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

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

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

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

Cédric Pralet  
ONERA—The French Aerospace Lab  
F-31055, Toulouse  
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 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

1This work is partially funded by the ANR project “Business Recommendation for Configurable Products” (BR4CP), ANR-11-B502-008
of values, and a set of constraints \( C \) that specify allowed combinations of values for subsets of variables. For \( y \in X \), \( \text{Dom}(y) \) denotes the domain of \( y \). For \( Y = \{y_1, \ldots, y_k\} \subseteq X \), \( \text{Dom}(Y) \) denotes \( \text{Dom}(y_1) \times \cdots \times \text{Dom}(y_k) \), and \( \bar{y} \) denotes a \( Y \)-assignment of variables from \( Y \), i.e., \( \bar{y} \in \text{Dom}(Y) \). When \( Z \cap Y = \emptyset \), \( \bar{z} \cdot \bar{y} \) is the concatenation of \( \bar{z} \) and \( \bar{y} \). Last, \( y_{i|y_i} \) denotes the value assigned to \( y_i \) in \( \bar{y} \).

The set of solutions \( \text{Sol}(P) \) of a CSP \( P = (X, C) \) is the set of elements in \( \text{Dom}(X) \) that satisfy all constraints in \( C \). \( \text{Sol}(P) \) can be represented as a multivalued decision diagram on \( X \) \cite{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 \( \phi = (N, \mathcal{E}) \) where \( N \) is a set of nodes containing at most one root (denoted Root(\( \phi \))) and at most one leaf (the sink, denoted Sink(\( \phi \))).

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

The size of \( \phi \), 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 (\( x_1 < x_2 < x_3 \)).](image)

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 \( \vec{x} = (a_1, \ldots, a_n) \in \text{Dom}(X) \) belongs to \( \text{Sol}(P) \), 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 \( \text{Sol}(P) \). 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 \( \text{Dom}(X) \)—or equivalently, any Boolean function over \( \text{Dom}(X) \).

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

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

Existing proposals referring to decisions diagrams as a way to compile CSP assume that they are deterministic and ordered \cite{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 \( \prec \) be a total order over \( X \). An MDD is said to be ordered \( w.r.t. \prec \), if, for every couple of nodes \( (N, M) \) such that \( N \) an ancestor of \( M \), \( \text{Var}(N) \prec \text{Var}(M) \) 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;
- OMDD\(_\prec\) is the language of MDDs ordered \( w.r.t. \prec \) (these MDDs may be non-deterministic);
- OMDD is the union of all OMDD\(_\prec\) languages;
- dOMDD\(_\prec\) is the language of all deterministic MDDs that are ordered \( w.r.t. \prec \);
- dOMDD is the union of all the dOMDD\(_\prec\) languages.

It obviously holds that \( \text{dOMDD}\(_\prec\) \subseteq \text{OMDD} \subseteq \text{MDD} \), that \( \text{dOMDD}\(_\prec\) \subseteq \text{OMDD} \subseteq \text{OMDD} \subseteq \text{MDD} \), and that \( \text{dOMDD} \subseteq \text{OMDD} \).

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 \{dOMDD\(_\prec\), OMDD\(_\prec\), dOMDD, OMDD, dMDD, MDD\}. There exists a polytime algorithm that transforms any \( \phi \) in \( L \) into an equivalent reduced \( \phi' \) 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 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 \( L^2 \) be the sublanguage of L obtained when restricting it to Boolean variables. \( d\text{MDD}^B \), \( d\text{MDD}^\prec \), and \( \text{OMDD}^\prec \) correspond to BDD, OBDD, and OBDD, respectively, up to a linear transformation.

**Definition 7.** A sublanguage \( L_2 \) of MDD is polynomially translatable (resp. linearly translatable) into another sublanguage \( L_1 \) of MDD, which we denote \( L_1 \leq P L_2 \) (resp. \( L_1 \leq L L_2 \)), if and only if there exists a polytime (resp. linear time) algorithm \( A_{L_2 \rightarrow L_1} \), mapping any element in \( L_2 \) to an equivalent element in \( L_1 \).

When translations \( A_{L_1 \rightarrow L_2} \) and \( A_{L_2 \rightarrow L_1} \) are linear and stable, i.e., when \( A_{L_1 \rightarrow L_2} = A_{L_2 \rightarrow L_1} \), we say that \( L_1 \) and \( L_2 \) are linearly equivalent, denoted by \( L_1 \equiv_L L_2 \).

**Definition 8 (Succinctness).** A sublanguage \( L_1 \) of MDD is at least as succinct as another sublanguage \( L_2 \) of MDD (denoted \( L_1 \leq s L_2 \)) if and only if there exists a polynomial \( P(\cdot) \) such that for each element \( \phi \) of \( L_1 \), there exists an equivalent representation \( \psi \) of \( L_1 \) 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, \( L_1 \leq s L_2 \Rightarrow L_1 \leq P L_2 \Rightarrow L_1 \leq s L_2 \).

The BDD and \( \text{dMDD}^B \) 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 \( Lb l(E) = 1 \) and by 1 otherwise.

**Proposition 9.** \( \text{dMDD}^B \equiv_L \text{BDD}, \text{dMDD}^B \equiv_L \text{OBDD}, \) and \( \text{dMDD}^\prec \equiv_L \text{OBDD}^\prec \).

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. \( \text{OMDD} \) is hence strictly more succinct than DNF—it can be seen as a proper “superset” of this fragment.

**Proposition 10.** It holds that \( \text{OMDD}^B \leq s \text{ DNF} \).

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

### 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 \( \text{dOMDD} \) and \( \text{dOMDD}^\prec \) collapse into \( \text{OBDD} \) and \( \text{OBDD}^\prec \) when domains are Boolean: \( \text{dOMDD}^\prec \leq s \text{ dOMDD} \) is a straightforward consequence of \( \text{OBDD}^\prec \leq s \text{ OBDD} \). Some other results are derived from the NNF map in a less immediate way; for instance \( \text{dOMDD} \leq s \text{ OMDD} \) (and thus \( \text{OMDD} \leq s \text{ dOMDD} \)) holds since \( \text{OMDD}^B \leq s \text{ DNF} \) and \( \text{OBDD} \sim s \text{ dOMDD}^B \); if \( \text{dOMDD} \leq s \text{ OMDD} \) were true, we could derive \( \text{OBDD} \leq s \text{ DNF} \), which has been proven false [9].

![Fig. 2. \( \text{dOMDD}^B \) 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.](image-url)

Some results are harder to get. For instance, in order to prove \( \text{OMDD} \leq s \text{ MDD} \), 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 \( \text{Sol} \) 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 \( \text{dOMDD} \) or \( \text{OMDD} \) representing \( \text{Sol} \) has a size exponential in \( n \). Figure 1 shows the corresponding blow-up on a small instance.

We get \( \text{dOMDD}^\prec \leq s \text{ OMDD}^\prec \) 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 \( x_n < \ldots < x_2 < x_1 \): the \( \text{dOMDD}^\prec \) 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 \( \text{OMDD}^\prec \) (with the same order \( < \)) whose size is polynomial in \( n \).
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 ¬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 (∨C), disjunction (∨C) and the aforementioned negation (¬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 \( P(\cdot) \) and an algorithm that maps every MDD φ from L, any set of variables \( \{y_1, \ldots, y_k\} \subseteq \text{Var}(\phi) \), and any sequence \( \{A_1, \ldots, A_k\} \) of finite sets of integers, to 1 if \( I(\phi) \models [y_1 \in A_1] \lor \cdots \lor [y_k \in A_k] \) (resp. \( [y_1 \in A_1] \land \cdots \land [y_k \in A_k] \models I(\phi) \)) 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 \( \text{Var}(\phi) \)-assignment \( \bar{x} \) to 1 if \( \bar{x} \) 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 \( y \in \text{Var}(\phi) \), 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.
Definition 13. Let $I, J$ be the interpretation functions on $\text{Var}(I), \text{Var}(J)$ of some MDDs.

- The existential projection of $I$ on $Y \subseteq \text{Var}(I)$ is the function $I^Y$ on the variables of $Y$ defined by: $I^Y(\vec{y}) = \top$ iff there exists a $Z$-assignment $\vec{z}$ (with $Z = \text{Var}(I) \setminus Y$) s.t. $I(\vec{z} \cdot \vec{y}) = \top$. The “forgetting” operation is the dual one: Forget($I$, $Y$) = $I^{|\text{Var}(I)\setminus Y|}$.

- The universal projection of $I$ on $Y \subseteq \text{Var}(I)$ is the function $I^Y$ on the variables of $Y$ defined by: $I^Y(\vec{y}) = \top$ iff for all $Z$-assignment $\vec{z}$ (with $Z = \text{Var}(I) \setminus Y$), $I(\vec{z} \cdot \vec{y}) = \top$. The “ensuring” operation is the dual one: Ensure($I$, $Y$) = $I^{|\text{Var}(I)\setminus Y|}$.

- The restriction of $I$ to $J$, denoted $I_{[J]}$, is defined by $I_{[J]}(I \wedge J, \text{Var}(J))$.

- Given an assignment $\vec{y}$ of some set of variables $Y \subseteq \text{Var}(I)$, the conditioning of $I$ by $\vec{y}$ is the function $I_{[\vec{y}]}$ on the variables in $Z = \text{Var}(I) \setminus Y$ defined by: $I_{[\vec{y}]}(\vec{z}) = I(\vec{z} \cdot \vec{y})$.

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

- $L$ satisfies FO (resp. EN) iff there exists a polynomial algorithm that maps every MDD $\phi$ from $L$ and every $Y \subseteq \text{Var}(\phi)$ to an MDD $\phi'$ in $L$ such that $I(\phi') = \text{Forget}(I(\phi), Y)$ (resp. $I(\phi') = \text{Ensure}(I(\phi), Y)$).

- $L$ 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 $P(\cdot)$ and an algorithm that outputs, for every MDD $\phi$ from $L$, any set of variables $\{y_1, \ldots, y_k\} \subseteq \text{Var}(\phi)$ and any sequence $(A_1, \ldots, A_k)$ of finite sets of integers, an MDD $\phi'$ in $L$ such that $I(\phi') = I(\phi)[y_1 \in A_1] \wedge \ldots \wedge [y_k \in A_k]$ in time $P(||\phi||; n_A)$, where $n_A = \max_{1 \leq i \leq k} |A_i|$.

- $L$ satisfies CD iff there exists a polytime algorithm that maps every MDD $\phi$ in $L$ and every assignment $\vec{y}$ of $Y \subseteq \text{Var}(\phi)$ to an MDD $\phi'$ in $L$ such that $I(\phi') = I(\phi)_{[\vec{y}]}$.

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 $d$MDD, $d$OMDD, and $d$OMDD$_<$ are generally known or follow directly from previous works [10], [2], [9]. $d$OMDDs 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 OBDDs, are not satisfied by $d$OMDD; 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: $d$OMDD$_<$ supports the same transformations as DNF but SEN, and more transformations than $d$OMDD$_<$ but the negation, while being strictly more succinct than these two classes.

As for the performances w.r.t. queries, OMDD and $d$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 DNF 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 $d$OMDD compiler we use follows the bottom-up approach [1]: each constraint is compiled as a $d$OMDD$_<$, and the elementary $d$OMDDs obtained are then combined by conjunction. After these steps, we get a $d$OMDD$_<$. 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>
<thead>
<tr>
<th>L</th>
<th>F</th>
<th>TR</th>
<th>FO</th>
<th>SFO</th>
<th>EN</th>
<th>SEN</th>
<th>C</th>
<th>VBC</th>
<th>C</th>
<th>ABC</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dMDD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OMDD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dOMDD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dOMDD$_&lt;$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DNF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DNNF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OBDD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OBDD$_&lt;$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>%T</td>
<td>%C</td>
<td>#SOL</td>
<td>#N OMDD</td>
<td>#N dOMDD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>------</td>
<td>---------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>290888073</td>
<td>80</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1336056826</td>
<td>1338</td>
<td>1558</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>50065762</td>
<td>5662</td>
<td>8132</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>95131</td>
<td>3315</td>
<td>5005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>2367</td>
<td>737</td>
<td>897</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>391615179</td>
<td>79</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>1586148506</td>
<td>2572</td>
<td>2932</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>189551100</td>
<td>12223</td>
<td>16370</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>11577737</td>
<td>20501</td>
<td>35486</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>10358848</td>
<td>13815</td>
<td>25240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>70185</td>
<td>5776</td>
<td>9253</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>4662</td>
<td>1719</td>
<td>2246</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>229</td>
<td>54</td>
<td>401</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 polynome: typically, all transformations (save SEN, which depend on the domain size) satisfied by DNF are also satisfied by 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, 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, OMDD< is a new fragment in the NNF map, below DNNF and above DNF and OBDD. Moreover, 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


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< in linear time.

Proof: In order to represent a term \( t = l_1 \land \cdots \land l_k \) by a doMDD<, let us order its literals in such a way that \( i < j \) if and only if \( \text{Var}(l_i) < \text{Var}(l_j) \). Then build a chain of nodes \( N_1, \ldots, N_k, N_{k+1} \) where \( \text{Var}(N_i) = \text{Var}(l_i), i = 1, k \) and \( N_{k+1} \) will be the sink of the new doMDD<. 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 \( c = l_1 \lor \cdots \lor l_k \) by a doMDD<, let us order its literals in such a way that \( i < j \) if and only if \( \text{Var}(l_i) < \text{Var}(l_j) \). Then build a chain of nodes \( N_1, \ldots, N_k, N_{k+1} \) where \( \text{Var}(N_i) = \text{Var}(l_i), i = 1, k \) and \( N_{k+1} \) will be the sink of the new doMDD<. 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 $N_{k+1}$ 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 $Y = \{y_1, \ldots, y_k\}$ be a set of variables from $X$, and $(A_1, \ldots, A_k)$ a sequence of subsets of $\text{Dom}(y_1), \ldots, \text{Dom}(y_k)$. Given a total ordering $<_{\text{on X}}$ on $X$, the “term” $[y_1 \in A_1] \land \cdots \land [y_k \in A_k]$ can be represented as a $d\text{OMDD}_{<}$ 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 $d\text{OMDD}_{<}$ has $k + 1$ nodes and $\sum_{i=1}^{n_A} |A_i|$ edges.

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

**Proof:** Suppose that $\phi$ has at least one edge, and consider a path from the root to the sink of $\phi$. Because the variables in $\phi$ are ordered, they are encountered only once. It is thus possible to build an assignment $\vec{x}$ of $\phi$ as follows: for any $(N, N', a)$ on the path, assign $a$ to $\text{Var}(N)$; 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), $I(\vec{x}) = T$.

**A. Succinctness Proofs (Proposition 11)**

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

  II has a $d\text{OMDD}_{<n}$ representation the size of which is exponential in $n$; let us denote it $\phi_a$. We show that the size of any of the $\text{OMDD}_{<n}$ representations of $\text{Sol}(\Pi)$ is exponential with $n$.

  Let $\phi_b$ be an $\text{OMDD}_{<n}$ representation of $\text{Sol}(\Pi)$. Consider an edge $E = (N, N', v)$ in $\phi_b$ 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 $p : \text{Root}(\phi) \rightarrow (N, N', v) \rightarrow \text{Sink}(\phi)$. On this path, the value of $z_i$ must be $v$. So, a path $p'$ obtained from $p$ by using another edge $(N, N', u)$ instead of $(N, N', v)$ cannot belong to $\phi_a$. 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 $\{y_1, \ldots, y_n\}$. Each of the possible assignments of $Y$ is represented by (at least) one path entering a distinct $z_i$ node. The $Y$ assignments are not constrained: $\phi_a$ has at least $2^n$ nodes at level $z_1$.

  $\phi_a$ is thus a $d\text{OMDD}_{<n}$ of size polynomial in $n$ that has no polynomial representation as an $\text{OMDD}_{<n}$.

- $\text{OMDD} \not\subseteq_{s} \text{dOMDD}$. We get a direct proof of $\text{OMDD} \not\subseteq_{s} \text{dOMDD}$ 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 $i$th level, there must be as many nodes as there are possibilities of choosing $i$ values in $\{1, \ldots, n\}$ (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 $i$th level, and the second one afterwards). Hence the total number of nodes is at least $\sum_{i=1}^{n}(n) = 2^{n-1}$.

  Yet, the problem can obviously be represented as a poly-size dOMDD, since each constraint $x_i \neq x_j$ can be represented as a $n(n - 1)$-edge dOMDD, and dOMDD satisfies $\land$ (see Proposition 15).

- $\text{dOMDD} \not\subseteq \text{OMDD}_{<}$. Proposition 10 states that $\text{OMDD}_{<} \subseteq_{s} \text{DNF}$. If $\text{dOMDD} \subseteq_{s} \text{OMDD}_{<}$, and thus $\text{dOMDD}_{<} \not\subseteq \text{OMDD}_{<}$ were true, since $\text{OBDD} \equiv_{C} \text{OMDD}_{<}$, $\text{OBDD} \subseteq_{s} \text{DNF}$ would also be true—yet it has been proven false [9].

- $L \subseteq L'$. All positive results come directly from simple language inclusions (if $L \supseteq L'$, then surely $L \subseteq L'$).

- $L \not\subseteq L'$. All remaining negative results come from the fact that if $L_1 \not\subseteq L_2$, then every $L$ such that $L_1 \subseteq L$ verifies $L \not\subseteq L_2$, and every $L$ such that $L \subseteq L_2$ verifies $L_1 \not\subseteq L$, or we could derive $L_1 \subseteq L_2$ by transitivity.

**B. Proofs of Queries (Proposition 15)**

- $\text{MDD and its subclasses satisfy MC}$. Since these languages satisfy $\text{CD}$ (see Section C), we can test whether $\vec{x}$ is a model of $\phi$ 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 $\text{MC}$.

- $\text{Queries on dOMDD and MDD}$. BDD is a subclass of dMDD and MDD, so dMDD and MDD do not satisfy any of the other considered queries unless $P = \text{NP}$.

- $\text{OMDD and OMDD}_{<} \not\text{do satisfy VA, IM, EQ, SE, CT}$. Lemma 16 states that any term can be transformed into $\text{OMDD}_{<}$ in linear time, and $\text{OMDD}_{<}$ satisfies $\lor$ (Prop. 15); hence $\text{OMDD}_{<} \not\subseteq_{C} \text{DNF}$, therefore $\text{OMDD}_{<} \not\subseteq_{C} \text{DNF}$ (by inclusion), and we obtain the result since DNF does not satisfy VA, IM, EQ, SE, CT unless $P = \text{NP}$.

- $\text{OMDD and OMDD}_{<} \text{ satisfy CE}$. $\text{CO}$ holds since the only reduced OMDD that is not consistent is the empty graph (Lemma 18).

- $\text{OMDD and OMDD}_{<} \text{ satisfy CE}$. Checking whether $\phi$ entails $[x_1 \in A_1] \lor \cdots \lor [x_k \in A_k]$ is equivalent to checking whether $\phi \land [x_1 \notin A_1] \land \cdots \land [x_k \notin A_k]$ is inconsistent. Since (i) $[x \notin A] \equiv [x \in \text{Dom(x)} \setminus A]$, (ii) a “term” can be represented in polytime as an OMDD of the same variable order as $\phi$ (Lemma 17), and (iii) $\text{OMDD}_{<}$ satisfies $\land$ (see below), we can infer that OMDD satisfies CE.

- $\text{OMDD and OMDD}_{<} \text{ satisfy ME, MX, CX}$. These hold because 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 $\phi_{[x_i=v]}$ 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 $n_{Sink}(\phi) = 1$, then traverse the graph from the sink to the root, associating to each edge $E$ the number

$$n_E = n_{Dest(E)} \times \prod_{x_i \leftarrow E} |\text{Dom}(x)|$$

(where $x_a$ and $x_d$ denote Var(Src($E$)) and Var(Dest($E$)), respectively), and to each node $N$ the number

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

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

From CT, we can test in poltime 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$^B \equiv_C$ 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 poltime procedure that transforms any dOMDD into an OBDD (see e.g. [10]; we use here a direct encoding).

Let $\phi$ be a dOMDD on $X = \{x_1, \ldots, x_n\}$, with variable ordering $x_1 < \cdots < x_n$. For each variable $x_i$ and each value $\omega_j \in \text{Dom}(x_i) = \{\omega_1, \ldots, \omega_k\}$, 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$^B$ in which each $x$-node of $\phi$ is replaced by an ordered sequence of $x_i^j$ nodes.

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

Let us denote $X_{bin}$ the set of binary variables we introduced; an acceptable assignment $\vec{x}_{bin}$ 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 $\vec{x}_{bin}$. Formally,

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

Fig. 4. Encoding of a dOMDD $\phi$ into a dOMDD$^B$ $\phi_{bin}$

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

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

We denote $\text{Dom}_{bin}(X)$ the set of acceptable assignments of $X_{bin}$. By construction, there is a bijection between $\text{Dom}(X)$ and $\text{Dom}_{bin}(X)$, and for each model of $\phi$, its corresponding assignment in $\text{Dom}_{bin}(X)$ is a model of $\phi_{bin}$, and reciprocally.

Thus, given two MDDs $\phi$ and $\psi$ and their matching $\phi_{bin}$ and $\psi_{bin}$, it holds that $\phi \equiv \psi \iff \phi_{bin} \equiv \psi_{bin}$.

Now, dOMDD$^B \equiv_C$ OBDD, and OBDD supports EQ [14, Corollary 8.12].

For testing the equivalence of two dOMDDs $\phi$ and $\psi$ we only have to build their corresponding $\phi_{bin}$ and $\psi_{bin}$ (which is done in poltime), 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$^B$ satisfies SE.

Same indirect proof as the previous one : for testing whether $\phi$ entails $\psi$, we only have to build their corresponding $\phi_{bin}$ and $\psi_{bin}$ (which is done in poltime), transform them into OBDDs, and check whether $\phi_{bin}$ entails $\psi_{bin}$. It is the case if and only if $\phi \equiv \psi$.

- dOMDD$^B$ satisfies CO, VA, CE, EQ, MX, CX, CT, ME. Straightforward from the fact that dOMDD support these queries and that dOMDD$^B \subseteq$ dOMDD.

- dOMDD$^B$ satisfies IM. Checking whether $[y_1 \in A_1] \land \cdots \land \lceil y_k \in A_k \rceil$ entails $\phi$ is equivalent to checking whether $\phi \vee \neg([y_1 \in A_1] \land \cdots \land \lceil y_k \in A_k \rceil)$ is valid.
Lemma 19 (Conditioning). The algorithm in Figure 5:

1. Number the variables of \( \phi \) according to \(<\) (if relevant)
2. for all \( E = (N, N', v) \) such that \( \text{Var}(N) \in Y \) do
3. if \( v \neq \bar{y} \mid \text{Var}(N) \) then
4. Remove \( E \) from \( \phi \): \( \text{Out}(N) := \text{Out}(N) \setminus \{E\} \)
5. else
6. Label \( E \) with \( T \)
7. for all \( x_i \in Y, i \) going from \( n \) to 1 do
8. for all \( N \) such that \( \text{Var}(N) = x_i \) do
9. Let \( I = \text{In}(N), O = \text{Out}(N), \mathcal{E}_{\text{new}} = \emptyset \)
10. for all \( E_i = (N_I, N_O, a) \in I \) do
11. for all \( E_O = (N, N_O, T) \in O \) do
12. Add an edge \( E = (N_I, N_O, a) \) to \( \mathcal{E}_{\text{new}} \)
13. Remove \( N, I \) and \( O \) from the graph
14. Add \( \mathcal{E}_{\text{new}} \) to the graph

Fig. 5. Conditioning of an MDD \( \phi \) by an assignment \( \bar{y} \) of \( Y \subseteq \text{Var}(\phi) \).

Lemma 17 states that we can obtain a dOMDD\(_<\) equivalent to \([y_1 \in A_1] \land \cdots \land [y_k \in A_k]\) 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 \(-C \) and \( VBC \), 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 \( \phi, \phi \) is a dOMDD\(_<\). We can thus apply the previous technique to check whether \([y_1 \in A_1] \land \cdots \land [y_k \in A_k]\) entails \( \phi \).

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 \( \phi \) by some assignment \( \bar{y} \) of \( Y \subseteq X \). Let us denote \( \phi|_{\bar{y}} \) the MDD built by this algorithm.

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

1. preserves determinism and ordering;
2. is polytime when \( \phi \) is deterministic;
3. for each dMDD \( \phi, \|\phi|_{\bar{y}}\| \) is bounded by \( \|\phi\| \);
4. for each MDD \( \phi, \|\phi|_{\bar{y}}\| \) is bounded by \( |\text{Nodes}(\phi)|^2 \);
5. for each dMDD \( \phi, I(\phi|_{\bar{y}}) = I(\phi)\bar{y} \).

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 \( \phi \) are ordered according to \(<\), the variables in \( \phi|_{\bar{y}} \) are ordered according to \(<\).

- DETERMINISM: This algorithm preserves the property of determinism: when \( \phi \) is deterministic, at most one edge in \( \text{Out}(N) \) can be marked by \( T \). Hence, for any edge \( E = (N_I, N_O, a) \) 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 \( |\text{In}(N)| + |\text{Out}(N)| \) edges and creates at most \( |\text{In}(N)| \times |\text{Out}(N)| \) edges, that are only labeled with values mentioned in \( \phi \); thus the number of outgoing edges is bounded by \( |\text{Edges}(\phi)| \times |\text{Nodes}(\phi)| \) (in the worst case, there is one edge for each variable 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, \( |\text{Out}(N)| = 1 \) at the end of the first loop: the second loops visits each node \( N \) once, deletes \( |\text{In}(N)| + |\text{Out}(N)| \) edges and creates at most \( |\text{In}(N)| \times 1 \) edges. Its complexity is bounded by \( \|\phi\| \).

EQUVALENCE: Let \( T = X \setminus Y \) and consider any assignment \( \bar{t} \) of \( T \). We show that \( I(\phi|_{\bar{y}})\bar{y}(\bar{t}) = I(\phi)\bar{y}(\bar{t}) \).

Suppose \( I(\phi)|_{\bar{y}}(\bar{t}) = T \). Then there is a path \( p \) in \( \phi \) that is compatible with both \( \bar{y} \) and \( \bar{t} \). By construction, there is a path \( p' \) in \( \phi|_{\bar{y}} \) that is compatible with \( \bar{t} \); nodes along \( p' \) that are labeled with variables in \( Y \) have been suppressed, but edges with values compatible with \( \bar{y} \) have been “shortcut”. Hence \( I(\phi|_{\bar{y}})\bar{y}(\bar{t}) = T \).

Suppose \( I(\phi)|_{\bar{y}}(\bar{t}) = T \); there is a path \( p \) in \( \phi|_{\bar{y}} \) that is compatible with \( \bar{t} \). This path corresponds to several paths in \( \phi \), in which there are similar \( T \)-edges (so they are all compatible with \( \bar{t} \)). At least one of these paths is compatible with \( \bar{y} \); 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 \( \phi \) that is compatible with both \( \bar{y} \) and \( \bar{t} \); \( I(\phi)|_{\bar{y}}(\bar{t}) = T \).

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

1. preserves determinism and ordering;
2. is polynomial in \( \|\phi\| \);
3. for each dMDD \( \phi, \text{Compl}(\phi) \equiv \neg \phi \).

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
Lemma 21. It holds that

\[
\text{Forget}(I, \{x\}) = \bigvee_{\vec{x} \in \text{Dom}(\{x\})} I|_{\vec{x}}.
\]

\[
\text{Ensure}(I, \{x\}) = \bigwedge_{\vec{x} \in \text{Dom}(\{x\})} I|_{\vec{x}}.
\]

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

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

- **MDD and its subclasses satisfy CD.** Since the algorithm from Figure 5 is polytime, computes a conditioning of \( \phi \) 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: MDD and its subclasses even support CD in linear time.

- **dMDD, dOMDD and dOMDD do not satisfy ¬C.** Straightforward from Lemma 20: the algorithm from Figure 6 preserves determinism and variable ordering.

- **OMDD< and OMDD do not satisfy ¬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 ¬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 P = NP.

- **MDD satisfies ∧C ∧BC.** To make the conjunction of \( k \) MDDs \( \phi_1, \ldots, \phi_k \), replace the sink of \( \phi_i \) by the root of \( \phi_{i+1} \), for all \( 1 \leq i \leq k-1 \). The root of the new MDD is that of \( \phi_1 \), and its sink is that of \( \phi_k \).

- **dMDD satisfies ∧C ∧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 ∧C ∧BC.** Thanks to Proposition 9, any OBDD< can be turned into an equivalent OMDD< in linear time and size. If OMDD supported ∧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 ∧BC, and a fortiori does not support ∧C, unless \( P = NP \).

- **dOMDD does not satisfy ∧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 ∧BC unless \( P = NP \).

- **dOMDD and dOMDD do not satisfy ∧C.** We have a stronger result for ∧C: Since dOMDD \( \equiv_C \) OBDD (resp. dOMDD< \( \equiv_C \) OBDD<) thanks to Lemma 9, as OBDD (resp. OBDD<) does not support ∧C, dOMDD (resp. dOMDD<) does not either.

- **OMDD< does not satisfy ∧C.** Lemma 16 states that propositional clauses can be represented in the form of OMDD< in linear time; if OMDD< supported ∧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 \( P = NP \).

- **dOMDD< and OMDD< satisfy ∧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 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 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 \(\text{Var}(N_1) = \text{Var}(N_2)\) then
8: let \(x := \text{Var}(N_1) = \text{Var}(N_2)\)
9: Create a node \(N'\) labeled by \(x\)
10: for all \(\omega \in \text{Dom}(x)\), \(E_1 \in \text{Out}(N_1)\), \(E_2 \in \text{Out}(N_2)\)
11: if \(\omega = \text{Lbl}(E_1) = \text{Lbl}(E_2)\) then
12: let \(\phi_\omega := \text{conjunct_step}(\text{Dest}(E_1), \text{Dest}(E_2))\)
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: \(N_i = \text{Argmin}_< (\text{Var}(N_1), \text{Var}(N_2))\), \(N_i = \text{Argmax}_> (\text{Var}(N_1), \text{Var}(N_2))\)
16: Create a node \(N'_i\) labeled by \(\text{Var}(N_i)\)
17: for all \(E \in \text{Out}(N_i)\)
18: let \(\phi_E := \text{conjunct_step}(\text{Dest}(E), N'_i)\)
19: Add an edge coming out of \(N'_i\), labeled by \(\text{Lbl}(E)\) and pointing to the root of \(\phi_E\)
20: return \(N'\)

Fig. 7. \text{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.

For each execution of \text{conjunct_step}, it maintains determinism when the inputs are deterministic.

The procedure is hence polytime.

\text{OMDD}_{\leq} satisfies \(\forall C, \forall BC\). To make the disjunction of \(k\) OMDDs \(\phi_1, \ldots, \phi_k\) 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 \(\phi_i\). Thus \text{OMDD}_{\leq} supports \(\forall C\) and \(a \text{fortiori} \forall BC\).

- \text{OMDD}_{\leq} satisfies \(\forall BC\). This holds because \text{OMDD}_{\leq} satisfies both \(\forall BC\) and \(\forall C\). To compute the disjunction of two \text{OMDD}s of the same order, it is sufficient to compute their negation, the conjunction of the two resulting \text{OMDD}s, and again the negation of the result.

- \text{MDD does not satisfy FO}. Given any MDD \(\phi, \phi\) is consistent if and only if \(\text{Forget}(\text{I}(\phi), \text{Var}(\phi)) \equiv \top\). The only reduced MDDs without any variable are the empty and the sink-only graph, and testing the emptiness of an MDD is done in 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 \(P = NP\).

- \text{MDD does not satisfy FO}. Same proof as the previous one (\text{OMDD} does not satisfy \(CO\) unless \(P = NP\)).

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

We show that \(I(F(\phi, Y)) \equiv \text{Forget}(I(\phi), Y), Y, T = X \setminus Y\), and let \(\tilde{t}\) be a \(T\)-assignment. Suppose \(\text{Forget}(I(\phi), Y)(\tilde{t}) = \top\); then there exists a \(Y\)-assignment \(\bar{y}\) such that \(\bar{y}. \tilde{t}\) is a model of \(\phi\). Let \(p\) be a path in \(\phi\) compatible with both \(\bar{y}\) and \(\tilde{t}\). By construction, there is a path in \(F(\phi, Y)\) that is compatible with \(\tilde{t}\); 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, \(\tilde{t}\) is a model of \(F(\phi, Y)\); \(I(F(\phi, Y))(\tilde{t}) = \top\).

Suppose that \(\text{Forget}(I(\phi), Y)(\tilde{t}) = \bot\). There is no \(Y\)-assignment \(\bar{y}\) such that \(\bar{y}. \tilde{t}\) is a model of \(\phi\). Let \(p\) be a path in \(\phi\); it cannot be compatible with \(\tilde{t}\). Indeed, there necessarily exists at least one \(Y\)-assignment \(\bar{y}\) that is compatible with \(p\), since \(\phi\) is read-once (no variable is repeated along \(p\)). If \(p\) were also compatible with \(\tilde{t}\), it would contradict our hypothesis. Hence, \(p\) is not compatible with \(\tilde{t}\); 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 $\bar{t}$. Since we chose an arbitrary path $p$ in $\phi$, no path in $F(\phi, Y)$ can be compatible with $\bar{t}$, and finally, $I(F(\phi, Y))(\bar{t}) = \bot$.

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 $\not\leq_n$ DNF (resp. OBDD $\not\leq_n$ DNF). Hence, dOMDD and dOMDD cannot satisfy SFO, and thus FO.

- **MDD satisfies SFO.** From Lemma 21, it holds that Forget(I, $\{x\}$) = $\bigvee_{x \in \text{Dom}(\{x\})} I_{\bar{x}}$. Let $V_x = \{\text{Llv}(E) : \text{Var}(E) = x\}$ the set of values that appear on an $x$-edge in $\phi$, and $d_x = |V_x|$; to obtain an MDD whose interpretation function is equal to Forget(I, $\{x\}$), it is sufficient to make $d_x$ copies of $\phi$, 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 $\forall C$; thus SFO is feasible in time polynomial in $||\phi||$.

- **dMDD satisfies SFO.** Same proof as the previous one, as dMDD satisfies CD and $\forall C$.

- **MDD does not satisfy EN.** Given any MDD $\phi$, $\phi$ is valid if and only if $\text{Ensure}(I(\phi), \text{Var}(\phi)) \equiv \top$. 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 $\forall A$ unless $P = NP$.

- **dMDD, OMDD, and OMDD do not satisfy EN.** Because they do not satisfy $\forall A$ unless $P = NP$ (same proof as the previous one).

- **dOMDD does not satisfy EN, SEN.** This is a consequence of the fact that dOMDD supports $\neg C$ and does not support FO and SFO. Indeed, for any interpretation function $I$ and any set of variables $Y \subseteq X$, Forget(I, $Y$) = $\neg \text{Ensure}(\neg I, Y)$. If dOMDD satisfied EN (resp. SEN), we would have a polytime algorithm for performing a forgetting (resp. single forgetting) of some $Y$ in $\phi$: compute a negation of $\phi$, 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 $\neg C$ but does not support FO or SFO.

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

- **dMDD satisfies SEN.** Same proof as the previous one (dMDD satisfies CD and $\land C$).

- **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 $P = NP$.

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 OMDDs, 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 $P = NP$. Hence, OMDD (resp. OMDD) cannot satisfy SEN unless $P = NP$.

- **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 $[y_1 \in \text{Dom}(y_1)] \land \cdots \land [y_k \in \text{Dom}(y_k)]$, MDD and dMDD do not satisfy TR unless $P = NP$, 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 $\land C$ and FO; it thus satisfies TR. Any OMDD being an element of OMDD for some $<$, OMDD also satisfies TR.