Representing CSPs with Set-labeled Diagrams: A Compilation Map

Alexandre Niveau[1], Hélène Fargier[2], and Cédric Pralet[3]

[1] CRIL/Université d'Artois, Lens, France
niveau@cril.fr
[2] IRIT/CNRS, Toulouse, France
fargier@irit.fr
[3] Onera/DCSD, Toulouse, France
cedric.pralet@onera.fr

Abstract. Constraint Satisfaction Problems (CSPs) offer a powerful framework for representing a great variety of problems. Unfortunately, most of the operations associated with CSPs are NP-hard. As some of these operations must be addressed online, compilation structures for CSPs have been proposed, *e.g.* finite-state automata and Multivalued Decision Diagrams (MDDs).
The aim of this paper is to draw a compilation map of these structures. We cast all of them as fragments of a more general framework that we call Set-labeled Diagrams (SDs), as they are rooted, directed acyclic graphs with variable-labeled nodes and set-labeled edges; contrary to MDDs and Binary Decision Diagrams, SDs are not required to be deterministic (the sets labeling the edges going out of a node are not necessarily disjoint), ordered nor even read-once.
We study the relative succinctness of different subclasses of SDs, as well as the complexity of classically considered queries and transformations. We show that a particular subset of SDs, satisfying a focusing property, has theoretical capabilities very close to those of Decomposable Negation Normal Forms (DNNFs), although they do not satisfy the decomposability property *stricto sensu*.

1 Introduction

Constraint Satisfaction Problems (CSPs) [RBW06] offer a powerful framework for representing a great variety of problems, *e.g.* planning or configuration problems. Different kinds of operations can be posted on a CSP, such as extraction of a solution (the most classical query), strong consistency of the domains, addition or retraction of new constraints (dynamic CSP), counting of the number of solutions, and even combinations of these operations. For instance, the interactive solving of a configuration problem amounts to a series of (unary) constraints additions and retractions while maintaining the strong consistency of the domains, *i.e.* each value in a domain is involved in at least one solution.

Most of these operations are NP-hard, but must sometimes be addressed online. A possible way of solving this contradiction is to use knowledge compilation, which consists in transforming the problem offline in such a way that its online resolution becomes tractable. As a matter of fact, Multivalued Decision Diagrams (MDDs) [SKMB90, Vem92, KVBSV98, AHHT07] have been proposed as a way to "compile" CSPs, and successfully used in product configuration [AFM02]. Figure 1 shows how an MDD can represent the set of solutions of a CSP.

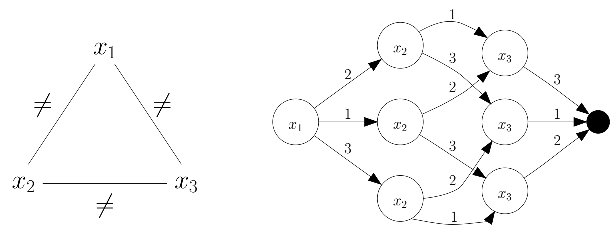


Fig. 1. This figure shows the constraint graph (on the left) of the 3-coloring problem (the domain of the variables is $\{1, 2, 3\}$), and an MDD (on the right) representing the set of solutions of this CSP.

In the present paper, we investigate this landscape by capturing these existing compilation structures as subsets of a more general framework called "set-labeled diagrams". The latter also covers new structures relaxing the requirements of determinism and ordering, which, as we show, can lead to exponentially more compact graphs without losing much in efficiency. In particular, we identify a subset of set-labeled diagrams that has theoretical capabilities very close to those of DNNFs (Decomposable Negation Normal Forms) [DM02], although it does not satisfy decomposability *stricto sensu*. Moreover, while most of the operations considered in classical knowledge compilation maps deal with reasoning problems, we introduce in the present map a few new operations, that are motivated by the use of the CSP framework for some more decision-oriented applications, such as planning and configuration. Proofs are gathered in Appendix A and B.

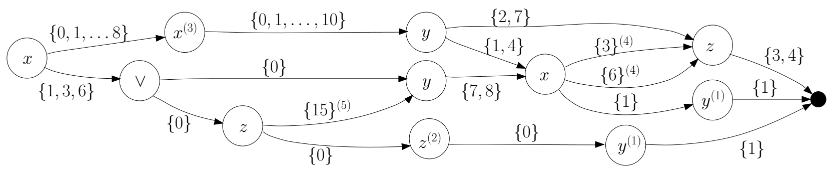


Fig. 2. An example of non-reduced SD. Variable domains are all $\{0, 1, \dots, 10\}$. The two nodes marked $^{(1)}$ are isomorphic; node $^{(2)}$ is stammering; node $^{(3)}$ is undecisive; the edges marked $^{(4)}$ are contiguous; edge $^{(5)}$ is dead.

2 Set-labeled Diagrams

Let us first formally define set-labeled diagrams, their interpretation, and their place among other knowledge compilation languages.

2.1 Structure and Semantics

The definition of set-labeled diagrams is similar to the one of classical decision diagram structures:

Definition 2.1 (Set-labeled diagram). *A* set-labeled diagram *(SD) is a couple $\phi = \langle X, \mathit{\Gamma} \rangle$, with*

- *X (also denoted $\mathrm{Var}(\phi)$) a finite and totally ordered set of variables whose domains are finite sets of integers;*
- *$\mathit{\Gamma}$ a directed acyclic graph[4] with at most one root and at most one leaf (the* sink*). Non-leaf nodes are labeled by a variable of X or by the disjunctive symbol $\veebar$. Each edge is labeled by a finite subset of $\mathbb{N}$.*

This definition allows the graph to be empty (no node at all, only case when there is not exactly one root and one leaf) or to contain only one node (together root and sink). It does not prevent edges' labels to be empty, and ensures that every edge belongs to at least one path from the root to the sink. Figure 2 gives an example of SD.

W will know introduce useful notations: For $x \in X$, $\mathrm{Dom}(x) \subseteq \mathbb{N}$ denotes the *domain* of x. By convention, $\mathrm{Dom}(\veebar) = \{0\}$. For $Y = \{y_1, \ldots, y_k\} \subseteq X$, such that the y_i are sorted in ascending order, $\mathrm{Dom}(Y)$ denotes $\mathrm{Dom}(y_1) \times \cdots \times \mathrm{Dom}(y_k)$, and $\vec{y}$ denotes a *Y-assignment* of variables from Y, *i.e.* $\vec{y} \in \mathrm{Dom}(Y)$. When $Y \cap Z = \emptyset$, $\vec{y}.\vec{z}$ is the *concatenation* of $\vec{y}$ and $\vec{z}$. Last, $\vec{y}(y_i)$ denotes the value assigned to y_i in $\vec{y}$ (by convention, $\vec{y}(\veebar) = 0$). Let $\phi = \langle X, \mathit{\Gamma} \rangle$ be a set-labeled diagram, N a node and E an edge in $\mathit{\Gamma}$; let us use the following notations:

- $\mathrm{Root}(\phi)$ the root of $\mathit{\Gamma}$ and $\mathrm{Sink}(\phi)$ its sink;
- $|\phi|$ the *size* of ϕ, *i.e.* the sum of the cardinalities of all labels in ϕ plus the cardinalities of the variables' domains;
- $\mathrm{Out}_\phi(N)$ (resp. $\mathrm{In}_\phi(N)$) the set of outgoing (resp. incoming) edges of N;
- $\mathrm{Var}_\phi(N)$ the variable labeling N (not defined for $\mathrm{Sink}(\phi)$);
- $\mathrm{Src}_\phi(E)$ the node from which E comes and $\mathrm{Dest}(E)$ the node to which E points;
- $\mathrm{Lbl}_\phi(E)$ the set labeling E;
- $\mathrm{Var}_\phi(E) = \mathrm{Var}_\phi(\mathrm{Src}(E))$ the variable associated to E.

We shall drop the ϕ subscript whenever there is no ambiguity.

An SD can be seen as a compact representation of a Boolean function over discrete variables. This function is the *interpretation function* of the set-labeled diagram:

[4] Actually, depending on the definition $\mathit{\Gamma}$ may not strictly be a graph, but rather a *multigraph*, since we allow two edges to go in parallel from one node to another (see *e.g.* Figure 2): the set of edges is a subset of $N \times N \times 2^{\mathbb{N}}$, N being the set of nodes.

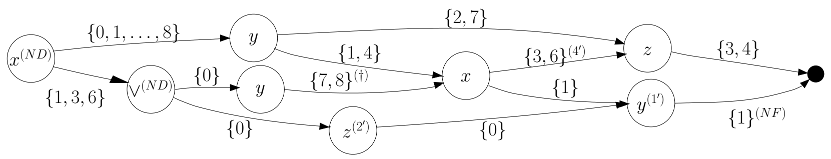


Fig. 3. In this SD, all edges are focusing but the one marked $^{(NF)}$ (it is not included in the one marked $^{(\dagger)}$), and all nodes are deterministic but the ones marked $^{(ND)}$. This SD is the reduced form of the SD presented in Figure 2: isomorphic nodes marked $^{(1)}$ have been merged into node $^{(1')}$, stammering node $^{(2)}$ has been collapsed into node $^{(2')}$, contiguous edges marked $^{(4)}$ have been merged into edge $^{(4')}$, and undecisive node $^{(3)}$ and dead edge $^{(5)}$ have been removed.

Definition 2.2 (Semantics of a set-labeled diagram). *A set-labeled diagram ϕ on $X = \mathrm{Var}(\phi)$ represents a function $\mathrm{I}(\phi)$ from $\mathrm{Dom}(X)$ to $\{\top, \bot\}$, called the* interpretation function *of ϕ, and defined as follows: for a given 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 along p, $\vec{x}(\mathrm{Var}(E)) \in \mathrm{Lbl}(E)$.*

We say that $\vec{x}$ is a model *of ϕ whenever $\mathrm{I}(\phi)(\vec{x}) = \top$. $\mathrm{Mod}(\phi)$ denotes the set of models of ϕ.*

ϕ is said to be equivalent *to another SD ψ (denoted $\phi \equiv \psi$) iff $\mathrm{Mod}(\phi) = \mathrm{Mod}(\psi)$.*

Note that the interpretation function of the empty SD always returns $\bot$, since it contains no path from the root to the sink. Conversely, the interpretation function of the one-node SD always returns $\top$, since in the one-node SD, the only path from the root to the sink contains no edge.

From these two definitions it follows that SDs are strongly related to ordered binary decision diagrams [Bry86] and multivalued decision diagrams [SKMB90, Vem92, APV99, AHHT07] as a way to represent a set of assignments of discrete variables (or typically, the set of solutions of a CSP). They actually generalize these data structures twofold. First, there is no restriction on the order in which the variables are encountered along a path, and variables can be repeated along a path. Second, SDs are not necessarily deterministic: the sets labeling edges going out of a node are not due to be pairwise disjoint, and thus a single model can be captured by several paths. SDs even support pure non-deterministic "OR" nodes (the $\veebar$-nodes) that allow the unrestricted union of several subgraphs. Putting away these two restrictions is valuable both theoretically, to generalize a large class of data structures, and practically, since SDs can be more compact than their ordered and deterministic variants (see Section 3.1). Let us define determinism formally and then introduce useful concepts.

Definition 2.3 (Deterministic set-labeled diagrams). *A node N in a set-labeled diagram ϕ is* deterministic *if the sets labeling its outgoing edges are pairwise disjoint.*

A deterministic *set-labeled diagram (dSD) is an SD containing only deterministic nodes.*

The notion of determinism is illustrated on Figure 3.

Definition 2.4 (Consistency, validity, context). *Let ϕ be a set-labeled diagram on X.*

ϕ is said to be consistent *(resp.* valid*) if and only if* $\mathrm{Mod}(\phi) \neq \emptyset$ *(resp.* $\mathrm{Mod}(\phi) = \mathrm{Dom}(X)$*).*

A value $v \in \mathbb{N}$ is said to be consistent *for a variable $y \in X$ in ϕ if and only if there exists an X-assignment $\vec{x}$ in $\mathrm{Mod}(\phi)$ such that $\vec{x}(y) = v$.*

The set of all consistent values for y in ϕ is called the context *of y in ϕ and denoted* $\mathrm{Ctxt}_\phi(y)$.

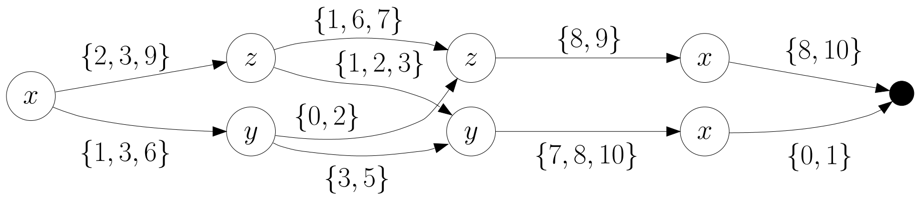


Fig. 4. Before an SD is proven inconsistent, every path must be checked. Here is an example of SD whose every path is inconsistent.

We will see in the following that deciding whether an SD is consistent is not tractable. One of the reasons is that the sets restricting a variable along a path can be conflicting, hence in the worst case all paths must be explored before a consistent one is found. Figure 4 shows an example of SD with no consistent path at all. To avoid this, we will consider SDs in which labeling sets can only shrink from the root to the sink, thus preventing conflicts:

Definition 2.5 (Focusing set-labeled diagrams). *A* focusing *edge in a set-labeled diagram ϕ is an edge E such that all edges E' on a path from the root of ϕ to* $\mathrm{Src}(E)$ *such that* $\mathrm{Var}(E) = \mathrm{Var}(E')$ *verify* $\mathrm{Lbl}(E) \subseteq \mathrm{Lbl}(E')$.

A focusing *set-labeled diagram (FSD) is an SD containing only focusing edges.*

The notion of focusing edge is illustrated in Figure 3. It is sufficient for the consistency query to be polynomial, but for some other operations (such as obtaining the conjunction of two SDs), a stricter restriction is necessary. An interesting one, very common in knowledge compilation, is to impose an order on the variables encountered along the paths; applying to SDs, we recover Multivalued Decision Diagrams (MDDs) in their practical acception[5] [SKMB90, Vem92, AHHT07].

Definition 2.6 (Ordered diagrams). *Let $<$ be a total order on the variables of X. A set-labeled diagram is said to be* ordered w.r.t. $<$ *iff, for any two nodes N and M, if N is an ancestor of M then* $\mathrm{Var}(N) < \mathrm{Var}(M)$.

A dSD ordered w.r.t. $<$ is called an MDD$_<$. The language `MDD` *is the union of all* `MDD`$_<$ *languages.*[6]

[5] The original definition of MDDs does not require determinism, nor introduces an order on the variables. Nevertheless, the papers resorting to these structures work only with ordered and deterministic MDDs; that is why we abusively designate ordered dSDs as MDDs.

[6] A *language* is a set of graph structures, fitted up with an interpretation function. We denote `SD` the language of SDs, `dSD` the language of dSDs, and so on.

Obviously, if there are not two occurrences of the same variable in a path, all edges are focusing. Hence: $\texttt{MDD}_< \subseteq \texttt{MDD} \subseteq \texttt{dFSD} \subseteq \texttt{FSD} \subseteq \texttt{SD}$ and $\texttt{dFSD} \subseteq \texttt{dSD} \subseteq \texttt{SD}$. We will study in the next sections the main properties of the SD family, and their relationships with classical Boolean decision diagrams. But before that, let us show how to reduce an SD in order to make it as compact as possible — and save space.

2.2 Reduction

Like a BDD, an SD can be reduced in size without changing its semantics. Reduction is based on several operations; some of them are straightforward generalizations of those introduced in the context of BDDs [Bry86], namely merging isomorphic nodes (that are equivalent) and collapsing undecisive edges (that are always crossed), while others are specific to set-labeled diagrams, namely suppressing dead edges (that are never crossed), merging contiguous edges (that have the same source and the same destination) and collapsing stammering nodes (successive decisions that pertain to the same variable). All these notions are illustrated in the SD of Figure 2, and the reduced form of this SD is shown on Figure 3. Formally:

Definition 2.7. – *Two edges* E_1, E_2 *are* contiguous *iff* $\mathrm{Src}(E_1) = \mathrm{Src}(E_2)$ *and* $\mathrm{Dest}(E_1) = \mathrm{Dest}(E_2)$.

- *Two nodes* N_1, N_2 *are* isomorphic *iff* $\mathrm{Var}(N_1) = \mathrm{Var}(N_2)$ *and there exists a bijection* σ *from* $\mathrm{Out}(N_1)$ *onto* $\mathrm{Out}(N_2)$, *such that* $\forall E \in \mathrm{Out}(N_1), \mathrm{Lbl}(E) = \mathrm{Lbl}(\sigma(E))$ *and* $\mathrm{Dest}(E) = \mathrm{Dest}(\sigma(E))$.
- *An edge* E *is* dead *iff* $\mathrm{Lbl}(E) \cap \mathrm{Dom}(\mathrm{Var}(E)) = \emptyset$.
- *A node* N *is* undecisive *iff* $|\mathrm{Out}(N)| = 1$ *and* $E \in \mathrm{Out}(N)$ *is such that* $\mathrm{Dom}(\mathrm{Var}(E)) \subseteq \mathrm{Lbl}(E)$.
- *A non-root node* N *is* stammering *iff all parent nodes of* N *are labeled by* $\mathrm{Var}(N)$, *and either* $\sum_{E\in\mathrm{Out}(N)} |E| = 1$ *or* $\sum_{E\in\mathrm{In}(N)} |E| = 1$.

Definition 2.8 (Reduced form). *A set-labeled diagram* ϕ *is said to be* reduced *iff no node of* ϕ *is isomorphic to another, stammering, or undecisive; and no edge of* ϕ *is dead or contiguous to another.*

In the following, we can consider only reduced SDs since reduction can be done in time polynomial in their size; indeed, each reduction step (removal of isomorphic nodes, contiguous edges, etc.) is polytime and removes more nodes and edges than it adds, hence even if we have to traverse the graph several times, the global complexity remains polynomial.

Proposition 2.9 (Reduction). *Let* L *be a sublanguage of* SD *among* $\{\texttt{SD}, \texttt{FSD}, \texttt{dSD}, \texttt{dFSD}, \texttt{MDD}, \texttt{MDD}_<\}$. *There exists a polytime algorithm that transforms any* ϕ *in* L *into an equivalent reduced* ϕ' *in* L *such that* $|\phi'| \leq |\phi|$.

We have seen that SDs are strongly related to BDDs and MDDs; we will now detail these relations.

2.3 SDs and the Decision Diagram Family

Binary Decision Diagrams (BDDs, [Bry86]) are rooted, directed acyclic graphs that represent Boolean functions of Boolean variables. They have two leaves, respectively labeled $\perp$ and $\top$; their non-leaf nodes are labeled by a Boolean variable and have two outgoing edges, respectively labeled $\perp$ and $\top$. A free BDD (FBDD) is a BDD that satisfies the read-once property (each path contains at most one occurrence of each variable). Whenever a same order is imposed on the variables along every path, we get an ordered BDD (OBDD). OBDDs have been extended to enumerated domains as MDDs by [SKMB90, Vem92, APV99] and later on, worked out by [AHHT07].

SDs are obviously not decision diagrams in the sense of Bryant since they do not have a $\perp$ sink, but classical MDDs do not either. Adding or not such a sink is actually harmless, and does not represent a real difference. The first main difference between decision diagrams and SDs is that SDs can be non-deterministic. Relationships between SDs and their Boolean counterparts are formally provided thereafter.

Definition 2.10 (Polynomial translatability). *A sublanguage* $\mathtt{L}_2$ *of* SD *is* polynomially translatable *into another sublanguage* $\mathtt{L}_1$ *of* SD, *which we denote* $\mathtt{L}_1 \leq_{\mathcal{P}} \mathtt{L}_2$, *if and only if there exists a polytime algorithm mapping any element from* $\mathtt{L}_2$ *to an equivalent element from* $\mathtt{L}_1$.

For any subclass L of SD, any $D \subseteq \mathbb{N}$, let $\mathtt{L}_D$ be the sublanguage of L for which all domains are included in D. We will consider in particular classes $\mathtt{dSD}_{\{0,1\}}$, $\mathtt{dFSD}_{\{0,1\}}$, and $\mathtt{FSD}_{\{0,1\}}$, that generalize BDD, FBDD, and DNF respectively.

When an order is imposed on the variables, $\mathtt{MDD}_{\{0,1\}}$ and OBDD are equivalent representation languages, up to the existence of a $\perp$ sink, which, once again, is harmless. More generally, [SKMB90] have shown that a log encoding of the domains allow to transform any MDD into an equivalent OBDD, providing by the way a convenient way to implement a MDD package on top of a BDD package.

Let us put the emphasis on the new languages, namely focusing SDs:

Proposition 2.11 **($\mathtt{FSD}_{\{0,1\}} \leq_{\mathcal{P}} \mathtt{DNF}$).** *Any formula in the* DNF *language can be expressed in the form of a* $FSD_{\{0,1\}}$ *in linear time.*

Proposition 2.12 **($\mathtt{dFSD}_{\{0,1\}} \leq_{\mathcal{P}} \mathtt{FBDD}$).** *Any FBDD (and thus any OBDD) can be expressed in the form of an equivalent* $dFSD_{\{0,1\}}$ *in time linear in the FBDD's size.*

dFSD actually generalizes FBDD, and we will see that it reaches the same performances as this fragment, except for the counting query. But it is worth noticing that, contrary to FBDD, dFSD allows a variable to be met twice on a path. $\mathtt{dFSD}_{\{0,1\}}$ is thus a proper superset of FBDD.

FSD are more general than usual MDD compilations of CSPs, since they do not require any order nor even determinism; we will see in the following that this can lead to exponential savings in space.

Last, but not least, it should be noticed that dFSDs and FSDs are not decomposable structures in the sense of Negation Normal Forms (NNFs). Indeed, the definition of decomposability [Dar01], when applied to a decision diagram, implies that variables cannot be repeated along a path. Since they are not decomposable, dFSDs do not define a subclass of AND/OR MDDs [MD06], of structured DNNFs [PD08], nor of tree automata [FV04], that are decomposable (and ordered) structures.

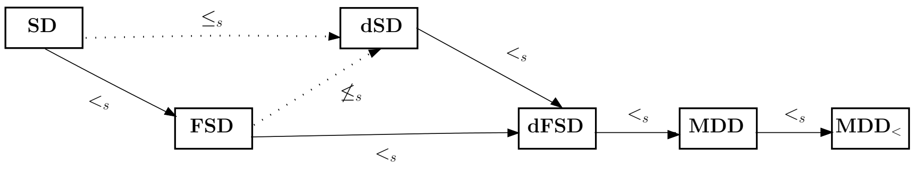
3 A Compilation Map of SDs

In the following, we put forward the properties of the SD language and its sublanguages, according to their spatial efficiency (succinctness), and to their capacity to make queries and transformations tractable.

3.1 Succinctness

Definition 3.1 (Succinctness). *A sublanguage* $\mathtt{L}_1$ *of* $\mathtt{SD}$ *is* at least as succinct *as another sublanguage* $\mathtt{L}_2$ *of* $\mathtt{SD}$*, which is denoted* $\mathtt{L}_1 \leq_s \mathtt{L}_2$*, if and only if there exists a polynomial* $P(\cdot)$ *s.t. for each element* ϕ *of* $\mathtt{L}_2$*, there exists an equivalent* ψ *of* $\mathtt{L}_1$ *verifying* $|\psi| \leq P(|\phi|)$.

Of course, $\leq_s$ is a preorder: let us denote $\sim_s$ its symmetric part and $<_s$ its asymmetric part.


$$\begin{tikzcd}
\boxed{\text{SD}} \arrow[rr, dotted, "\leq_s"] \arrow[dr, "<_s"'] & & \boxed{\text{dSD}} \arrow[dr, "<_s"] & & & \\
& \boxed{\text{FSD}} \arrow[ur, dotted, "\not\leq_s"'] \arrow[rr, "<_s"'] & & \boxed{\text{dFSD}} \arrow[r, "<_s"] & \boxed{\text{MDD}} \arrow[r, "<_s"] & \boxed{\text{MDD}_<}
\end{tikzcd}$$

L	SD	dSD	FSD	dFSD	MDD	MDD$_<$
SD	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$
dSD	?	$\leq_s$	?	$\leq_s$	$\leq_s$	$\leq_s$
FSD	$\not\leq_s^*$	$\not\leq_s^*$	$\leq_s$	$\leq_s$	$\leq_s$	$\leq_s$
dFSD	$\not\leq_s^*$	$\not\leq_s^*$	$\not\leq_s^*$	$\leq_s$	$\leq_s$	$\leq_s$
MDD	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\leq_s$	$\leq_s$
MDD$_<$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\not\leq_s$	$\leq_s$

Table 1. Results about succinctness. *means that the result is true unless the polynomial hierarchy PH collapses. The graph above illustrates those results, dotted lines meaning that we lack of information to prove both directions (*i.e.* it is proven that one of the languages is at least as succinct as the other, but it isn't known whether the converse is true or false).

Proposition 3.2 (Succinctness). *The results of Table 1 hold.*

This shows that SD $<_s^*$ FSD, FSD $<_s^*$ dFSD, and dFSD $<_s^*$ MDD, which means that *imposing focusingness, determinism, or ordering may lead to an exponential increase in space.*

3.2 Operations on Set-labeled Diagrams

We now need to extend to enumerated domains the queries and transformations considered in classical compilation maps. The importance of most of them is discussed in depth in [DM02], so we refrain from recalling it here. We add to these operations two new ones: The **CX**, "Context Extraction" query aims at providing the user with all possible values of some variable of interest. The $\wedge$**tC** "conjunction with a term" transformation restricts the possible values of some variables to a subset of their domains. These two operations are widely used in configuration, where the user iteratively looks for the possible values of the next configuration variables and restricts its values according to her preferences [AFM02]. For space reasons, we do not include here the most straightforward operation extensions; they can however be found in the long version of this paper.

L	CO	VA	MC	CE	IM	EQ	SE	MX	CX	CT	ME
SD	○	○	√	○	○	○	○	○	○	○	○
dSD	○	○	√	○	○	○	○	○	○	○	○
FSD	√	○	√	√	○	○	○	√	√	○	√
dFSD	√	√	√	√	√	?	○	√	√	?	√
MDD	√	√	√	√	√	√	○	√	√	√	√
MDD$_<$	√	√	√	√	√	√	√	√	√	√	√
DNNF	√	○	√	√	○	○	○	√	√	○	√
d-DNNF	√	√	√	√	√	?	○	√	√	√	√
OBDD	√	√	√	√	√	√	○	√	√	√	√
OBDD$_<$	√	√	√	√	√	√	√	√	√	√	√

L	CD	SCD	$\wedge$tC	FO	SFO	EN	SEN	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
SD	√	√	√	○	√	○	√	√	√	√	√	?
dSD	√	√	√	○	√	○	√	√	√	√	√	√
FSD	√	√	√	√	√	○	○	○	○	√	√	○
dFSD	√	√	√	○	○	○	○	○	○	○	○	?
MDD	√	√	√	●	●	●	●	●	○	●	○	√
MDD$_<$	√	√	√	●	●	●	●	●	√	●	√	√
DNNF	√	√	√	√	√	○	○	○	○	√	√	○
d-DNNF	√	√	√	○	○	○	○	○	○	○	○	?
OBDD	√	√	√	●	√	●	√	●	○	●	○	√
OBDD$_<$	√	√	√	●	√	●	√	●	√	●	√	√

Table 2. Results about queries and transformations; √ means "satisfies", ● means "does not satisfy", and ○ means "does not satisfy, unless P $=$ NP". Results for DNNF, d-DNNF, OBDD and OBDD$_<$ are from Darwiche and Marquis' map are given here as a baseline.

Definition 3.3 (Queries). *Let* L *denote a subset of the* SD *language.*

- L *satisfies consistency* (**CO**) *(resp. validity* (**VA**)*) iff there exists a polytime algorithm that maps every SD ϕ from* L *to* 1 *if ϕ is consistent (resp. valid), and to* 0 *otherwise.*
- L *satisfies equivalence* (**EQ**) *(resp. sentential entailment* (**SE**)*) iff there exists a polytime algorithm that maps every pair of SDs (ϕ, ϕ') from* L *to* 1 *if $\phi \equiv \phi'$ (resp. $\mathrm{I}(\phi) \models \mathrm{I}(\phi')$) and to* 0 *otherwise.*

- $\mathtt{L}$ *satisfies clausal entailment (*$\mathbf{CE}$*) (resp. implication (*$\mathbf{IM}$*)) iff there exists a polynomial* $P(;)$ *and an algorithm that maps every SD* ϕ *from* $\mathtt{L}$*, any set of variables* $\{y_1, \ldots, y_k\} \subseteq \mathrm{Var}(\phi)$*, and any sequence* $(A_1, \ldots, A_k)$ *of sets of integers, to* 1 *if* $\mathrm{I}(\phi) \models f_{y_1,A_1} \vee \cdots \vee f_{y_k,A_k}$ *(resp.* $f_{y_1,A_1} \wedge \cdots \wedge f_{y_k,A_k} \models \mathrm{I}(\phi)$*) and to* 0 *otherwise, in time* $P(|\phi|; n_A)$*, where* $n_A = \max_{1 \leq i \leq k} |A_i|$ *and* $f_{x,A}$ *is the function defined on* $Y = \{x\}$ *by* $f_{x,A}(\vec{y}) = \top \Leftrightarrow \vec{y}(x) \in A$*.*
- $\mathtt{L}$ *satisfies model checking (*$\mathbf{MC}$*) iff there exists a polytime algorithm that maps every SD* ϕ *from* $\mathtt{L}$ *and any* $\mathrm{Var}(\phi)$*-assignment* $\vec{x}$ *to* 1 *if* $\vec{x}$ *is a model of* ϕ *and to* 0 *otherwise.*
- $\mathtt{L}$ *satisfies model enumeration (*$\mathbf{ME}$*) iff there exists a polynomial* $P(;)$ *and an algorithm that outputs, for any SD* ϕ *from* $\mathtt{L}$*, all models of* ϕ *in time* $P(|\phi|; |\operatorname{Mod}(\phi)|)$*.*
- $\mathtt{L}$ *satisfies model extraction (*$\mathbf{MX}$*) iff there exists a polytime algorithm that maps every SD* ϕ *in* $\mathtt{L}$ *to one model of* ϕ *if there is one, and stops without returning anything otherwise.*
- $\mathtt{L}$ *satisfies context extraction (*$\mathbf{CX}$*) iff there exists a polytime algorithm that outputs, for any* ϕ *in* $\mathtt{L}$ *and any* $y \in \mathrm{Var}(\phi)$*,* $\mathrm{Ctxt}_\phi(y)$*.*

Definition 3.4. *Let* I, J *be the interpretation functions on* $\mathrm{Var}(I), \mathrm{Var}(J)$ *of some SDs.*

- *The conjunction (resp. disjunction) of* I *and* J *is the function* $\mathrm{I} \wedge \mathrm{J}$ *(resp.* $\mathrm{I} \vee \mathrm{J}$*) on the variables in* $X = \mathrm{Var}(\mathrm{I}) \cup \mathrm{Var}(\mathrm{J})$ *defined by* $(\mathrm{I} \wedge \mathrm{J})(\vec{x}) = \mathrm{I}(\vec{x}) \wedge \mathrm{J}(\vec{x})$ *(resp.* $(\mathrm{I} \vee \mathrm{J})(\vec{x}) = \mathrm{I}(\vec{x}) \vee \mathrm{J}(\vec{x})$*).*
- *The existential projection of* I *on* $Y \subseteq \mathrm{Var}(\mathrm{I})$ *is the function* $\mathrm{I}^{\downarrow Y}$ *on the variables of* Y *defined by:* $\mathrm{I}^{\downarrow Y}(\vec{y}) = \top$ *iff there exist a* Z*-assignment* $\vec{z}$ *(with* $Z = \mathrm{Var}(\mathrm{I}) \setminus Y$*) s.t.* $\mathrm{I}(\vec{z}.\vec{y}) = \top$*. The "forgetting" operation is the dual one:* $\operatorname{forget}(\mathrm{I}, Y) = \mathrm{I}^{\downarrow \mathrm{Var}(\mathrm{I}) \setminus Y}$*.*
- *The universal projection of* I *on* $Y \subseteq \mathrm{Var}(\mathrm{I})$ *is the function* $\mathrm{I}^{\Downarrow Y}$ *on the variables of* Y *defined by:* $\mathrm{I}^{\Downarrow Y}(\vec{y}) = \top$ *iff for any* Z*-assignment* $\vec{z}$ *(with* $Z = \mathrm{Var}(\mathrm{I}) \setminus Y$*),* $\mathrm{I}(\vec{z}.\vec{y}) = \top$*. The "ensuring" operation is the dual one:* $\operatorname{ensure}(\mathrm{I}, Y) = \mathrm{I}^{\Downarrow \mathrm{Var}(\mathrm{I}) \setminus Y}$*.*
- *Given an assignment* $\vec{y}$ *of some set of variables* $Y \subseteq \mathrm{Var}(\mathrm{I})$*, the conditioning of* I *by* $\vec{y}$ *is the function* $\mathrm{I}_{|\vec{y}}$ *on the variables in* $Z = \mathrm{Var}(\mathrm{I}) \setminus Y$ *defined by:* $\mathrm{I}_{|\vec{y}}(\vec{z}) = \mathrm{I}(\vec{y}.\vec{z})$*.*

Definition 3.5 (Transformations). *Let* $\mathtt{L}$ *denote a subset of the* $\mathtt{SD}$ *language.*

- $\mathtt{L}$ *satisfies conditioning (*$\mathbf{CD}$*) iff there exists a polytime algorithm that maps every SD* ϕ *in* $\mathtt{L}$ *and every assigment* $\vec{y}$ *of* $Y \subseteq \mathrm{Var}(\phi)$ *to an SD* ϕ' *in* $\mathtt{L}$ *such that* $\mathrm{I}(\phi') = \mathrm{I}(\phi)_{|\vec{y}}$*.*
- $\mathtt{L}$ *satisfies forgetting (*$\mathbf{FO}$*) (resp. ensuring (*$\mathbf{EN}$*)) iff there exists a polytime algorithm that maps every SD* ϕ *from* $\mathtt{L}$ *and every* $Y \subseteq \mathrm{Var}(\phi)$ *to an SD* ϕ' *in* $\mathtt{L}$ *such that* $\mathrm{I}(\phi') = \operatorname{forget}(\mathrm{I}(\phi), Y)$ *(resp.* $\mathrm{I}(\phi') = \operatorname{ensure}(\mathrm{I}(\phi), Y)$*).*
- $\mathtt{L}$ *satisfies* $\mathbf{SCD}$ *(resp.* $\mathbf{SFO}$*, resp.* $\mathbf{SEN}$*) iff it satisfies* $\mathbf{CD}$ *(resp.* $\mathbf{FO}$*, resp.* $\mathbf{EN}$*) when limited to a single variable (*i.e. $\mathrm{Card}(Y) = 1$*).*
- $\mathtt{L}$ *satisfies conjunction (*$\wedge\mathbf{C}$*) (resp. disjunction (*$\vee\mathbf{C}$*)) iff there exists a polytime algorithm that maps every finite set of SDs* $\Phi = \{\phi_1, \ldots, \phi_k\}$ *from* $\mathtt{L}$ *to an SD* ϕ *in* $\mathtt{L}$ *such that* $\mathrm{I}(\phi) = \mathrm{I}(\phi_1) \wedge \cdots \wedge \mathrm{I}(\phi_k)$ *(resp.* $\mathrm{I}(\phi) = \mathrm{I}(\phi_1) \vee \cdots \vee \mathrm{I}(\phi_k)$*).*
- $\mathtt{L}$ *satisfies bounded conjunction (*$\wedge\mathbf{BC}$*) (resp. bounded disjunction (*$\vee\mathbf{BC}$*)) iff it satisfies* $\wedge\mathbf{C}$ *(resp.* $\vee\mathbf{C}$*) when limited to a pair of SDs (*i.e. $\mathrm{Card}(\Phi) = 2$*)*

– `L` *satisfies term conjunction* ($\wedge$**tC**) *iff there exists a polynomial* $P(;)$ *and an algorithm that outputs, for every SD* ϕ *from* `L`*, any set of variables* $\{y_1, \dots, y_k\} \subseteq \mathrm{Var}(\phi)$ *and any sequence* $(A_1, \dots, A_k)$ *of sets of integers, an SD* ϕ' *in* `L` *such that* $\mathrm{I}(\phi') = \mathrm{I}(\phi) \wedge f_{y_1,A_1} \wedge \cdots \wedge f_{y_k,A_k}$ *in time* $P(|\phi|; n_A)$*, where* $n_A = \max_{1 \leq i \leq k} |A_i|$*.*
– `L` *satisfies negation* ($\neg$**C**) *iff there exists a polytime algorithm that maps every SD* ϕ *in* `L` *to an SD* ϕ' *in* `L` *such that* $\mathrm{I}(\phi') = \neg\, \mathrm{I}(\phi)$*.*

Proposition 3.6. *The results of Table 2 hold.*

The results pertaining to MDDs are generally known or follows directly from [SKMB90, KVBSV98]: in short, since MDDs (i) are a generalization of OBDDs to non-Boolean domains and (ii) can be encoded as OBDDs, they obviously have the same capabilities than OBDDs, except for **SEN** and **SFO**, for which the domain size has a more important role.

Table 2 puts forward the attractivity of the new `dFSD` and `FSD` classes: their performances are comparable to those of the `d-DNNF` (resp. `DNNF`) fragments, although, once again, `dFSD` and `FSD` *do not satisfy decomposability* but a distinct property also guaranteeing **CO**: focusingness.

From a more practical point of view, when one needs to use queries on compiled forms, but no transformation, like in configuration applications, dFSDs are much more interesting than MDDs — they can be much more compact, while satisfying almost the same queries. It is worthwhile imposing a variable order (*i.e.* going from dFSDs to MDDs) when one of the $\neg$**C**, **SEN**, **SFO** transformation is required.

Relaxing the requirement of determinism, we get the `FSD` fragment which moreover satisfies **FO** and $\vee$**C**; `FSD` fits particularly applications such as planning, where one needs to often check consistency, forget variables and extract models.

3.3 Capturing Continuous Variables

For the sake of simplicity, we chose to represent labeling sets as enumerations (the size of a set is defined as its cardinality). However, they could be represented in a more compact way, as a union of integer intervals. In this case, the effective size of a label would be the number of bounds necessary to represent this union, and this could lead to space savings linear in the size of the domains.

This idea of grouping elements raises another thought: there can be, in an SD, subsets whose elements are never separated, *i.e.* values that are interchangeable. One could then define meta-values, each one corresponding to one of these subsets. This would preserve the structural properties (determinism, focusingness…) of the considered SD.

It is also worth noticing that these principles provide us with a way to capture continuous variables, and to transform "continuous SDs" into discrete SDs. Classically, a continuous variable is "discretized" by partitioning its domain arbitrarily and replacing it by an integer variable, each value of which is associated with a unique element of the partition. Here, given a continuous SD, obtained for example by compiling the trace of a continuous CSP solver [NFPV10], it is easy to compute a discrete labeling that fits exactly the configuration of the graph. The idea is to recover, for each variable x, the set B of bounds of all intervals pertaining to x in the continuous SD. The partition chosen for $\mathrm{Dom}(x)$ is then the set containing all singleton intervals $[a, a]$ where $a \in B$, and all open intervals $]a, b[$ where a and b are successive bounds in B. Thus, all continuous intervals associated with x in the continuous SD and all intersections of such intervals can be represented as the exact union of elements in the partition.

Doing so, we can build an SD representing a CSP with real variables without resorting to an arbitrary (and spatially costly) discretization of the domains. The resulting SD can be embarked along with a translation table, allowing to make queries and transformations on the discrete structure and translating inputs (resp. results) from (resp. to) real numbers.

4 Conclusion

This paper draws the compilation map of set-labeled diagrams, a new knowledge compilation language able to represent solution sets of CSPs. We identified a structural property, focusingness, which is not equivalent to decomposability, yet when imposed on SDs allows the same queries and transformations. On the practical side, having considered CSPs rather than Boolean logic as a source language for compilation also raised the need for specific operations, that weren't covered by classical compilation maps (context extraction, conjunction with a term); it is noticeable that focusingness is sufficient to allow them in polytime. Finally, we sketched a way to use SDs to compile continuous CSPs.

This work introduces the first brick of a complexity map of CSP compilation; it is dedicated to “linear” decision diagrams. The next step is obviously to extend it to parallel focusing structures, the introduction of “AND” nodes. This will explicitly raise the question of a property capturing both the notions of focusingness and decomposability.

Future work also involves studying of how to build SDs, notably the experimentation of a compiler based on the trace of a search-like algorithm, and the comparison of various *dynamic* search heuristics w.r.t. SDs' size.

References

[AFM02] J. Amilhastre, H. Fargier, and P. Marquis. Consistency restoration and explanations in dynamic CSPs — application to configuration. *Artificial Intelligence*, 135(1-2):199–234, 2002.

[AHHT07] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In *CP*, pages 118–132, 2007.

[APV99] J. Amilhastre, P., and M.-C. Vilarem. FA Minimisation Heuristics for a Class of Finite Languages. In *WIA*, pages 1–12, 1999.

[Bry86] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[Dar01] Adnan Darwiche. Decomposable Negation Normal Form. *Journal of the ACM*, 48(4):608–647, 2001.

[DM02] A. Darwiche and P. Marquis. A Knowledge Compilation Map. *JAIR*, 17:229–264, 2002.

[FV04] H. Fargier and M.-C. Vilarem. Compiling CSPs into tree-driven automata for interactive solving. *Constraints*, 9:263–287, 2004.

[KS92] H.A. Kautz and B. Selman. Forming concepts for fast inference. In *Proc. of AAAI'92*, pages 786–793, San Jose (CA), 1992.

[KVBSV97] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. *Synthesis of Finite State Machines: Functional Optimization*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[KVBSV98] T. Kam, T. Villa, R.K. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued Decision Diagrams: Theory and Applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.

[MD06] R. Mateescu and R. Dechter. Compiling Constraint Networks into AND/OR Multi-valued Decision Diagrams. In *CP*, pages 329–343, 2006.

[MT98] C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer, 1998.

[NFPV10] A. Niveau, H. Fargier, C. Pralet, and G. Verfaillie. Knowledge compilation using interval automata and applications to planning. In *ECAI*, pages 459–464, 2010.

[PD08] K. Pipatsrisawat and A. Darwiche. New compilation languages based on structured decomposability. In *AAAI'08*, 2008. 517-522.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[SKMB90] A. Srinivasan, T. Kam, S. Malik, and R.K. Brayton. Algorithms for discrete function manipulation. In *ICCAD-90*, pages 92 –95, November 1990.

[Vem92] N. R. Vempaty. Solving Constraint Satisfaction Problems Using Finite State Automata. In *AAAI*, pages 453–458, 1992.

A Proofs of Reduction, Translatability, and Succinctness

In all of these proofs, we consider for short that for an interpretation function I with $Y = \mathrm{Var}(\mathrm{I})$, a set of variables Z s.t. $Z \cap Y = \emptyset$, $\vec{y}$ a Y-assignment and $\vec{z}$ a Z-assignment, $\mathrm{I}(\vec{y}.\vec{z})$ is simply defined as $\mathrm{I}(\vec{y})$.

Let ϕ be an SD on X and $\vec{x}$ an X-assignment. A path p from the root to the sink of ϕ is said to be *compatible with* $\vec{x}$ if and only if for each edge E along p, $\vec{x}(\mathrm{Var}(E)) \in \mathrm{Lbl}(E)$.

For any I, we denote $\vec{x} \models \mathrm{I}$ the fact that $\vec{x}$ is a model of I, and $\vec{x} \models \phi$ iff $\vec{x} \models \mathrm{I}(\phi)$. We also denote $\vec{x} \models p$ if p is a path compatible with $\vec{x}$.

A.1 Reduction

Lemma A.1. *Algorithm 1 reduces SDs in polytime.*

Proof. Let us apply Algorithm 1 on an SD ϕ.

For a given node N:

- the operation of l. 4 suppresses N if it is stammering
- the operation of l. 10 ensures that N has no more dead outgoing edges
- the operation of l. 15, ensures that N has no more contiguous outgoing edges
- the operation of l. 18 suppresses N if it is undecisive
- the operation of l. 24 suppresses all nodes that are isomorphic to N

Algorithm 1 Reduction algorithm. At any time during process, if an edge has no source or destination, it is suppressed; so are non-leaf nodes without outgoing edges and non-root nodes without incoming edges.

```
1: repeat
2:   number the nodes of $\phi$ in such a way that if $N_i \in \mathrm{Ch}(N_j)$ then $i < j$
3:   for $i$ from 1 to the number of nodes in $\Gamma(\phi)$ do
4:     if $N_i$ is stammering then
5:       for all $(E_{\mathrm{in}}, E_{\mathrm{out}}) \in \mathrm{In}(N_i) \times \mathrm{Out}(N_i)$ do
6:         add an edge from $\mathrm{Src}(E_{\mathrm{in}})$ to $\mathrm{Dest}(E_{\mathrm{out}})$ labeled by $\mathrm{Lbl}(E_{\mathrm{in}}) \cap \mathrm{Lbl}(E_{\mathrm{out}})$
7:       suppress $N_i$
8:     else
9:       for all $E \in \mathrm{Out}(N_i)$ do
10:        if $E$ is dead then
11:          suppress $E$
12:        else
13:          mark $E$
14:          for all $E' \in \mathrm{Out}(N)$ such that $E'$ is not marked do
15:            if $E$ and $E'$ are contiguous then
16:              label $E$ with $\mathrm{Lbl}(E) \cup \mathrm{Lbl}(E')$
17:              suppress $E'$
18:      if $N_i$ is undecisive then
19:        for all $E_{\mathrm{in}} \in \mathrm{In}(N_i)$ do
20:          redirect $E_{\mathrm{in}}$ to the child of $N_i$
21:        suppress $N_i$
22:      else
23:        for $j$ from 1 to the number of nodes in $\Gamma(\phi)$ do
24:          if $N_i$ and $N_j$ are isomorphic then
25:            for all $E_{\mathrm{in}} \in \mathrm{In}(N_i)$ do
26:              redirect $E_{\mathrm{in}}$ to $N_j$
27:            suppress $N_i$
28: until $\phi$ has not changed during process
```

and the algorithm stops when no operation has to be applied, so obviously the resulting SD is reduced. Moreover, it is easy to verify that each operation leaves the semantics of ϕ unchanged. Finally, every operation removes stricly more edges or nodes than it creates;[7]this proves that (i) the algorithm eventually stops (once the SD is empty, it does not change anymore), and (ii) the size of the resulting SD is lower than $|\phi|$ (the only case where the size does not change is when the input SD is already reduced).

The computation for each node is obviously polynomial and the traversal loop (l. 3) treats each node once; as a result, what is inside of the repeating loop (from l. 2 to l. 27) is processed in polytime.

As the reducibility properties are not mutually independant, the traversal must be repeated while it has modified something in ϕ (l. 28). This does not change polynomiality: since a traversal lowers the size of ϕ (except of course for the last one), the traversal loop will not be repeated more than $|\phi|$ times.

Note that there obviously exists more efficient methods to reduce an SD, but the only point here is to show that this operation is polytime. □

Definition A.2 (SD focusing w.r.t. a variable). *An SD ϕ is said to be* focusing w.r.t. y, *with $y \in \mathrm{Var}(\phi)$ iff every edge E in ϕ s.t. $\mathrm{Var}(E) = y$ is focusing.*

Lemma A.3. *Algorithm 1 maintains the property of focusing w.r.t. a given variable.*

Proof. Let ϕ be an SD that is focusing w.r.t. $y \in \mathrm{Var}(\phi)$. Let us suppose that we are at step i in the algorithm, with $\mathrm{Var}(N_i) = y$.

- "stammering" operation: let $E \in \mathrm{Out}(N_i)$ and E' be an edge on a path from $\mathrm{Dest}(E)$ to the sink such that $\mathrm{Var}(E') = y$. E' is focusing, so $\mathrm{Lbl}(E') \subseteq \mathrm{Lbl}(E)$. Thus $\mathrm{Lbl}(E') \subseteq \mathrm{Lbl}(E) \cap \mathrm{Lbl}(E_{\mathrm{in}})$ for any $E_{\mathrm{in}} \in \mathrm{In}(N_i)$. Hence E' is still focusing after the "stammering" operation.
- "dead" operation: suppressing edges does not have any influence on the focusingness of other edges in the graph.
- "contiguous" operation: the two contiguous edges points to the same node, so every descendant edge E associated with y is such that $\mathrm{Lbl}(E)$ is included in the labeling interval of either one of the contiguous edge; hence it is included in their union.
- "undecisive" operation: every descendant edge of the child of N_i is also a descendant edge of N_i, so this operation does not compromise their focusingness.
- "isomorphic" operation: since every outgoing edge of every node isomorphic to N_i is focusing, redirecting all the parent edges to N_i is harmless.

□

Lemma A.4 (SDs focusing w.r.t. all their variables). *A reduced SD ϕ that is focusing w.r.t. every variable in $\mathrm{Var}(\phi)$ is focusing.*

Proof. Let E be an edge in ϕ. Either $\mathrm{Var}(E) = y \neq \veebar$, in which case E is focusing, as ϕ is focusing w.r.t. y; or $\mathrm{Var}(E) = \veebar$, in which case E is also focusing, as every edge E' in ϕ such that $\mathrm{Var}(E') = \veebar$ is labeled by $\{0\}$ (since ϕ is reduced). Hence ϕ is focusing. □

[7] The only operation that creates anything is the stammering one, and recall that either $\mathrm{In}(N_i)$ or $\mathrm{Out}(N_i)$ contains only one element.

Lemma A.5. *Algorithm 1 maintains determism.*

Proof. Let ϕ be a dSD. Let us suppose that we are at step i in the algorithm, with $\mathrm{Var}(N_i) = y$.

- "stammering" operation: parent labels are modified only by value removal, so they are still deterministic after the operation.
- "dead" operation: suppressing edges does not have any influence on the determinism.
- "contiguous" operation: the two contiguous edges were disjoint with the other edges, so their union also is.
- "undecisive" operation: no edge is modified or added.
- "isomorphic" operation: no edge is modified or added.

□

Proof (Proof of Proposition 2.9 (reduction)).

`SD`*:* Straightforward from Lemma A.1

`FSD`*:* Let ϕ be an FSD. ϕ is *a fortiori* focusing w.r.t. all of its variables. Lemma A.3 states that the SD ϕ' obtained by Algorithm 1 is also focusing w.r.t. all of its variables. Since ϕ' is reduced, we use Lemma A.4 to infer that ϕ' is focusing. Thus our reduction algorithm maintains the focusing property, hence the result.

`dSD`*:* Thanks to Lemma A.5, we know that our reduction algorithm maintains determinism.

`dFSD`*:* Our reduction algorithm maintains both focusingness (as shown above in this proof) and determinism (Lemma A.5), hence the result.

□

A.2 Polynomial Translatability w.r.t. Decision Diagrams

In the following, we suppose that SDs (resp. dSDs, dFSDs, FSDs, MDDs) are in reduced form. This is harmless thanks to Proposition 2.9.

As usual with compilation maps, may proofs take advantage of the fact that all the languages presented allow to represent a term (resp. a clause) of Boolean logic in polytime and linear space.

Lemma A.6 **($\mathtt{MDD}_{\{0,1\},<} \leq_{\mathcal{P}} \mathtt{term}, \mathtt{MDD}_{\{0,1\},<} \leq_{\mathcal{P}} \mathtt{clause}$).**

- *Given an order $<$ between variables Any term or clause in propositional logic can be expressed in the form of a MDD$_{\{0,1\},<}$ in polytime.*
- *Any term or clause in propositional logic can be expressed in the form of a MDD (resp. FSD, dFSD) in polytime.*

Proof. To represent a term $t = l_1 \wedge \cdots \wedge l_k$ by a MDD$_{\{0,1\},<}$, first order its literal in such a way that $i < j$ iff $var(l_i) < var(l_j)$. Then build a chain of nodes $N_1, \ldots, N_k, N_{k+1}$ where $var(N_i) = var(l_i), i = 1, k$ and N_{k+1} is the sink of the SD. The edges are the following: each node N_i (but the sink) has a unique child N_{i+1}, and the edge's label is $\{1\}$ if l_i is a positive literal and $\{0\}$ if l_i is a negative literal.

To represent a clause $cl = l_1 \vee \cdots \vee l_k$ by a MDD$_{\{0,1\},<}$, first order its literals in such a way that $i < j$ iff $var(l_i) < var(l_j)$. Then build a chain of nodes $N_1, \ldots, N_k, N_{k+1}$ where $var(N_i) = var(l_i), i = 1, k$ and N_{k+1} is the sink of the SD. The edges are the following: each node $N_i, i < k$ has a two children, N_{i+1} and the sink, N_{k+1}. If l_i is a positive literal, the edge pointing to N_{i+1} is labeled by $\{0\}$ and the one pointing to N_{k+1}, by $\{1\}$; If l_i is a negative literal, this is the opposite. Finally, N_k has a a unique successor, the sink, and the edge is labeled by $\{1\}$ if l_k is a positive literal, and by $\{0\}$ otherwise.

The second item holds since `MDD`$_{\{0,1\},<}$ is respectively included in `MDD`, `FSD`, and `dFSD`. □

Proof (Proof of Proposition 2.11 (`FSD`$_{\{0,1\}} \leq_{\mathcal{P}}$ `DNF`*)).* Lemma A.6 states that each of the k terms of a DNF can be turned into an FSD$_{\{0,1\}}$ in linear time; Now, to make the disjunction of k FSDs$_{\{0,1\}}$ $\phi_1, \ldots, \phi_k$, simply define a new node, say N, labeled by $\veebar$. Then, for each ϕ_i, add an edge from N to $\mathrm{Root}(\phi_i)$ labeled by $\{0\}$. Fuse the sink nodes of the ϕ_i into a single one. This construction is linear in time and space w.r.t. the size of the DNF and preserves the property of focusing. □

Algorithm 2 Transformation of a BDD ϕ into a dSD$_{\{0,1\}}$

```
 1: let $\psi$ be the sink-only graph.
 2: for all node $N$ in $\phi$, ordered from the $\perp$-labeled leaf to the root do
 3:     add to $\psi$ a node $N'$ labeled by the same variable as $N$.
 4:     for all edge $E$ in $\phi$ coming out of $N$ do
 5:         let $D$ be the node to which $E$ points
 6:         if $D$ has a corresponding node $D'$ in $\psi$ then
 7:             add to $\psi$ an edge $E'$ coming from $N'$ and pointing to $D'$
 8:             if $E$ is labeled by $\perp$ then
 9:                 label $E'$ with $\{0\}$
10:             else
11:                 label $E'$ with $\{1\}$
12: if the root $R$ of $\phi$ has a corresponding node $R'$ in $\psi$ then
13:     let $\mathrm{Root}(\psi) = R'$
14: else
15:     let $\psi$ be the empty graph
16: return $\psi$
```

Lemma A.7.

– *Algorithm 2 transforms a BDD into an equivalent SD in time linear in the BDD's size.*

- *When applied on an OBDD$_<$, Algorithm 2 outputs an MDD$_<$ (respecting the same variable order).*

Proof.

- Algorithm 2 transforms a BDD into an equivalent SD by removing its $\bot$-labeled node, and by recursively removing all edges pointing to no node, and all non-leaf nodes without outgoing edges. The graph obtained then becomes an SD because each $\top$-labeled edge is replaced by a $\{1\}$-labeled edge, and each $\bot$-labeled edge is replaced by a $\{0\}$-labeled edge.
  It is obvious that the obtained SD is deterministic, due to the BDD structure. Moreover, this construction is linear in time and space w.r.t. the size of the BDD (each edge is computed once).
- The procedure does not modify the structure of the graph; thus, the read-once property and the variable order are preserved.

□

Algorithm 3 Transformation of a dSD$_{\{0,1\}}$ ϕ into a BDD

```
1: **if** $\phi$ is empty **then**
2:     **return** the $\bot$ BDD
3: let $\psi$ be a BDD with its two leaves
4: **for all** node $N$ in $\phi$, ordered from the sink to the root, excluding the sink **do**
5:     add to $\psi$ a node $N'$ labeled by the same variable as $N$.
6:     let $U := \emptyset$
7:     **for all** edge $E \in \mathrm{Out}(N)$ **do**
8:         let $D := \mathrm{Dest}(E)$
9:         let $D'$ be the node in $\psi$ corresponding to $D$ (the node corresponding to the sink being
           the $\top$-leaf)
10:        **if** $0 \in \mathrm{Lbl}(E)$ **then**
11:            add to $\psi$ a $\bot$-edge coming from $N'$ and pointing to $D'$
12:        **if** $1 \in \mathrm{Lbl}(E)$ **then**
13:            add to $\psi$ a $\top$-edge coming from $N'$ and pointing to $D'$
14:        $U := U \cup \mathrm{Lbl}(E)$
15:    **if** $0 \notin U$ **then**
16:        add to $\psi$ a $\bot$-edge coming from $N'$ and pointing to the $\bot$-leaf
17:    **if** $1 \notin U$ **then**
18:        add to $\psi$ a $\top$-edge coming from $N'$ and pointing to the $\bot$-leaf
19: set the root of $\psi$ to be the corresponding node of $\mathrm{Root}(\phi)$
20: **return** $\psi$
```

Lemma A.8.

- *Algorithm 3 transforms a dSD into an equivalent BDD in time linear in the dSD's size.*

– *When applied on an MDD$_{\{0,1\},<}$, Algorithm 3 outputs an OBDD$_<$ (respecting the same variable order).*

Proof.

– Algorithm 3 use the reverse procedure as of Algorithm 2: replace each $\{1\}$-labeled edge by a $\top$-labeled edge, each $\{0\}$-labeled edge by a $\bot$-labeled edge, each $\{0,1\}$-labeled edge by two edges (same source, same destination), one labeled by $\top$ and one labeled by $\bot$. Replace the sink by a $\top$-labeled leaf and add one $\bot$-labeled leaf. Finally, for each node that as only one outgoing edge, add to it another outgoing edge labeled by the other value and pointing to the $\bot$-leaf. The process is obviously linear (each edge is encountered once, and we add at most one edge for each node).
– Again, the procedure does not modify the structure of the graph; thus, the read-once property and the variable order are preserved.

□

Lemma A.9 **($\mathtt{dSD}_{\{0,1\}} \sim_{\mathcal{P}} \mathtt{BDD}$).** *Any BDD can be expressed in the form of an equivalent dSD$_{\{0,1\}}$ in time linear in the BDD's size.*

Any dSD$_{\{0,1\}}$ can be expressed in the form of an equivalent BDD, in time linear in the graph's size.

Proof (Proof of Lemma A.9).

$\mathtt{dSD}_{\{0,1\}} \leq_{\mathcal{P}} \mathtt{BDD}$ holds thanks to Lemma A.7, which also states that the procedure is linear.

$\mathtt{BDD} \leq_{\mathcal{P}} \mathtt{dSD}_{\{0,1\}}$ holds thanks to Lemma A.8, which also states that the procedure is linear. □

Proof (Proof of Proposition 2.12($\mathtt{dFSD}_{\{0,1\}} \leq_{\mathcal{P}} \mathtt{FBDD}$)). If we use Algorithm 2 on an FBDD, the resulting dSD is focusing. Indeed, as each variable can only be encountered once on each path, there is no risk that an interval conflicts with another.

Lemma A.7 states that the complexity of the algorithm is linear w.r.t. the size of the input.

□

Lemma A.10. $\mathtt{MDD}_{\{0,1\},<} \sim_{\mathcal{P}} \mathtt{OBDD}_<$ *holds.*

Proof. Straightforward from Lemmas A.7 and A.8.

□

A.3 Succinctness

Lemma A.11. *Let γ be a clause; a dFSD$_{\{0,1\}}$ equivalent to $\Sigma \vee l$ can be constructed in polytime in the size of Σ.*

Proof. Let Σ be a dFSD$_{\{0,1\}}$; let $l_1, \dots, l_n$ the literals that appear in γ.

$\Sigma \vee l_1 \equiv (\Sigma_{|\neg l_1} \wedge \neg l_1) \vee l_1$. Since $\mathtt{dFSD}$ satisfies **CD**, a dFSD$_{\{0,1\}}$ β representing $\Sigma_{|\neg l_1}$ can be built in polytime. A dFSD$_{\{0,1\}}$ equivalent to $(\beta \wedge \neg l_1) \vee l_1$ can be built, whose root is labeled by $var(l1)$ and has two outgoing edges: the first one is labeled by $\{1\}$ if l_1 is a positive literal, by $\{0\}$ otherwise; the second one one is labeled by $\{0\}$ if l_1 is a positive literal, by $\{1\}$ otherwise, and links the root to β. The procedure is polynomial and the result is linear in the size of Σ (the size $\Sigma_{|\neg l_1}$ is lower than the one of Σ; then one node and two edges are added).

Iterating the operation for $i = \{1, \dots, n\}$, we get a dFSD$_{\{0,1\}}$ equivalent to $\Sigma \vee \gamma$; its number of nodes (and thus, of edges) in bounded by $n + |nodes(\Sigma)|$ $\square$

Many proofs of succinctness rely on the following lemma, due to [KS92]:

Lemma A.12. *It is impossible to find a polysize compilation function* comp *such that for any CNF Σ and any clause γ, checking whether $\Sigma \models \gamma$ using* $\texttt{comp}(\Sigma)$ *can be done in polytime, unless the polynomial hierarchy* PH *collapses at the second level.*

In particular, it is impossible, unless the polynomial hierarchy PH collapses at the second level, to find a target language at least as succinct as CNF that supports **CE**.

Proof (Proof of Proposition 3.2 (succinctness table)).

$\texttt{SD} \leq_s \{\texttt{dSD}, \texttt{FSD}, \texttt{dFSD}, \texttt{MDD}, \texttt{MDD}_<\}$. Of course, $\texttt{MDD}_< \subseteq \texttt{MDD} \subseteq \texttt{dFSD} \subseteq \texttt{FSD} \subseteq \texttt{SD}$ and $\texttt{dFSD} \subseteq \texttt{dSD} \subseteq \texttt{SD}$. This proves the first line of the table.

$\{\texttt{dSD}, \texttt{FSD}, \texttt{dFSD}, \texttt{MDD}\} \leq_s \texttt{MDD}_<$. The same basic inclusions prove the last column of the table.

$\{\texttt{dSD}, \texttt{FSD}, \texttt{dFSD}\} \leq_s \texttt{MDD}$. Again from the same basic inclusions.

$\{\texttt{dSD}, \texttt{FSD}\} \leq_s \texttt{dFSD}$. Again from the same basic inclusions.

$\texttt{FSD} \not\leq_s^* \texttt{dSD}$. Any clause can be expressed as in polytime as a dSD$_{\{0,1\}}$. Making the conjunction of two dSDs is linear (just let the source of the first one be the sink of the second one, see also Prop. 3.6). Hence $\texttt{dSD}_{\{0,1\}} \leq_{\mathcal{P}} \texttt{CNF}$.

Suppose we had $\texttt{FSD}_{\{0,1\}} \leq_s \texttt{dSD}_{\{0,1\}}$: by transitivity it would be true that $\texttt{FSD}_{\{0,1\}} \leq_s \texttt{CNF}$.

Yet, FSD supports **CE** (Prop. 3.6), and it is impossible to find a target language at least as succinct as CNF and supporting **CE**, unless the polynomial hierarchy collapses (Lemma A.12).

As a result, $\texttt{FSD}_{\{0,1\}} \not\leq_s \texttt{dSD}_{\{0,1\}}$ and thus $\texttt{FSD} \not\leq_s \texttt{dSD}$, unless the polynomial hierarchy collapses.

$\{\texttt{FSD}, \texttt{dFSD}\} \not\leq_s^* \{\texttt{SD}, \texttt{dSD}\}$. This result comes from the fact that $\texttt{FSD} \not\leq_s^* \texttt{dSD}$, and that $\texttt{dFSD} \subseteq \texttt{FSD}$ and $\texttt{dSD} \subseteq \texttt{SD}$.

$\texttt{dFSD} \not\leq_s^* \texttt{FSD}$. The proof is close to the one of $\texttt{d-DNNF} \not\leq_s \texttt{DNF}$ [DM02].

Suppose we had $\texttt{dFSD}_{\{0,1\}} \leq_s \texttt{FSD}_{\{0,1\}}$: since $\texttt{FSD}_{\{0,1\}} \leq_{\mathcal{P}} \texttt{DNF}$ (Proposition 2.11), by transitivity it would be true that $\texttt{dFSD}_{\{0,1\}} \leq_s \texttt{DNF}$. Then any DNF Δ can be compiled into an equivalent polysize dFSD$_{\{0,1\}}$ $(\Delta)^*$.

Now, checking whether a clause γ in entailed by a CNF sentence Σ is equivalent to checking whether the sentence $\neg\Sigma \vee \gamma$ is valid, where $\neg\Sigma$ is a DNF. As a DNF,

$\neg\Sigma$ can be compiled into a dFSD$_{\{0,1\}}$ $(\neg\Sigma)^*$ in time and size polynomial. Thanks to Lemma A.11, we can get in polytime a dFSD$_{\{0,1\}}$ equivalent to $(\neg\Sigma)^* \vee \gamma$, and, since dFSD satisfy **VA**, we can check in polytime whether $(\neg\Sigma)^* \vee \gamma$ is valid.

In summary $(\neg\Sigma)^*$ is a polysize compilation of Σ, allowing clausal entailment to be achieved in polytime. The existence of such a $(\neg\Sigma)^*$ for every CNF Σ implies the collapse of the polynomial hierarchy (Lemma A.12).

Hence, $\mathtt{dFSD}_{\{0,1\}} \leq_s \mathtt{FSD}_{\{0,1\}}$ and thus $\mathtt{dFSD} \not\leq_s \mathtt{FSD}$, unless the polynomial hierarchy collapses.

$\mathtt{MDD} \not\leq_s \mathtt{dFSD}$. Suppose that $\mathtt{MDD} \leq_s \mathtt{dFSD}$. Since $\mathtt{dFSD}_{\{0,1\}} \leq_{\mathcal{P}} \mathtt{FBDD}$ (Prop. 2.12), it holds that $\mathtt{MDD}_{\{0,1\}} \leq_s \mathtt{FBDD}$. Then from $\mathtt{MDD}_{\{0,1\}} \sim_{\mathcal{P}} \mathtt{OBDD}$ (consequence of Lemma A.10), we deduce $\mathtt{OBDD} \leq_s \mathtt{FBDD}$, which is false (see [DM02]).

$\mathtt{MDD} \not\leq_s \{\mathtt{dSD}, \mathtt{FSD}, \mathtt{SD}\}$. This holds since $\mathtt{dFSD} \leq_s \mathtt{MDD}$ and $\mathtt{dFSD} \not\leq_s \{\mathtt{dSD}, \mathtt{FSD}, \mathtt{SD}\}$.

$\mathtt{MDD}_< \not\leq_s \mathtt{MDD}$. It is easy to show that $\mathtt{OBDD}_< \not\leq_s \mathtt{OBDD}$ [DM02]. Since $\mathtt{MDD}_{<\{0,1\}} \sim_{\mathcal{P}} \mathtt{OBDD}_<$ (Lemma A.10) this is straightforward that $\mathtt{MDD}_{\{0,1\}} \sim_{\mathcal{P}} \mathtt{OBDD}$, and from those two equivalences, $\mathtt{MDD}_< \not\leq_s \mathtt{MDD}$.

$\mathtt{MDD}_< \not\leq_s \{\mathtt{SD}, \mathtt{dSD}, \mathtt{FSD}, \mathtt{dFSD}\}$. Immediate from the above proof and from the fact that MDD is included in all of these fragments.

□

B Queries and Tranformations

This section is entirely devoted to the proof of Proposition 3.6.

B.1 Model Checking

SD *supports* **MC**. First, condition the SD by the X-assignment that is to be checked, say $\vec{x}$. The resulting SD contains only $\veebar$-nodes, so reduction removes them all. We get either the empty SD (then the assigmement is not a model) or the sink-only SD (then $\vec{x}$ is a model). □

FSD, dSD, dFSD, MDD *and* $\mathtt{MDD}_<$ *support* **MC**. Immediate from the fact that all these languages are subsets of SD, and SD satisfies **MC**. □

B.2 Conjunction with a Term

SD, dSD, MDD *and* $\mathtt{MDD}_<$ *support* $\wedge$**tC**. Straightforward from the fact that these languages support $\wedge$**BC**. □

`FSD` *and* `dFSD` *support* $\wedge$**tC**. To conjunct an SD ψ with $x \in A$, compute the restriction $\psi_{|x\in A}$ by simply removing from the x-edges the values that do not belong to A. The procedure is linear in the size of ψ and A (bounded by $\mathcal{O}(|A| \cdot |\phi|)$). The graph of ψ and $\psi_{|x\in A}$ are the same — only the labels on the edges change. Then add before the root of $\psi_{|x\in A}$ a node labeled by x, with a single outgoing edge labeled by A and pointing to the former root (since there may be paths that do not mention x).

The operation is linear and does not increase the size of that graph. Moreover, it preserves the properties of focusingness ($E \subseteq F \implies E \cap A \subseteq F$) and determinism ($E \cap F = \emptyset \implies (E \cap A) \cap (F \cap A) = \emptyset$)

Moreover, $\mathrm{I}(\psi') = \mathrm{I}(\psi) \wedge f_{x,A}$: By construction, for any $\vec{z}$, if $\vec{z}$ is a counter model for ψ, it is also a counter model for ψ'. Moreover any $\vec{z}$, when $\vec{z}(x)$ belongs to A, is supported by the very same path in ψ and ψ: $\vec{z}$ is a model of $\mathrm{I}(\psi)$ and satisfies $x \in A$, it is a model of $\mathrm{I}(\psi_{|x\in A})$. Hence $\mathrm{I}(\psi') = \mathrm{I}(\psi) \wedge f_{x,A}$.

To conjunct ϕ by a term $x_1 \in A_1, \ldots, x_k \in A_k$, repeat the operation for each $x_i \in A_i$. The graph of ψ and $\psi_{|x\in A}$ are the same — only the labels on the edges change. Since each elementary restriction does not increase the size of that graph, the complexity of the full procedure is bounded by $\mathcal{O}(k \cdot |\phi| \cdot \max_{1\leq i\leq k} |A_i|)$.

As a consequence, `FSD` and `dFSD` satisfy $\wedge$**tC**. □

B.3 Queries on `dSD` and `SD`

`dSD` *does not support* **CO**, **VA**, **CE**, **IM**, **EQ**, **SE**, **CT**, **ME**. From $\mathtt{dSD}_{\{0,1\}} \leq_{\mathcal{P}}$ `BDD` (Lemma . A.9) and `BDD` does not satisfy **CO**, **VA**, **CE**, **IM**, **EQ**, **SE**, **CT**, **ME** unless $\mathsf{P} = \mathsf{NP}$. □

`dSD` *does not support* **MX**, **CX**. Since **MX** implies **CO** (resp. **CX** implies **CO**): if **MX** (resp. **CX**) were polynomial, we would have a polytime algorithm for deciding whether a dSD is consistent or not, whereas `dSD` does not support **CO** unless $\mathsf{P} = \mathsf{NP}$. □

`SD` *does not support* **CO**, **VA**, **CE**, **IM**, **EQ**, **SE**, **MX**, **CX**, **CT**, **ME**. `SD` does not satisfy **CO**, **VA**, **CE**, **IM**, **EQ**, **SE**, **MX**, **CX**, **CT**, **ME** since `dSD` $\subseteq$ `SD` and `dSD` no not satisfy any of these requests unless $\mathsf{P} = \mathsf{NP}$. □

B.4 Queries on `FSD`

`FSD` *does not support* **VA**, **IM**, **EQ**, **SE**, **CT**. `FSD` does not satisfy **VA**, **IM**, **EQ**, **SE**, **CT**, since $\mathtt{FSD}_{\{0,1\}} \leq_{\mathcal{P}}$ `DNF` (Proposition 2.11) and `DNF` does not satisfy **VA**, **IM**, **EQ**, **SE** nor **CT**, unless $\mathsf{P} = \mathsf{NP}$. □

`FSD` *supports* **CO**. **CO** holds since (i) any FSD can be reduced in polytime, and (ii) the only reduced SD that is not consistent is the empty graph. Indeed, suppose that a reduced FSD ϕ has at least one variable-edge E (a reduced EA cannot contain only $\veebar$-nodes); as $\mathrm{Lbl}(E) \cap \mathrm{Dom}(\mathrm{Var}(E)) \neq \emptyset$, there is at least a value v in $\mathrm{Lbl}(E)$ that is included in the domain of its variable. As E is focusing, v is also coherent with the preceding edges in ϕ. Since it is the case for all edges, ϕ cannot be inconsistent. □

`FSD` *supports* **ME**. Let ϕ be a FSD. We check (in polytime) whether ϕ is consistent ; if it is not the case, the empty set is returned. Otherwise, we build a decision tree representing a $\mathrm{Mod}(\phi)$. Let us denote $\mathrm{Var}(\phi) = \{x_1, \dots, x_n\}$; we create a tree T, initially containing only one node, labeled by the empty assignment. We complete the tree thanks to the following process:

```
1: for all $i$ from 1 to $|\mathrm{Var}(\phi)|$ do
2:   let $\mathcal{F}$ be the set of $T$'s leaves.
3:   for all $F$ in $\mathcal{F}$ do
4:     let $\vec{z}$ be the $\{x_1, \dots, x_{i-1}\}$-assignment labeling $F$.
5:     for $v \in \mathrm{Dom}(x_i)$ do
6:       let $\vec{v}$ be the $\{x_i\}$-assignment such that $\vec{v}(x_i) = v$
7:       if $\mathrm{I}(\phi)_{|\vec{z}.\vec{v}}$ is consistent then
8:         add a child to $F$, labeled by $\vec{z}.\vec{v}$.
9:     remove $F$ from $\mathcal{F}$.
```

The model set of ϕ is then the set of assignments labeling the leaves of T, as we tested each possible value for each variable. At the end of the algorithm, all the leaves of T are at the same level (*i.e.* the paths of T are of equal length); indeed, for each node, at least one of the tests of l. 7 must pass (as the current FSD is consistent). It implies that at each incrementation of i, $|\mathcal{F}| \leq |\mathrm{Mod}(\phi)|$. Moreover, for each variable x, $|\mathrm{Dom}(x) \leq |\phi|$ by definition of the size. Hence, the test of l. 7 is not done more than $|\mathrm{Mod}(\phi)| \cdot |\phi|$ in the whole algorithm.

Now, this test is made in time polynomial w.r.t. $|\phi|$, as `FSD` supports **CO** and **SCD**. Hence, the global algorithm is polytime w.r.t. $|\mathrm{Mod}(\phi)|$ and $|\phi|$. □

`FSD` *supports* **MX**. Let ϕ be an FSD. If it is consistent (which can be verified in polytime), let $\vec{x} \in \mathrm{Dom}(\mathrm{Var}(\phi)$. Starting from the sink, we chose a path to the root. For each edge E of this path, we chose a value $v \in \mathrm{Lbl}(E) \cap \mathrm{Dom}(\mathrm{Var}(E))$ (as ϕ is reduced, we know that this set is not empty), and assign it to $\vec{x}$: $\vec{x}(\mathrm{Var}(E)) := v$, except if we already encountered this variable (we know that the value we chosed before is also compatible with this edge, as ϕ is focusing) or if $\mathrm{Var}(E) = \veebar$. When the root is reached, $\vec{x}$ is a model of ϕ (for every unencountered variable, $\vec{x}$ was set to a domain-compatible value at the beginning). This procedure is done in time polynomial in $|\phi|$. □

`FSD` *supports* **CX**. The following polynomial algorithm (each edge is encountered once) computes the context of y in ϕ:

```
1: reduce $\phi$
2: let $\mathcal{C} := \emptyset$
3: mark the sink of $\phi$
4: for all node $N$ in $\phi$, ordered from the sink to the root do
5:   if $N$ is marked then
6:     for all $E \in \mathrm{In}(N)$ do
7:       if $\mathrm{Var}(E) = y$ then
8:         add $\mathrm{Lbl}(E)$ to $\mathcal{C}$
9:       else
10:        mark $\mathrm{Src}(E)$
```

```
11: if the root of $\phi$ is marked then
12:     $\mathcal{C} := \mathrm{Dom}(y)$
13: return $\mathcal{C} \cap \mathrm{Dom}(y)$
```

The idea is to find the y-frontier of the sink (*e.g.* the set of the y-labeled nodes N such that there exists a path from a child of N to the sink not mentioning y), by pulling up a mark meaning that no y-labeled node has been encountered. If a mark reaches the root, there is at least one path from the root to the sink on which there is no y-labeled node, so the context of y in ϕ is $\mathrm{Dom}(y)$ (because ϕ is reduced, so the path is trivially satisfiable, in the same way that a non-empty reduced FSD is satisfiable). If not, the context of y is the union $\mathcal{C}$ of the intervals labeling edges by which the y-frontier access the sink, intersected with the domain of y (this intersection being polytime w.r.t. the number of intervals in $\mathcal{C}$, and thus w.r.t. $|\phi|$). □

FSD *supports* **CE**. Checking whether ϕ entails $f_{x_1,A_1} \vee \cdots \vee f_{x_k,A_k}$ is equivalent to checking whether $\phi \wedge \neg f_{x_1,A_1} \wedge \cdots \wedge \neg f_{x_k,A_k}$ is inconsistent. As $\neg f_{x,A} \equiv f_{x,\mathrm{Dom}(x)\setminus A}$, and FSD supports $\wedge$**tC** and **CO**, it supports **CE**. □

B.5 Queries on dFSD

dFSD *supports* **CO***,* **CE***,* **MX***,* **CX***,* **ME**. All these operations are supported, since dFSDs are particular FSDs. □

dFSD *does not support* **SE**. dFSD does not satisfy **SE**, since $\text{dFSD}_{\{0,1\}} \leq_{\mathcal{P}}$ FBDD (Proposition 2.12) and FBDD does not satisfy **SE** unless $\mathsf{P} = \mathsf{NP}$. □

For to prove **IM**, we need the following lemma, that extends Lemma A.11:

Lemma B.1. *Let Y be a set of variables $Y = \{y_1, \dots, y_k\} \subseteq \mathrm{Var}(\phi)$, and let $(A_1, \dots, A_k)$ be a sequence of sets of integers. For any dFSD ϕ, we can compute in polytime a dFSD equivalent to $\phi \vee f_{y_1,A_1} \vee \cdots \vee f_{y_k,A_k}$.*

Proof. Let us consider one of the $f_{y,A}$. $\phi \vee f_{y,A} \equiv (\phi \wedge \neg f_{y,A}) \vee f_{y,A}$.

Obviously $\neg f_{y,A} \equiv f_{y,\mathrm{Dom}(y)\setminus A}$, and $|\mathrm{Dom}(y) \setminus A|$ can be obtained in time linear in the size of A and $\mathrm{Dom}(y)$, thus in time linear in $|A|$ and $|\phi|$ (the sizes of the domains are comprised in the size of ϕ).

The procedure described in the proof of $\wedge$**tC** on dFSD is also done in time linear, and the resulting dFSD contains only edges whose label is included in A.

Hence, we can obtain a linear dFSD β representing $\phi \wedge \neg f_{y,A}$, that contains only edges whose label is included in A.

Then we can build a dFSD equivalent to $\beta \vee f_{y,A}$ as follows: its root is labeled by y and has two outgoing edges, the first one labeled by A and pointing to the sink, the second one labeled by $\mathrm{Dom}(y) \setminus A$ and pointing to the root of β.

Obviously, the resulting dFSD is equivalent to $(\phi \wedge \neg f_{y,A}) \vee f_{y,A}$ and thus to $\phi \vee f_{y,A}$, and it is deterministic and focusing (the $\wedge$**tC** procedure preserves those two properties, and the root node we add is deterministic and contains all edges' labels of β). This procedure is linear in the size of ϕ and A.

Iterating the operation for $y \in \{y_1, \dots, y_k\}$, we get a dFSD equivalent to $\phi \vee f_{y_1,A_1} \vee \cdots \vee f_{y_k,A_k}$; its number of nodes (and thus, of edges) is polynomial in $|\phi|$, $\max_{1 \leq i \leq k} |A_i|$, and k which is bound by $|\phi|$. $\square$

`dFSD` *supports* **IM**. Checking whether $f_{x_1,A_1} \wedge \cdots \wedge f_{x_k,A_k}$ entails ϕ is equivalent to checking whether $\phi \vee \neg f_{x_1,A_1} \vee \cdots \vee \neg f_{x_k,A_k}$ is valid. Lemma B.1 states that we can obtain a dFSD equivalent to this formula in polytime. Since `dFSD` supports **VA**, it implies that it supports **IM**. $\square$

`dFSD` *supports* **VA**. The following algorithm, applied to a dFSD ϕ, returns true if and only if phi is valid. In this algorithm we consider all labels to be included in the variables' domains (this can be done in polytime).

```
1:  for all node $N$ in $\phi$ do
2:      for all $x \in \mathrm{Var}(\phi)$ do
3:          let $S_{x,N} := \emptyset$
4:  for all $x \in \mathrm{Var}(\phi)$ do
5:      let $S_{x,\mathrm{Root}(\phi)} := \mathrm{Dom}(x)$
6:  for all node $N$ in $\phi$, ordered from the root to the sink do
7:      let $x := \mathrm{Var}(N)$
8:      let $U := \emptyset$
9:      for all $E \in \mathrm{Out}(N)$ do
10:         $U := U \cup \mathrm{Lbl}(E)$
11:         let $S_{x,\mathrm{Dest}(E)} := S_{x,\mathrm{Dest}(E)} \cup \mathrm{Lbl}(E)$
12:         for all $y \in \mathrm{Var}(\phi)$ do
13:             let $S_{y,\mathrm{Dest}(E)} := S_{y,\mathrm{Dest}(E)} \cup S_{y,N}$
14:     if $U \not\supseteq S_{x,N}$ then
15:         return false
16: return true
```

The idea is to check whether what comes in each node is included into what comes out, *i.e.* no value is "lost" (what comes in the root being the whole $\mathrm{Dom}(\mathrm{Var}(\phi))$). Formally, let us first define the notion of "last ancestor edge": For a node N, labeled by x, an edge E is a last ancestor of N if and only if $\mathrm{Var}(E) = x$ and there exists a path p from $\mathrm{Dest}(E)$ to N that contains no other x-node than N.

Now, our algorithm checks whether the following property (denoted $\mathrm{InEqOut}(N)$) is true for all node N:

- if there exists a path from the root to N that contains no other x-node than N, then

$$\forall v \in \mathrm{Dom}(\mathrm{Var}(N)), \exists E_v \in \mathrm{Out}(N), v \in \mathrm{Lbl}(E_v)$$

- if not, then for every ancestor edge of N

$$\forall v \in \mathrm{Lbl}(E), \exists E_v \in \mathrm{Out}(N), v \in \mathrm{Lbl}(E_v)$$

Our algorithm does this by gathering the last label encountered for each variable (Line 11), and making at each node, for each variable, the union of gathered labels of all incoming paths (Lines 11 and 13). It checks on Line 14 whether the union of outgoing labels contains all the computed "label set" for the node's variable. The root is treated as a special case, allowing to check the first item of $\mathrm{InEqOut}(N)$'s definition.

Now, let us prove that

$$\forall N \in \mathrm{Nodes}(\phi), \mathrm{InEqOut}(N) \quad \Leftrightarrow \quad \phi \models \top.$$

- **($\Rightarrow$)** We show that for any assignment $\vec{y} \in \mathrm{Dom}(\mathrm{Var}(\phi))$, there exists a path in the graph that is compatible with $\vec{y}$. Let us consider $\vec{y} \in \mathrm{Dom}(\mathrm{Var}(\phi))$. At each node N labeled by any variable x, we are assured to have the possibility to choose an outgoing edge compatible with $\vec{y}(x)$. Indeed, either this is the first x-node we encounter, and thus there exists an outgoing edge compatible with any value in $\mathrm{Dom}(x)$, or we already encounter at least one x-node, and chosen a value v for x; then the second item of $\mathrm{InEqOut}(N)$'s definition ensures that there exists an outgoing edge compatible with v.
  As it is true for each node, there is always a compatible edge that can be taken, therefore we are assured to reach the sink; thus, there exists a path from the root to the sink of phi that is compatible with $\vec{y}$. This proves that $\vec{y}$ is a model.
  As it is true for any assignment $\vec{y} \in \mathrm{Dom}(\mathrm{Var}(\phi))$, ϕ is valid.
- **($\Leftarrow$)** Suppose there exists a node N that does not verify $\mathrm{InEqOut}(N)$. Denoting $\mathrm{Var}(N) = x$, this means that
  - either (a) there exists a path p_R from the root to N that contains no other x-node than N, and then there is a value $v \in \mathrm{Dom}(x)$ such that $\forall E \in \mathrm{Out}(N), v \notin \mathrm{Lbl}(E)$;
  - or (b) there is no such path, and then there exists an ancestor edge E_{ancestor} of N and a value $v \in \mathrm{Lbl}(E_{\mathrm{ancestor}})$ such that $\forall E\ \mathrm{Out}(N), v \notin \mathrm{Lbl}(E)$;

  Let us consider a path p, the one denoted p_R in the case (a), or any path containing E_{ancestor} in the case (b). Let $\vec{y} \in \mathrm{Dom}(\mathrm{Var}(\phi))$ be an assignment that is (i) compatible with p and such that (ii) $\vec{y}(x) = v$. Such an assignment exists, because (i) as ϕ is focusing, all paths from the root to N are compatible with at least one model (there is no contradiction between the edges, as long as ϕ is reduced of course), and (ii) p has been defined in both cases (a) and (b) to be compatible with v.
  Let us suppose $\vec{y} \in \mathrm{Mod}(\phi)$. There exists a path p', from the root to the sink of ϕ, that is compatible with $\vec{y}$. Determinism imposes that p and p' be equal from the root to N; indeed, only one acceptable choice is possible at each node. Since p' is compatible with $\vec{y}$, this means there is an edge E going out of N and such that $v \in \mathrm{Lbl}(E)$, which has been supposed to be false. Then $\vec{y} \notin \mathrm{Mod}(\phi)$, hence ϕ is not valid.

$\square$

B.6 Queries on `MDD`

`MDD` *supports* **CO**, **VA**, **CE**, **IM**, **MX**, **CX**, **ME**. All these operations are supported, since MDDs are particular dFSDs. $\square$

`MDD` *supports* **CT**. As MDDs are read-once and deterministic, given a MDD ϕ with variable order $<$, we simply have to associate to the sink the number $n_{\text{Sink}(\phi)} = 1$, then traverse the graph from the sink to the root, associating to each edge E the number

$$n_E = |\operatorname{Lbl}(E)| \cdot \prod_{\operatorname{Var}(\operatorname{Src}(E)) < x < \operatorname{Var}(\operatorname{Dest}(E))} |\operatorname{Dom}(x)|$$

and to each node N the number

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

The number of models is then $n_{\operatorname{Root}(\phi)}$. This process being polynomial in $|\phi|$, `MDD` satisfies **CT**. □

`MDD` *supports* **EQ**. Let ϕ be an MDD on $X = \{x_1, \dots, x_n\}$, with variable ordering $x_1 < \cdots < x_n$. For each variable x_i and each value $v_j \in \operatorname{Dom}(x) = \{v_1, \dots, v_{k_i}\}$, let us define a $\{0,1\}$-variable x_i^j, taking the value 1 when $x_i = v_j$ and 0 otherwise. Following [SKMB90, KVBSV97, KVBSV98] can build an MDD in which each x-node of ϕ is replaced by an ordered sequence of x_i^j nodes, using the following procedure:

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

The return diagram ϕ_{bin} is obviously an $\text{MDD}_{\{0,1\},<}$, the order being $x_1^1 < \cdots < x_1^{k_1} < \dots < x_n^1 < \cdots < x_n^{k_n}$. Its size is polynomial in the one of the original MDD, each x-node being replaced by at most $|\operatorname{Dom}(x)|$ nodes having at most two outgoing edges. The procedure runs in polytime (each domain value is explored once for each node).

Let us denote X_{bin} the set of binary variables we introduced; an *acceptable* assignment $\vec{x}_{\text{bin}}$ of the variables from X_{bin} is defined as follows: for each variable $x_i \in X$, there is exactly one of the x_i^j that takes the value 1 in $\vec{x}_{\text{bin}}$. Formally, it is an assignment $\vec{x}_{\text{bin}} \in \text{Dom}(\bigcup_{x_i \in X} \bigcup_{j=1}^{k_{x_i}} x_i^j)$ verifying:

$$\forall x_i \in X, \qquad |\{j / \vec{x}_{\text{bin}}(x_i^j) = 1\}| = 1.$$

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

Now, $\texttt{MDD}\{0,1\} \sim_{\mathcal{P}} \texttt{OBDD}$, and `OBDD` supports **EQ** (Lemma 8.14 of [MT98]). For testing the equivalence of two MDDs ϕ and ψ, we only have to build their corresponding ϕ_{bin} and ψ_{bin} (which is done in polytime), transform them into OBDDs, and check whether they are equivalent; it is the case if and only if $\phi \equiv \psi$. □

`MDD` *does not support* **SE**. `MDD` does not satisfy **SE**, since $\texttt{MDD}_{\{0,1\}} \leq_{\mathcal{P}} \texttt{OBDD}$ (Lemma A.10) and `OBDD` does not satisfy **SE** unless $\mathsf{P} = \mathsf{NP}$. □

B.7 Queries on $\texttt{MDD}_<$

$\texttt{MDD}_<$ *supports* **CO**, **VA**, **CE**, **IM**, **EQ**, **MX**, **CX**, **CT**, **ME**. All these operations are supported, since $\texttt{MDD}_< \subseteq \texttt{MDD}$. □

$\texttt{MDD}_<$ *supports* **SE**. Checking whether an MDD ϕ entails another MDD ψ is equivalent to checking whether $\neg\phi \vee \psi$ is valid. Since $\texttt{MDD}_<$ satisfy $\vee$**BC**, $\neg$**C** and **VA**, it also satisfy **SE**. □

B.8 Conditioning

The proof of **CD** is based on the equivalence between (semantic) conditioning as defined in Definition 3.4 and the following notion of syntactic conditioning:

Definition B.2 (Syntactic conditioning). *Let ϕ be an SD, $Y \subseteq \text{Var}(\phi)$ and $\vec{y}$ a Y-assignment. We denote $\phi_{|\vec{y}}$ the SD obtained by (i) replacing, for each node N such as $\text{Var}(N) \in Y$, its label by $\veebar$ and the label of each $E \in \text{Out}(N)$ by $\{0\}$ if $\vec{y}(\text{Var}(E)) \in \text{Lbl}(E)$ and by $\emptyset$ otherwise.*

Lemma B.3 (Syntactic conditioning is linear). *$\phi_{|\vec{y}}$ is obtained in linear time.*

Proof. It is trivial, since each node and each edge is scanned at most once. □

Lemma B.4 (Syntactical conditioning provides a semantical conditioning). *$\phi_{|\vec{y}}$ is a conditioning of ϕ by $\vec{y}$.*

Proof. By definition of the conditioning, $\vec{z} \in \mathrm{Dom}(\mathrm{Var}(\phi) \setminus \{Y\})$ is a model of $\mathrm{I}(\phi)_{|\vec{y}}$ iff $\vec{y}.\vec{z}$ is a model of ϕ.

Suppose that $\vec{y}.\vec{z}$ is a model of ϕ. Then there is in ϕ a path p such that $\vec{z}.\vec{y} \models p$. By construction, a copy $p_{|\vec{y}}$ of p exists in $\phi_{|\vec{y}}$ and $\vec{z} \models p_{|\vec{y}}$: $\vec{z}$ is a model of $\phi_{|\vec{y}}$.

Suppose that $\vec{y}.\vec{z}$ is a not model of ϕ: for any path p in ϕ, there is an edge E on this path such that $\vec{y}.\vec{z}(\mathrm{Var}(E)) \notin \mathrm{Lbl}(E)$. Recall that the paths in $\phi_{|\vec{y}}$ are the same than those in ϕ and let $p_{|\vec{y}}$ be the one corresponding to p. If $\mathrm{Var}(E) \in Y$, $\vec{y}.\vec{z}(\mathrm{Var}(E)) \notin \mathrm{Lbl}(E)$ has led to label the corresponding edge in $\phi_{|\vec{y}}$ by $\emptyset$: $\vec{z}$ cannot be compatible with this path. If $\mathrm{Var}(E) \in Y$, $\vec{y}.\vec{z}(\mathrm{Var}(E)) \notin \mathrm{Lbl}(E)$ means that $\vec{z}(\mathrm{Var}(E)) \notin \mathrm{Lbl}(E)$: because these edges remain unchanged, $\vec{z}$ cannot be compatible with this path. Therefore $\vec{z}$ is not compatible with any path in $\phi_{|\vec{y}}$: $\vec{z}$ is not a model of $\phi_{|\vec{y}}$.

Hence $\mathrm{I}(\phi_{|\vec{y}}) = \mathrm{I}(\phi)_{|\vec{y}}$. □

`SD` *supports* **CD**, **SCD**. This is a direct consequence of Lemmas B.4 and B.3. Note that these transformations are supported in linear time. □

`FSD` *supports* **CD**, **SCD**. Let ϕ be an FSD, $Y \subseteq \mathrm{Var}(\phi)$ and $\vec{y}$ a Y-assignment. Let $\phi^*_{|\vec{y}}$ be the FSD obtained by suppressing $\emptyset$-labeled edges and their subgraphs in $\phi_{|\vec{y}}$. We show that $\phi^*_{|\vec{y}}$ is focusing.

Indeed, in $\phi^*_{|\vec{y}}$ the only edges that have been modified are those whose corresponding edge in ϕ is associated with a variable in Y. In $\phi^*_{|\vec{y}}$, they are all associated with $\veebar$ and labeled $\{1\}$ (since we removed $\emptyset$-labeled edges).

As for the other edges in $\phi^*_{|\vec{y}}$, since they *all* remain unchanged, they are still focusing. It is thus obvious that $\phi^*_{|\vec{y}}$ is focusing w.r.t. all variables in $\mathrm{Var}(\phi) \setminus Y = \mathrm{Var}(\phi^*_{|\vec{y}})$. Then Lemma A.4 shows that $\phi_{|\vec{y}}$ is focusing.

As $\mathrm{I}(\phi_{|\vec{y}}) = \mathrm{I}(\phi^*_{|\vec{y}}) = \mathrm{I}(\phi)_{|\vec{y}}$ and since $\phi^*_{|\vec{y}}$ is obtained in linear time, we get that `FSD` supports **CD** (and hence **SCD**) in linear time. □

`dSD` *supports* **CD**, **SCD**. Let ϕ be a dSD, $Y \subseteq \mathrm{Var}(\phi)$ and $\vec{y}$ a Y-assignment. We show that $\phi_{|\vec{y}}$ is deterministic.

Let us consider a node $N_{|\vec{y}}$ in $\phi_{|\vec{y}}$, and denote N its corresponding node in ϕ. If the two nodes are the same, $N_{|\vec{y}}$ is obviously deterministic. Suppose it has been modified: it is now labeled by $\veebar$. Since the outgoing edges of N are labeled by disjoint sets, at most one of them can include the value $\vec{y}(\mathrm{Var}(N))$. Consequently, at most one of $N_{|\vec{y}}$'s outgoing edges can be labeled by a non-empty set. This proves that $\phi_{|\vec{y}}$ is deterministic.

As $\mathrm{I}(\phi_{|\vec{y}}) = \mathrm{I}(\phi)_{|\vec{y}}$ and since $\phi_{|\vec{y}}$ is obtained in linear time, it holds that `dSD` supports **CD** (and hence **SCD**) in linear time. □

`dFSD` *satisfies* **CD**, **SCD**. Let ϕ be a dFSD, $Y \subseteq \mathrm{Var}(\phi)$ and $\vec{y}$ a Y-assignment. The two previous proofs (kcfdSD satisfies **CD**, `FSD` satisfies **CD**) show that $\phi_{|\vec{y}}$ is deterministic when ϕ is and that $\phi_{|\vec{y}}$ is focusing when ϕ. Hence $\phi_{|\vec{y}}$ is a dFSD.

As $\mathrm{I}(\phi_{|\vec{y}}) = \mathrm{I}(\phi)_{|\vec{y}}$ and since $\phi_{|\vec{y}}$ is obtained in linear time, it holds that `dFSD` supports **CD** (and hence **SCD**) in linear time. □

`MDD` *and* `MDD`$_<$ *support* **CD**, **SCD**. The procedure described above still works here, as it preserves determinism and obviously variable ordering. □

B.9 Negation

Algorithm 4 Given a dSD ϕ, builds a dSD called $\mathrm{compl}(\phi)$

```
 1: let $\psi$ be the sink-only graph.
 2: for all node $N$ in $\phi$, ordered from the sink to the root, excluding the sink do
 3:     create a node $N'$ labeled by the same variable $x$ as $N$.
 4:     Let $U := \mathrm{Dom}(x) \setminus \bigcup_{E \in \mathrm{Out}(N)} \mathrm{Lbl}(E)$
 5:     if $U \neq \emptyset$ then
 6:         add to $N'$ an outgoing edge $E_{\text{compl}}$ labeled by $U$ and pointing to the sink of $\psi$
 7:     for all edge $E$ in $\phi$ coming out of $N$ do
 8:         let $D = \mathrm{Dest}(E)$
 9:         if $D$ has a corresponding node $D'$ in $\psi$ then
10:             add to $N'$ an outgoing edge $E'$ labeled by $\mathrm{Lbl}(E)$ and pointing to $D'$
11:     if $N'$ has at least one outgoing edge then
12:         add $N'$ to $\psi$
13: if $\mathrm{Root}(\phi)$ has no corresponding node in $\psi$ then
14:     return $\psi$
15: else
16:     let $\mathrm{Root}(\psi)$ be the corresponding node of $\mathrm{Root}(\phi)$
17: return $\psi$
```

Lemma B.5. *The following properties hold:*

- *Algorithm 4 preserves determinism;*
- *Algorithm 4 runs in time polynomial in $|\phi|$;*
- *for each dSD ϕ, $\mathrm{compl}(\phi) \equiv \neg\phi$.*

Proof.

- **Determinism:** This is obvious, as we do not add nodes, and only add edges that are disjoint with the other edges coming out of a given node.
- **Complexity:** Each node is encountered once. The size of union U is bounded by the size of the variable's domain. Hence the size of the E_{compl} edge is bounded by $|\phi|$. In the worst case, there is an E_{compl} edge for each node of $\mathrm{compl}(\phi)$; the total size of these added edges is bounded by $|\phi|^2$.
- **Equivalence:** We prove this by induction on the number of nodes of ϕ. For $n \in \mathbb{N}$, let $\mathcal{P}(n)$ be the following proposition: "For any dSD ϕ containing n nodes, $\mathrm{compl}(\phi) \equiv \neg\phi$". $\mathcal{P}(n)$ is obviously true for $n \leq 2$. Let $n \geq 3$, and suppose that $\mathcal{P}(k)$ is true for all $k < n$.
  Let us denote R the root of ϕ, R' its corresponding node in $\mathrm{compl}(\phi)$, and $\mathrm{Var}(R) = x$. We consider an assignment $\vec{y} \in \mathrm{Dom}(\mathrm{Var}(\phi))$. We can meet two cases:
  - if there exists an edge $E \in \mathrm{Out}(R)$ that is compatible with $\vec{y}$, as ϕ is deterministic, it means that $\vec{y} \in \mathrm{Mod}(\phi) \Leftrightarrow \vec{y} \in \mathrm{Mod}(\phi_D)$, with ϕ_D the subgraph rooted at $D = \mathrm{Dest}(E)$. By construction of $\mathrm{compl}(\phi)$, if there is no corresponding edge E' in $\mathrm{compl}(\phi)$, it means that $\mathrm{compl}(\phi_D) = \emptyset$, and if there

is one, it means that $\text{compl}(\phi_D) = \phi'_D$, with $D' = \text{Dest}(E')$. Using our induction hypothesis (ϕ_D has at most n nodes), $\text{compl}(\phi_D) \equiv \neg\phi_D$, and thus $\vec{y} \in \text{Mod}(\phi_D) \Leftrightarrow \vec{y} \notin \text{compl}(\phi_D)$. As $\text{compl}(\phi)$ is also deterministic, we have $\vec{y} \notin \text{compl}(\phi_D) \Leftrightarrow \vec{y} \notin \text{compl}(\phi)$, and by putting the pieces together we get $\vec{y} \in \text{Mod}(\phi) \Leftrightarrow \vec{y} \notin \text{compl}(\phi)$.

- if it is not the case, then obviously $\vec{y} \notin \text{Mod}(\phi)$. Now, by construction, we know that there exists an edge E_{compl} coming out of R' and compatible with assignments that are compatible with no edge in $\text{Out}(R)$, that points to the sink of $\text{compl}(\phi)$. Thus $\vec{y} \in \text{Mod}(\text{compl}(\phi))$, therefore $\vec{y} \in \text{Mod}(\phi) \Leftrightarrow \vec{y} \notin \text{compl}(\phi)$.

In all cases, $\vec{y} \in \text{Mod}(\phi) \Leftrightarrow \vec{y} \notin \text{compl}(\phi)$ holds, hence $\text{compl}(\phi) \equiv \neg\phi$.
Since both the basis and the inductive step have been proven, we showed by induction that $\mathcal{P}(n)$ holds for all $n \in \mathbb{N}$.

□

FSD *does not support* $\neg\mathbf{C}$. By negating a CNF one obtains a DNF, that is polynomially translatable into an FSD (Proposition 2.11). Thus, if FSD satisfied $\neg\mathbf{C}$, we could, for any CNF, build an equivalent FSD in polytime. As FSD supports **CO**, we would have a polytime algorithm for deciding whether a CNF is consistent, which is impossible unless P = NP. □

dSD *supports* $\neg\mathbf{C}$. Straightforward from Lemma B.5. □

MDD *and* MDD$_<$ *support* $\neg\mathbf{C}$. Algorithm 4 preserves the ordering of the variables. □

B.10 Conjunction

SD *supports* $\wedge\mathbf{C}$, $\wedge\mathbf{BC}$. To make the conjunction of k SDs $\phi_1, \dots, \phi_k$, replace the sink of ϕ_i by the root of ϕ_{i+1}, for all $1 \leq i \leq k-1$. The root of the new SD is the one of ϕ_1, its sink the one of ϕ_k. Note that this process is linear. □

dSD *supports* $\wedge\mathbf{C}$, $\wedge\mathbf{BC}$. To make the conjunction of k dSDs, use the previous polytime procedure: the result is obviously a dSD, since no edge is modified or added. The process is also linear. □

FSD *does not support* $\wedge\mathbf{C}$, $\wedge\mathbf{BC}$. Thanks to Prop. 2.12, any OBDD can be turned into an equivalent FSD in polytime. If FSD supported $\wedge\mathbf{BC}$, we would have a polytime algorithm to decide whether the conjunction of two OBDDs (the variable orderings being possibly different in each OBDD) is consistent, forasmuch as FSD supports **CO**; yet, this problem is NP-complete, as shown in Lemma 8.14 of [MT98]. Therefore FSD does not support $\wedge\mathbf{BC}$, and *a fortiori* does not support $\wedge\mathbf{C}$, unless P = NP. □

dFSD *does not support* $\wedge\mathbf{C}$, $\wedge\mathbf{BC}$. Same proof as for FSDs, since any OBDD can be turned into an equivalent dFSD in polytime (Prop. 2.12) and dFSD supports **CO**. □

`MDD` *does not support* $\wedge\mathbf{C}$, $\wedge\mathbf{BC}$. Since $\mathtt{MDD}_{\{0,1\}} \sim_{\mathcal{P}}$ `OBDD` (Lemma A.10), as `OBDD` does not support $\wedge\mathbf{C}$, and does not support $\wedge\mathbf{BC}$ unless $\mathsf{P} = \mathsf{NP}$, it is the same for `MDD`. □

$\mathtt{MDD}_<$ *does not support* $\wedge\mathbf{C}$. Since $\mathtt{MDD}_{\{0,1\},<} \sim_{\mathcal{P}} \mathtt{OBDD}_<$ (Lemma A.10), as $\mathtt{OBDD}_<$ does not support $\wedge\mathbf{C}$, $\mathtt{MDD}_<$ does not either. □

Algorithm 5 $\mathtt{conjunct\_step}(N_1, N_2)$: returns a MDD$_<$ that is the conjunction of the two MDDs$_<$ of which N_1 and N_2 are roots.

```
1: if the cache contains the key $(N_1, N_2)$ then
2:     return the MDD corresponding to this key in the cache
3: if $N_1$ is the sink then
4:     let $\phi$ be the MDD rooted at $N_2$
5: else if $N_2$ is the sink then
6:     let $\phi$ be the MDD rooted at $N_1$
7: else if $\mathrm{Var}(N_1) < \mathrm{Var}(N_2)$ then
8:     create a node $N_1'$ labeled by $\mathrm{Var}(N_1)$
9:     for all $E \in \mathrm{Out}(N_1)$ do
10:        let $\phi_E := \mathtt{conjunct\_step}(\mathrm{Dest}(E), N_2)$
11:        add an edge coming out of $N_1'$, labeled by $\mathrm{Lbl}(E)$ and pointing to the root of $\phi_E$
12:    let $\phi$ be the MDD rooted at $N_1'$
13: else if $\mathrm{Var}(N_1) > \mathrm{Var}(N_2)$ then
14:    create a node $N_2'$ labeled by $\mathrm{Var}(N_2)$
15:    for all $E \in \mathrm{Out}(N_2)$ do
16:        let $\phi_E := \mathtt{conjunct\_step}(N_1, \mathrm{Dest}(E))$
17:        add an edge coming out of $N_2'$, labeled by $\mathrm{Lbl}(E)$ and pointing to the root of $\phi_E$
18:    let $\phi$ be the MDD rooted at $N_2'$
19: else
20:    create a node $N'$ labeled by $x = \mathrm{Var}(N_1) = \mathrm{Var}(N_2)$
21:    for all $v \in \mathrm{Dom}(x)$ do
22:        if there are edges $E_1 \in \mathrm{Out}(N_1)$ and $E_2 \in \mathrm{Out}(N_2)$ such that $v \in \mathrm{Lbl}(E_1)$ and
           $v \in \mathrm{Lbl}(E_2)$ then
23:            let $\phi_v := \mathtt{conjunct\_step}(\mathrm{Dest}(E_1), \mathrm{Dest}(E_2))$
24:            add an edge coming out of $N'$, labeled by $\{v\}$ and pointing to the root of $\phi_v$
25:    let $\phi$ be the MDD rooted at $N'$
26: add $\phi$ to the cache, at the key $(N_1, N_2)$
27: return $\phi$
```

$\mathtt{MDD}_<$ *supports* $\wedge\mathbf{BC}$. We can use Algorithm 5, adapted from the one on OBDDs [Bry86]. It applies on non-empty MDDs of a same variable order (if one MDD is empty, it is trivial to compute the conjunction). A cache is maintained to avoid computing twice the same couple of nodes, thus `conjunct_step` is not called more than $|\phi_1| \cdot |\phi_2|$ times. When the couple of nodes bear on the same variable, each value of its domain is explored once, and the size of the domain is lower than either $|\phi_1|$ or $|\phi_2|$ (by definition of the size function). The procedure is hence polytime. □

B.11 Disjunction

`SD` *supports* $\vee$**C**, $\vee$**BC**. To make the disjunction of k SDs $\phi_1, \dots, \phi_k$, simply define a new node, say N, labeled by $\veebar$. Then, for each ϕ_i, add an edge from N to $\mathrm{Root}(\phi_i)$ labeled by $\{0\}$. Fuse the sink nodes of the ϕ_i into a single one. This process is linear (this will be useful for the proof of **SFO**). □

`FSD` *supports* $\vee$**C**, $\vee$**BC**. Use the same procedure. If the ϕ_i are focusing, the resulting SD is also obviously focusing. As for SDs, the process is linear. □

`dSD` *supports* $\vee$**C**, $\vee$**BC**. This holds since `dSD` satisfies $\wedge$**C** and $\neg$**C**. Indeed, to make the disjunction of $\phi_1, \dots \phi_n$, compute the negation of each disjunct (`dSD` satisfies $\neg$**C**), then make their conjunction (`dSD` satisfies $\wedge$**C** in linear time), and again compute the negation of the result. □

`dFSD` *does not support* $\vee$**C**, $\vee$**BC**. If `dFSD` supported $\vee$**BC**, we would have a polytime algorithm to decide whether the conjunction of two OBDDs ϕ_1 and ϕ_2 (the variable orderings being possibly different in each OBDD) is consistent. Indeed, $\phi_1 \wedge \phi_2$ is consistent iff $\neg\phi_1 \vee \neg\phi_2$ is valid. We can obtain in polytime OBDDs representing $\neg\phi_1$ and $\neg\phi_2$ (switch the labels of the leaves). Now, thanks to Prop. 2.12, any OBDD can be turned into an equivalent dFSD in polytime, so we can get dFSDs representing $\neg\phi_1$ and $\neg\phi_2$; As `dFSD` supports **VA**, if it supported $\vee$**BC**, we could check whether $\neg\phi_1 \vee \neg\phi_2$ is valid, and thus whether $\phi_1 \wedge \phi_2$ is consistent. Yet, this problem is NP-complete, as shown in Lemma 8.14 of [MT98]. Therefore `dFSD` does not support $\vee$**BC**, and *a fortiori* does not support $\vee$**C**, unless $\mathsf{P} = \mathsf{NP}$. □

`MDD` *does not support* $\vee$**C**, $\vee$**BC**. Since $\texttt{MDD}_{\{0,1\}} \sim_{\mathcal{P}} \texttt{OBDD}$ (consequence of Lemma A.10), as `OBDD` does not support $\vee$**C**, and does not support $\vee$**BC** unless $\mathsf{P} = \mathsf{NP}$, it is the same for `MDD`. □

$\texttt{MDD}_<$ *does not support* $\vee$**C**. Since $\texttt{MDD}_{\{0,1\},<} \sim_{\mathcal{P}} \texttt{OBDD}_<$ (Lemma A.10), as $\texttt{OBDD}_<$ does not support $\vee$**C**, $\texttt{MDD}_<$ does not either. □

$\texttt{MDD}_<$ *supports* $\vee$**BC**. This comes from the fact that $\texttt{MDD}_<$ supports both $\wedge$**BC** and $\neg$**C**: we just have to compute $\neg(\neg\phi_1 \wedge \neg\phi_2)$. □

B.12 Forgetting

Lemma B.6. $\mathrm{forget}(\mathrm{I}, \{x\}) = \bigvee_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}$.

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

SD *does not support* **FO**. Given any SD ϕ, ϕ is consistent iff $\mathrm{forget}(\mathrm{I}(\phi), \mathrm{Var}(\phi)) \equiv \top$. The only reduced SDs without any variable are the empty and the sink-only graph, and testing the emptyness of an SD is done is constant time. If SD were satisfying **FO**, we would have a polytime algorithm for deciding the consistency of any SD, yet SD does not satisfy **CO** unless $\mathsf{P} = \mathsf{NP}$. □

dSD *does not support* **FO**. Same proof as the previous one (dSD does not satisfy **CO** unless $\mathsf{P} = \mathsf{NP}$). □

SD *supports* **SFO**. From Lemma B.6, it holds that $\mathrm{forget}(\mathrm{I}, \{x\}) = \bigvee_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}$. Let $n = |\,\mathrm{Card}(\mathrm{Dom}(x))|$; to obtain an SD whose interpretation function is equal to $\mathrm{forget}(\mathrm{I}, \{x\})$, it is sufficient to make n simple conditionings and $n - 1$ disjunctions. These two operations are linear (since SD satisfies **SCD** and $\vee$**BC** in linear time), and n is linear in $|\phi|$, thus **SFO** is feasible in time polynomial in $|\phi|$. □

dSD *supports* **SFO**. Same proof as the previous one (dSD satisfies **SCD** and $\vee$**BC** in linear time). □

dFSD *does not support* **SFO**, **FO**. Let ϕ_1 and ϕ_2 be two dFSDs. Let $Z = \mathrm{Var}(\phi_1) \cup \mathrm{Var}(\phi_2)$, and a variable $x \notin Z$ of domain $\{0, 1\}$. We build the SD ψ by merging the sinks of ϕ_1 and ϕ_2, and adding a node which will be the root of ψ, labeled by x, with one outgoing edge labeled by $\{0\}$ and pointing to the root of ϕ_1, and a second outgoing edge labeled by $\{1\}$ and pointing to the root of ϕ_2. ψ is obviously focusing, since ϕ_1 and ϕ_2 are, and x is mentioned in one node only, and deterministic, since ϕ_1 and ϕ_2 are, and our new root is.

We will prove that $\mathrm{forget}(\mathrm{I}(\psi), \{x\} = \mathrm{I}(\phi_1) \vee \mathrm{I}(\phi_2)$, *i.e.* for any Z-assignment $\vec{z}$, $(\mathrm{I}(\phi_1) \vee \mathrm{I}(\phi_2))(\vec{z}) = \top \Leftrightarrow \exists \vec{x} \in \mathrm{Dom}(\{x\}), \mathrm{I}(\psi)(\vec{x}.\vec{z}) = \top$.

($\Rightarrow$) Let $\vec{z}$ be a Z-assignment verifying $(\mathrm{I}(\phi_1) \vee \mathrm{I}(\phi_2))(\vec{z}) = \top$. Either $\mathrm{I}(\phi_1)(\vec{z}) = \top$, or $\mathrm{I}(\phi_2)(\vec{z}) = \top$. In the first case, let us take $\vec{x}$ such that $\vec{x}(x) = 0$: by construction there exists a path in ψ that is compatible with $\vec{x}.\vec{z}$. Symmetrically, in the second case, taking $\vec{x}$ such that $\vec{x}(x) = 1$, there exists a path in ψ that is compatible with $\vec{x}.\vec{z}$. Hence, there always exists a $\vec{x}$ such that $\mathrm{I}(\psi)(\vec{x}.\vec{z}) = \top$.

($\Leftarrow$) Let $\vec{z}$ be a Z-assignment verifying $(\mathrm{I}(\phi_1) \vee \mathrm{I}(\phi_2))(\vec{z}) = \bot$. Let $\vec{x}$ be a $\{x\}$-assignment: either $\vec{x}(x) = 0$, in which case $\mathrm{I}(\psi)(\vec{x}.\vec{z}) = \bot$, as $\mathrm{I}(\phi_1)(\vec{z}) = \bot$; or $\vec{x}(x) = 1$, in which case $\mathrm{I}(\psi)(\vec{x}.\vec{z}) = \bot$ too, because this time $\mathrm{I}(\phi_2)(\vec{z}) = \bot$. Hence there always exists no $\vec{x}$ such that $\mathrm{I}(\psi)(\vec{x}.\vec{z}) = \top$.

We thus proved that $\mathrm{forget}(\mathrm{I}(\psi), \{x\}) = \mathrm{I}(\phi_1) \vee \mathrm{I}(\phi_2)$. It is possible to build ψ in time linear in the size of ϕ_1 and ϕ_2. Therefore, if dFSD supported **SFO**, we would have a polytime algorithm allowing to build a dFSD equivalent to the disjunction of two dFSDs. Yet it is impossible, unless $\mathsf{P} = \mathsf{NP}$ (see $\vee$**BC**). Hence dFSD does not support **SFO**, and *a fortiori* **FO**, unless $\mathsf{P} = \mathsf{NP}$. □

`MDD`$_<$ *and* `MDD` *do not support* **SFO**, **FO**. The proof can be done in a similar fashion, building ψ from k MDDs$_<$ with a root variable having a domain of size k. ψ is obviously an MDD bearing on the same variable order (with the root variable being inferior to all other variables). If `MDD`$_<$ supported **SFO**, we would have a polytime algorithm allowing to build an MDD$_<$ equivalent to the disjunction of k MDDs$_<$, which is impossible. Thus `MDD`$_<$ does not support **SFO**, and *a fortiori* **FO**.

This also proves the result for `MDD`, as only one MDD is considered for this transformation. □

`FSD` *supports* **FO**, **SFO**. Let ψ be any reduced FSD such that $\mathrm{Var}(\psi) \neq \emptyset$, and let $x \in \mathrm{Var}(\psi)$. We denote $\psi^{\downarrow x}$ the FSD obtained by changing every x-labeled node in ψ by an $\veebar$-labeled node and every outgoing edge of such nodes by a $\{0\}$-labeled edge. We will show that $\mathrm{I}(\psi^{\downarrow x}) = \mathrm{forget}(\mathrm{I}(\psi), \{x\})$.

Let $Z = \mathrm{Var}\,\psi \setminus \{x\}$. We have to prove that

$$\forall \vec{z} \in \mathrm{Dom}(Z), \qquad \mathrm{I}(\psi^{\downarrow x})(\vec{z}) = \top \quad \Leftrightarrow \quad \exists \vec{x} \in \mathrm{Dom}(\{x\}), \mathrm{I}(\psi)(\vec{x}.\vec{z}) = \top$$

Let $\vec{z}$ be any Z-assignment:

($\Rightarrow$) if $\mathrm{I}(\psi^{\downarrow x})(\vec{z}) = \top$, there is a path p in $\psi^{\downarrow x}$ that is compatible with $\vec{z}$. Let us denote as p' the path in ψ that corresponds to p. Obviously, $\vec{z}$ is compatible with p': nodes labeled with variables in Z are not modified.
If p' contains no x-node, then any $\vec{x} \in \mathrm{Dom}(\{x\})$ trivially satisfies the requirement $\mathrm{I}(\psi)(\vec{x}.\vec{z}) = \top$. Otherwise, p' contains x-edges the labels of which are not disjoint (ψ is reduced and focusing), so we can also find $\vec{x} \in \mathrm{Dom}(\{x\})$ such that $\mathrm{I}(\psi)(\vec{x}.\vec{z}) = \top$.

($\Leftarrow$) if $\mathrm{I}(\psi^{\downarrow x})(\vec{z}) = \bot$, there is no path in $\psi^{\downarrow x}$ that is compatible with $\vec{z}$. Let us suppose there exists $\vec{x} \in \mathrm{Dom}(\{x\})$ such that $\mathrm{I}(\psi)(\vec{x}.\vec{z}) = \top$. Then, there is path p in ψ which is compatible with both $\vec{x}$ and $\vec{z}$. In $\psi^{\downarrow x}$, the x-nodes have been replaced by $\veebar$-nodes, and the Z-nodes have not been modified. Hence, the path p' in $\psi^{\downarrow x}$ that corresponds to p is compatible with $\vec{z}$. This is contradictory, since we supposed $\mathrm{I}(\psi^{\downarrow x})(\vec{z}) = \bot$. Consequently, there is no $\vec{x} \in \mathrm{Dom}(\{x\})$ such that $\mathrm{I}(\psi)(\vec{x}.\vec{z}) = \top$.

The process of obtaining $\psi^{\downarrow x}$ is linear, and we can forget as many variables as we need with a single traversal of the graph, hence `FSD` supports **FO** and thus **SFO**. □

B.13 Ensuring

Lemma B.7. $\mathrm{ensure}(\mathrm{I}, \{x\}) = \bigwedge_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}$.

Proof. Straightforward from the definition: let $W = X \setminus \{x\}$. $\mathrm{ensure}(\mathrm{I}, \{x\}) = \mathrm{I}^{\Downarrow W}$. Then $\mathrm{ensure}(\mathrm{I}, \{x\})(\vec{w}) = \top$ iff $\forall \vec{x} \in \mathrm{Dom}(\{x\})$ it holds that $\mathrm{I}(\vec{w}.\vec{x}) = \top$. Hence $\mathrm{ensure}(\mathrm{I}, \{x\}) = \bigwedge_{\vec{x} \in \mathrm{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}$. □

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

`dSD` *does not support* **EN**. Same proof as the previous one (`dSD` does not support **VA** unless $\mathsf{P} = \mathsf{NP}$). □

`SD` *supports* **SEN**. From Lemma B.6, it holds that $\operatorname{ensure}(\mathrm{I}, \{x\}) = \bigwedge_{\vec{x} \in \operatorname{Dom}(\{x\})} \mathrm{I}_{|\vec{x}}$. Let $n = |\operatorname{Card}(\operatorname{Dom}(x))|$; to obtain an SD whose interpretation function is equal to $\operatorname{ensure}(\mathrm{I}, \{x\})$. it is sufficient to make n simple conditionings and $n-1$ conjunctions. These two operations are linear, and n is linear in $|\phi|$, thus **SEN** is feasible in time polynomial in $|\phi|$. □

`dSD` *supports* **SEN**. Same proof as the previous one (`dSD` satisfies **SCD** and $\wedge$**BC** in linear time). □

`FSD` *does not support* **SEN**, **EN**. Let ϕ_1 and ϕ_2 be two FSDs.

Let $Z = \operatorname{Var}(\phi_1) \cup \operatorname{Var}(\phi_2)$, and a variable $x \notin Z$ of domain $\{0, 1\}$.

We build the graph ψ by merging the sinks of ϕ_1 and ϕ_2, and adding a node which will be the root of ψ, labeled by x, with one outgoing edge labeled by $\{0\}$ and pointing to the root of ϕ_1, and a second outgoing edge labeled by $\{1\}$ and pointing to the root of ϕ_2.

ψ is obviously focusing, since ϕ_1 and ϕ_2 are, and x is mentioned in one node only.

Let $\vec{x}$ be the 0 assignment of $\{x\}$ and $\vec{x}'$ be the 1 assignment of $\{x\}$

By construction, $\mathrm{I}(\phi_1) = \mathrm{I}(\psi_{|\vec{x}})$ (*i.e.* $\vec{z}$ is a model of $\mathrm{I}(\phi_1)$ iff $\vec{z}(x) = 0$). Similarly, $\mathrm{I}(\phi_2) = \mathrm{I}(\psi_{|\vec{x}'})$ (*i.e.* $\vec{z}$ is a model of $\mathrm{I}(\phi_1)$ iff $\vec{z}(x) = 1$).

From Lemma B.7, it holds that $\operatorname{ensure}(\mathrm{I}, \{x\}) = \bigwedge_{\vec{x} \in \operatorname{Dom}(\{x\})} \mathrm{I}_{|\vec{x}})$

Hence $\operatorname{ensure}(\mathrm{I}(\psi), \{x\}) = \mathrm{I}(\psi_{|\vec{x}}) \wedge \mathrm{I}(\psi_{|\vec{x}'})$, that is to say: $\operatorname{ensure}(\mathrm{I}(\psi), \{x\}) = \mathrm{I}(\phi_1) \wedge \mathrm{I}(\phi_2)$.

It is possible to build ψ in time linear in the size of ϕ_1 and ϕ_2. Therefore, if `FSD` supported **SEN**, we would have a polytime algorithm allowing to build an FSD equivalent to the conjunction of two FSDs. Yet it is impossible, unless $\mathsf{P} = \mathsf{NP}$ (see $\wedge$**BC**). Hence `FSD` does not support **SEN**, and *a fortiori* **EN**, unless $\mathsf{P} = \mathsf{NP}$. □

`dFSD` *does not support* **SEN**, **EN**. The proof is similar the previous one, from two dFSDs ϕ_1 and ϕ_2 — the previous construction of ψ is a dFSD. □

`MDD`$_<$ *and* `MDD` *do not support* **SEN**, **EN**. Again, the proof is similar, building ψ from k MDDs$_<$ with a root variable having a domain of size k. ψ is obviously an MDD bearing on the same variable order (with the root variable being inferior to all other variables). If `MDD`$_<$ supported **SEN**, we would have a polytime algorithm allowing to build an MDD$_<$ equivalent to the conjunction of k MDDs$_<$, which is impossible. Thus `MDD`$_<$ does not support **SEN**, and *a fortiori* **EN**.

This also proves the result for `MDD`, as only one MDD is considered for this transformation. □