.** *Let* L *be one of the languages in* $\{\mathtt{dOMDD}_<, \mathtt{OMDD}_<, \mathtt{dOMDD}, \mathtt{OMDD}, \mathtt{dMDD}, \mathtt{MDD}\}$. *There exists a polytime algorithm that transforms any* ϕ *in* L *into an equivalent reduced* ϕ' *in* L *such that* $\|\phi'\| \leq \|\phi\|$.

[3]A *language* is a set of graph structures, together with an interpretation function and a size function. We should write $\mathtt{MDD}_X$, but we omit the subscript since there is no ambiguity on X. An element of the MDD language is an MDD.

III. Decision Diagrams in the NNF KC Map

To highlight the relationship between MDDs and the NNF KC map [9], let us consider the Boolean case. For any subclass L of MDD, let $\mathtt{L}^{\mathfrak{B}}$ be the sublanguage of L obtained when restricting it to Boolean variables. $\mathtt{dMDD}^{\mathfrak{B}}$, $\mathtt{dOMDD}^{\mathfrak{B}}$, and $\mathtt{dOMDD}_{<}^{\mathfrak{B}}$ correspond to BDD, OBDD, and $\mathtt{OBDD}_{<}$ respectively, up to a linear transformation.

Definition 7. A sublanguage $\mathtt{L}_2$ of MDD is *polynomially translatable* (resp. *linearly translatable*) into another sublanguage $\mathtt{L}_1$ of MDD, which we denote $\mathtt{L}_1 \leq_{\mathcal{P}} \mathtt{L}_2$ (resp. $\mathtt{L}_1 \leq_{\mathcal{L}} \mathtt{L}_2$), if and only if there exists a polytime (resp. linear time) algorithm $A_{\mathtt{L}_2 \mapsto \mathtt{L}_1}$ mapping any element in $\mathtt{L}_2$ to an equivalent element in $\mathtt{L}_1$.

When translations $A_{\mathtt{L}_1 \mapsto \mathtt{L}_2}$ and $A_{\mathtt{L}_2 \mapsto \mathtt{L}_1}$ are linear and stable, i.e., when $A_{\mathtt{L}_1 \mapsto \mathtt{L}_2} = A^{-1}_{\mathtt{L}_2 \mapsto \mathtt{L}_1}$, we say that $\mathtt{L}_1$ and $\mathtt{L}_2$ are linearly equivalent, denoted by $\mathtt{L}_1 \equiv_{\mathcal{L}} \mathtt{L}_2$.

Definition 8 (Succinctness)**.** A sublanguage $\mathtt{L}_1$ of MDD is *at least as succinct* as another sublanguage $\mathtt{L}_2$ of MDD (denoted $\mathtt{L}_1 \leq_s \mathtt{L}_2$) if and only if there exists a polynomial $P(\cdot)$ such that for each element ϕ of $\mathtt{L}_2$, there exists an equivalent representation ψ of $\mathtt{L}_1$ verifying $\|\psi\| \leq P(\|\phi\|)$.

Relation $\leq_s$ is a preorder. We denote $\sim_s$ its symmetric part, and $<_s$ its asymmetric part. Of course, $\mathtt{L}_1 \leq_{\mathcal{L}} \mathtt{L}_2 \Rightarrow \mathtt{L}_1 \leq_{\mathcal{P}} \mathtt{L}_2 \Rightarrow \mathtt{L}_1 \leq_s \mathtt{L}_2$.

The BDD and $\mathtt{dMDD}^{\mathfrak{B}}$ languages are linearly equivalent: to transform a BDD into a deterministic Boolean MDD, simply remove the $\perp$ sink and reduce the graph. To transform a deterministic Boolean MDD into a BDD, simply add a $\perp$ sink, and for any node that has only one outgoing edge E, add an edge pointing to the $\perp$ sink, labeled by 0 if $\mathrm{Lbl}(E) = 1$ and by 1 otherwise.

Proposition 9. $\mathtt{dMDD}^{\mathfrak{B}} \equiv_{\mathcal{L}}$ BDD, $\mathtt{dOMDD}^{\mathfrak{B}} \equiv_{\mathcal{L}}$ OBDD, *and* $\mathtt{dOMDD}_{<}^{\mathfrak{B}} \equiv_{\mathcal{L}} \mathtt{OBDD}_{<}$.

Multivalued decision diagrams also capture fragments beyond the BDD family. DNFs, for instance, can be represented as OMDDs in linear time, although some DNFs have no polynomial OBDD representation. OMDD is hence strictly more succinct than DNF—it can be seen as a proper "superset" of this fragment.

Proposition 10. *It holds that* $\mathtt{OMDD}_{<}^{\mathfrak{B}} <_s$ DNF.

Finally, it holds that $\mathtt{dOMDD}^{\mathfrak{B}} \subseteq$ d-DNNF and $\mathtt{OMDD}^{\mathfrak{B}} \subseteq$ DNNF; moreover, $\mathtt{dOMDD}^{\mathfrak{B}} <_s$ d-DNNF (since $\mathtt{dOMDD}^{\mathfrak{B}} \equiv_{\mathcal{L}}$ OBDD). Figure 2 summarizes our results: $\mathtt{OMDD}^{\mathfrak{B}}$ appears as a new fragment in the NNF succinctness map, that takes place below DNNF and above DNF and OBDD. Deciding whether $\mathtt{OMDD}^{\mathfrak{B}}$ and DNNF coincide, and more generally extending multivalued diagrams to DNNF-like forms, is a work left to further research.

TABLE I
Results about succinctness.

L	MDD	dMDD	OMDD	dOMDD	$\mathtt{OMDD}_{<}$	$\mathtt{dOMDD}_{<}$
MDD	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$
dMDD	?	$\leq_s$	?	$\leq_s$	?	$\leq_s$
OMDD	$\not\leq_s$	$\not\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$
dOMDD	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\leq_s$	$\not\leq_s$	$\leq_s$
$\mathtt{OMDD}_{<}$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\leq_s$	$\leq_s$
$\mathtt{dOMDD}_{<}$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\leq_s$

IV. The MDD Knowledge Compilation Map

A. Succinctness

The results of our succinctness analysis are depicted in Table I (see also Figure 2 for the Boolean case).

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

Some of these results are not surprising and directly follow from the fact that dOMDD and $\mathtt{dOMDD}_{<}$ collapse into OBDD and $\mathtt{OBDD}_{<}$ when domains are Boolean: $\mathtt{dOMDD}_{<} \not\leq_s$ dOMDD is a straightforward consequence of $\mathtt{OBDD}_{<} \not\leq_s$ OBDD. Some other results are derived from the NNF map in a less immediate way; for instance dOMDD $\not\leq_s$ OMDD (and thus OMDD $<_s$ dOMDD) holds since $\mathtt{OMDD}^{\mathfrak{B}} \leq_{\mathcal{P}}$ DNF and OBDD $\sim_s \mathtt{dOMDD}^{\mathfrak{B}}$: if dOMDD $\leq_s$ OMDD were true, we could derive OBDD $\leq_s$ DNF, which has been proven false [9].

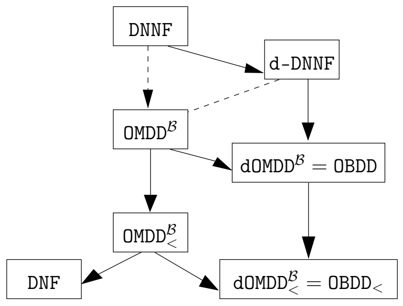
DNNF
d-DNNF
$\mathtt{OMDD}^{\mathcal{B}}$
$\mathtt{dOMDD}^{\mathcal{B}} = \mathtt{OBDD}$
$\mathtt{OMDD}_{<}^{\mathcal{B}}$
DNF
$\mathtt{dOMDD}_{<}^{\mathcal{B}} = \mathtt{OBDD}_{<}$

Fig. 2. $\mathtt{OMDD}^{\mathfrak{B}}$ and its sublanguages in the DNNF succinctness map. An edge $L_1 \rightarrow L_2$ indicates that L_1 is strictly more succinct than L_2. Dashed edges indicate incomplete results.

Some results are harder to get. For instance, in order to prove OMDD $\not\leq_s$ MDD, we used the n-coloring problem of a clique containing n vertices. On the one hand, it can be shown that the set of solutions Sol of this problem can be represented using an MDD whose size is polynomial in n. On the other hand, it is possible to prove that any dOMDD or OMDD representing Sol has a size exponential in n. Figure 1 shows the corresponding blow-up on a small instance.

We get $\mathtt{dOMDD}_{<} \not\leq_s \mathtt{OMDD}_{<}$ by considering another CSP, the n-coloring problem of a star graph with n vertices (see Figure 3 for an example with $n = 3$). Let x_1 be the center of the star, and consider the order $x_n < \cdots < x_2 < x_1$: the $\mathrm{dOMDD}_{<}$ representing this problem contains at least 2^n nodes and $n \cdot 2^{n-1}$ edges, whereas it can be shown that this CSP can be represented by an $\mathrm{OMDD}_{<}$ (with the same order $<$) whose size is polynomial in n.

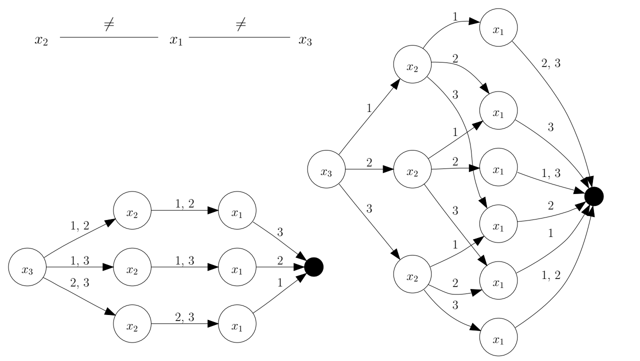

Fig. 3. The coloring problem on a star graph (3 variables, 3 colors), an OMDD and a dOMDD representing its set of solutions (both for $x_3 < x_2 < x_1$).

B. Queries and Transformations

As outlined by Darwiche and Marquis [9], evaluating the suitability of a target compilation language for a particular application consists in balancing its succinctness against the set of requests that it supports in polytime. The requests identified by Darwiche and Marquis are oriented towards applications in knowledge management, yet most of them are still meaningful in some applications targeted by the CSP framework. We enrich this set by a few new requests, raised by more decision-oriented applications.

The most usual requests are checking the consistency of the CSP (**CO**), extracting one or all solutions (we denote these requests **MX** and **ME**), and counting (**CT**) the number of solutions.

The "context extraction" query (**CX**) aims at providing the user with all possible values of a variable of interest.

The "conditioning" operation (**CD**) assigns some values to some variables. More generally, the "term restriction" transformation (**TR**) restricts the possible values of some variables to a subset of their domains. **TR**, **CD** and **CX** are often used in sequence in interactive CSP solving, where the user iteratively looks for the possible values of the next variables and restricts their values according to her preferences [4].

"Clausal entailment" (**CE**) is a request coming from reasoning problems: it consists in determining whether all the solutions of a problem satisfy a disjunction of elementary conditions (unary constraints). Reasoning AI applications raise other requests, like **VA** (is a formula valid?) and $\neg$**C** (compute the negation of a given formula).

The equivalence (**EQ**) and implication (**IM**) requests come from model checking. They can be useful for some CSP modeling problem: **IM** corresponds to checking whether a set of constraints is a semantic relaxation of another one (whether the assignments satisfying the latter also satisfy the former); **EQ** is the equivalence test.

Interactive modeling applications can also involve the manipulation of constraints represented in a compiled form, namely conjunction ($\wedge$**C**), disjunction ($\vee$**C**) and the aforementioned negation ($\neg$**C**), so as to build composed constraints from a few operators. This interactive modeling process can also rely on other operations to check the resulting constraint (or problem), e.g., by projecting, through a **CX** operation, the constraint on one of its variables.

The forgetting operation (**FO**) allows one to eliminate some (intermediate) variables from the problem—this amounts to an existential projection of the problem. Its dual operation, ensuring (**EN**), performs a universal variable elimination. The forgetting and ensuring operations are notably relevant for compilation-based approaches of planning, such as the "planning as model checking" paradigm [6].

All these operations are often performed in sequence. Conditioning (**CD**) followed by model extraction (**MX**) can for instance be useful in planning under uncertainty: if we suppose that a decision policy π associating decisions with states has already been compiled, **CD** allows π to be conditioned by the current state, whereas **MX** allows a valid decision for that state to be extracted. Another example is given by online diagnosis applications: the (compiled) model of the system is first conditioned by the observations (**CD**), then **CX** can give the possible failure modes of each component in the system. More complex failure hypotheses can be checked via clausal entailment (**CE**) queries.

More generally, the operations mentioned above can be partitioned into two sets: the transformations and the queries. Transformations take as input a (compiled) problem and return another one (e.g., the conditioning of a CSP by some assignment produces a CSP with less variables). Queries do not modify the problem, but simply answer a question (**CO**, **CT**, **CE** are queries).

Let us now formalize these operations as properties that a compilation language may satisfy or not. For space reasons, we refrain from stating definitions of queries transformations that are straightforward generalizations of the Boolean case [9].

Definition 12 (Queries)**.** Let L denote a sublanguage of MDD.

- L satisfies **CE** (resp. **IM**) iff there exists a polynomial $P(;)$ and an algorithm that maps every MDD ϕ from L, any set of variables $\{y_1, \dots, y_k\} \subseteq \mathrm{Var}(\phi)$, and any sequence $(A_1, \dots, A_k)$ of finite sets of integers, to 1 if $\mathrm{I}(\phi) \models [y_1 \in A_1] \vee \cdots \vee [y_k \in A_k]$ (resp. $[y_1 \in A_1] \wedge \cdots \wedge [y_k \in A_k] \models \mathrm{I}(\phi)$) and to 0 otherwise, in time bounded by $P(\|\phi\|; n_A)$, where $n_A = \max_{1 \leq i \leq k} |A_i|$.
- L satisfies **MC** iff there exists a polytime algorithm that maps every MDD ϕ from L and any $\mathrm{Var}(\phi)$-assignment $\vec{x}$ to 1 if $\vec{x}$ is a model of ϕ and to 0 otherwise.
- L satisfies **MX** iff there exists a polytime algorithm that maps every MDD ϕ in L to one model of ϕ if there is one, and stops without returning anything otherwise.
- L satisfies **CX** iff there exists a polytime algorithm that outputs, for any ϕ in L and any $y \in \mathrm{Var}(\phi)$, the set of all values taken by y in at least one model of ϕ.

Before defining transformations, we present the semantic operations on which they are based.

TABLE II

RESULTS ABOUT QUERIES AND TRANSFORMATIONS; √ MEANS "SATISFIES", ● MEANS "DOES NOT SATISFY", AND ○ MEANS "DOES NOT SATISFY, UNLESS P = NP". BRACKETS [·] DENOTE A CONJECTURE. MOST RESULTS FOR DNF, DNNF, d-DNNF, OBDD, AND OBDD$_<$ ARE FROM THE NNF MAP AND ARE GIVEN HERE AS A BASELINE.

L	CO	VA	MC	CE	IM	EQ	SE	MX	CX	CT	ME
MDD	○	○	√	○	○	○	○	○	○	○	○
dMDD	○	○	√	○	○	○	○	○	○	○	○
OMDD	√	○	√	√	○	○	○	√	√	○	√
dOMDD	√	√	√	√	√	√	○	√	√	√	√
OMDD$_<$	√	○	√	√	○	○	○	√	√	○	√
dOMDD$_<$	√	√	√	√	√	√	√	√	√	√	√
DNF	√	○	√	√	○	○	○	√	√	○	√
DNNF	√	○	√	√	○	○	○	√	√	○	√
d-DNNF	√	√	√	√	√	?	○	√	√	√	√
OBDD	√	√	√	√	√	√	○	√	√	√	√
OBDD$_<$	√	√	√	√	√	√	√	√	√	√	√

L	CD	TR	FO	SFO	EN	SEN	$\vee$C	$\vee$BC	$\wedge$C	$\wedge$BC	$\neg$C
MDD	√	○	○	√	○	√	√	√	√	√	?
dMDD	√	○	○	√	○	√	√	√	√	√	√
OMDD	√	√	√	√	○	○	[○]	[○]	○	○	○
dOMDD	√	●	●	●	●	●	●	○	●	○	√
OMDD$_<$	√	√	√	√	○	○	√	√	○	√	○
dOMDD$_<$	√	●	●	●	●	●	●	√	●	√	√
DNF	√	√	√	√	○	√	√	√	●	√	●
DNNF	√	√	√	√	○	○	√	√	○	○	○
d-DNNF	√	√	○	○	○	○	○	○	○	○	?
OBDD	√	●	●	√	●	√	●	○	●	○	√
OBDD$_<$	√	●	●	√	●	√	●	√	●	√	√

Definition 13. Let I, J be the interpretation functions on $\mathrm{Var}(\mathrm{I}), \mathrm{Var}(\mathrm{J})$ of some MDDs.

- The existential projection of I on $Y \subseteq \mathrm{Var}(\mathrm{I})$ is the function $\mathrm{I}^{\downarrow Y}$ on the variables of Y defined by: $\mathrm{I}^{\downarrow Y}(\vec{y}) = \top$ iff there exists a Z-assignment $\vec{z}$ (with $Z = \mathrm{Var}(\mathrm{I}) \setminus Y$) s.t. $\mathrm{I}(\vec{z} \,.\, \vec{y}) = \top$. The "forgetting" operation is the dual one: $\mathrm{Forget}(\mathrm{I}, Y) = \mathrm{I}^{\downarrow \mathrm{Var}(\mathrm{I}) \setminus Y}$.
- The universal projection of I on $Y \subseteq \mathrm{Var}(\mathrm{I})$ is the function $\mathrm{I}^{\Downarrow Y}$ on the variables of Y defined by: $\mathrm{I}^{\Downarrow Y}(\vec{y}) = \top$ iff for all Z-assignment $\vec{z}$ (with $Z = \mathrm{Var}(\mathrm{I}) \setminus Y$), $\mathrm{I}(\vec{z} \,.\, \vec{y}) = \top$. The "ensuring" operation is the dual one: $\mathrm{Ensure}(\mathrm{I}, Y) = \mathrm{I}^{\Downarrow \mathrm{Var}(\mathrm{I}) \setminus Y}$.
- The restriction of I to J, denoted $\mathrm{I}_{|\,\mathrm{J}}$, is defined by $\mathrm{I}_{|\,\mathrm{J}} = \mathrm{Forget}(\mathrm{I} \wedge \mathrm{J}, \mathrm{Var}(\mathrm{J}))$.
- Given an assignment $\vec{y}$ of some set of variables $Y \subseteq \mathrm{Var}(\mathrm{I})$, the conditioning of I by $\vec{y}$ is the function $\mathrm{I}_{|\vec{y}}$ on the variables in $Z = \mathrm{Var}(\mathrm{I}) \setminus Y$ defined by: $\mathrm{I}_{|\vec{y}}(\vec{z}) = \mathrm{I}(\vec{y} \,.\, \vec{z})$.

Definition 14 (Transformations)**.** Let L denote a subset of the MDD language.

- L satisfies **FO** (resp. **EN**) iff there exists a polytime algorithm that maps every MDD ϕ from L and every $Y \subseteq \mathrm{Var}(\phi)$ to an MDD ϕ' in L such that $\mathrm{I}(\phi') = \mathrm{Forget}(\mathrm{I}(\phi), Y)$ (resp. $\mathrm{I}(\phi') = \mathrm{Ensure}(\mathrm{I}(\phi), Y)$).
- L satisfies **SFO** (resp. **SEN**) iff it satisfies **FO** (resp. **EN**) when limited to a single variable (i.e., $|Y| = 1$).
- L satisfies **TR** iff there exists a polynomial $P(;)$ and an algorithm that outputs, for every MDD ϕ from L, any set of variables $\{y_1, \dots, y_k\} \subseteq \mathrm{Var}(\phi)$ and any sequence $(A_1, \dots, A_k)$ of finite sets of integers, an MDD ϕ' in L such that $\mathrm{I}(\phi') = \mathrm{I}(\phi)_{|\left[y_1 \in A_1\right] \wedge \dots \wedge \left[y_k \in A_k\right]}$ in time $P(\|\phi\|; n_A)$, where $n_A = \max_{1 \le i \le k} |A_i|$.
- L satisfies **CD** iff there exists a polytime algorithm that maps every MDD ϕ in L and every assignment $\vec{y}$ of $Y \subseteq \mathrm{Var}(\phi)$ to an MDD ϕ' in L such that $\mathrm{I}(\phi') = \mathrm{I}(\phi)_{|\vec{y}}$.

The results of our analysis of the complexity of queries and transformations are depicted in Table II.

Proposition 15. *The results in Table II hold.*

Results for dMDD, dOMDD, and dOMDD$_<$ are generally known or follow directly from previous works [10], [2], [9]. dOMDDs have almost the same capabilities as OBDDs, which is not surprising, given how strong their relationship is. However, note that single forgetting and single ensuring, while satisfied by OBDD, are not satisfied by dOMDD; this is due to the domain sizes being unbounded.

More interestingly, Proposition 15 puts forward the attractiveness of *non-deterministic* OMDDs w.r.t. transformations: OMDD$_<$ supports the same transformations as DNF but **SEN**, and more transformations than dOMDD$_<$ but the negation, while being strictly more succinct than these two classes.

As for the performances w.r.t. queries, OMDD and OMDD$_<$ are less interesting than their deterministic counterparts. But "only" **CT**, **VA**, **EQ**, **IM**, and **SE** are lost when determinism is relaxed (this is not surprising, considering that the same requests are lost when comparing DNNF to d-DNNF). Hence the interest of these languages for many applications that do not involve these requests, such as planning, in which one needs to often check consistency, forget variables and extract models, or online configuration, that relies essentially on conditioning and context extraction.

V. EXPERIMENTAL RESULTS

Table III reports some results obtained on randomly generated CSPs containing 15 variables whose domain size is equal to 6. For each line, 50 random problems are generated. For each of these problems, the order $<$ on the variables used in the compiled forms is built via the MCSInv heuristics [11], which iteratively chooses a variable connected to the greatest number of remaining variables in the constraint graph. The dOMDD compiler we use follows the bottom-up approach [1]: each constraint is compiled as a dOMDD$_<$, and the elementary dOMDDs obtained are then combined by conjunction. After these steps, we get a dOMDD$_<$. The compilation can be pursued by using additional compacting operations, which do not preserve determinism [12]. In the end, an OMDD$_<$ is obtained.

It appears that the interest of non-deterministic structures is not limited to a few specific problems like those used in proofs:

TABLE III

RESULTS FOR RANDOMLY GENERATED BINARY CSPS (15 VARIABLES, DOMAIN SIZE EQUAL TO 6). %T DENOTES THE PERCENTAGE OF TUPLES SATISFYING EACH CONSTRAINT; %C THE DENSITY OF THE CONSTRAINT GRAPH; #SOL THE NUMBER OF SOLUTIONS OF THE CSP; #N THE NUMBER OF NODES IN THE COMPILED FORM.

%T	%C	#SOL	#N OMDD	#N dOMDD
70	10	290888073	80	81
	20	136056826	1338	1558
	30	5006576	5662	8132
	40	95131	3315	5005
	50	2367	737	897
80	10	391615179	79	80
	20	1581648506	2572	2932
	30	189551100	12223	16370
	40	11557737	20501	35486
	50	1035884	13815	25240
	60	70185	5776	9253
	70	4662	1719	2246
	80	229	54	401

indeed, even on random CSPs, allowing non-deterministic compacting operations can sometimes divide by 2 or more the number of nodes in the graph.

VI. CONCLUSION

Both theoretical complexity results and experiments show that relaxing the determinism requirement in ordered decision diagrams can improve succinctness. Relaxing determinism also allows more transformations to be achieved in polytime: typically, all transformations (save **SEN**, which depend on the domain size) satisfied by DNF are also satisfied by $\mathtt{OMDD}_<$. This includes forgetting and disjunction, which are not satisfied by deterministic languages. The price to pay when putting determinism away is the loss of the negation transformation, and of the counting, validity, equivalence, and implication queries (note that the same operations are lost when going from deterministic to non-deterministic DNNFs). As a result, $\mathtt{OMDD}_<$ is particularly appealing for applications relying on transformations (save negation) and on basic consistency queries (**CO**, **MX**, **ME**, **CX**), such as planning and online configuration. From the theoretical point of view, we also established that, when restricted to Boolean domains, $\mathtt{OMDD}_<$ is a new fragment in the NNF map, below DNNF and above DNF and OBDD. Moreover, $\mathtt{OMDD}_<$ satisfies more queries and transformations than DNNF does.

The next step is to introduce decomposable AND nodes in the MDD framework. This should allow AND/OR graphs to be captured, and also new fragments to be defined, such as non-deterministic AND/OR graphs.

REFERENCES

[1] N. R. Vempaty, "Solving Constraint Satisfaction Problems Using Finite State Automata," in *AAAI*, 1992, pp. 453–458.

[2] T. Kam, T. Villa, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Multi-valued Decision Diagrams: Theory and Applications," *Multiple-Valued Logic*, vol. 4, no. 1–2, pp. 9–62, 1998.

[3] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann, "A Constraint Store Based on Multivalued Decision Diagrams," in *CP*, 2007, pp. 118–132.

[4] J. Amilhastre, H. Fargier, and P. Marquis, "Consistency Restoration and Explanations in Dynamic CSPs — Application to Configuration," *AIJ*, vol. 135, no. 1–2, pp. 199–234, 2002.

[5] H. Cambazard, T. Hadzic, and B. O'Sullivan, "Knowledge Compilation for Itemset Mining," in *ECAI*, 2010, pp. 1109–1110.

[6] F. Giunchiglia and P. Traverso, "Planning as Model Checking," in *ECP*, 1999, pp. 1–20.

[7] J. Hoey, R. St-Aubin, A. J. Hu, and C. Boutilier, "SPUDD: Stochastic Planning Using Decision Diagrams," in *UAI*, 1999, pp. 279–288.

[8] P. Torasso and G. Torta, "Model-Based Diagnosis Through OBDD Compilation: A Complexity Analysis," in *Reasoning, Action and Interaction in AI Theories and Systems*, 2006, pp. 287–305.

[9] A. Darwiche and P. Marquis, "A Knowledge Compilation Map," *JAIR*, vol. 17, pp. 229–264, 2002. [Online]. Available: citeseer.ist.psu.edu/darwiche02knowledge.html

[10] A. Srinivasan, T. Ham, S. Malik, and R. Brayton, "Algorithms for discrete function manipulation," in *ICCAD-90*, Nov. 1990, pp. 92–95.

[11] R. E. Tarjan and M. Yannakakis, "Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs," *SIAM J. Comput.*, vol. 13, no. 3, pp. 566–579, 1984.

[12] J. Amilhastre, P. Vilarem, and M.-C. Vilarem, "FA Minimisation Heuristics for a Class of Finite Languages," in *WIA*, 1999, pp. 1–12.

[13] A. Niveau, H. Fargier, and C. Pralet, "Representing CSPs with Set-labeled Diagrams: A Compilation Map," in *GKR 2011 — Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 7205. Springer, 2012, pp. 137–171.

[14] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer, 1998.

[15] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

APPENDIX

This appendix gathers all proofs of the paper. Note that most results are already known, straightforward, or inherited from the BDD framework. We tried to be exhaustive to save the reader the need to refer to numerous sources while checking proofs. These sources include, as far as we know, the original KC map [9], works about MDDs [10], [2], and our KC map of set-labeled diagrams [13] (from which a number of the following proofs are directly taken or lightly adapted).

As usual with compilation maps, many proofs take advantage of the fact that all the languages presented allow Boolean terms and clauses to be represented in linear time and space. The proof is similar to the Boolean case [9].

Lemma 16. *Given an order $<$ between variables, any term or clause in propositional logic can be expressed in the form of a dOMDD$_<^{\mathfrak{B}}$ in linear time.*

Proof: In order to represent a term $t = l_1 \wedge \cdots \wedge l_k$ by a dOMDD$_<^{\mathfrak{B}}$, let us order its literals in such a way that $i < j$ if and only if $\mathrm{Var}(l_i) < \mathrm{Var}(l_j)$. Then build a chain of nodes $N_1, \ldots, N_k, N_{k+1}$ where $\mathrm{Var}(N_i) = \mathrm{Var}(l_i), i = 1, k$ and N_{k+1} will be the sink of the new dOMDD$_<^{\mathfrak{B}}$. The edges are the following: each node N_i (but the sink) has a unique child N_{i+1}, and the edge's label is 1 if l_i is a positive literal and 0 if l_i is a negative literal.

In order to represent a clause $cl = l_1 \vee \cdots \vee l_k$ by a dOMDD$_<^{\mathfrak{B}}$, let us order its literals in such a way that $i < j$ if and only if $\mathrm{Var}(l_i) < \mathrm{Var}(l_j)$. Then build a chain of nodes $N_1, \ldots, N_k, N_{k+1}$ where $\mathrm{Var}(N_i) = \mathrm{Var}(l_i), i = 1, k$ and N_{k+1}will be the sink of the new dOMDD$_<^{\mathfrak{B}}$. The edges are the following: each node $N_i, i < k$ has two children, N_{i+1}

and the sink, N_{k+1}. If l_i is a positive literal, the edge pointing to N_{i+1} is labeled by 0 and the one pointing to N_{k+1}, by 1. If l_i is a negative literal, it is the opposite. Finally, N_k has a unique successor, the sink, and the edge is labeled by 1 if l_k is a positive literal, and by 0 otherwise. ■

We need the following lemma, that extends Lemma 16.

Lemma 17. *Let $Y = \{y_1, \dots, y_k\}$ be a set of variables from X, and $(A_1, \dots, A_k)$ a sequence of subsets of $\mathrm{Dom}(y_1), \dots, \mathrm{Dom}(y_k)$. Given a total ordering $<$ on X, the "term" $\big[y_1 \in A_1\big] \wedge \dots \wedge \big[y_k \in A_k\big]$ can be represented as a dOMDD$_<$ in time linear in k and $n_A = \max_{1 \leq i \leq k} |A_i|$.*

Proof: The proof is the same as the one of Lemma 16, with the difference that we add as many edges as there are values in the current A_i. The dOMDD$_<$ has $k+1$ nodes and $\sum_{i=1}^{k} |A_i|$ edges. ■

Lemma 18. *The only reduced OMDD that is inconsistent is the empty graph.*

Proof: Suppose that ϕ has at least one edge, and consider a path from the root to the sink of ϕ. Because the variables in ϕ are ordered, they are encountered only once. It is thus possible to build an assignment $\vec{x}$ of X as follows: for any (N, N', a) on the path, assign a to $\mathrm{Var}(N)$; a variable not present on the path can take any value in its domain. By construction (a path from the root to the sink is followed), $I(\phi)(\vec{x}) = \top$. ■

A. Succinctness Proofs (Proposition 11)

• $\texttt{OMDD}_< \not\leq_s \texttt{dOMDD}$. Let us consider the CSP Π over a set of $2n$ Boolean variables partitioned into $Y = \{y_1, \dots, y_n\}$ and $Z = \{z_1, \dots, z_n\}$. The sets of contraints of Π is $C = \{y_i = z_i, 1 \leq i \leq n\}$. Consider the orders $y_1 <^a z_1 <^a y_2 <^a z_2 <^a \dots <^a y_n <^a z_n$ and $y_1 <^b y_2 <^b \dots <^b y_n <^b z_1 <^b \dots <^b z_n$.

Π has a $\texttt{dOMDD}_{<^a}$ representation the size of which is polynomial in n; let us denote it ϕ_a. We show that the size of any of the $\texttt{OMDD}_{<^b}$ representations of $\mathrm{Sol}(\Pi)$ is exponential with n.

Let ϕ_b be an $\texttt{OMDD}_{<^b}$ representation of $\mathrm{Sol}(\Pi)$. Consider an edge $E = \langle N, N', v\rangle$ in ϕ_b representing the assignment of some variable from Y (i.e., an edge in the first half of the graph). Let y_i be the variable labeling N, and let us consider a path $p : \mathrm{Root}(\phi) \rightsquigarrow \langle N, N', v\rangle \rightsquigarrow \mathrm{Sink}(\phi)$. On this path, the value of z_i must be v. So, a path p' obtained from p by using another edge $\langle N, N', u\rangle$ instead of $\langle N, N', v\rangle$ cannot belong to ϕ_b. For the same reason, any two edges entering a Y node must be labeled by the same value, and any two paths from the root to N' are either disjoint or represent the same assignment of $\{y_1, \dots, y_i\}$. Each of the possible assignments of Y is represented by (at least) one path entering a distinct z_1 node. The Y assignments are not constrained: ϕ_b has at least 2^n nodes at level z_1.

ϕ_a is thus a dOMDD$_{<^a}$ of size polynomial in n that has no polynomial representation as an OMDD$_{<^b}$. ■

• $\texttt{OMDD} \not\leq_s \texttt{dMDD}$. We get a direct proof of $\texttt{OMDD} \not\leq_s \texttt{dMDD}$ by considering that the problem of n-coloring a clique with n vertices (or equivalently, the AllDifferent constraint over the n variables, see Figure 1) has no polynomial representation as an OMDD; indeed, in a similar fashion as in the previous proof, we can show that at the ith level, there must be as many nodes as there are possibilities of choosing i values in $\{1, \dots, n\}$ (if there are fewer nodes at this level, we can build a path on which two different values are assigned to the same variable, the first time before the ith level, and the second one afterwards). Hence the total number of nodes is at least $\sum_{i=1}^{n} \binom{n}{i}$, which is 2^n.

Yet, the problem can obviously be represented as a polysize dMDD, since each constraint $x_i \neq x_j$ can be represented as a $n(n-1)$-edge dMDD, and $\texttt{dMDD}$ satisfies $\wedge\mathbf{C}$ (see Proposition 15). ■

• $\texttt{dOMDD} \not\leq_s \texttt{OMDD}_<$. Proposition 10 states that $\texttt{OMDD}^{\mathfrak{B}}_< \leq_s \texttt{DNF}$. If $\texttt{dOMDD} \leq_s \texttt{OMDD}_<$, and thus $\texttt{dOMDD}^{\mathfrak{B}} \leq_s \texttt{OMDD}^{\mathfrak{B}}_<$ were true, since $\texttt{OBDD} \equiv_{\mathcal{L}} \texttt{dOMDD}^{\mathfrak{B}}$, $\texttt{OBDD} \leq_s \texttt{DNF}$ would also be true—yet it has been proven false [9]. ■

• $\texttt{L} \leq_s \texttt{L}'$. All positive results come directly from simple language inclusions (if $\texttt{L} \supseteq \texttt{L}'$, then surely $\texttt{L} \leq_s \texttt{L}'$). ■

• $\texttt{L} \not\leq_s \texttt{L}'$. All remaining negative results come from the fact that if $\texttt{L}_1 \not\leq_s \texttt{L}_2$, then every $\texttt{L}$ such that $\texttt{L}_1 \leq_s \texttt{L}$ verifies $\texttt{L} \not\leq_s \texttt{L}_2$, and every $\texttt{L}$ such that $\texttt{L} \leq_s \texttt{L}_2$ verifies $\texttt{L}_1 \not\leq_s \texttt{L}$, or we could derive $\texttt{L}_1 \leq_s \texttt{L}_2$ by transitivity. ■

B. Proofs of Queries (Proposition 15)

• **$\texttt{MDD}$ and its subclasses satisfy MC.** Since these languages satisfy **CD** (see Section C), we can test whether $\vec{x}$ is a model of ϕ by conditioning the MDD by $\vec{x}$; indeed, we get either the empty MDD (then the assignment is not a model) or the sink-only MDD (then $\vec{x}$ is a model). Hence all these languages satisfy **MC**. ■

• **Queries on $\texttt{dMDD}$ and $\texttt{MDD}$.** $\texttt{BDD}$ is a subclass of $\texttt{dMDD}$ and $\texttt{MDD}$, so $\texttt{dMDD}$ and $\texttt{MDD}$ do not satisfy any of the other considered queries unless $\mathsf{P} = \mathsf{NP}$. ■

• **$\texttt{OMDD}$ and $\texttt{OMDD}_<$ do not satisfy VA, IM, EQ, SE, CT.** Lemma 16 states that any term can be transformed into $\texttt{OMDD}^{\mathfrak{B}}_<$ in linear time, and $\texttt{OMDD}^{\mathfrak{B}}_<$ satisfies $\vee\mathbf{C}$ (Prop. 15); hence $\texttt{OMDD}^{\mathfrak{B}}_< \leq_{\mathcal{L}} \texttt{DNF}$, therefore $\texttt{OMDD}^{\mathfrak{B}} \leq_{\mathcal{L}} \texttt{DNF}$ (by inclusion), and we obtain the result since $\texttt{DNF}$ does not satisfy **VA**, **IM**, **EQ**, **SE**, **CT** unless $\mathsf{P} = \mathsf{NP}$. ■

• **$\texttt{OMDD}$ and $\texttt{OMDD}_<$ satisfy CO.** **CO** holds since the only reduced OMDD that is not consistent is the empty graph (Lemma 18). ■

• **$\texttt{OMDD}$ and $\texttt{OMDD}_<$ satisfy CE.** Checking whether ϕ entails $\big[x_1 \in A_1\big] \vee \dots \vee \big[x_k \in A_k\big]$ is equivalent to checking whether $\phi \wedge \big[x_1 \notin A_1\big] \wedge \dots \wedge \big[x_k \notin A_k\big]$ is inconsistent. Since (i) $\big[x \notin A\big] \equiv \big[x \in \mathrm{Dom}(x) \setminus A\big]$, (ii) a "term" can be represented in polytime as an OMDD of the same variable order as ϕ (Lemma 17), and (iii) $\texttt{OMDD}_<$ satisfies $\wedge\mathbf{BC}$ (see below), we can infer that $\texttt{OMDD}$ satisfies **CE**. ■

• **$\texttt{OMDD}$ and $\texttt{OMDD}_<$ satisfy ME, MX, CX.** These hold because $\texttt{OMDD}$ satisfies **CD** and **CO**. Enumerating models can be done by exploring the entire assignment tree, as done in the Boolean case [9] (checking consistency after each branching avoids having to backtrack, so the size of the search tree is polynomial in the size of the ouput). Model extraction is a subcase of **ME**, and context extraction can be done by checking, for each variable x_i and each value v in its domain, whether $\phi_{|\left[x_i=v\right]}$ is consistent. ∎

• `dOMDD` **satisfies CO, CE, MX, CX, ME.** All these operations are supported, since `dOMDD` is included in `OMDD` which supports these operations. ∎

• `dOMDD` **satisfies CT, VA.** To count the number of model of a dOMDD, we simply have to associate to the sink the number $n_{\mathrm{Sink}(\phi)} = 1$, then traverse the graph from the sink to the root, associating to each edge E the number

$$n_E = n_{\mathrm{Dest}(E)} \times \prod_{x_\mathrm{s} < x < x_\mathrm{d}} |\mathrm{Dom}(x)|$$

(where x_s and x_d denote $\mathrm{Var}(\mathrm{Src}(E))$ and $\mathrm{Var}(\mathrm{Dest}(E))$, respectively), and to each node N the number

$$n_N = \sum_{E \in \mathrm{Out}(N)} n_E.$$

The number of models is then $n_{\mathrm{Root}(\phi)}$. This process being polynomial in $\|\phi\|$ (each edge is crossed (backwards) once), `dOMDD` satisfies **CT**.

From **CT**, we can test in polytime whether the number of models is equal to the number of possible assignments of the variables, i.e., whether the dOMDD is valid, thus `dOMDD` supports **VA**. ∎

• `dOMDD` **does not satisfy SE.** `dOMDD` does not satisfy **SE**, since `dOMDD`$^{\mathfrak{B}} \equiv_{\mathcal{L}}$ `OBDD` (Proposition 9) and `OBDD` does not satisfy **SE** unless P = NP. ∎

• `dOMDD` **satisfies EQ.** We give an indirect proof, using the algorithm in Figure 4, a polytime procedure that transforms any dOMDD into an OBDD (see e.g. [10]; we use here a direct encoding).

Let ϕ be a dOMDD on $X = \{x_1, \dots, x_n\}$, with variable ordering $x_1 < \dots < x_n$. For each variable x_i and each value $\omega_j \in \mathrm{Dom}(x) = \{\omega_1, \dots, \omega_{k_i}\}$, let us define a Boolean variable x_i^j, taking the value 1 when $x_i = \omega_j$ and 0 otherwise. The algorithm in Figure 4 builds a dOMDD$_<$ in which each x-node of ϕ is replaced by an ordered sequence of x_i^j nodes.

When applied on a dOMDD this algorithm obviously return a dOMDD$^{\mathfrak{B}}_{<'}$, the order being $x_1^1 <' \dots <' x_1^{k_1} <' \dots <' x_n^1 <' \dots <' x_n^{k_n}$. Its size is polynomial in the one of the original dOMDD, each x-node being replaced by at most $|\mathrm{Dom}(x)|$ nodes having at most two outgoing edges. The procedure runs in polytime (each domain value is explored once for each node).

Let us denote X_bin the set of binary variables we introduced; an *acceptable* assignment $\vec{x}_\mathrm{bin}$ of the variables from X_bin is defined as follows: for each variable $x_i \in X$, there is exactly one of the x_i^j that takes the value 1 in $\vec{x}_\mathrm{bin}$. Formally,

```
1: if $\phi$ is empty then
2:    Let $\phi_\mathrm{bin}$ be the empty MDD
3: else
4:    Let $\phi_\mathrm{bin}$ be the sink-only MDD
5:    for all node $N$ in $\phi$, ordered from the sink to the root, excluding the sink do
6:       Let $x_i := \mathrm{Var}(N)$
7:       for all $\omega_j \in \mathrm{Dom}(x_i)$, in decreasing order do
8:          if $\exists E \in \mathrm{Out}(N), \omega_j \in \mathrm{Lbl}(E)$ then
9:             Add to $\phi_\mathrm{bin}$ a node $N_i^j$ labeled by $x_i^j$
10:            Add to $N_i^j$ an outgoing edge labeled 1 and pointing to the corresponding node of $\mathrm{Dest}(E)$ (the corresponding node of $\phi$'s sink being $\phi_\mathrm{bin}$'s sink)
11:            if $N'$ has been defined then
12:               Add to $N_i^j$ an outgoing edge labeled 0 and pointing to $N'$
13:            Let $N' := N_i^j$
14:      Let $N'$ be the corresponding node of $N$ in $\phi_\mathrm{bin}$
15:   Set $\mathrm{Root}(\phi_\mathrm{bin})$ to be the corresponding node of $\mathrm{Root}(\phi)$
16:   return $\phi_\mathrm{bin}$
```

Fig. 4. Encoding of a dOMDD ϕ into a dOMDD$^{\mathfrak{B}}$ ϕ_bin

it is an assignment $\vec{x}_\mathrm{bin} \in \mathrm{Dom}(\bigcup_{x_i \in X} \bigcup_{j=1}^{k_i} x_i^j)$ verifying:

$$\forall x_i \in X, \qquad \left| \left\{ j \in \{1, \dots, k_i\} : \vec{x}_\mathrm{bin}(x_i^j) = 1 \right\} \right| = 1.$$

We denote $\mathrm{Dom}_\mathrm{bin}(X)$ the set of acceptable assignments of X_bin. By construction, there is a bijection between $\mathrm{Dom}(X)$ and $\mathrm{Dom}_\mathrm{bin}(X)$, and for each model of ϕ, its corresponding assignment in $\mathrm{Dom}_\mathrm{bin}(X)$ is a model of ϕ_bin, and reciprocally. Thus, given two MDDs ϕ and ψ and their matching ϕ_bin and ψ_bin, it holds that $\phi \equiv \psi \iff \phi_\mathrm{bin} \equiv \psi_\mathrm{bin}$.

Now, `dOMDD`$^{\mathfrak{B}} \equiv_{\mathcal{L}}$ `OBDD`, and `OBDD` supports **EQ** [14, Corollary 8.12].

For testing the equivalence of two dOMDDs ϕ and ψ we only have to build their corresponding ϕ_bin and ψ_bin (which is done in polytime), transform them into OBDDs, and check whether they are equivalent; it is the case if and only if $\phi \equiv \psi$.

Hence `dOMDD` satisfies **EQ**. ∎

• `dOMDD`$_<$ **satisfies SE.**

Same indirect proof as the previous one : for testing whether ϕ entails ψ, we only have to build their corresponding ϕ_bin and ψ_bin (which is done in polytime), transform them into OBDDs, and check whether ϕ_bin entails ψ_bin. It is the case if and only if ϕ entails ψ. ∎

• `dOMDD`$_<$ **satisfies CO, VA, CE, EQ, MX, CX, CT, ME.** Straightforward from the fact that `dOMDD` support these queries and that `dOMDD`$_< \subseteq$ `dOMDD`. ∎

• `dOMDD`$_<$ **satisfies IM.** Checking whether $\left[y_1 \in A_1\right] \wedge \dots \wedge \left[y_k \in A_k\right]$ entails ϕ is equivalent to checking whether $\phi \vee \neg(\left[y_1 \in A_1\right] \wedge \dots \wedge \left[y_k \in A_k\right])$ is valid.

```
1: Number the variables of $\phi$ according to $<$ (if relevant)
2: for all $E = \langle N, N', v\rangle$ such that $\mathrm{Var}(N) \in Y$ do
3:   if $v \neq \vec{y}_{|\,\mathrm{Var}(N)}$ then
4:     Remove $E$ from $\phi$: $\mathrm{Out}(N) := \mathrm{Out}(N) \setminus \{E\}$
5:   else
6:     Label $E$ with $\top$
7: for all $x_i \in Y$, $i$ going from $n$ to 1 do
8:   for all $N$ such that $\mathrm{Var}(N) = x_i$ do
9:     Let $I = \mathrm{In}(N)$, $O = \mathrm{Out}(N)$, $\mathcal{E}_{\mathrm{new}} = \emptyset$
10:    for all $E_I = \langle N_I, N, a\rangle \in I$ do
11:      for all $E_O = \langle N, N_O, \top\rangle \in O$ do
12:        Add an edge $E = \langle N_I, N_O, a\rangle$ to $\mathcal{E}_{\mathrm{new}}$
13:    Remove $N$, $I$ and $O$ from the graph
14:    Add $\mathcal{E}_{\mathrm{new}}$ to the graph
```

Fig. 5. Conditioning of an MDD ϕ by an assignment $\vec{y}$ of $Y \subseteq \mathrm{Var}(\phi)$.

```
1: Let $\psi$ be the sink-only graph.
2: for all node $N$ in $\phi$, ordered from the sink to the root,
   excluding the sink do
3:   Create a node $N'$ labeled by the same variable $x$ as $N$
4:   Let $U = \{v \in \mathrm{Dom}(x) : \forall E \in \mathrm{Out}(N), \mathrm{Lbl}(E) \neq v\}$
5:   for all $v \in U$ do
6:     Add to $N'$ an outgoing edge $E_{\mathrm{out}} = \langle N', \mathrm{Sink}(\psi), v\rangle$
7:   for all edge $E = \langle N, D, v\rangle$ coming out of $N$ do
8:     if $D$ has a corresponding node $D'$ in $\psi$ then
9:       Add to $N'$ an outgoing edge $E' = \langle N', D', v\rangle$
10:  if $N'$ has at least one outgoing edge then
11:    Add $N'$ to $\psi$
12: return $\psi$
```

Fig. 6. Given a dMDD ϕ, builds a dMDD called $\mathrm{Compl}(\phi)$.

Lemma 17 states that we can obtain a dOMDD$_<$ equivalent to $[y_1 \in A_1] \wedge \cdots \wedge [y_k \in A_k]$ in time linear in n_A and k (which is bounded by $|X|$ and thus by the size of the graph); we then just have to apply a negation and a single disjunction (dOMDD$_<$ supports $\neg\mathbf{C}$ and $\vee\mathbf{BC}$, see below), and then check the validity of the resulting dOMDD$_<$, which is possible since dOMDD$_<$ supports **VA**. ■

• **dOMDD satisfies IM.** Only one dOMDD is considered in this query: if we denote $<$ the ordering of a dOMDD ϕ, ϕ is a dOMDD$_<$. We can thus apply the previous technique to check whether $[y_1 \in A_1] \wedge \cdots \wedge [y_k \in A_k]$ entails ϕ. ■

C. Proofs of Transformations (Proposition 15)

Many of the following proofs rely on the fact that an MDD can be conditioned in polytime (linear time for deterministic MDDs), thanks to the algorithm from Figure 5, which computes a syntactical conditioning of ϕ by some assignment $\vec{y}$ of $Y \subseteq X$. Let us denote $\phi_{|\vec{y}}$ the MDD built by this algorithm.

Lemma 19 (Conditioning)**.** *The algorithm in Figure 5:*

- *preserves determinism and ordering;*
- *is polytime (linear when ϕ is deterministic);*
- *for each dMDD ϕ, $\|\phi_{|\vec{y}}\|$ is bounded by $\|\phi\|$;*
- *for each MDD ϕ, $\|\phi_{|\vec{y}}\|$ is bounded by $|\mathrm{Nodes}(\phi)|^2$;*
- *for each dMDD ϕ, $\mathrm{I}(\phi_{|\vec{y}}) = \mathrm{I}(\phi)_{|\vec{y}}$.*

Proof:

VARIABLE ORDERING: This algorithm preserves variable ordering, since the only edges that are added link an ancestor of N to a successor of N: if the variables in ϕ are ordered according to $<$, the variables in $\phi_{|\vec{y}}$ are ordered according to $<$.

DETERMINISM: This algorithm preserves the property of determinism: when ϕ is deterministic, at most one edge in $\mathrm{Out}(N)$ can be marked by $\top$. Hence, for any edge $E = (N_I, N, a)$ going out of N_I, only one edge $E = (N_I, N_O, a)$ is created: if N_I was deterministic, it is still the case at the end of the procedure.

TIME AND SPACE COMPLEXITY (GENERAL CASE): The algorithm from Figure 5 runs in polytime. Indeed, the first loop visits each edge at most once. The second loops visits each node N once, deletes $|\mathrm{In}(N)| + |\mathrm{Out}(N)|$ edges and creates at most $|\mathrm{In}(N)| \times |\mathrm{Out}(N)|$ edges, that are only labeled with values mentioned in ϕ: thus the number of outgoing edges is bounded by $|\mathrm{Edges}(\phi)| \times |\mathrm{Nodes}(\phi)|$ (in the worst case, there is one edge for each value and each node). The complexity of the operation is thus polynomial; as no node is created, it is the case for the entire procedure.

TIME AND SPACE COMPLEXITY (DETERMINISTIC CASE): Same proof as the previous one, noticing that in the deterministic case, $|\mathrm{Out}(N)| = 1$ at the end of the first loop: the second loops visits each node N once, deletes $|\mathrm{In}(N)| + |\mathrm{Out}(N)|$ edges and creates at most $|\mathrm{In}(N)| \times 1$ edges. Its complexity is bounded by $\|\phi\|$.

EQUIVALENCE: Let $T = X \setminus Y$ and consider any assignment $\vec{t}$ of T. We show that $\mathrm{I}(\phi_{|\vec{y}})(\vec{t}) = \mathrm{I}(\phi)_{|\vec{y}}(\vec{t})$.

Suppose $\mathrm{I}(\phi)_{|\vec{y}}(\vec{t}) = \top$. Then there is a path p in ϕ that is compatible with both $\vec{y}$ and $\vec{t}$. By construction, there is a path p' in $\phi_{|\vec{y}}$ that is compatible with $\vec{t}$: nodes along p that are labeled with variables in Y have been suppressed, but edges with values compatible with $\vec{y}$ have been "shortcut". Hence $\mathrm{I}(\phi_{|\vec{y}})(\vec{t}) = \top$.

Suppose $\mathrm{I}(\phi_{|\vec{y}})(\vec{t}) = \top$; there is a path p in $\phi_{|\vec{y}}$ that is compatible with $\vec{t}$. This path corresponds to several paths in ϕ, in which there are similar T-edges (so they are all compatible with $\vec{t}$). At least one of these paths is compatible with $\vec{y}$: if it were not the case, they would all have had an edge removed after the first step of the procedure, and thus cannot correspond to path p. Hence, there is at least one path in ϕ that is compatible with both $\vec{y}$ and $\vec{t}$: $\mathrm{I}(\phi)_{|\vec{y}}(\vec{t}) = \top$. ■

Lemma 20 (Negation)**.** *The algorithm in Figure 6:*

- *preserves determinism and ordering;*
- *is polynomial in $\|\phi\|$;*
- *for each dMDD ϕ, $\mathrm{Compl}(\phi) \equiv \neg\phi$.*

Proof:

ORDERING: This is obvious since (i) no "new" nodes are added, and (ii) the only added edges point to the sink.

DETERMINISM: This is obvious, as no "new" nodes are added, and the only added edges are disjoint with the other edges coming out of their source node.

TIME AND SPACE COMPLEXITY: Each node is encountered once. For each node N, all the values in $\mathrm{Dom}(\mathrm{Var}(N))$ are considered; at most one edge per value is added. Hence a time and space complexity in $\mathcal{O}(d \times |\mathrm{Nodes}(\phi)|)$, where $d = \max_{x \in X} |\mathrm{Dom}(x)|$, and thus polynomial in $\|\phi\|$, since both d and $|\mathrm{Nodes}(\phi)|$ are bounded by $\|\phi\|$.

EQUIVALENCE: We prove this by induction on the number of nodes in ϕ. For $n \in \mathbb{N}$, let $\mathcal{P}(n)$ be the following proposition: "For any dMDD ϕ containing n nodes, $\mathrm{Compl}(\phi) \equiv \neg\phi$". $\mathcal{P}(n)$ is obviously true for $n \leq 1$. Let $n \geq 2$, and suppose that $\mathcal{P}(k)$ is true for all $k < n$.

Let ϕ be an n-node dMDD, of root R, with $\mathrm{Var}(R) = x$. We show that $\mathrm{Compl}(\phi) \equiv \neg\phi$.

- Let $\vec{y} \in \mathrm{Mod}(\phi)$; surely there exists an edge E going out of R and compatible with $\vec{y}$. Consequently, $\vec{y}$ cannot be compatible with E_{out}. Since ϕ is deterministic, $\vec{y}$ is necessarily compatible with the subgraph rooted at $\mathrm{Dest}(E)$. Let us call it ϕ_E. If $\mathrm{Dest}(E)$ has no corresponding node in $\mathrm{Compl}(\phi)$, then clearly $\vec{y}$ cannot be a model of $\mathrm{Compl}(\phi)$. If it has one, then by construction, the root of $\mathrm{Compl}(\phi)$ has an outgoing edge with the same label as E, pointing to a subgraph corresponding to $\mathrm{Compl}(\phi_E)$. Since ϕ_E has less than n nodes, our induction hypothesis tells us that $\mathrm{Compl}(\phi_E) \equiv \neg\phi_E$. Hence $\vec{y}$ is not a model of $\mathrm{Compl}(\phi)$.
- Let $\vec{y} \notin \mathrm{Mod}(\phi)$. If R has no outgoing edge compatible with $\vec{y}$, then E_{out} in $\mathrm{Compl}(\phi)$ is, and it goes directly to the sink; $\vec{y}$ is a model of $\mathrm{Compl}(\phi)$. If there is such an edge E, then $\vec{y}$ cannot be a model of ϕ_E, the subgraph rooted at $\mathrm{Dest}(E)$ (recall that if E exists, it is unique, since ϕ is deterministic). Now, by construction, the root of $\mathrm{Compl}(\phi)$ has an outgoing edge labeled the same as E and pointing to $\mathrm{Compl}(\phi_E)$. Since ϕ_E has less than n nodes, our induction hypothesis implies that $\mathrm{Compl}(\phi_E) \equiv \neg\phi_E$, consequently $\vec{y}$ is a model of $\mathrm{Compl}(\phi_E)$, and thus of $\mathrm{Compl}(\phi)$.

All in all, $\vec{y} \in \mathrm{Mod}(\phi) \iff \vec{y} \notin \mathrm{Mod}(\mathrm{Compl}(\phi))$ holds, hence $\mathrm{Compl}(\phi) \equiv \neg\phi$.

Since both the basis and the inductive step have been proven, we showed by induction that $\mathcal{P}(n)$ holds for all $n \in \mathbb{N}$. ■

Lemma 21. *It holds that*

$$\mathrm{Forget}(\mathrm{I}, \{x\}) = \bigvee_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}},$$

$$\mathrm{Ensure}(\mathrm{I}, \{x\}) = \bigwedge_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}.$$

Proof: Straightforward from the definition: let $W = X \setminus \{x\}$. $\mathrm{Forget}(\mathrm{I}, \{x\}) = \mathrm{I}^{\downarrow W}$. Then $\mathrm{Forget}(\mathrm{I}, \{x\})(\vec{w}) = \top$ if and only if there exists $\vec{x} \in \mathrm{Dom}(\{x\})$ such that $\mathrm{I}(\vec{w} \, . \, \vec{x}) = \top$. Hence $\mathrm{Forget}(\mathrm{I}, \{x\}) = \bigvee_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}$.

Similarly, let $W = X \setminus \{x\}$. $\mathrm{Ensure}(\mathrm{I}, \{x\}) = \mathrm{I}^{\Downarrow W}$. Then $\mathrm{Ensure}(\mathrm{I}, \{x\})(\vec{w}) = \top$ if and only if for all $\vec{x} \in \mathrm{Dom}(\{x\})$, it holds that $\mathrm{I}(\vec{w} \, . \, \vec{x}) = \top$. Hence $\mathrm{Ensure}(\mathrm{I}, \{x\}) = \bigwedge_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}$. ■

• **`MDD` and its subclasses satisfy CD.** Since the algorithm from Figure 5 is polytime, computes a conditioning of ϕ and preserves determinism and variable ordering (Lemma 19), we can derive that all languages identified in this paper, viz., `MDD`, `dMDD`, `OMDD`, `dOMDD`, `OMDD`$_<$, and `dOMDD`$_<$, satisfy **CD**. `dMDD` and its subclasses even support **CD** in linear time. ■

• **`dMDD`, `dOMDD` and `dOMDD`$_<$ satisfies $\neg$C.** Straightforward from Lemma 20; the algorithm from Figure 6 preserves determinism and variable ordering. ■

• **`OMDD`$_<$ and `OMDD` do not satisfy $\neg$C.** By negating a CNF one obtains a DNF, which is linearly translatable into an OMDD of any order (Proposition 10). Thus, if `OMDD`$_<$ or `OMDD` satisfied $\neg$**C**, we could, for any CNF, build an equivalent OMDD in polytime. As `OMDD`$_<$ and `OMDD` support **CO**, we would have a polytime algorithm for deciding whether a CNF is consistent, which is impossible unless $\mathsf{P} = \mathsf{NP}$. ■

• **`MDD` satisfies $\wedge$C, $\wedge$BC.** To make the conjunction of k MDDs $\phi_1, \ldots, \phi_k$, replace the sink of ϕ_i by the root of ϕ_{i+1}, for all $1 \leq i \leq k-1$. The root of the new MDD is that of ϕ_1, and its sink is that of ϕ_k. ■

• **`dMDD` satisfies $\wedge$C, $\wedge$BC.** To make the conjunction of k dMDDs, use the previous polytime procedure: the result is obviously a dMDD, since no edge is modified or added. ■

• **`OMDD` does not satisfy $\wedge$C, $\wedge$BC.** Thanks to Proposition 9, any OBDD$_<$ can be turned into an equivalent OMDD$_<^{\mathfrak{B}}$ in linear time and size. If `OMDD` supported $\wedge$**BC**, we would have a polytime algorithm to decide whether the conjunction of two OBDDs (the variable orderings being possibly different in each OBDD) is consistent, since `OMDD` supports **CO**; yet, this problem is NP-complete [14, Lemma 8.14]. Therefore `OMDD` does not support $\wedge$**BC**, and *a fortiori* does not support $\wedge$**C**, unless $\mathsf{P} = \mathsf{NP}$. ■

• **`dOMDD` does not satisfy $\wedge$BC.** Same proof as for OMDDs, since any OBDD$_<$ can be turned into an equivalent dOMDD$_<$ in polytime (Proposition 9) and `dOMDD` supports **CO**. Thus `dOMDD` does not support $\wedge$**BC** unless $\mathsf{P} = \mathsf{NP}$. ■

• **`dOMDD` and `dOMDD`$_<$ do not satisfy $\wedge$C.** We have a stronger result for $\wedge$**C**: Since `dOMDD`$^{\mathfrak{B}} \equiv_{\mathcal{L}}$ `OBDD` (resp. `dOMDD`$_<^{\mathfrak{B}} \equiv_{\mathcal{L}}$ `OBDD`$_<$) thanks to Lemma 9, as `OBDD` (resp. `OBDD`$_<$) does not support $\wedge$**C**, `dOMDD` (resp. `dOMDD`$_<$) does not either. ■

• **`OMDD`$_<$ does not satisfy $\wedge$C.** Lemma 16 states that propositional clauses can be represented in the form of OMDDs$_<$ in linear time; if `OMDD`$_<$ supported $\wedge$**C**, we could thus build a polynomial `OMDD`$_<$ representation of any CNF, and it would be possible to check whether it is consistent, as `OMDD`$_<$ supports **CO**. Yet it is impossible, unless $\mathsf{P} = \mathsf{NP}$. ■

• **`dOMDD`$_<$ and `OMDD`$_<$ satisfy $\wedge$BC.** We can use the algorithm in Figure 7, adapted from that on OBDDs [15]. It applies on non-empty OMDDs of a same variable order (if

```
1:  if the cache contains the key $(N_1, N_2)$ then
2:      return the OMDD corresponding to this key in the cache
3:  if $\mathrm{Out}(N_1) = \emptyset$ (the sink of $N_1$ is reached) then
4:      return a copy of the OMDD rooted at $N_2$
5:  if $\mathrm{Out}(N_2) = \emptyset$ (the sink of $N_2$ is reached) then
6:      return a copy of the OMDD rooted at $N_1$
7:  if $\mathrm{Var}(N_1) = \mathrm{Var}(N_2)$ then
8:      Let $x := \mathrm{Var}(N_1) = \mathrm{Var}(N_2)$
9:      Create a node $N'$ labeled by $x$
10:     for all $\omega \in \mathrm{Dom}(x)$, $E_1 \in \mathrm{Out}(N_1)$, $E_2 \in \mathrm{Out}(N_2)$ do
11:         if $\omega = \mathrm{Lbl}(E_1) = \mathrm{Lbl}(E_2)$ then
12:             Let $\phi_\omega :=$ conjunct_step$(\mathrm{Dest}(E_1), \mathrm{Dest}(E_2))$
13:             Add an edge coming out of $N'$, labeled by $\omega$ and pointing to the root of $\phi_\omega$
14:     return the OMDD rooted at $N'$
15: $N_i = \mathrm{Argmin}_<(\mathrm{Var}(N_1), \mathrm{Var}(N_2))$,
16: $N_j = \mathrm{Argmax}_<(\mathrm{Var}(N_1), \mathrm{Var}(N_2))$
17: Create a node $N_i'$ labeled by $\mathrm{Var}(N_i)$
18: for all $E \in \mathrm{Out}(N_i)$ do
19:     Let $\phi_E :=$ conjunct_step$(\mathrm{Dest}(E), N_j)$
20:     Add an edge coming out of $N_i'$, labeled by $\mathrm{Lbl}(E)$ and pointing to the root of $\phi_E$
21: return $N'$
```

Fig. 7. `conjunct_step`(N_1, N_2): returns an OMDD$_<$ that is the conjunction of the two OMDDs$_<$ of which N_1 and N_2 are roots.

one OMDD is empty, it is trivial to compute the conjunction). A cache is maintained to avoid computing twice the same couple of nodes, thus `conjunct_step` is not called more than $\|\phi_1\| \cdot \|\phi_2\|$ times. The procedure is polynomial and it maintains determinism when the inputs are deterministic. For each execution of `conjunct_step`, each value of a given variable's domain is explored once, and the size of the domain is lower than either $\|\phi_1\|$ or $\|\phi_2\|$ (by definition of the size function). The procedure is hence polytime. ∎

• `MDD` **satisfies $\vee$C, $\vee$BC.** To make the disjunction of k MDDs $\phi_1, \dots, \phi_k$, add to the root of ϕ_i an edge per value in the domain of $\mathrm{Var}(\mathrm{Root}(\phi)$, pointing to the root of ϕ_{i+1}. Then merge all the sinks into a single one. ∎

• `dMDD` **satisfies $\vee$C, $\vee$BC.** This holds since `dMDD` satisfies $\wedge$**C** and $\neg$**C**. Indeed, to make the disjunction of $\phi_1, \dots \phi_k$, compute the negation of each disjunct (`dMDD` satisfies $\neg$**C**), then make their conjunction (`dMDD` satisfies $\wedge$**C**), and again compute the negation of the result. ∎

• `dOMDD` **does not satisfy $\vee$C, $\vee$BC.** Since $\texttt{dOMDD}^{\mathfrak{B}} \equiv_{\mathcal{L}}$ `OBDD` (Lemma 9), as `OBDD` does not support $\vee$**C**, and does not support $\vee$**BC** unless $\mathsf{P} = \mathsf{NP}$, `dOMDD` does not either. ∎

• `dOMDD`$_<$ **does not satisfy $\vee$C.** Since $\texttt{dOMDD}_<^{\mathfrak{B}} \equiv_{\mathcal{L}} \texttt{OBDD}_<$ (Lemma 9), as `OBDD`$_<$ does not support $\vee$**C**, `dOMDD`$_<$ does not either. ∎

• `OMDD`$_<$ **satisfies $\vee$C, $\vee$BC.** To make the disjunction of

```
1:  Number the variables of $\phi$ according to $<$
2:  for all $E = \langle N, N', v\rangle$ such that $\mathrm{Var}(N) \in Y$ do
3:      Label $E$ with $\top$
4:  for all $x_i \in Y$, $i$ going from $n$ to 1 do
5:      for all $N$ such that $\mathrm{Var}(N) = x_i$ do
6:          Let $I = \mathrm{In}(N)$, $O = \mathrm{Out}(N)$, $\mathcal{E}_{\text{new}} = \emptyset$
7:          for all $E_I = \langle N_I, N, a\rangle \in I$ do
8:              for all $E_O = \langle N, N_O, \top\rangle \in O$ do
9:                  Add an edge $E = \langle N_I, N_O, a\rangle$ to $\mathcal{E}_{\text{new}}$
10:         Remove $N$, $I$ and $O$ from the graph
11:         Add $\mathcal{E}_{\text{new}}$ to the graph
```

Fig. 8. Forgetting of the variables Y in an OMDD ϕ.

k OMDDs $\phi_1, \dots, \phi_k$ with the same ordering $<$, first make sure all roots are labeled by the same variable (add a new root if it is not the case, with one edge for each value of the domain, pointing to the former root); then merge all the roots into one. This process is linear in the sizes of the ϕ_i. Thus `OMDD`$_<$ supports $\vee$**C**, and *a fortiori* $\vee$**BC**. ∎

• `dOMDD`$_<$ **satisfies $\vee$BC.** This holds because `dOMDD`$_<$ satisfies both $\wedge$**BC** and $\neg$**C**. To compute the disjonction of two dOMDDs of the same order, it is sufficient to compute their negation, the conjunction of the two resulting dOMDDs, and again the negation of the result. ∎

• `MDD` **does not satisfy FO.** Given any MDD ϕ, ϕ is consistent if and only if $\mathrm{Forget}(\mathrm{I}(\phi), \mathrm{Var}(\phi)) \equiv \top$. The only reduced MDDs without any variable are the empty and the sink-only graph, and testing the emptyness of an MDD is done is constant time. If `MDD` satisfied **FO**, we would have a polytime algorithm for deciding the consistency of any MDD, yet `MDD` does not satisfy **CO** unless $\mathsf{P} = \mathsf{NP}$. ∎

• `dMDD` **does not satisfy FO.** Same proof as the previous one (`dMDD` does not satisfy **CO** unless $\mathsf{P} = \mathsf{NP}$). ∎

• `OMDD`$_<$ **and** `OMDD` **support FO, SFO.** The algorithm from Figure 8 computes a syntactical forgetting of $Y \subseteq X$ in ϕ; let us denote it $F(\phi, Y)$. This algorithm simply replaces, for any N such that $\mathrm{Var}(N) \in Y$, any path $\langle N_1, N, u\rangle \rightsquigarrow \langle N, N_2, v\rangle$ by a shortcut $\langle N_1, N_2, u\rangle$. The proof is similar to that of **CD**.

We show that $\mathrm{I}(F(\phi, Y)) \equiv \mathrm{Forget}(\mathrm{I}(\phi), Y)$. Let $T = X \setminus Y$, and let $\vec{t}$ be a T-assignment. Suppose $\mathrm{Forget}(\mathrm{I}(\phi), Y)(\vec{t}) = \top$; then there exists a Y-assignment $\vec{y}$ such that $\vec{y} \,.\, \vec{t}$ is a model of ϕ. Let p be a path in ϕ compatible with both $\vec{y}$ and $\vec{t}$. By construction, there is a path in $F(\phi, Y)$ that is compatible with $\vec{t}$: either there is no Y-node along p, and the result is clear, or there is a Y-node, and then it has been "shortcut". Consequently, $\vec{t}$ is a model of $F(\phi, Y)$: $\mathrm{I}(F(\phi, Y))(\vec{t}) = \top$.

Suppose that $\mathrm{Forget}(\mathrm{I}(\phi), Y)(\vec{t}) = \bot$. There is no Y-assignment $\vec{y}$ such that $\vec{y} \,.\, \vec{t}$ is a model of ϕ. Let p be a path in ϕ: it cannot be compatible with $\vec{t}$. Indeed, there necessarily exists at least one Y-assignment $\vec{y}$ that is compatible with p, since ϕ is read-once (no variable is repeated along p). If p were also compatible with $\vec{t}$, it would contradict our hypothesis. Hence, p is not compatible with $\vec{t}$; let p' be the path in $F(\phi, Y)$

corresponding to p. It is the same as p, except that Y-nodes have been bypassed: T-edges are the same. Thus p' is not compatible with $\vec{t}$. Since we chose an arbitrary path p in ϕ, no path in $F(\phi, Y)$ can be compatible with $\vec{t}$, and finally, $\mathrm{I}(F(\phi, Y))(\vec{t}) = \bot$.

Notice that this algorithm does not preserve determinism. ■

• **dOMDD and dOMDD$_<$ do not satisfy SFO, FO.** Suppose dOMDD (resp. dOMDD$_<$) satisfied **SFO**. Then we could obtain in polytime a dOMDD (resp. a dOMDD$_<$) representing the disjunction of an arbitrary number of dOMDDs$_<$, by joining them thanks to a new root node labeled with a new variable, and forgetting this variable. Since any term can be represented as a polysize dOMDD$_<$ (Lemma 16), we could build a polysize dOMDD (resp. dOMDD$_<$) representing some DNF. Yet, it is impossible, since OBDD $\not\leq_s$ DNF (resp. OBDD$_<$ $\not\leq_s$ DNF). Hence, dOMDD and dOMDD$_<$ cannot satisfy **SFO**, and thus **FO**. ■

• **MDD satisfies SFO.** From Lemma 21, it holds that $\mathrm{Forget}(\mathrm{I}, \{x\}) = \bigvee_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}$. Let $V_x = \{\mathrm{Lbl}(E)\colon \mathrm{Var}(E) = x\}$ the set of values that appear on an x-edge in ϕ, and $d_x = |V_x|$; to obtain an MDD whose interpretation function is equal to $\mathrm{Forget}(\mathrm{I}, \{x\})$, it is sufficient to make d_x copies of ϕ, to condition each of them by a value in V_x and to make $d_x - 1$ disjunctions. These two operations are polytime since MDD satisfies **CD** and **∨C**; thus **SFO** is feasible in time polynomial in $\|\phi\|$. ■

• **dMDD satisfies SFO.** Same proof as the previous one, as dMDD satisfies **CD** and **∨C**. ■

• **MDD does not satisfy EN.** Given any MDD ϕ, ϕ is valid if and only if $\mathrm{Ensure}(\mathrm{I}(\phi), \mathrm{Var}(\phi)) \equiv \top$. Again, the only reduced MDDs without any variable are the empty and the sink-only graphs. If MDD were satisfying **EN**, we would have a polytime algorithm for deciding the validity of any MDD, yet MDD does not support **VA** unless $\mathsf{P} = \mathsf{NP}$. ■

• **dMDD, OMDD, and OMDD$_<$ do not satisfy EN.** Because they do not satisfy **VA** unless $\mathsf{P} = \mathsf{NP}$ (same proof as the previous one). ■

• **dOMDD$_<$ does not satisfy EN, SEN.** This is a consequence of the fact that dOMDD$_<$ supports **¬C** and does not support **FO** and **SFO**. Indeed, for any interpretation function I and any set of variables $Y \subseteq X$, $\mathrm{Forget}(\mathrm{I}, Y) = \neg\,\mathrm{Ensure}(\neg\,\mathrm{I}, Y)$. If dOMDD$_<$ satisfied **EN** (resp. **SEN**), we would have a polytime algorithm for performing a forgetting (resp. single forgetting) of some Y in ϕ: compute a negation of ϕ, ensure Y in this negation, and return a negation of the result. ■

• **dOMDD does not satisfy EN, SEN.** The proof is the same than the previous one, since dOMDD supports **¬C** but does not support **FO** or **SFO**. ■

• **MDD satisfies SEN.** From Lemma 21, it holds that $\mathrm{Ensure}(\mathrm{I}, \{x\}) = \bigwedge_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}$. Let $d = |\mathrm{Dom}(x)|$; to obtain an MDD whose interpretation function is equal to $\mathrm{Ensure}(\mathrm{I}(\phi), \{x\})$, it is sufficient to make d copies of ϕ, to condition each of them by one of the possible assigment of x and to make $d - 1$ conjunctions. These two operations are polytime (since MDD satisfies **CD** and **∧C**) thus **SEN** is feasible in time polynomial in $\|\phi\|$. ■

• **dMDD satisfies SEN.** Same proof as the previous one (dMDD satisfies **CD** and **∧C**). ■

• **OMDD and OMDD$_<$ do not satisfy SEN.** We prove that if OMDD or OMDD$_<$ satisfied **SEN**, we could translate any CNF into one of these languages in polytime, and thus check its consistency, which is impossible unless $\mathsf{P} = \mathsf{NP}$.

Let us suppose OMDD (resp. OMDD$_<$) satisfied **SEN**. Then we could obtain in polytime an OMDD (resp. an OMDD$_<$) representing the conjunction of an arbitrary number of dOMDDs$_<$, by joining them with a new root node labeled with a new variable, and ensuring this variable. Since any clause can be represented as a polysize OMDD$_<$ (Lemma 16), we could thus build a polysize OMDD (resp. OMDD$_<$) representing some CNF. Since OMDD (resp. OMDD$_<$) satisfies **CO**, we could check in polytime whether any CNF is consistent, which is impossible unless $\mathsf{P} = \mathsf{NP}$. Hence, OMDD (resp. OMDD$_<$) cannot satisfy **SEN** unless $\mathsf{P} = \mathsf{NP}$. ■

• **MDD, dMDD, dOMDD, and dOMDD$_<$ do not satisfy TR.** Since the satisfaction of **TR** implies the satisfaction of **FO** (forgetting boils down to a restriction to $\left[y_1 \in \mathrm{Dom}(y_1)\right] \wedge \cdots \wedge \left[y_k \in \mathrm{Dom}(y_k)\right]$), MDD and dMDD do not satisfy **TR** unless $\mathsf{P} = \mathsf{NP}$, and dOMDD and dOMDD$_<$ do not satisfy **TR**. ■

• **OMDD and OMDD$_<$ satisfy TR.** Any “term” can be turned in polytime into an OMDD of any order (Lemma 17), and OMDD$_<$ satisfies **∧BC** and **FO**; it thus satisfies **TR**. Any OMDD being an element of OMDD$_<$ for some $<$, OMDD also satisfies **TR**. ■