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

Jerome Amilhastre, Hélène Fargier, Alexandre Niveau, Cédric Pralet

▶ **To cite this version:**

Jerome Amilhastre, Hélène Fargier, Alexandre Niveau, Cédric Pralet. Compiling CSPs: A Complexity Map of (Non-Deterministic) Multivalued Decision Diagrams. International Journal on Artificial Intelligence Tools, 2014, 23 (4), pp.1460015. 10.1142/S021821301460015X . hal-01303816

HAL Id: hal-01303816
https://hal.science/hal-01303816

Submitted on 18 Apr 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive TOULOUSE Archive Ouverte (OATAO)
OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : http://oatao.univ-toulouse.fr/
Eprints ID : 15480

To link to this article : DOI : 10.1142/S021821301460015X
URL : http://dx.doi.org/10.1142/S021821301460015X

To cite this version : Amilhastre, Jerome and Fargier, Helene and Niveau, Alexandre and Pralet, Cedric *Compiling CSPs: A Complexity Map of (Non-Deterministic) Multivalued Decision Diagrams.* (2014) International Journal on Artificial Intelligence Tools, Vol. 23 (n° 4). ISSN 0218-2130

Any correspondence concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

COMPILING CSPs: A COMPLEXITY MAP OF (NON-DETERMINISTIC) MULTIVALUED DECISION DIAGRAMS*

JÉRÔME AMILHASTRE
Cameleon Software, 185 rue Galilée
F-31670 Labège, France
jamilhastre@cameleon-software.com

HÉLÈNE FARGIER
IRIT–CNRS, Université Paul Sabatier
F-31062 Toulouse Cedex 9, France
fargier@irit.fr

ALEXANDRE NIVEAU†
CRIL–CNRS, 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

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. When these requests have to 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 tractable 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, they capture OBDDs and DNFs as proper subsets and have performances close to those of DNNFs. 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 polynomial time (typically, the disjunctive ones). Experiments on random problems confirm the gain in succinctness.

Keywords: Knowledge Compilation; CSP; MDD.

1. Introduction

The powerful framework of Constraint Satisfaction Problems (CSPs) allows the representation a great variety of problems. Different kinds of requests can be posted

*This is a revised, detailed version of the eponymous paper published in the proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI 2012). Note that this revision features a change in terminology, saving the name "MDD" for syntactically deterministic structures, and introducing a dedicated name for non-deterministic ones.

†Corresponding author.

on a CSP, such as the classical extraction of a solution, but also enforcement of global inverse consistency for the domains,[1] 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 global inverse consistency.[2,3]

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 (MDD),[4,5,6] that is, as a graph the nodes of which are labeled with variables and the edges of which represent assignments of the variables. In such diagrams, each path from the root to the sink represents a solution of the CSP. MDDs allow several operations, like those previously cited, to be achieved in time polynomial in the size of the diagram. This size can theoretically be exponentially higher than the size of the original CSPs, 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,[3] in recommender systems,[7] or, in their original Boolean form, in planning[8,9] and diagnosis[10].

Up to our knowledge, these applications always consider *(syntactically) deterministic* MDDs, that is, MDDs in which edges going out of a node have mutually exclusive labels. However, this assumption is probably not compulsory for many operations, notably the aforementioned ones. *Non-deterministic* structures could thus be appealing, depending on what is lost and gained when assuming or relaxing the determinism assumption. To evaluate the interest of this relaxation, we propose to draw a compilation map of MDDs, in the spirit of the NNF knowledge compilation map.[11] Such a map provides a way to identify the most succinct language supporting in polynomial time 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 framework of *multivalued variable diagrams* (MVDs), which correspond to "non-deterministic MDDs"; it includes several sublanguages, yielded from the application of the *decision* and *ordering* properties. In Section 3, we study the case of Boolean domains, in order to picture MDDs and MVDs in the NNF knowledge compilation map: we show that, beyond ordered MDDs, which somehow correspond to OBDDs, the language of non-deterministic ordered MVDs is strictly more succinct than both OBDDs and DNFs. Section 4 is devoted to the MVD knowledge compilation map, including a succinctness analysis of the different languages of the family of MVDs, as well as a complexity analysis of many queries and transformations. Section 5 then presents our first experimental results about the relative succinctness of deterministic and non-deterministic MDDs. Last, all proofs are gathered in the appendix.

2. Compiling Constraint Satisfaction Problems

2.1. *Constraint satisfaction problems*

We consider variables with finite domains of values. For a variable x, $\mathrm{Dom}(x)$ denotes the domain of x. For a set of variables $X = \{x_1, \ldots, x_k\}$, $\mathrm{Dom}(X)$ denotes the set of assignments of variables from X, or *X-assignments*, that is to say, $\mathrm{Dom}(X) = \mathrm{Dom}(x_1) \times \cdots \times \mathrm{Dom}(x_k)$, and $\vec{x}$ denotes an X-assignment: $\vec{x} \in \mathrm{Dom}(X)$. When X and Y are disjoint sets of variables, $\vec{x} \,.\, \vec{y}$ is the *concatenation* of $\vec{x}$ and $\vec{y}$. Last, for an X-assignment $\vec{x}$ and a set of variables Z, $\vec{x}|_Z$ denotes the restriction of $\vec{x}$ to the variables in Z, and $\vec{x}|_{x_i}$ the value assigned to x_i in $\vec{x}$.

Definition 2.1 (Constraint satisfaction problem)**.** A *Constraint Satisfaction Problem* (CSP) is an ordered pair $P = \langle X, C \rangle$, where $X = \{x_1, \ldots, x_n\}$ is a finite set of finite-domain variables and C is a finite set of constraints. Each constraint $c \in C$ has an associated *scope*, denoted $\mathrm{Scope}(c)$ and included in X, and consists of a set of $\mathrm{Scope}(c)$-assignments: these are the assignments allowed by the constraint.

A *solution* of P is an X-assignment $\vec{x}$ compatible with every constraint in C, that is, $\forall c \in C, \vec{x}|_{\mathrm{Scope}(c)} \in C$. The set of solutions of P is denoted $\mathrm{Sol}(P)$.

In this paper, we study the *compilation* of the solution set of CSPs. To this end, we consider a solution set as a Boolean function on the CSP's variables X, mapping each X-assignment $\vec{x}$ to $\top$ if $\vec{x}$ is a solution, and to $\bot$ otherwise. We will consider several target compilation languages representing Boolean functions on finite-domain variables.

Definition 2.2 (Target compilation language)**.** Let X be a finite set of finite-domain variables; a *target compilation language* on X is a set $\mathtt{L}_X$ of graph structures, together with an interpretation function $\mathrm{I}_{\mathtt{L}_X}$, mapping each graph ϕ to a Boolean function $\mathrm{I}_{\mathtt{L}_X}(\phi)\colon \mathrm{Dom}(X) \to \{\top, \bot\}$, called its *interpretation*, and a size function $\|\cdot\|_{\mathtt{L}_X}$. A subset of a language $\mathtt{L}_X$ is called a *sublanguage* of $\mathtt{L}_X$.

In the following, we omit the X subscript when there is no ambiguity. We denote languages using a typewriter font, as usual in the knowledge compilation map.[11]

2.2. *Multivalued variable diagrams*

Languages used to compile CSPs have been given various names in the literature; sometimes the same name has been used to denote slightly different languages, yielding ambiguity. We will adopt an explicit terminology here, formally defining one "root" language (that of "multivalued variable diagrams"), and retrieving several sublanguages by the application of structural restrictions. We will discuss in Section 2.3 the other names and forms of the languages we define.

Definition 2.3 (Multivalued variable diagram)**.** A *Multivalued Variable Diagram* (MVD) ϕ on a finite set of finite-domain variables X (denoted $\mathrm{Scope}(\phi)$) 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 and

at most one leaf (the *sink*, denoted $\mathrm{Sink}(\phi)$). Each non-sink node $N \in \mathcal{N}$ is labeled with a variable $\mathrm{Var}(N)$ in X, and each edge going out of N is labeled with a value in the domain of $\mathrm{Var}(N)$.

We denote by $\mathrm{Out}(N)$ (resp. $\mathrm{In}(N)$) the set of outgoing (resp. incoming) edges of N. An edge $E \in \mathcal{E}$ is often denoted by the 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, denoted $\mathrm{Lbl}(E)$.

Multivalued diagrams constitute a generalization of well-known *binary* decision diagrams,[12] which represent Boolean functions of Boolean variables. In the same fashion, multivalued diagrams represent Boolean functions of *multivalued* variables.

Definition 2.4 (Semantics of MVDs)**.** An MVD 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, $\vec{x}|_{\mathrm{Var}(N)} = a$.

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. Figure 1 shows examples of MVDs. To check whether an assignment $\vec{x} = \langle a_1, \ldots, a_n\rangle \in \mathrm{Dom}(X)$ belongs to $\mathrm{Mod}(\phi)$, 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 there exists one; $\vec{x}$ is a model if and only if the sink is reached.

We define the target compilation language of MVDs using this semantics, together with a size function taking domains into account, since some operations rely on values that are not mentioned in the graph itself.

Definition 2.5 (The `MVD` language)**.** The $\mathtt{MVD}_X$ language is the set of all MVDs on X, together with the interpretation function I of Definition 2.4, and the size function associating every MVD ϕ with the sum of its number of edges and of the cardinalities of all domains of variables from X.

Note that MVDs are *non-deterministic*, in the sense that edges going out of a node are not necessarily disjoint: starting from the root, choosing a value for each variable does not impose one to follow a *single* path. In other words, a given assignment can correspond to several complete paths in the graph.

This is not the case in existing proposals referring to the compilation of CSPs[4,3,6]: they use multivalued decision diagrams (MDDs), that are (syntactically) deterministic. We define the `MDD` language as the restriction of `MVD` to syntactically deterministic graphs; we call this restriction the *decision property*,[13] refraining from using "determinism" for the sake of consistency with the `NNF` knowledge compilation map,[11] in which determinism is a *semantic* property.

Definition 2.6 (Decision)**.** A node N in an MVD is a *decision node* if and only if values labeling edges going out of N are pairwise distinct. An MVD satisfies the

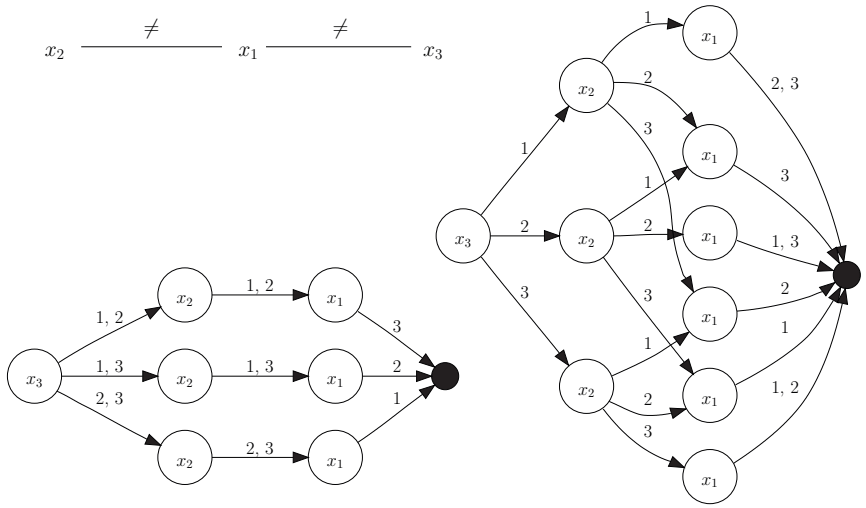

Fig. 1. The coloring problem on a star graph (3 variables, 3 colors), an OMVD and a OMDD representing its set of solutions (both for $x_3 < x_2 < x_1$).

decision property, and is called a *multivalued decision diagram* (MDD), if and only if all its nodes are decision nodes. The language `MDD` is the sublanguage of `MVD` containing all and only MVDs satisfying the decision property.

We also define another syntactic restriction, that of "ordering", which is generally assumed in the literature about CSP compilation,[4,3,6] but was only an option in the original definition of MDDs.[14]

Definition 2.7 (Ordering)**.** Let $<$ be a strict total order over a set X of variables. An MVD on X is said to be *ordered with respect to* $<$ if and only if for every pair of nodes $\langle N, M\rangle$ such that N an ancestor of M, $\mathrm{Var}(N) < \mathrm{Var}(M)$ holds.

We call for short "ordered MVD (OMVD) on $<$", an MVD that is ordered with respect to $<$. Decision and ordering are the basis for several sublanguages of `MVD`.

Definition 2.8 (Family of OMVDs)**.** We define the following languages:

- `OMVD` is the sublanguage of `MVD` containing all and only ordered MVDs;
- $\texttt{OMVD}_<$ is the sublanguage of `OMVD` containing all and only MVDs that are ordered with respect to a given $<$;
- `OMDD` (resp. $\texttt{OMDD}_<$) is the sublanguage of `OMVD` (resp. $\texttt{OMVD}_<$). containing all and only OMVDs that satisfy the decision property.

In other words, `OMVD` is the union of all $\texttt{OMVD}_<$ (for any $<$), and `OMDD` (resp. $\texttt{OMDD}_<$) is the intersection of `OMVD` (resp. $\texttt{OMVD}_<$) with `MDD`. It obviously holds that $\texttt{OMDD}_< \subseteq \texttt{OMDD} \subseteq \texttt{MDD}$ and that $\texttt{OMVD}_< \subseteq \texttt{OMVD} \subseteq \texttt{MVD}$.

Similarly to binary decision diagrams, we suppose that MVDs are in *reduced* form, that is, (i) isomorphic nodes (labeled with 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 edge per value in the variable's domain) have been skipped, and (iii) nodes with no parent (except for the root) or no successor (except for the sink) have been removed. Assuming that MVDs are reduced is harmless, because reduction can be done in time polynomial in the size of the diagram.

Proposition 2.9 (Reduction)**.** *Let* `L` *be one of the languages we defined. There exists a polynomial algorithm that transforms any* ϕ *in* `L` *into a reduced* ϕ' *in* `L` *such that* $\mathrm{I}(\phi') = \mathrm{I}(\phi)$ *and* $\|\phi'\| \leq \|\phi\|$.

2.3. *Related languages*

The `MVD` framework covers several languages that have been used to compile CSPs; we discuss them in this section and show how they can come down as MVDs.

2.3.1. *Finite-state automata*

Up to our knowledge, the first paper about CSP compilation is due to Vempaty,[4] who used deterministic finite-state automata (DFAs) from computability theory. The idea is to represent X-assignments as *words* over an alphabet $\Sigma_X = \bigcup_{x \in X} \mathrm{Dom}(x)$; for example, the word *abc* corresponds to the $\{x, y, z\}$-assignment in which $x = a$, $y = b$, and $z = c$ (considering the variable order $x < y < z$). The solution set of any CSP then corresponds to a set of words, i.e., to a finite formal language; any finite formal language being regular, this solution set can thus be represented by a deterministic finite-state automaton accepting this language.

Remark that the usual notion of DFA is more general than the one used in CSP compilation. For instance, all paths in a DFA representing a CSP must have the same (finite) length; it is obviously not the case for general DFAs, which can recognize even infinite languages. In the context of CSP compilation,[4,3,15] the use of the term "DFA" is metonymic.

These structures are very close to ordered multivalued decision diagrams. Except for the differences in terminology, the main distinction, as shown by Hadzic et al.,[15] is the fact that a path in an OMDD needs not mention all the CSP variables in the scope, whereas automata cannot "skip" a variable, since it would change the formal language they accept. However, this restriction is harmless: in an OMDD ϕ, the number of "skipped" edges is bounded by $e \times d \times |X|$ edges, where e is the number of edges and d the cardinal of the largest domain (at worst, we have to add $|X|$ nodes with d edges for each edge already in the graph). This number being bounded by $\|\phi\|^3$, adding all "skipped" edges in ϕ makes it only polynomially larger.

Deterministic finite-state automata thus correspond to `OMDD`; similarly, using *non-deterministic* finite-state automata (NFAs) to compile CSPs would correspond to using `OMVD`. Finally, note that by using a different kind of alphabet (such as

$\Sigma'_X = \{\langle x, a\rangle : x \in X, a \in \mathrm{Dom}(x)\})$, it is possible to cast non-ordered MDDs and MVDs as specific DFAs and NFAs, respectively.

2.3.2. *MDDs with two leaves*

In their original definition,[14] MDDs are not necessarily ordered, and can have several leaves (to represent functions with several output values). However, in the literature, it is often assumed that MDDs are ordered and that they have exactly two leaves, in the manner of BDDs—a $\top$-leaf, indicating assignments that are models of the function represented, and a $\bot$-leaf, indicating countermodels.

Yet, since two-leaf MDDs are syntactically deterministic, each assignment corresponds to exactly one complete path from the root to a leaf; such a path leads either to the $\top$-leaf or to the $\bot$-leaf. Consequently, the $\top$-leaf plays the role of the sink in our definition of MDDs, whereas the $\bot$-leaf is actually redundant: removing it, and reducing the graph to remove dangling edges and nodes, we get exactly our (single-leaf) MDDs. Moreover, adding a $\bot$-leaf to a single-leaf MDD ϕ is also easy: for each node N, and each value $a \in \mathrm{Dom}(\mathrm{Var}(N))$ for which N has no outgoing edge, add an edge $\langle N, \bot, a\rangle$. The number of added edges is bounded by $n \times d$, where n is the number of nodes in ϕ and d the cardinal of the largest domain. Both n and d being bounded by $\|\phi\|$, the number of added edges is at most polynomial.

Consequently, our choice (also made by Hadzic et al.[15]) to define the MDD language as containing single-leaf MDDs rather than two-leaf MDDs, is harmless from the point of view of the knowledge compilation map, while offering a consistent notation for non-deterministic structures (in which the $\bot$-leaf is meaningless, since a given assignment can be associated with several paths).

3. MVDs in the Boolean Knowledge Compilation Map

In this section, we study the relationship between MVDs and languages in the Boolean knowledge compilation map, notably BDDs. Let us start by defining concepts allowing a formal comparison of target languages. The first one concerns the "translation" of elements of some language into another language; when this translation is tractable, the first language inherits many properties of the second one.[16]

Definition 3.1 (Translatability)**.** A target language $\mathtt{L}_2$ is *polynomially translatable* (resp. *linearly translatable*) into another target language $\mathtt{L}_1$, 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 polynomial (resp. linear) algorithm $A_{\mathtt{L}_2 \mapsto \mathtt{L}_1}$ mapping any element in $\mathtt{L}_2$ to an element in $\mathtt{L}_1$ with the same interpretation.

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 $\mathtt{L}_1 \equiv_{\mathcal{L}} \mathtt{L}_2$. Linearly equivalent languages are so close that they can somehow be considered as identical; the complexity of operations is the same for both languages.

Another important concept is *succinctness*, that compares languages with respect to their ability to represent data in a compact manner.[17,11]

Definition 3.2 (Succinctness)**.** A target language $\mathtt{L}_1$ is *at least as succinct* as another target language $\mathtt{L}_2$ (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 element ψ of $\mathtt{L}_1$ having the same interpretation and 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, $\mathtt{L}_1 \leq_{\mathcal{L}} \mathtt{L}_2 \Rightarrow \mathtt{L}_1 \leq_{\mathcal{P}} \mathtt{L}_2 \Rightarrow \mathtt{L}_1 \leq_s \mathtt{L}_2$.

To highlight the relationship between MVDs and the NNF knowledge compilation map,[11] let us consider the Boolean case. For any sublanguage L of MVD, let $\mathtt{L}^{\mathcal{B}}$ be the sublanguage of L obtained when restricting it to Boolean variables.

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

Proposition 3.3. $\mathtt{MDD}^{\mathcal{B}} \equiv_{\mathcal{L}} \mathtt{BDD}$, $\mathtt{OMDD}^{\mathcal{B}} \equiv_{\mathcal{L}} \mathtt{OBDD}$, *and* $\mathtt{OMDD}^{\mathcal{B}}_{<} \equiv_{\mathcal{L}} \mathtt{OBDD}_{<}$.

As they are not necessarily deterministic, multivalued variable diagrams capture fragments beyond the BDD family. DNFs, for instance, can be represented as OMVDs in linear time, although some DNFs have no polynomial OBDD (and thus OMVD) representation. OMVD can hence be seen as a proper "superset" of DNF.

Proposition 3.4. *It holds that* $\mathtt{OMVD}^{\mathcal{B}}_{<} \leq_{\mathcal{L}} \mathtt{DNF}$ *and* $\mathtt{OMVD}^{\mathcal{B}}_{<} \not\geq_s \mathtt{DNF}$.

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

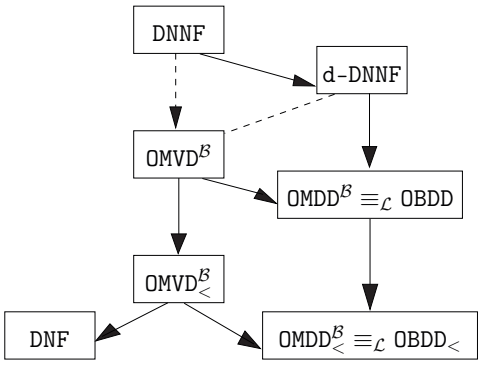

Fig. 2. $\mathtt{OMVD}^{\mathcal{B}}$ and its sublanguages in the DNNF succinctness map. An edge $L_1 \to L_2$ indicates that L_1 is strictly more succinct than L_2. Dashed edges indicate incomplete results. Relations deducible by transitivity are not represented, which means that two fragments not being ancestors to each other are incomparable with respect to succinctness.

4. The Knowledge Compilation Map of MVDs

4.1. *Succinctness*

The results of our succinctness analysis are depicted in Table 1 (see also Figure 2 for the Boolean case).

Table 1. Results about succinctness.

L	MVD	MDD	OMVD	OMDD	$\texttt{OMVD}_<$	$\texttt{OMDD}_<$
MVD	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$
MDD	?	$\leq_s$	?	$\leq_s$	?	$\leq_s$
OMVD	$\not\leq_s$	$\not\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$
OMDD	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\leq_s$	$\not\leq_s$	$\leq_s$
$\texttt{OMVD}_<$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\leq_s$	$\leq_s$
$\texttt{OMDD}_<$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\leq_s$

Theorem 4.1. *The results in Table 1 hold.*

Some of these results are not surprising and directly follow from the fact that OMDD and $\texttt{OMDD}_<$ collapse into OBDD and $\texttt{OBDD}_<$ when domains are Boolean: $\texttt{OMDD}_< \not\leq_s$ OMDD is a straightforward consequence of $\texttt{OBDD}_< \not\leq_s$ OBDD. Some other results are derived from the NNF map in a less immediate way; for instance OMDD $\not\leq_s$ $\texttt{OMVD}_<$ holds since $\texttt{OMVD}_<^{\mathcal{B}} \leq_{\mathcal{L}}$ DNF and OBDD $\sim_s \texttt{OMDD}^{\mathcal{B}}$: if OMDD $\leq_s$ $\texttt{OMVD}_<$ were true, we could derive OBDD $\leq_s$ DNF, which has been proven false.[11]

Some results are harder to get. For instance, in order to prove that OMVD $\not\leq_s$ 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 as an MDD of size polynomial in n. On the other hand, it is possible to prove that any OMVD representing Sol contains at least 2^n nodes. Figure 3 shows the corresponding blow-up on a small instance.

We get $\texttt{OMVD}_< \not\leq_s$ OMDD by generalizing the classical proof of $\texttt{OMDD}_< \not\leq_s$ OBDD: we can show that when compiling the CSP $\bigwedge_{i=1}^{n}[y_i = z_i]$, non-determinism cannot

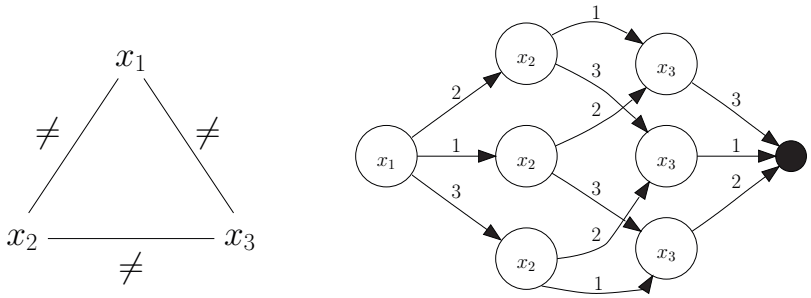

Fig. 3. The "Alldifferent" coloring problem on a complete graph (3 variables, 3 colors) and an OMDD representing its set of solutions ($x_1 < x_2 < x_3$).

compensate for the exponential blow-up caused by ordering all y_i before all z_i.

We also get a direct proof of $\texttt{OMDD}_< \not\leq_s \texttt{OMVD}_<$ by considering another CSP, the problem of n-coloring a star graph with n vertices (see Figure 1 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 OMDD on $<$ 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 OMVD on the same order $<$ and of size polynomial in n.

4.2. *Queries and transformations*

As outlined by Darwiche and Marquis,[11] 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 polynomial time. 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 solution (**MX**), enumerating all solutions (**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 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 users iteratively look for the possible values of the next variables and restrict them according to their preferences.[3]
- "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 sentential entailment (**SE**) requests come from model checking. They can be useful for CSP modeling problems: **SE** 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 compiled constraints, with 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.[8]

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.

Let us now adopt a more formal stance. In a compilation map, the ability of a language L to efficiently achieve operations defines properties of L, which L can satisfy or not. These properties serve as criteria for deciding whether L is appropriate as a target compilation language for a given problem. More precisely, the operations mentioned above can be partitioned into two sets, viz., 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 (e.g., checking the consistency of a CSP).

We formally define the queries and transformations described above; while some are straightforward generalizations of the Boolean case,[11] some others need to be carefully adapted. Let us begin with queries.

Definition 4.2 (Queries)**.** Let L denote a sublanguage of MVD.

- L satisfies **CO** (resp. **VA**) if and only if there exists a polynomial algorithm that maps every MVD ϕ in L to 1 if ϕ has a model (resp. has no countermodel), and to 0 otherwise.
- L satisfies **MC** if and only if there exists a polynomial algorithm that maps every MVD ϕ in L and any $\mathrm{Scope}(\phi)$-assignment $\vec{x}$ to 1 if $\vec{x}$ is a model of ϕ and to 0 otherwise.
- L satisfies **CE** (resp. **IM**) if and only if there exists a polynomial algorithm that maps any MVD ϕ in L, any set of variables $\{x_1, \ldots, x_k\} \subseteq \mathrm{Scope}(\phi)$, and any sequence $\langle A_1, \ldots, A_k \rangle$ of finite sets of integers such that $\forall i \in \{1, \ldots, k\}, A_i \subseteq \mathrm{Dom}(x_i)$, to 1 if $\mathrm{Mod}(\phi)) \subseteq \mathrm{Mod}([x_1 \in A_1] \vee \cdots \vee [x_k \in A_k])$ (resp. $\mathrm{Mod}([x_1 \in A_1] \wedge \cdots \wedge [x_k \in A_k]) \subseteq \mathrm{Mod}(\phi)$) and to 0 otherwise.
- L satisfies **SE** (resp. **EQ**) if and only if there exists a polynomial algorithm

that maps every pair of MVDs $\langle \phi, \psi \rangle$ in L such that $\text{Scope}(\phi) = \text{Scope}(\psi)$, to 1 if $\text{Mod}(\phi) \subseteq \text{Mod}(\psi)$ (resp. $\text{Mod}(\phi) = \text{Mod}(\psi)$), and to 0 otherwise.
- L satisfies **MX** if and only if there exists a polynomial algorithm that maps every MVD ϕ in L to a model of ϕ if there exists one, and that stops without returning anything otherwise.
- L satisfies **CX** if and only if there exists a polynomial algorithm that outputs, for any ϕ in L and any $y \in \text{Scope}(\phi)$, the set of all values taken by y in at least one model of ϕ.
- L satisfies **CT** if and only if there exists a polynomial algorithm that maps every MVD ϕ in L to its number of models $|\text{Mod}(\phi)|$.
- L satisfies **ME** if and only if there exists a polynomial $P(\cdot, \cdot)$ and an algorithm that enumerates, for every MVD ϕ in L, its set of models $\text{Mod}(\phi)$, in time $P(\|\phi\|, |\text{Mod}(\phi)|)$.

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

Definition 4.3. Let I and J be the interpretation functions of some MVDs.

- Given a set of variables $Y \subseteq \text{Scope(I)}$, the *forgetting* of Y in I is the function $\text{Forget}(\text{I}, Y)$ of scope $Z = \text{Scope(I)} \setminus Y$, such that $\text{Forget}(\text{I}, Y)(\vec{z}) = \top$ if and only if there exists a Y-assignment $\vec{y}$ verifying $\text{I}(\vec{z} \,.\, \vec{y}) = \top$.
- Given a set of variables $Y \subseteq \text{Scope(I)}$, the *ensuring* of Y in I is the function $\text{Ensure}(\text{I}, Y)$ of scope $Z = \text{Scope(I)} \setminus Y$, such that $\text{Ensure}(\text{I}, Y)(\vec{z}) = \top$ if and only if for all Y-assignment $\vec{y}$, it holds that $\text{I}(\vec{z} \,.\, \vec{y}) = \top$.
- The *restriction* of I to J, denoted $\text{I}|_{\text{J}}$, is defined by $\text{I}|_{\text{J}} = \text{Forget}(\text{I} \wedge \text{J}, \text{Scope(J)})$.
- Given an assignment $\vec{y}$ of some set of variables $Y \subseteq \text{Scope(I)}$, the *conditioning* of I by $\vec{y}$ is the function $\text{I}|_{\vec{y}}$ of scope $Z = \text{Scope(I)} \setminus Y$, defined by $\text{I}|_{\vec{y}}(\vec{z}) = \text{I}(\vec{y} \,.\, \vec{z})$.

Definition 4.4 (Transformations)**.** Let L denote a sublanguage of MVD.

- L satisfies **CD** if and only if there exists a polynomial algorithm that maps every MVD ϕ in L and every assignment $\vec{x}$ of $X \subseteq \text{Scope}(\phi)$ to an MVD ϕ' in L such that $\text{I}(\phi') = \text{I}(\phi)|_{\vec{x}}$.
- L satisfies **TR** if and only if there exists a polynomial algorithm mapping any MVD ϕ in L, any set of variables $\{x_1, \ldots, x_k\} \subseteq \text{Scope}(\phi)$ and any sequence $\langle A_1, \ldots, A_k \rangle$ of finite sets of integers such that $\forall i \in \{1, \ldots, k\}, A_i \subseteq \text{Dom}(x_i)$, to an MVD ϕ' in L verifying $\text{I}(\phi') = \text{I}(\phi)|_{[x_1 \in A_1] \wedge \cdots \wedge [x_k \in A_k]}$.
- L satisfies **FO** (resp. **EN**) if and only if there exists a polynomial algorithm that maps every MVD ϕ in L and every $Y \subseteq \text{Scope}(\phi)$ to an MVD ϕ' in L such that $\text{I}(\phi') = \text{Forget}(\text{I}(\phi), Y)$ (resp. $\text{I}(\phi') = \text{Ensure}(\text{I}(\phi), Y)$).
- L satisfies **SFO** (resp. **SEN**) if and only if it satisfies **FO** (resp. **EN**) when limited to a single variable (i.e., $|Y| = 1$).

- L satisfies $\vee\mathbf{C}$ (resp. $\wedge\mathbf{C}$) if and only if there exists a polynomial algorithm that maps every finite tuple $\langle \phi_1, \ldots, \phi_k \rangle$ of MVDs in L to an MVD ϕ' in L such that $\mathrm{I}(\phi') = \bigvee_{i=1}^{k} \mathrm{I}(\phi_i)$ (resp. $\mathrm{I}(\phi') = \bigwedge_{i=1}^{k} \mathrm{I}(\phi_i)$).
- L satisfies $\vee\mathbf{BC}$ (resp. $\wedge\mathbf{BC}$) if and only if it satisfies $\vee\mathbf{C}$ (resp. $\wedge\mathbf{C}$) when limited to a pair of MVDs (i.e., $k = 2$).
- L satisfies $\neg\mathbf{C}$ if and only if there exists a polynomial algorithm that maps every MVD ϕ in L to an MVD ϕ' in L such that $\mathrm{I}(\phi') = \neg\, \mathrm{I}(\phi)$.

The results of our analysis of the complexity of queries and transformations are depicted in Tables 2 and 3.

Theorem 4.5. *The results in Tables 2 and 3 hold.*

Results for MDD, OMDD, and $\mathtt{OMDD}_<$ are generally known or follow directly from previous works.[14,5,11] OMDDs have almost the same capabilities as OBDDs, which is not surprising, given how strong their relationship is. However, note that single

Table 2. Results about queries.

L	CO	VA	MC	CE	IM	EQ	SE	MX	CX	CT	ME
MVD	○	○	√	○	○	○	○	○	○	○	○
MDD	○	○	√	○	○	○	○	○	○	○	○
OMVD	√	○	√	√	○	○	○	√	√	○	√
OMDD	√	√	√	√	√	√	○	√	√	√	√
$\mathtt{OMVD}_<$	√	○	√	√	○	○	○	√	√	○	√
$\mathtt{OMDD}_<$	√	√	√	√	√	√	√	√	√	√	√
DNF	√	○	√	√	○	○	○	√	√	○	√
DNNF	√	○	√	√	○	○	○	√	√	○	√
d-DNNF	√	√	√	√	√	?	○	√	√	√	√
OBDD	√	√	√	√	√	√	○	√	√	√	√
$\mathtt{OBDD}_<$	√	√	√	√	√	√	√	√	√	√	√

Note: √ means "satisfies", and ○ means "does not satisfy, unless P = NP". Most results for DNF, DNNF, d-DNNF, OBDD, and $\mathtt{OBDD}_<$ are from the NNF map and are given here as a baseline.

Table 3. Results about transformations.

L	CD	TR	FO	SFO	EN	SEN	$\vee$C	$\vee$BC	$\wedge$C	$\wedge$BC	$\neg$C
MVD	√	○	○	√	○	√	√	√	√	√	?
MDD	√	○	○	√	○	√	√	√	√	√	√
OMVD	√	√	√	√	○	○	?	?	○	○	○
OMDD	√	•	•	•	•	•	•	○	•	○	√
$\mathtt{OMVD}_<$	√	√	√	√	○	○	√	√	○	√	○
$\mathtt{OMDD}_<$	√	•	•	•	•	•	•	√	•	√	√
DNF	√	√	√	√	○	√	√	√	•	√	•
DNNF	√	√	√	√	○	○	√	√	○	○	○
d-DNNF	√	√	○	○	○	○	○	○	○	○	?
OBDD	√	•	•	√	•	√	•	○	•	○	√
$\mathtt{OBDD}_<$	√	•	•	√	•	√	•	√	•	√	√

Note: √ means "satisfies", • means "does not satisfy", and ○ means "does not satisfy, unless P = NP". Most results for DNF, DNNF, d-DNNF, OBDD, and $\mathtt{OBDD}_<$ are from the NNF map and are given here as a baseline.

forgetting and single ensuring, while satisfied by OBDD, are not satisfied by OMDD; this is due to the domain sizes being unbounded.

More interestingly, Theorem 4.5 puts forward the attractiveness of *non-deterministic* OMVDs with respect to transformations. Indeed, $\texttt{OMVD}_<$ satisfies more transformations than $\texttt{OMDD}_<$, "losing" only the negation transformation; yet it is strictly more succinct than $\texttt{OMDD}_<$. Similarly, $\texttt{OMVD}_<$ is strictly more succinct than DNF, and yet they satisfy the same transformations; the only exception is **SEN**, but DNF only satisfies it because it is limited to Boolean variables (it does not satisfy it when extended to finite-domain variables).

As for performances with respect to queries, OMVD and $\texttt{OMVD}_<$ are less interesting than their deterministic counterparts. However, "only" **CT**, **VA**, **EQ**, **IM**, and **SE** are lost when determinism is relaxed (remark 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, which relies essentially on conditioning and context extraction.

5. Experimental Results

Table 4 reports some results[18] 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,[19] which iteratively chooses a variable connected to the greatest number of remaining variables in the constraint graph. The OMDD compiler we use follows the bottom-up approach[4]: each constraint is compiled as an OMDD, and the elementary OMDDs obtained are then combined

Table 4. Results for randomly generated binary CSPs (15 variables, domain size equal to 6).

%T	%C	#SOL	#N OMVD	#N OMDD
70	10	290888073	80	81
	20	136056826	1338	1558
	30	5006576	5662	8132
	40	95131	3315	5005
	50	2367	737	897
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

Note: %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.

by conjunction. The compilation can be pursued by using additional compacting operations, which generalize the merging of isomorphic nodes without preserving determinism[20]; in the end, an OMVD is obtained.

It appears that the interest of non-deterministic structures is not limited to a few specific problems like those used in proofs: indeed, even on random CSPs, allowing non-deterministic compacting operations can sometimes divide by 2 or more the number of nodes in the graph.

6. 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 polynomial time: typically, all transformations (except for **SEN**, which depends on the domain size) satisfied by DNF are also satisfied by $\texttt{OMVD}_<$. 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, $\texttt{OMVD}_<$ is especially appealing for applications relying on transformations (with the exception of 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, $\texttt{OMVD}_<$ is a new fragment in the NNF map, below DNNF and above DNF and OBDD. Moreover, $\texttt{OMVD}_<$ satisfies more queries and transformations than DNNF does.

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

Acknowledgements

This work is partially supported by the project BR4CP ANR-11-BS02-008 of the French National Agency for Research.

References

1. E. C. Freuder and C. D. Elfe, "Neighborhood inverse consistency preprocessing," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1996, pp. 202–208.
2. E. Gelle and R. Weigel, "Interactive configuration using constraint satisfaction techniques," in *Second International Conference on Practical Application of Constraint Technology (PACT)*. Menlo Park, AAAI Press, 1996, pp. 37–44.
3. J. Amilhastre, H. Fargier, and P. Marquis, "Consistency restoration and explanations in dynamic CSPs—Application to configuration," *Artificial Intelligence Journal*, vol. 135, no. 1–2, pp. 199–234, 2002.

4. N. R. Vempaty, "Solving constraint satisfaction problems using finite state automata," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1992, pp. 453–458.
5. 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.
6. H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann, "A constraint store based on multivalued decision diagrams," in *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, 2007, pp. 118–132.
7. H. Cambazard, T. Hadzic, and B. O'Sullivan, "Knowledge compilation for itemset mining," in *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2010, pp. 1109–1110.
8. F. Giunchiglia and P. Traverso, "Planning as model checking," in *Proceedings of the European Conference on Planning (ECP)*, 1999, pp. 1–20.
9. J. Hoey, R. St-Aubin, A. J. Hu, and C. Boutilier, "SPUDD: Stochastic planning using decision diagrams," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999, pp. 279–288.
10. 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.
11. A. Darwiche and P. Marquis, "A knowledge compilation map," *Journal of Artificial Intelligence Research (JAIR)*, vol. 17, pp. 229–264, 2002.
12. R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.
13. H. Fargier and P. Marquis, "On the use of partially ordered decision graphs in knowledge compilation and quantified Boolean formulae," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2006.
14. A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton, "Algorithms for discrete function manipulation," in *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, Nov. 1990, pp. 92–95.
15. T. Hadzic, E. R. Hansen, and B. O'Sullivan, "On automata, MDDs and BDDs in constraint satisfaction," in *Proceedings of the ECAI Workshop on Inference methods based on Graphical Structures of Knowledge (WIGSK)*, 2008.
16. H. Fargier and P. Marquis, "Extending the knowledge compilation map: Krom, Horn, affine and beyond," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2008, pp. 442–447.
17. G. Gogic, H. A. Kautz, C. H. Papadimitriou, and B. Selman, "The comparative linguistics of knowledge representation," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, pp. 862–869.
18. J. Amilhastre, "Représentation par automate d'ensemble de solutions de problèmes de satisfaction de contraintes," Ph.D. dissertation, Université Montpellier II, 1999.
19. 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 Journal on Computing*, vol. 13, no. 3, pp. 566–579, 1984.
20. J. Amilhastre, P. Janssen, and M.-C. Vilarem, "FA minimisation heuristics for a class of finite languages," in *International Workshop on Implementing Automata (WIA)*, 1999, pp. 1–12.
21. A. Niveau, H. Fargier, and C. Pralet, "Representing CSPs with set-labeled diagrams: A compilation map," in *Proceedings of the International Workshop on Graph Structures for Knowledge Representation and Reasoning (GKR)2011 — Revised Selected Papers*,

ser. Lecture Notes in Computer Science, vol. 7205. Springer, 2012, pp. 137–171.
22. C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer, 1998.

Appendix A. Proofs

This appendix gathers all proofs of propositions and theorems in 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 knowledge compilation map,[11] works about MDDs,[14,5] and our knowledge compilation map of set-labeled diagrams[21] (from which a number of the following proofs are directly taken or lightly adapted).

As usual with compilation maps, several proofs use the fact that all languages presented allow Boolean terms and clauses to be represented in linear time and space. The proof is taken from the Boolean case.[11]

Lemma A.1. *Given an order $<$ between variables, any term or clause in propositional logic can be expressed in the $\mathtt{OMDD}^{\mathcal{B}}_{<}$ language in linear time.*

Proof. In order to represent a term $t = \ell_1 \wedge \cdots \wedge \ell_k$ or a clause $c = \ell_1 \vee \cdots \vee \ell_k$ as an OMDD, let us order its literals in such a way that $i < j$ if and only if $\mathrm{Var}(\ell_i) < \mathrm{Var}(\ell_j)$. Then, let us build a chain of nodes $N_1, \ldots, N_k, N_{k+1}$, where for each $i \in \{1, \ldots, k\}$, $\mathrm{Var}(N_i) = \mathrm{Var}(\ell_i)$ and where N_{k+1} is a sink.

For the term, edges are as follows: each non-sink node N_i has one child N_{i+1}, and the edge label is $\top$ if ℓ_i is a positive literal and $\bot$ if ℓ_i is a negative literal.

For the clause, edges are as follows: each node N_i for $i < k$ 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 $\bot$ and the one pointing to N_{k+1} is labeled $\top$. If ℓ_i is a negative literal, it is the opposite. Finally, N_k has the sink as unique successor, via an edge labeled $\top$ if ℓ_k is a positive literal, and $\bot$ otherwise. □

A.1. *MVDs in the NNF map*

Proof of Proposition 3.4. We first show that $\mathtt{OMVD}^{\mathcal{B}}_{<} \leq_{\mathcal{L}} \mathtt{DNF}$. Let ϕ be a DNF, and $<$ a strict total order on $\mathrm{Scope}(\phi)$. First, let us transform each term in ϕ into a Boolean-variable OMDD, ordered with respect to $<$; thanks to Lemma A.1, we know that this process is in linear time, and thus that the term-OMDDs we obtain are of size linear in the size of ϕ. Let x be the smallest variable (for $<$) mentioned in any of the term-OMDDs. Note that x can only appear in a root node; merge all the roots labeled with x into a single root node R. Then, for each term-OMDD that does not mention x, denoting N its root, add two edges $\langle R, N, \top\rangle$ and $\langle R, N, \bot\rangle$.

The resulting graph, denoted ψ, is an element of $\mathtt{OMVD}^{\mathcal{B}}_{<}$ (it does not satisfy the decision property when ϕ contains at least two terms), and has been obtained in time linear in the size of ϕ. We now show that ϕ and ψ have the same interpretation.

Let $\vec{y} \in \mathrm{Mod}(\phi)$; $\vec{y}$ satisfies at least one term in ϕ. If this term contains x, then the root of the corresponding term-OMDD has been merged into the root of ψ; if it does not contain x, by construction, there exists an edge labeled with $\vec{y}|_x$ between the root of ψ and the root of the corresponding term-OMDD. In both cases, there exists a path from the root to the sink of ψ that is compatible with $\vec{y}$.

Reciprocally, let $\vec{y} \in \mathrm{Mod}(\psi)$. There exists a path from the root to the sink of ψ that is compatible with $\vec{y}$. This path traverses one of the term-OMDDs, so $\vec{y}$ satisfies one of the terms in ϕ, therefore it is a model of ϕ. All in all, we can translate any DNF into $\mathtt{OMVD}^{\mathcal{B}}_{<}$ in linear time, hence $\mathtt{OMVD}^{\mathcal{B}}_{<} \leq_{\mathcal{L}} \mathtt{DNF}$.

We now show that $\mathtt{OMVD}^{\mathcal{B}}_{<} \not\geq_s \mathtt{DNF}$. It is clear that $\mathtt{OMDD}^{\mathcal{B}}_{<} \geq_s \mathtt{OMVD}^{\mathcal{B}}_{<}$, since any OMDD is an OMVD. If $\mathtt{OMVD}^{\mathcal{B}}_{<} \geq_s \mathtt{DNF}$ held, we could infer that $\mathtt{OMDD}^{\mathcal{B}}_{<} \geq_s \mathtt{DNF}$, and since $\mathtt{OMDD}^{\mathcal{B}}_{<} \equiv_{\mathcal{L}} \mathtt{OBDD}$ (Prop. 3.3), that $\mathtt{OBDD} \geq_s \mathtt{DNF}$. However, this is false,[11] hence $\mathtt{OMVD}^{\mathcal{B}}_{<} \not\geq_s \mathtt{DNF}$ holds. □

A.2. *Succinctness proofs (Theorem 4.1)*

Proposition A.2. $\mathtt{OMVD}_{<} \not\leq_s \mathtt{OMDD}$.

Proof. Let X be a set of $2n$ Boolean variables, and let $<^b$ be a total strict order on X. We are going to show that there exists a family Γ_n of Boolean functions over X, and a total strict order $<^a$ on X, such that

- Γ_n can be represented as an OMDD on $<^a$ of size polynomial in n;
- all representations of Γ_n as OMVDs on $<^b$ are of size exponential in n.

Let us consider that X is partitioned into two sets, $Y = \{y_1, \ldots, y_n\}$ and $Z = \{z_1, \ldots, z_n\}$, such that the total strict order $<^b$ on X verifies

$$y_1 <^b y_2 <^b \ldots <^b y_n \quad <^b \quad z_1 <^b z_2 <^b \ldots <^b z_n.$$

Let Γ_n be the Boolean function $\bigwedge_{i=1}^{n}[y_i = z_i]$. We consider the total strict order $<^a$ on X, defined as

$$y_1 <^a z_1 \quad <^a \quad y_2 <^a z_2 \quad <^a \quad \ldots \quad <^a \quad y_n <^a z_n.$$

Γ_n can be represented as an OMDD on $<^a$ of size polynomial in n: each constraint $x_i = y_i$ can be represented as an OMDD with only 3 nodes and 4 edges (recall variables in X are Boolean), and since they do not share variables, these small OMDDs can be chained (replacing the sink of each one by the root of the next one) into an OMDD respecting the order $<^a$, which we denote as ϕ^a_n.

Now, we show that the size of any OMVD on $<^b$ representing Γ_n is exponential in n. Let ϕ^b_n be an OMVD on $<^b$ representing Γ_n. Consider an edge $E\langle N, N', a\rangle$ in ϕ^b_n, representing the assignment of some variable y_i (that is, an edge in the first half of the graph).

Let us consider a path p from the root the sink of ϕ^b_n including E. On this path, the value of z_i must be a, by definition of Γ_n. Therefore, there can be no path from the root to N on which y_i is assigned a different value than a.

More generally, any two paths from the root to N' correspond either to identical or to completely disjoint assignments of $y_1, \ldots, y_i$, Each of the possible assignments of variables from Y is thus represented by (at least) one path pointing to a distinct z_1-node. Since there are 2^n assignments of variables from Y, there are at least 2^n nodes labeled z_1. Hence $\|\phi_n^b\| \geq 2^n$: all representations of Γ_n as OMVDs on $<^b$ are of size exponential in n.

All in all, for any total strict variable order $<$, there exists a family Γ_n of Boolean functions that have polynomial representations as OMDDs but no polynomial representation in $\texttt{OMVD}_<$: $\texttt{OMVD}_< \not\leq_s \texttt{OMDD}$.

□

Proposition A.3. $\texttt{OMVD} \not\leq_s \texttt{MDD}$.

Proof. Let $n \in \mathbb{N}^*$. Let Z_n be a set of n variables $\{z_1, \ldots, z_n\}$ of domain $\{1, \ldots, n\}$. Let Σ_n be the Boolean function defined as $\Sigma_n \equiv \text{alldiff}(z_1, \ldots, z_n)$, or equivalently:

$$\Sigma_n \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{i-1} [z_i \neq z_j].$$

There exists an MDD of size polynomial in n representing Σ_n. Indeed, each constraint $[z_i \neq z_j]$ can be represented as an MDD of n^2 edges (n edges for z_i, and for each one, a z_j-node with $n-1$ outgoing edges). Chaining these small MDDs (replacing the sink of each one by the root of the next one), we obtain an MDD of size polynomial in n.

Now, let ϕ be an OMVD representing of Σ_n; we consider, without loss of generality, that its variable order $<$ verifies $z_1 < \cdots < z_n$. We prove that $\|\phi\|$ is exponential in n.

Let $i \in \{1, \ldots, n-1\}$; consider two distinct subsets S and S' of $\{1, \ldots, n\}$ such that $|S| = |S'| = i$. Let $a \in \{1, \ldots, n\}$ such that $a \in S$ and $a \notin S'$. Let $\vec{z}_S$ and $\vec{z}_{S'}$ be two $\{z_1, \ldots, z_i\}$-assignments that cover all values in S and S', respectively. We consider two paths p_S and $p_{S'}$ from the root to the sink of ϕ, compatible with $\vec{z}_S$ and $\vec{z}_{S'}$, respectively. Suppose they go through a same z_{i+1}-node N; p_S assigns a to some variable before encountering N, and $p_{S'}$ assigns a to some variable after having encountered N. Then, the path obtained by joining the first part of p_S (from the root to N) and the second part of $p_{S'}$ (from N to the sink) is a consistent path assigning a to two different variables. Since such a path violates the "alldifferent" constraint, it is impossible, so p_S and $p_{S'}$ must go through two different z_{i+1}-nodes.

Hence, there are at least one z_{i+1}-node for each $S \subseteq \{1, \ldots, n\}$ of cardinal i; since there are $\binom{n}{i}$ such subsets, there are at least $\binom{n}{i}$ z_{i+1}-nodes. The number of nodes in ϕ is thus greater than $\sum_{i=1}^{n-1} \binom{n}{i} = 2^n - 2$.

All in all, family Σ_n has polynomial representations as MDDs, but only exponential representations as OMVDs, which proves $\texttt{OMVD} \not\leq_s \texttt{MDD}$. □

Proposition A.4. $\texttt{OMDD} \not\leq_s \texttt{OMVD}_<$.

Proof. Proposition 3.4 states that $\texttt{OMVD}^{\mathcal{B}}_{<} \leq_s \texttt{DNF}$. If $\texttt{OMDD} \leq_s \texttt{OMVD}_{<}$, it would hold that $\texttt{OMDD} \leq_s \texttt{DNF}$. Since the edges in OMDDs representing propositional formulæ are only labeled with 0 or 1, this would imply that $\texttt{OMDD}^{\mathcal{B}} \leq_s \texttt{DNF}$. But thanks to Proposition 3.3, it would mean that $\texttt{OBDD} \leq_s \texttt{DNF}$—yet it is false.[11] □

Proof of Theorem 4.1. All positive results come directly from simple language inclusions (if $\texttt{L} \supseteq \texttt{L}'$, then surely $\texttt{L} \leq_s \texttt{L}'$).

All negative results stem from Propositions A.2, A.3 and A.4, and from the fact that if $\texttt{L}_1 \not\leq_s \texttt{L}_2$, then every L such that $\texttt{L}_1 \leq_s \texttt{L}$ verifies $\texttt{L} \not\leq_s \texttt{L}_2$, and every 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. □

A.3. *First proofs of transformations (Theorem 4.5)*

In this section, we prove the satisfaction of several transformations on some languages; these results are used in subsequent proofs.

Lemma A.5. *Algorithm 1 runs in time polynomial in the size of its input MVD. When given an MVD ordered with respect to its input order $<$, the output MVD is also ordered with respect to $<$.*

Proof. Let us consider an MVD ϕ, a set $X = \{x_1, \ldots, x_k\}$ of variables in $\mathrm{Scope}(\phi)$, a sequence $\mathcal{A} = \langle A_1, \ldots, A_k \rangle$ of sets of values such that $\forall i \in \{1, \ldots, k\}, A_i \subseteq \mathrm{Dom}(x_i)$, and a strict total order $<$ on $\mathrm{Scope}(\phi)$. Let $d = \max_{1 \leq i \leq k} |A_i|$.

Complexity. The number of iterations of the loop starting on line 5 is bounded by the number of edges in ϕ, and thus by $\|\phi\|$. Reduction (line 9) is polynomial (Prop. 2.9). On line 10, it is well known that indexing nodes in a DAG in ascending order can be done in time polynomial. Note that r is simply the number of non-sink nodes.

Next, on line 15, it should be noted that the number of incoming edges of a node is always bounded by dr (in the worst case, there is one edge per node and per value); this is independant from the fact that the algorithm adds edges as the loop goes. Moreover, on line 16, the number of children of a node is obviously always bounded by r. All in all, on line 20, we only add at most dr^2 edges to the graph; note that there can be no duplicate edges in a graph (edges having the same source, the same destination, and the same label), so if an edge is already present, it is not "added again". In the loop that starts on line 12, the number of added edges is bounded by dr^3.

Finally, the root merging process on line 23 is linear in dr, and the loop starting on line 25 adds at most dr edges to the new root node R.

To conclude, the algorithm runs in time polynomial in $\|\phi\|$, d, and r. Since $\forall i \in \{1, \ldots, k\}, |A_i| \leq |\mathrm{Dom}(x_i)| \leq \|\phi\|$, it holds that $d \leq \|\phi\|$, and since there cannot be more non-sink nodes than edges (every non-sink node has at least one outgoing edge), $r \leq \|\phi\|$. Consequently, the algorithm is polynomial in $\|\phi\|$.

Ordering. Let us suppose that ϕ is ordered with respect to $<$, the order given as input to the algorithm. In the first part (until line 21), edges and nodes are

Algorithm 1 Syntactic restriction of an MVD to a "multivalued term".

1: **input:** an MVD ϕ
2: **input:** a set $X = \{x_1, \ldots, x_k\}$ of variables in $\mathrm{Scope}(\phi)$
3: **input:** a sequence $\mathcal{A} = \langle A_1, \ldots, A_k \rangle$ of sets of values, such that $\forall i \in \{1, \ldots, k\}, A_i \subseteq \mathrm{Dom}(x_i)$
4: **input:** a strict total order $<$ on $\mathrm{Scope}(\phi)$
5: **for each** edge $E = \langle N, N', a \rangle$ in ϕ such that $\mathrm{Var}(N) \in X$ **do**
6: let $x_i := \mathrm{Var}(N)$
7: **if** $a \notin A_i$ **then**
8: remove E from ϕ
9: reduce ϕ
10: associate with each node an index i, such that N_0 is the sink, and if there is an edge from N_i to N_j, then $i > j$
11: let r be the index of the root
12: **for** $i := 1$ **to** $r - 1$ **do**
13: **if** $\mathrm{Var}(N_i) \in X$ **then**
14: let $\mathcal{E} := \emptyset$
15: **for each** $E \in \mathrm{In}(N_i)$ **do**
16: **for each** $N' \in \mathrm{Ch}(N_i)$ **do**
17: let $E' := \langle \mathrm{Src}(E), N', \mathrm{Lbl}(E) \rangle$
18: let $\mathcal{E} := \mathcal{E} \cup \{E'\}$
19: remove N_i, $\mathrm{In}(N_i)$, and $\mathrm{Out}(N_i)$ from ϕ
20: add all edges in $\mathcal{E}$ to ϕ
21: **if** $\mathrm{Var}(N_r) \in X$ **then**
22: let $\mathcal{R}$ be the set of children of the root that are labeled with a variable x minimal for $<$
23: create a new node R that results from the merging of all nodes in $\mathcal{R}$
24: remove from ϕ all nodes in $\mathcal{R}$ and their incoming and outgoing edges
25: **for each** $N' \in \mathrm{Ch}(N_r)$ **do**
26: **for each** $a \in \mathrm{Dom}(x)$ **do**
27: add an outgoing edge $\langle R, N', a \rangle$ to R
28: remove N_r and its outgoing edges, and add R as the new root
29: remove from $\mathrm{Scope}(\phi)$ all variables in X

removed from ϕ, but no nodes are added; the resulting MVD is thus still ordered with respect to $<$. Now, if the procedure enters the test on line 21, it removes the former root N_r and adds a new one R. However, this new root is labeled by the smallest variable x (with respect to $<$) labeling any child of N_r; and all children of N_r labeled with x have been merged into R. Hence, R has no child the label of which is larger or equal to x; and since the remainder of the graph is not modified, the resulting MVD is still ordered with respect to $<$. □

Proposition A.6. `MVD` *and its subclasses satisfy* **CD**.

Proof. We will use Algorithm 1, showing that when the size of each input A_i is 1, it computes a conditioning, and moreover maintains the decision property.

Let us consider an MVD ϕ, a set $X = \{x_1, \ldots, x_k\}$ of variables in $\text{Scope}(\phi)$, and an X-assignment $\vec{x}$. We will show that when given ϕ, X, the sequence $\mathcal{A} = \langle\{\vec{x}|_{x_i}\}\rangle_{1 \leq i \leq k}$ of the singletons of assigned values in $\vec{x}$, and any variable order $<$, Algorithm 1 outputs an MVD ϕ' such that $\text{I}(\phi') = \text{I}(\phi)|_{\vec{x}}$.

Let us denote $Z = \text{Scope}(\phi) \setminus X = \text{Scope}(\phi')$ (line 29), and let $\vec{z}$ be any Z-assignment. By definition of the conditioning, we must prove that

$$\text{I}(\phi')(\vec{z}) = \top \iff \text{I}(\phi)(\vec{x} \,.\, \vec{z}) = \top.$$

Sufficient Condition. Suppose $\text{I}(\phi)(\vec{x} \,.\, \vec{z}) = \top$; there exists at least one path from the root to the sink of ϕ that is compatible with $\vec{x} \,.\, \vec{z}$. The loop over edges, starting on line 5, only removes X-edges that are not compatible with $\vec{x}$, so all compatible paths remain. Let us consider one of these paths p, and let us consider two subsequent edges along p, denoted $E_1 = \langle N, N', a_1\rangle$ and $E_2\langle N', N'', a_2\rangle$, such that $\text{Var}(N') \in X$.

In the loop over nodes, N' is removed, together with its incoming and outgoing edges—thus E_1 and E_2 are removed. However, a new edge $\langle N, N'', a_1\rangle$ is added; consequently, there still exists at least one path p' compatible with $\vec{x} \,.\, \vec{z}$.

Finally, if the root of ϕ is labeled by a variable in X, it is replaced by a new root R. Let $\langle N_r, N', a\rangle$ be the first edge in p'. If $\text{Var}\, N'$ is minimal for $<$, then the new root R is labeled by $\text{Var}\, N'$ and all outgoing edges of N' are outgoing edges of R: removing the first edge of p', and replacing N' by R in the next edge, we get a path from the root to the sink of ϕ' that is compatible with $\vec{x} \,.\, \vec{z}$. If $\text{Var}\, N'$ is not minimal for $<$, then the new root R has one outgoing edge leading to N' per value in $\text{Dom}(\text{Var}(R))$: surely, one is compatible with $\vec{x}$. Adding this edge at the beginning of p', we get a path from the root to the sink of ϕ' that is compatible with $\vec{x} \,.\, \vec{z}$.

In all cases, there exists in ϕ' a path from the root to the sink that is compatible with $\vec{x} \,.\, \vec{z}$; since all X-nodes have been removed, the path is compatible with $\vec{z}$. Hence $\text{I}(\phi')(\vec{z}) = \top$.

Necessary Condition. Suppose $\text{I}(\phi')(\vec{z}) = \top$; let p be a path from the root to the sink of ϕ' that is compatible with $\vec{z}$, and thus with $\vec{x} \,.\, \vec{z}$, since ϕ' does not mention any variable from X. Even if the root of ϕ' has been modified (lines 21–28), we know that there was a path compatible with $\vec{x} \,.\, \vec{z}$ before the modification, since the new root R then was a child of the former root, via an edge compatible with $\vec{x}$ (all edges incompatible with $\vec{x}$ have been removed from ϕ in the loop starting on line 5).

Let $E = \langle N, N', a\rangle$ be an edge along p; either E was already in ϕ, or it has been added on line 20 to bypass an X-node N_x (with $\text{Var}(N_x) = x$), in which case it corresponds to two edges in the MVD of the previous loop, $\langle N, N_x, a\rangle$ and

$\langle N_x, N', a_x\rangle$, with $\vec{x}|_x = a_x$ (all edges incompatible with $\vec{x}$ had been removed). Since it is the case for any E along p, p results from the transformation of at least one path p' in ϕ, that is compatible with $\vec{x}\,.\,\vec{z}$. This proves that $\mathrm{I}(\phi)(\vec{x}\,.\,\vec{z}) = \top$.

Decision Property. Let us suppose ϕ is an MDD; we prove that ϕ' also is an MDD. First, remark that removing nodes or edges does not compromise the decision property. Let us study the two moments when the procedure actually adds edges, viz., lines 20 and 25–27.

Now, note that the loop over edges (starting from line 5) removes any edge incompatible with $\vec{x}$; since ϕ is an MDD, there can be at most one outgoing edge per node that is compatible with $\vec{x}$. Consequently, after this loop, every X-node has at most one outgoing edge—and thus at most one child; and on line 9, reduction removes those that have no child at all.

Therefore, on line 20, we simply replace each incoming edge $\langle N, N_i, a\rangle$ of N_i by a single edge $\langle N, N', a\rangle$, with N' the unique child of N_i. Since no other edge is added at this step, it preserves the decision property.

Finally, in the root replacement step, we know that the former root N_r has exactly one child N', and thus the new root R is simply a copy of N'. Since N' is removed from ϕ, the procedure does not even enter the loop on line 25. Once again, this step maintains the decision property. □

Proposition A.7. `OMVD` *and* $\texttt{OMVD}_<$ *satisfy* **TR**, **FO**, *and* **SFO**.

Proof. We show that Algorithm 1, when given an ordered graph, computes a term restriction. Let us consider an OMVD ϕ on some variable order $<$, a set $X = \{x_1, \ldots, x_k\}$ of variables in $\mathrm{Scope}(\phi)$, and a sequence $\mathcal{A} = \langle A_1, \ldots, A_k\rangle$ of sets of values such that $\forall i \in \{1, \ldots, k\}, A_i \subseteq \mathrm{Dom}(x_i)$. Let ϕ' be the output of Algorithm 1 when given as input ϕ, X, $\mathcal{A}$, and $<$. We show that $\mathrm{I}(\phi') \equiv \mathrm{I}(\phi)|_{[x_1\in A_1]\wedge\cdots\wedge[x_k\in A_k]}$.

Let us denote $Z = \mathrm{Scope}(\phi) \setminus X = \mathrm{Scope}(\phi')$ (line 29), and let $\vec{z}$ be any Z-assignment. By definition of restriction and forgetting, we must prove that

$$\mathrm{I}(\phi')(\vec{z}) = \top \iff \exists \vec{x} \in \mathrm{Dom}(X), (\mathrm{I}(\phi) \wedge [x_1 \in A_1] \wedge \cdots \wedge [x_k \in A_k])(\vec{z}\,.\,\vec{x}) = \top$$

Sufficient Condition. Suppose that there exists an X-assignment $\vec{x}$ such that $(\mathrm{I}(\phi) \wedge [x_1 \in A_1] \wedge \cdots \wedge [x_k \in A_k])(\vec{z}\,.\,\vec{x}) = \top$. Consequently, for any $x_i \in X$, it must hold that $\vec{x}|_{x_i} \in A_i$; and there must be at least one path p from the root to the sink of ϕ that is compatible with $\vec{x}\,.\,\vec{z}$. Since the loop over edges, starting on line 5, only removes X-edges that are not compatible with $\vec{x}$, p remains.

In a fashion similar to the sufficient condition in the proof of Proposition A.6, we can show that p corresponds to a path from the root to the sink of ϕ' that is compatible with $\vec{x}\,.\,\vec{z}$, and since all X-nodes have been removed, compatible with $\vec{z}$. Hence $\mathrm{I}(\phi')(\vec{z}) = \top$.

Necessary Condition. Suppose $\mathrm{I}(\phi')(\vec{z}) = \top$; let p be a path from the root to the sink of ϕ' that is compatible with $\vec{z}$. In a fashion similar to the necessary condition in the proof of Proposition A.6, we can show that p corresponds to at least one path from the root to the sink of ϕ that is compatible with $\vec{z}$. Let us

consider one of these paths, p'; there necessarily exists at least one X-assignment $\vec{x}$ that is compatible with p', because ϕ is ordered, and thus, each variable can only be encountered once along p'; each variable can be assigned at most once, there is no contradiction. Moreover, along p', all edges $\langle N, N', a\rangle$ such that $\mathrm{Var}(N) = x_i \in X$ verify $a \in A_i$, because edges that do not verify this are removed in the first loop (line 5), and we know that the procedure transforms p' into p. Hence, there exists at least one X-assignment $\vec{x}$ compatible with p' and verifying $\forall i \in \{1, \ldots, k\} \vec{x}|_{x_i} \in A_i$; such an assignment is a model of $[x_1 \in A_1] \wedge \cdots \wedge [x_k \in A_k]$, and p' being compatible with both $\vec{z}$ and $\vec{x}$, $\vec{z}\,.\,\vec{x}$ is a model of $\mathrm{I}(\phi)$: $(\mathrm{I}(\phi) \wedge [x_1 \in A_1] \wedge \cdots \wedge [x_k \in A_k])(\vec{z}\,.\,\vec{x}) = \top$.

In conclusion, Algorithm 1 computes a term restriction, and since we know by Lemma A.5 that it runs in polynomial time and maintains the ordering property (but not the decision property), it holds that `OMVD` and $\texttt{OMVD}_<$ satisfy **TR**, and thus **FO** (forgetting a variable x amounts to restricting it the term $[x \in \mathrm{Dom}(x)]$) and **SFO**. □

Proposition A.8. `MDD`*,* `OMDD`*, and* $\texttt{OMDD}_<$ *satisfy* $\neg$**C**.

Proof. Using the two-leaf form of MDDs (Section 2.3.2), "negating" an MDD is very simple: it is sufficient to swap its two leaves. Since any assignment corresponds to exactly one path from the root to a leaf, an assignment is a model of the original MDD if and only if it is not a model of the resulting MDD. This procedure is done in constant time, and obviously preserves ordering.

□

Proposition A.9. $\texttt{OMDD}_<$ *and* $\texttt{OMVD}_<$ *satisfy* $\wedge$**BC**.

Proof. We can use the algorithm in Figure 2, adapted from that on OBDDs.[12] It applies on non-empty OMVDs of a same variable order (if one OMVD is empty, it is trivial to compute the conjunction). A cache is maintained to avoid computing twice the same pair 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 in polynomial time. □

A.4. *Proofs of queries (Theorem 4.5)*

Proposition A.10. *Every sublanguage of* `MVD` *we defined satisfies* **MC**.

Proof. Since these languages satisfy **CD** (Proposition A.6), we can test whether $\vec{x}$ is a model of ϕ by conditioning the MVD by $\vec{x}$; indeed, we get either the empty MVD (then the assignment is not a model) or the sink-only MVD (then $\vec{x}$ is a model). Hence all these languages satisfy **MC**. □

Algorithm 2 `conjunct_step`(N_1, N_2): returns an OMVD$_<$ that is the conjunction of the two OMVDs$_<$ of which N_1 and N_2 are roots.

if the cache contains the key (N_1, N_2) **then**
  return the OMVD corresponding to this key in the cache
if $\mathrm{Out}(N_1) = \emptyset$ (the sink of N_1 is reached) **then**
  return a copy of the OMVD rooted at N_2
if $\mathrm{Out}(N_2) = \emptyset$ (the sink of N_2 is reached) **then**
  return a copy of the OMVD rooted at N_1
if $\mathrm{Var}(N_1) = \mathrm{Var}(N_2)$ **then**
  Let $x := \mathrm{Var}(N_1) = \mathrm{Var}(N_2)$
  Create a node N' labeled by x
  for each $\omega \in \mathrm{Dom}(x)$, $E_1 \in \mathrm{Out}(N_1)$, $E_2 \in \mathrm{Out}(N_2)$ **do**
    if $\omega = \mathrm{Lbl}(E_1) = \mathrm{Lbl}(E_2)$ **then**
      Let $\phi_\omega :=$ `conjunct_step`$(\mathrm{Dest}(E_1), \mathrm{Dest}(E_2))$
      Add an edge coming out of N', labeled by ω and pointing to the root of ϕ_ω
  return the OMVD rooted at N'
$N_i = \mathrm{Argmin}_<(\mathrm{Var}(N_1), \mathrm{Var}(N_2))$,
$N_j = \mathrm{Argmax}_<(\mathrm{Var}(N_1), \mathrm{Var}(N_2))$
Create a node N'_i labeled by $\mathrm{Var}(N_i)$
for each $E \in \mathrm{Out}(N_i)$ **do**
  Let $\phi_E :=$ `conjunct_step`$(\mathrm{Dest}(E), N_j)$
  Add an edge coming out of N'_i, labeled by $\mathrm{Lbl}(E)$ and pointing to the root of ϕ_E
return N'

Proposition A.11. `MVD` *and* `MDD` *do not satisfy any query we defined, besides* **MC***, unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Proposition 3.3 states that $\texttt{BDD} \equiv_{\mathcal{L}} \texttt{MDD}^{\mathcal{B}}$, and $\texttt{MDD}^{\mathcal{B}} \subseteq \texttt{MDD} \subseteq \texttt{MVD}$, so `MDD` and `MVD` cannot satisfy any query that `BDD` does not satisfy. We know from the Boolean knowledge compilation map[11] that `BDD` does not satisfy **CO** or **VA** unless $\mathsf{P} = \mathsf{NP}$, therefore neither `MDD` nor `MVD` satisfies **CO** or **VA** unless $\mathsf{P} = \mathsf{NP}$. Since the satisfaction of **CE**, **MX**, **CX**, or **ME** implies the satisfaction of **CO**, and the satisfaction of **IM**, **EQ**, **SE**, or **CT** implies the satisfaction of **VA**, none of these queries can be satisfied by `MDD` or `MVD` unless $\mathsf{P} = \mathsf{NP}$. □

A.4.1. *Ordered languages*

Proposition A.12. `OMVD` *and* $\texttt{OMVD}_<$ *do not satisfy* **VA***,* **IM***,* **EQ***,* **SE***, or* **CT***, unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Proposition 3.4 states that $\texttt{OMVD}^{\mathcal{B}}_< \leq_{\mathcal{L}} \texttt{DNF}$. If `OMVD` or $\texttt{OMVD}_<$ satisfied **VA**,

it would be the case for $\texttt{OMVD}^{\mathcal{B}}_{<}$ by inclusion: we could check in time polynomial the validity of any DNF, yet it is impossible unless $\mathsf{P} = \mathsf{NP}$.

Since the satisfaction of **IM**, **EQ**, **SE**, or **CT** implies the satisfaction of **VA**, OMVD and $\texttt{OMVD}_{<}$ cannot satisfy any of them unless $\mathsf{P} = \mathsf{NP}$. □

Proposition A.13. OMVD *and its sublanguages satisfy* **MX** *and* **CO**.

Proof. In an OMVD, every path from the root to the sink corresponds to at least one model. Indeed, the ordering property imposes that no variable can be encountered twice on a given path. Let ϕ be an OMVD, and p be a path from the root to the sink of ϕ. Denoting $\langle N_1, N_2, a_1\rangle, \langle N_2, N_3, a_2\rangle, \ldots, \langle N_k, N_{k+1}, a_k\rangle$ the edges along this path, any Scope(ϕ)-assignment $\vec{x}$ in which Var(N_i) is assigned to a_i (for $1 \leq i \leq k$) is a model of ϕ, by construction.

Consequently, extracting a model of an OMVD boils down to choosing a path, retrieving the associated assignment, and complete it by assigning random values to variables in Scope(ϕ) that are not mentioned along the chosen path. The only case when it does not work is when there is no path to choose, i.e., when the graph is empty; in this case the OMVD is inconsistent, and there is no model to find. Hence, OMVD satisfies **MX**, and thus **CO**. Any MVD in a sublanguage of OMVD is obviously an OMVD, so the procedure also works for $\texttt{OMVD}_{<}$, OMDD, and $\texttt{OMDD}_{<}$. □

Proposition A.14. OMVD *and its sublanguages satisfy* **CE**.

Proof. Checking whether I(ϕ) entails $[x_1 \in A_1] \vee \cdots \vee [x_k \in A_k]$ is equivalent to checking whether I(ϕ) $\wedge [x_1 \notin A_1] \wedge \cdots \wedge [x_k \notin A_k]$ is inconsistent. Clearly enough, $[x \notin A] = [x \in \mathrm{Dom}(x) \setminus A]$; and using the definition of restriction, we get that I(ϕ) entails $[x_1 \in A_1] \vee \cdots \vee [x_k \in A_k]$ if and only if $\mathrm{I}(\phi)|_{[x_1 \in \mathrm{Dom}(x_1) \setminus A_1] \wedge \cdots \wedge [x_k \in \mathrm{Dom}(x_k) \setminus A_k]}$ is inconsistent. Since OMVD satisfies **TR** (Prop. A.7) and **CO** (Prop. A.13), it satisfies **CE**. By inclusion, $\texttt{OMVD}_{<}$, OMDD, and $\texttt{OMDD}_{<}$ also satisfy **CE**. □

Proposition A.15. OMVD *and its sublanguages satisfy* **CX**.

Proof. Let ϕ be an OMVD and X be a set of variables in Scope(ϕ). The context of a variable x is the set of values that are part of at least one model of I(ϕ). To check whether a value a is in the context of x in ϕ, it is thus sufficient to assign x to a, which amounts to a conditioning, and to check whether the result is consistent. Since OMVD satisfies **CD** (Prop. A.6) and **CO** (Prop. A.13), this process is polynomial. Applying it for every value in Dom(x), we obtain the context of x; since $|\mathrm{Dom}(x)| \leq \|\phi\|$, the process is also polynomial. Last, $|X| \leq |\mathrm{Scope}(\phi)| \leq \|\phi\|$, so retrieving the context of all variables in X is polynomial. Hence OMVD satisfies **CX**, and by inclusion, $\texttt{OMVD}_{<}$, OMDD, and $\texttt{OMDD}_{<}$ also satisfy **CX**. □

Proposition A.16. OMVD *and its sublanguages satisfy* **ME**.

Proof. This holds because OMVD satisfies **CD** (Prop. A.6) and **CO** (Prop. A.13). Enumerating models can be done by exploring the entire assignment tree, as done

in the Boolean case[11] (checking consistency after each branching avoids having to backtrack, so the size of the search tree is polynomial in the size of the output). □

A.4.2. *Decision diagrams*

Proposition A.17. `OMDD` *and* $\texttt{OMDD}_<$ *satisfy* **CT** *and* **VA**.

Proof. Let ϕ be an OMDD, and p a path from the root to the sink of ϕ. Denoting $\langle N_1, N_2, a_1\rangle, \langle N_2, N_3, a_2\rangle, \ldots, \langle N_k, N_{k+1}, a_k\rangle$ the edges along p, any $\mathrm{Scope}(\phi)$-assignment $\vec{x}$ in which $\mathrm{Var}(N_i)$ is assigned to a_i (for $1 \leq i \leq k$) is compatible with p, by construction, and is thus a model of ϕ. However, it cannot be compatible with any other path in ϕ: if it were the case, ϕ would be non-deterministic.

All in all, each path in ϕ can be associated with the set of models of ϕ that are compatible with it, and these models are not compatible with any other path: models are partitioned among all paths. Counting the models of ϕ thus amounts to counting the models associated with each path.

The number of models associated with a given path p depends on the variables it mentions: if it mentions all variables in $\mathrm{Scope}(\phi)$, then it is compatible with exactly one model; if it mentions all variables but one, denoted x, then it is associated with $|\mathrm{Dom}(x)|$ models; etc.

Given a node N in an OMDD ϕ on $<$, let us denote $\mathrm{Scope}(N)$ the set of all variables greater or equal to $\mathrm{Var}(N)$ with respect to $<$. To count the number of models of ϕ, we will associate with each node N, the number n_N of $\mathrm{Scope}(N)$-assignments that are compatible with any path from N to the sink. We start by noticing that $n_{\mathrm{Sink}(\phi)} = 1$: indeed, $\mathrm{Scope}(\mathrm{Sink}(\phi)) = \emptyset$, and the unique $\emptyset$-assignment is compatible with all paths from the sink to itself (there is none). Then, the outgoing edges of an internal node N cannot be compatible with a same $\mathrm{Scope}(N)$-assignment, (recall that models are partitioned among paths, since ϕ satisfies the decision property). Hence

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

with n_E the number of $\mathrm{Scope}(N)$-assignments compatible with E. Denoting $D = \mathrm{Dest}(E)$, this number is simply n_D, multiplied by $\prod_{\mathrm{Var}(N)<x<\mathrm{Var}(D)} |\mathrm{Dom}(x)|$, to take into account all variables not mentioned in this subgraph.

The number $n_{\mathrm{Root}(\phi)}$ is then the number of $\mathrm{Scope}(\mathrm{Root}(\phi))$-assignments that are compatible with some path from the root to the sink of ϕ; to obtain the number of models of ϕ, we once again have to multiply it, this time by $\prod_{x \in \mathrm{Scope}(\phi) \setminus \mathrm{Scope}(\mathrm{Root}(\phi))} |\mathrm{Dom}(x)|$, to take into account the variables that are smaller than the root variable (with respect to $<$).

This number can be inductively computed, by traversing the graph from the sink to the root. The whole process being polynomial in $\|\phi\|$ (each edge is crossed (backwards) once), `OMDD` satisfies **CT**.

From **CT**, we can test in polynomial time whether the number of models is equal to the number of possible assignments of the variables, i.e., whether the OMDD is valid, thus OMDD supports **VA**. □

Proposition A.18. OMDD *and* $\text{OMDD}_<$ *satisfy* **IM**.

Proof. Let ϕ be an MVD; checking whether $[x_1 \in A_1] \wedge \cdots \wedge [x_k \in A_k]$ entails $\mathrm{I}(\phi)$ is equivalent to checking whether $\neg\,\mathrm{I}(\phi) \wedge [x_1 \in A_1] \wedge \cdots \wedge [x_k \in A_k]$ is inconsistent; now, it is consistent if and only if $(\neg\,\mathrm{I}(\phi))|_{[x_1 \in A_1] \wedge \cdots \wedge [x_k \in A_k]}$ is consistent. Since OMDD satisfies $\neg$**C** (Prop. A.8), **TR** (Prop. A.31), and **CO** (Prop. A.13), this verification can be done in time polynomial, and OMDD hence satisfies **IM**. By inclusion, $\text{OMDD}_<$ also satisfies **IM**. □

Proposition A.19. $\text{OMDD}_<$ *satisfies* **SE** *and* **EQ**.

Proof. Let ϕ and ψ be two MVDs; Checking whether $\mathrm{I}(\phi)$ entails $\mathrm{I}(\psi)$ is equivalent to checking whether $\mathrm{I}(\phi) \wedge \neg\,\mathrm{I}(\psi)$ is inconsistent. Since $\text{OMDD}_<$ satisfies $\neg$**C** (Prop. A.8), $\wedge$**BC** (Prop. A.9), and **CO** (Prop. A.13), this verification can be done in time polynomial, and $\text{OMDD}_<$ hence satisfies **SE**. This implies that it also satisfies **EQ**, since checking whether $\mathrm{I}(\phi)$ is equivalent to $\mathrm{I}(\psi)$ amounts to checking whether $\mathrm{I}(\phi)$ entails $\mathrm{I}(\psi)$ and $\mathrm{I}(\psi)$ entails $\mathrm{I}(\phi)$. □

Proposition A.20. OMDD *does not satisfy* **SE** *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Let ϕ and ψ be two MVDs; checking whether $\mathrm{I}(\phi) \wedge \mathrm{I}(\psi)$ is consistent is equivalent to checking whether $\mathrm{I}(\phi)$ does not entail $\neg\,\mathrm{I}(\psi)$. Since OMDD satisfies $\neg$**C** (Prop. A.8), if it also satisfied **SE**, we would have a polynomial algorithm for checking the consistency of the conjunction of two OMDDs, and in particular, of two OBDDs (possibly on different variable orders), since $\text{OMDD}^{\mathcal{B}} \equiv_{\mathcal{L}} \text{OBDD}$ (Prop. 3.3). However, Meinel and Theobald showed in Lemma 8.14 of their book about OBDDs[22] that checking the consistency of the conjunction of two OBDDs is NP-complete. Therefore OMDD cannot satisfy **SE** unless $\mathsf{P} = \mathsf{NP}$.

□

Proposition A.21. OMDD *satisfies* **EQ**.

Proof. The procedure described in Algorithm 3 is adapted from the proof of Theorem 8.11 of the OBDD book by Meinel and Theobald.[22]

The procedure is based on the following equation, holding for any variable $x \in \mathrm{Scope}(\Phi) \cup \mathrm{Scope}(\Psi)$:

$$\mathrm{I}(\Phi) \equiv \mathrm{I}(\Psi) \iff \forall a \in \mathrm{Dom}(x), \mathrm{I}(\Phi)|_{x=a} \equiv \mathrm{I}(\Psi)|_{x=a}. \tag{A.1}$$

Now, the procedure keeps a list L of pairs of subgraphs from Φ and Ψ respectively. We show that this list has the following property:

$$\mathrm{I}(\Phi) \equiv \mathrm{I}(\Psi) \iff \forall \langle \phi, \psi \rangle \in L, \mathrm{I}(\phi) \equiv \mathrm{I}(\psi). \tag{A.2}$$

Algorithm 3 Given two OMDDs Φ and Ψ, checks whether $\mathrm{I}(\Phi) \equiv \mathrm{I}(\Psi)$ holds.

```
1: let L := {⟨Φ, Ψ⟩}
2: for each node N in Φ, ordered from the root to the sink, except the sink do
3:    let φ be the subgraph rooted at N
4:    let ψ be one of the OMDDs such that ⟨φ, ψ⟩ ∈ L
5:    for each ψ' such that ⟨φ, ψ'⟩ ∈ L do
6:       if ψ' ≢ ψ then
7:          return false
8:       remove ⟨φ, ψ'⟩ from L
9:    let x := Var(N)
10:   for each a ∈ Dom(x) do
11:      if there exists E ∈ Out(N) such that Lbl(E) = a then
12:         let φ_E be the OMDD rooted at Dest(E)
13:      else
14:         let φ_E be the empty OMDD
15:      add the pair ⟨φ_E, ψ|_{x=a}⟩ to L
16:   remove the pair ⟨φ, ψ⟩ from L
17: for each pair ⟨φ, ψ⟩ ∈ L do
18:   if I(φ) ≢ I(ψ) then
19:      return false
20: return true
```

L is initialized with $\langle \Phi, \Psi \rangle$, so this is trivially true at the beginning. On line 8, we remove only redundant pairs. On line 16, we have replaced the last pair $\langle \phi, \psi \rangle$ by a set of pairs according to the decomposition scheme (A.1). Hence, while the procedure runs, we know that $\mathrm{I}(\Phi) \equiv \mathrm{I}(\Psi)$ holds if and only if $\forall \langle \phi, \psi \rangle \in L, \mathrm{I}(\phi) \equiv \mathrm{I}(\psi)$, q.e.d. (A.2).

We use this equivalence to show that the algorithm is sound and complete. On line 6, we encounter two inconsistent pairs: it is impossible that ϕ be equivalent to both ψ and ψ'. Thus there exists a pair in L such that $\mathrm{I}(\phi) \not\equiv \mathrm{I}(\psi)$, and hence $\mathrm{I}(\Phi) \not\equiv \mathrm{I}(\Psi)$. We can return false. At the end of the algorithm, we have tested the equivalence of all pairs in L (lines 17–19), we can thus return true.

Let us now show that the algorithm is polynomial. Each node of Φ is treated once. For a given node N, we add exactly $d_N = |\mathrm{Dom}(\mathrm{Var}(N))|$ pairs to L (line 15), d_N being bounded by $\|\Phi\|$. We also remove one pair (line 16). Hence L contains at most $d \cdot |\mathrm{Nodes}\, \Phi|$ pairs, with $d = \max_N d_N$.

For each node, we make at most $d \cdot |\mathrm{Nodes}\, \Phi|$ equivalence tests at line 6; equivalence tests being on OMDDs of the same variable order, they can be done in time polynomial (Prop. A.19). The traversal of the graph (lines 2–16) is thus polynomial.

Once we have traversed the entire Φ, we know by construction that the only pairs $\langle \phi, \psi \rangle$ left in L are such that ϕ is either the sink-only or the empty OMDD, so ϕ is an OMDD of the same variable order as Ψ: all the equivalence tests of line 18

(there can be at most $d \cdot |\text{Nodes}\,\Phi|$ of them) can be done in polynomial time. All in all, OMDD satisfies **EQ**. □

A.5. *Proofs of transformations (Theorem 4.5)*

Proposition A.22. $\texttt{OMVD}_<$ *and* OMVD *do not satisfy* $\neg\mathbf{C}$ *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. The negation of a CNF is a DNF, which is linearly translatable into an OMVD of any order (Proposition 3.4). Thus, if $\texttt{OMVD}_<$ or OMVD satisfied $\neg\mathbf{C}$, we would have a polynomial algorithm translating any CNF into an equivalent OMVD (take the negation of the CNF, translate the resulting DNF into an OMVD, and take the negation of the result). As $\texttt{OMVD}_<$ and OMVD satisfy **CO** (Prop. A.13), we would have a polynomial algorithm for deciding whether a CNF is consistent, which is impossible unless $\mathsf{P} = \mathsf{NP}$. □

Proposition A.23. MVD *and* MDD *satisfy* $\wedge\mathbf{C}$ *and* $\wedge\mathbf{BC}$.

Proof. To make the conjunction of k MVDs $\phi_1, \ldots, \phi_k$, replace the sink of ϕ_i by the root of ϕ_{i+1}, for all i in $\{1, \ldots, k-1\}$. Then the obtained MVD, rooted at the root of ϕ_1 and using the sink of ϕ_k, represents $\bigwedge_{i=1}^{k} \phi_i$. This procedure is linear in k, so MVD satisfies $\wedge\mathbf{C}$, and thus $\wedge\mathbf{BC}$.

Applying the previous procedure on MDDs, the result also is an MDD, since no edge is modified or added: it maintains the decision property. Hence MDD also satisfies $\wedge\mathbf{C}$ and $\wedge\mathbf{BC}$. □

Proposition A.24. OMVD *and* OMDD *do not satisfy* $\wedge\mathbf{BC}$ *or* $\wedge\mathbf{C}$ *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Thanks to Proposition 3.3, any OBDD can be turned into an equivalent OMVD in linear time. If OMVD (resp. OMDD) satisfied $\wedge\mathbf{BC}$, we would have a polynomial algorithm deciding whether the conjunction of two OBDDs (the variable order being possibly different in each OBDD) is consistent, since OMVD (resp. OMDD) supports **CO** (Prop. A.13); yet, this problem is NP-complete, as shown by Meinel and Theobald in Lemma 8.14 of their book about OBDDs.[22] Therefore OMVD (resp. OMDD) does not support $\wedge\mathbf{BC}$, and thus $\wedge\mathbf{C}$, unless $\mathsf{P} = \mathsf{NP}$. □

Proposition A.25. OMDD *and* $\texttt{OMDD}_<$ *do not satisfy* $\wedge\mathbf{C}$.

Proof. Lemma A.1 states that any propositional clause can be represented in the form of an OMDD on any order $<$ in linear time; if OMDD (resp. $\texttt{OMDD}_<$) satisfied $\wedge\mathbf{C}$, we could thus translate in polynomial time any propositional CNF into OMDD (resp. $\texttt{OMDD}_<$). Since the edges in OMDDs representing propositional formulæ can only be labeled with 0 or 1, it would mean that $\texttt{OMDD}^{\mathcal{B}} \leq_{\mathcal{P}} \texttt{CNF}$ (resp. $\texttt{OMDD}^{\mathcal{B}}_< \leq_{\mathcal{P}} \texttt{CNF}$).

But thanks to Proposition 3.3, $\texttt{OMDD}^{\mathcal{B}} \equiv_{\mathcal{L}} \texttt{OBDD}$ (resp. $\texttt{OMDD}^{\mathcal{B}}_< \equiv_{\mathcal{L}} \texttt{OBDD}_<$); yet we know from the Boolean map[11] that $\texttt{OBDD} \not\leq_s \texttt{CNF}$ (resp. $\texttt{OBDD}_< \not\leq_s \texttt{CNF}$). Hence OMDD (resp. $\texttt{OMDD}_<$) cannot satisfy $\wedge\mathbf{C}$. □

Proposition A.26. $\texttt{OMVD}_<$ *does not satisfy* $\wedge\mathbf{C}$ *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Lemma A.1 states that any propositional clause can be represented in the form of an OMVD on any order $<$ in linear time; if $\texttt{OMVD}_<$ satisfied $\wedge\mathbf{C}$, we could thus translate any propositional CNF into $\texttt{OMVD}_<$, and it would be possible to check whether it is consistent, as $\texttt{OMVD}_<$ satisfies **CO** (Prop. A.13). Yet it is impossible, unless $\mathsf{P} = \mathsf{NP}$. □

Proposition A.27. MVD *satisfies* $\vee\mathbf{C}$ *and* $\vee\mathbf{BC}$.

Proof. To make the disjunction of k MVDs $\phi_1, \ldots, \phi_k$, 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. □

Proposition A.28. MDD *satisfies* $\vee\mathbf{C}$ *and* $\vee\mathbf{BC}$*;* OMDD *and* $\texttt{OMDD}_<$ *do not satisfy* $\vee\mathbf{C}$*;* OMDD *does not satisfy* $\vee\mathbf{BC}$ *unless* $\mathsf{P} = \mathsf{NP}$*; and* $\texttt{OMDD}_<$ *satisfies* $\vee\mathbf{BC}$.

Proof. Let L be a sublanguage of MVD satisfying $\neg\mathbf{C}$; L verifies $\vee\mathbf{C}$ (resp. $\vee\mathbf{BC}$) if and only if it satisfies $\wedge\mathbf{C}$ (resp. $\wedge\mathbf{BC}$). Indeed, to make the disjunction of $\phi_1, \ldots \phi_k$, it is enough to compute the negation of each disjunct, then build their conjunction, and again compute the negation of the result.

MDD, OMDD, and $\texttt{OMDD}_<$ satisfy $\neg\mathbf{C}$ (Prop. A.8) so this property applies to them. We get the result since MDD satisfies $\wedge\mathbf{C}$ and $\wedge\mathbf{BC}$ (Prop. A.23), OMDD and $\texttt{OMDD}_<$ do not satisfy $\wedge\mathbf{C}$ (Prop. A.25), OMDD does not satisfy $\wedge\mathbf{BC}$ unless $\mathsf{P} = \mathsf{NP}$ (Prop. A.24), and $\texttt{OMDD}_<$ satisfies $\wedge\mathbf{BC}$ (Prop. A.9). □

Proposition A.29. $\texttt{OMVD}_<$ *satisfies* $\vee\mathbf{C}$ *and* $\vee\mathbf{BC}$.

Proof. Implied by the fact that $\texttt{OMVD}_<$ satisfies **SFO** (Prop. A.7): indeed, given k OMVDs $\phi_1, \ldots, \phi_k$ on the same order $<$, by joining them thanks to a new root node labeled with a new variable and forgetting this variable, we get an OMVD on $<$ that represents $\bigvee_{i=1}^{k} \mathrm{I}(\phi_i)$. The joining algorithm is linear in k, forgetting is polynomial, therefore $\texttt{OMVD}_<$ satisfies $\vee\mathbf{C}$ and thus $\vee\mathbf{BC}$. □

Proposition A.30. MVD *and* MDD *do not satisfy* **FO** *or* **TR** *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Given any MVD ϕ, ϕ is consistent if and only if $\mathrm{Forget}(\mathrm{I}(\phi), \mathrm{Scope}(\phi)) \equiv \top$. The only MVDs that do not mention any variable are the empty and the sink-only graph, and testing whether an MVD is empty is done is constant time. If MVD (resp. MDD) satisfied **FO**, we would have a polynomial algorithm for deciding the consistency of any MVD (resp. MDD), yet MVD (resp. MDD) does not satisfy **CO** unless $\mathsf{P} = \mathsf{NP}$ (Prop. A.11).

Hence, neither MVD nor MDD satisfies **FO** unless $\mathsf{P} = \mathsf{NP}$, and since **TR** implies **FO** (forgetting a variable x amounts to restricting it the term $[x \in \mathrm{Dom}(x)]$) we get the result. □

Proposition A.31. OMDD *and* $\texttt{OMDD}_<$ *do not satisfy* **SFO**, **FO**, *or* **TR**.

Proof. Suppose OMDD satisfied **SFO**. Then we could obtain in polynomial time an OMDD representing the disjunction of an arbitrary number of OMDDs on the same

order $<$, 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 OMDD on $<$ (Lemma A.1), we could transform in polynomial time any DNF into an OMDD (possibly on a different order than $<$). It would hence imply that $\mathtt{OMDD}^{\mathcal{B}} \leq_{\mathcal{P}} \mathtt{DNF}$, since edges in OMDDs representing propositional formulæ can only be labeled with 0 or 1. Since $\mathtt{OMDD}^{\mathcal{B}} \equiv_{\mathcal{L}} \mathtt{OBDD}$ (Prop. 3.3), we would have $\mathtt{OBDD} \leq_{\mathcal{P}} \mathtt{DNF}$, yet we know from the Boolean map[11] that $\mathtt{OBDD} \not\leq_s \mathtt{DNF}$. Hence, OMDD cannot satisfy **SFO**.

If $\mathtt{OMDD}_<$ satisfied **SFO**, it would also be the case for OMDD, so $\mathtt{OMDD}_<$ does not satisfy **SFO**. Finally, since **TR** implies **FO** (forgetting a variable x amounts to restricting it the term $[x \in \mathrm{Dom}(x)]$) and thus **SFO**, we get the result. □

Proposition A.32. MVD *and* MDD *satisfy* **SFO**, **SEN**.

Proof. First, we show that

$$\mathrm{Forget}(\mathrm{I}, \{x\}) = \bigvee_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}|_{\vec{x}} \qquad \text{and} \qquad \mathrm{Ensure}(\mathrm{I}, \{x\}) = \bigwedge_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}|_{\vec{x}}.$$

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

Now, let ϕ be an MVD, and $x \in \mathrm{Scope}(\phi)$. Let $d = |\mathrm{Dom}(x)|$. To obtain an MVD of interpretation $\mathrm{Forget}(\mathrm{I}(\phi), \{x\})$ (resp. $\mathrm{Ensure}(\mathrm{I}(\phi), \{x\})$), 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 $d - 1$ disjunctions (resp. conjunctions). Any sublanguage of MVD satisfying **CD** and $\vee$**C** (resp. **CD** and $\wedge$**C**) thus satisfies **SFO** (resp. **SEN**). Both MVD and MDD satisfy **CD** (Prop. A.6), $\vee$**C** (Prop. A.27, Prop. A.28), and $\wedge$**C** (Prop. A.23), so they satisfy **SFO** and **SEN**. □

Proposition A.33. MVD, MDD, OMVD, *and* $\mathtt{OMVD}_<$ *do not satisfy* **EN** *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Given any MVD ϕ, ϕ is valid if and only if $\mathrm{Ensure}(\mathrm{I}(\phi), \mathrm{Scope}(\phi)) \equiv \top$. The only MVDs that do not mention any variable are the empty and the sink-only graphs. If MVD (resp. MDD, OMVD, $\mathtt{OMVD}_<$) satisfed **EN**, we would have a polynomial algorithm for deciding the validity of any MVD (resp. MDD, OMVD, OMVD on $<$), yet MVD (resp. MDD, OMVD, $\mathtt{OMVD}_<$) does not support **VA** unless $\mathsf{P} = \mathsf{NP}$ (Prop. A.11 and A.12). □

Proposition A.34. OMDD *and* $\mathtt{OMDD}_<$ *do not satisfy* **EN** *or* **SEN**.

Proof. This is a consequence of the fact that $\mathtt{OMDD}_<$ (resp. OMDD) supports $\neg$**C** and does not support **SFO** (Prop. A.31). Indeed, for any interpretation function I and any set of variables $X \subseteq \mathrm{Scope}(\mathrm{I})$, $\mathrm{Forget}(\mathrm{I}, X) = \neg\,\mathrm{Ensure}(\neg\,\mathrm{I}, X)$. If $\mathtt{OMDD}_<$ (resp. OMDD) satisfied **SEN**, we would have a polynomial algorithm for performing

a forgetting of some variable x in ϕ: compute a negation of ϕ, ensure x in this negation, and return a negation of the result. Hence $\mathtt{OMDD}_<$ (resp. $\mathtt{OMDD}$) does not satisfy **SEN**, and thus **EN**. □

Proposition A.35. $\mathtt{OMVD}$ *and* $\mathtt{OMVD}_<$ *do not satisfy* **SEN** *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. We prove that if $\mathtt{OMVD}$ or $\mathtt{OMVD}_<$ satisfied **SEN**, we could translate any CNF into one of these languages in polynomial time, and thus check its consistency, which is impossible unless $\mathsf{P} = \mathsf{NP}$.

Let ϕ be a propositional CNF, and let $<$ be a total strict order on Scope(ϕ). All clauses in ϕ can be transformed in polynomial time into OMVDs ordered with respect to $<$ (Lemma A.1). If $\mathtt{OMVD}$ satisfied **SEN**, we could obtain in polynomial time an OMVD, possibly on a different order, representing the conjunction of all these OMVDs (that is, the original CNF), by joining them with a new root node labeled with a new variable, and ensuring this variable. Since $\mathtt{OMVD}$ satisfies **CO** (Prop. A.13), we would have a polynomial algorithm for checking the consistency of any CNF, yet this is impossible unless $\mathsf{P} = \mathsf{NP}$. We extend the result to $\mathtt{OMVD}_<$, since if $\mathtt{OMVD}_<$ satisfied **SEN**, then $\mathtt{OMVD}$ also would. □

A.6. *Final proof*

Proof of Theorem 4.5. All results from this theorem have been proved in previous propositions. The proposition corresponding to each result is indicated in Table 5. □

Table 5. Results about queries and transformations.

	MVD	MDD	OMVD	OMDD	OMVD$_<$	OMDD$_<$
CO	∘ A.11	∘ A.11	√ A.13	√ A.13	√ A.13	√ A.13
VA	∘ A.11	∘ A.11	∘ A.12	√ A.17	∘ A.12	√ A.17
MC	√ A.10	√ A.10	√ A.10	√ A.10	√ A.10	√ A.10
CE	∘ A.11	∘ A.11	√ A.14	√ A.14	√ A.14	√ A.14
IM	∘ A.11	∘ A.11	∘ A.12	√ A.18	∘ A.12	√ A.18
EQ	∘ A.11	∘ A.11	∘ A.12	√ A.21	∘ A.12	√ A.19
SE	∘ A.11	∘ A.11	∘ A.12	∘ A.20	∘ A.12	√ A.19
CX	∘ A.11	∘ A.11	√ A.15	√ A.15	√ A.15	√ A.15
MX	∘ A.11	∘ A.11	√ A.13	√ A.13	√ A.13	√ A.13
CT	∘ A.11	∘ A.11	∘ A.12	√ A.17	∘ A.12	√ A.17
ME	∘ A.11	∘ A.11	√ A.16	√ A.16	√ A.16	√ A.16
CD	√ A.6	√ A.6	√ A.6	√ A.6	√ A.6	√ A.6
TR	∘ A.30	∘ A.30	√ A.7	• A.31	√ A.7	• A.31
FO	∘ A.30	∘ A.30	√ A.7	• A.31	√ A.7	• A.31
SFO	√ A.32	√ A.32	√ A.7	• A.31	√ A.7	• A.31
EN	∘ A.33	∘ A.33	∘ A.33	• A.34	∘ A.33	• A.34
SEN	√ A.32	√ A.32	∘ A.35	• A.34	∘ A.35	• A.34
∨**C**	√ A.27	√ A.28	?	• A.28	√ A.29	• A.28
∨**BC**	√ A.27	√ A.28	?	∘ A.28	√ A.29	√ A.28
∧**C**	√ A.23	√ A.23	∘ A.24	• A.25	∘ A.26	• A.25
∧**BC**	√ A.23	√ A.23	∘ A.24	∘ A.24	√ A.9	√ A.9
¬**C**	?	√ A.8	∘ A.22	√ A.8	∘ A.22	√ A.8

Note: √ means "satisfies", • means "does not satisfy", and ∘ means "does not satisfy, unless $\mathsf{P} = \mathsf{NP}$".