Symmetry-Driven Decision Diagrams for Knowledge Compilation

Anicet Bart and **Frédéric Koriche** and **Jean-Marie Lagniez** and **Pierre Marquis**[1]

Abstract. In this paper, symmetries are exploited for achieving significant space savings in a knowledge compilation perspective. More precisely, the languages FBDD and DDG of decision diagrams are extended to the languages $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ of symmetry-driven decision diagrams, where X is a set of "symmetry-free" variables and Y is a set of "top" variables. Both the time efficiency and the space efficiency of $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ are analyzed, in order to put those languages in the knowledge compilation map for propositional representations. It turns out that each of $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ satisfies **CT** (the model counting query). We prove that no propositional language over a set $X \cup Y$ of variables, satisfying both **CO** (the consistency query) and **CD** (the conditioning transformation), is at least as succinct as any of $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ unless the polynomial hierarchy collapses. The price to be paid is that only a restricted form of conditioning and a restricted form of forgetting are offered by $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$. Nevertheless, this proves sufficient for a number of applications, including configuration and planning. We describe a compiler targeting $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ and give some experimental results on planning domains, highlighting the practical significance of these languages.

1 INTRODUCTION

It is well-known that many reasoning and optimization problems exhibit symmetries, and that recognizing and taking advantage of symmetries is a way to improve the computational time needed to solve those problems. Actually, much work has been devoted to this issue for decades. Among other highlights is the fact that the resolution system, equipped with a global symmetry rule, permits polynomial-length proofs of several combinatorial principles, including the pigeon/hole formulae [9], while such formulae require resolution proofs of exponential length [8, 14].

The main objective of this paper is to show that exploiting symmetries also proves valuable for *achieving space savings* in a knowledge compilation perspective, i.e., to derive more succinct compiled representations while preserving queries and transformations of interest. In order to reach this goal, we extend the language FBDD of free binary decision diagrams [7] to the language $\texttt{Sym-FBDD}_{X,Y}$ of symmetry-driven free binary decision diagrams, containing free binary decision diagrams equipped with symmetries. X is a (possibly empty) set of "symmetry-free" variables, and Y is a (possibly full) set of "top" variables. We also extend the language DDG of decomposable decision diagrams [5] (a superset of FBDD where decomposable $\wedge$-nodes are allowed in the representations) to the language Sym-DDG of symmetry-driven decomposable decision diagrams, where the same conditions on X and Y are considered. We analyze $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ along the lines of the knowledge compilation map for propositional representations [4], by identifying the queries and transformations of interest for which some polynomial-time algorithms exist when the input is a representation from one of those languages; we also investigate the space efficiency of $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$. Based on these investigations, it turns out that each of $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ satisfies the critical **CT** query (model counting) which is, for many languages, hard to satisfy (a #P-complete problem). We prove that no propositional language over a set $X \cup Y$ of variables, satisfying both **CO** (the consistency query) and **CD** (the conditioning transformation), is at least as succinct as any of $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ unless the polynomial hierarchy collapses. The price to be paid is that only restricted forms of conditioning and of projection are offered by $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$, namely conditioning over X and projection on Y. Nevertheless, this proves sufficient for a number of applications, including configuration and planning. We describe a compiler targeting $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ and give some experimental results on planning domains, highlighting the practical significance of these languages.

The paper is organized as follows. After introducing the formal background, the languages $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ are defined and analyzed. A CNF-to-$\texttt{Sym-DDG}_{X,Y}$ compiler is described in the next section. Before concluding, empirical results on some planning instances are presented, showing that the size of $\texttt{Sym-DDG}_{X,Y}$ compilations can be significantly smaller than the size of the state-of-the-art d-DNNF compilations. Proofs are not provided in the paper due to space limitations, but can be found in an extended version, available at `www.cril.fr/~marquis/symddg.pdf`.

2 FORMAL PRELIMINARIES

Let PS be a finite set of propositional variables. A permutation σ over L_{PS}, the set of all literals over PS, is a bijective mapping from $L_{PS} = PS \cup \{\neg x \mid x \in PS\}$ to L_{PS}. Any permutation σ can be extended easily to a morphism associating a propositional formula over PS with a propositional formula over PS, by stating that for every propositional connective c of arity k, we have $\sigma(c(\alpha_1, \ldots, \alpha_k)) = c(\sigma(\alpha_1), \ldots, \sigma(\alpha_k))$. We also note $\sigma(X) = \{\sigma(x) \mid x \in X\}$ for any subset X of PS.

Every permutation σ under consideration in this paper is assumed to satisfy the following *stability condition*: for any pair of literals ℓ_1, ℓ_2, $\sigma(\ell_1) = \ell_2$ iff $\sigma(\sim\ell_1) = \sim\ell_2$ where $\sim\ell$ is the opposite of ℓ, i.e., $\sim x = \neg x$ and $\sim\neg x = x$. Any permutation σ will be represented in a *simplified cycle notation*, i.e., as a product of cycles corresponding to its orbits (with at least two elements), where exactly one of

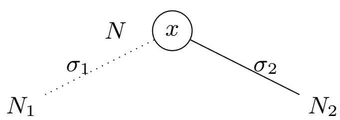
[1] CRIL-CNRS, Université d'Artois, France, email: name@cril.fr

the two orbits $(l_1 \ \ldots \ l_k)$ and $(\sim l_1 \ \ldots \ \sim l_k)$ are represented, whenever $(l_1 \ \ldots \ l_k)$ is an orbit of σ.

For instance, if $PS = \{x_1, \ldots, x_6\}$, $(x_1 \neg x_3 \ x_4)(x_5 x_6)$ denotes the permutation σ associating x_1 with $\neg x_3$, $\neg x_1$ with x_3, x_3 with $\neg x_4$, $\neg x_3$ with x_4, x_4 with x_1, $\neg x_4$ with $\neg x_1$, x_5 with x_6, $\neg x_5$ with $\neg x_6$, x_6 with x_5, and $\neg x_6$ with $\neg x_5$, while x_2 and $\neg x_2$ are left unchanged by σ. The identity permutation is represented by the empty word using the simplified cycle notation.

By Σ, we denote the set of all bijective mappings from L_{PS} to L_{PS} satisfying the stability condition. Clearly enough, Σ is closed by composition: if $\sigma_1, \sigma_2 \in \Sigma$ then $\sigma_1 \circ \sigma_2 \in \Sigma$. Since Σ is also closed for the inverse (if $\sigma \in \Sigma$, then $\sigma^{-1} \in \Sigma$) and it contains the identity element (which is the neutral element for composition), Σ is a permutation group. Clearly enough, applying a permutation $\sigma \in \Sigma$ to a propositional formula α does not change the *number of models* of the latter; especially, α is satisfiable (resp. valid) iff $\sigma(\alpha)$ is satisfiable (resp. valid).

In the rest of the paper, we focus on subsets of Sym−EDD, the language of symmetry-driven extended decision diagrams, where permutations are defined over Σ. Basically, Sym−EDD generalizes the language of "extended" decision diagrams (i.e., binary decision diagrams in which $\wedge$-nodes are allowed) by associating some permutations to the arcs and to the root node. Diagrams from Sym−EDD are based on decision nodes, where a decision node N labeled with $x \in PS$ is a node with two children, having the following form:



Such a node N is noted $ite(x, N_1, N_2)$, where "ite" stands for "if ... then ... else ...".

Definition 1 (Sym−EDD). Sym−EDD *is the set of all finite, single-rooted multi-DAGs* [2] *(also referred to as "formulae") α where:*

- *each leaf node of α is either the $\top$-node (a node labeled by the Boolean constant $\top$ – always true) or the $\bot$-node (a node labeled by the Boolean constant $\bot$ – always false)*
- *each internal node of α is labeled by $\wedge$ and has a finite number of children (≥ 1), or it is a decision node labeled with a variable from PS;*
- *each arc of α is labeled with a permutation from Σ;*
- *the root of α is labeled with a permutation from Σ.*

The size $|\alpha|$ of a Sym−EDD formula α is the number of nodes, plus the number of arcs in the DAG, plus the sizes of the permutations labeling the arcs of α and its root. The set $Var(\alpha)$ of variables of a Sym−EDD formula α rooted at node N is defined by $\{\sigma_N(x) \mid x \in Var(N)\}$, where σ_N is the permutation labeling N, and $Var(N)$ is defined as follows:

- if N is a leaf node labeled by a Boolean constant, then $Var(N) = \varnothing$;
- if N is a node labeled by $\wedge$ and having k children $N_1, \ldots, N_k$ such that $\forall i \in 1, \ldots, k$, σ_i is the label of the arc (N, N_i), then $Var(N) = \bigcup_{i=1}^{k} \sigma_i(Var(N_i))$;
- if $N = ite(x, N_1, N_2)$ is a decision node such that σ_1 is the label of the arc (N, N_1) and σ_2 is the label of the arc (N, N_2), then $Var(N) = \{x\} \cup \sigma_1(Var(N_1)) \cup \sigma_2(Var(N_2))$.

[2] More than one arc between two nodes is allowed.

Clearly enough, $Var(\alpha)$ can be computed in time linear in $|\alpha|$. Note that $Var(\alpha)$ may easily differ from the set of variables occurring in α, when no permutation is taken into account (or equivalently, when each permutation is equal to the identity permutation).

Let us now define the semantics of Sym−EDD formulae. A simple way to do so consists in associating with every Sym−EDD formula α a tree-shaped NNF formula $T(\alpha)$ which is logically equivalent to α. Formally, $T(\alpha)$ is given by $\sigma_N(T(N))$ where N is the root of α and $T(N)$ is defined inductively as follows:

- if N is a leaf node labeled by the Boolean constant $\top$ (resp. $\bot$), then $T(N) = \top$ (resp. $\bot$);
- if N is a node labeled by $\wedge$ and having k children $N_1, \ldots, N_k$ such that $\forall i \in 1, \ldots, k$, σ_i is the label of the arc (N, N_i), then $T(N) = \bigwedge_{i=1}^{k} \sigma_i(T(N_i))$;
- if $N = ite(x, N_1, N_2)$ is a decision node such that $\forall i \in 1, \ldots, 2$, σ_i is the label of the arc (N, N_i), then $T(N) = (\neg x \wedge \sigma_1(T(N_1))) \vee (x \wedge \sigma_2(T(N_2)))$.

Of course, the size of $T(\alpha)$ is exponentially larger than the size of α in the general case. Anyway, the models of α are precisely those of $T(\alpha)$. We denote by $\|\alpha\|$ the number of models of α over $Var(\alpha)$.

For space reasons, we assume the reader is familiar with the languages FBDD, DDG, DNNF [7, 5, 3] which are considered in the following, and with the KC map [4]. The basic queries considered in the KC map include tests for consistency **CO**, validity **VA**, implicates (clausal entailment) **CE**, implicants **IM**, sentential entailment **SE**, model counting **CT**, and model enumeration **ME**. We add to them the *model checking query* **MC**, which is not obvious for the languages we introduce in the paper. The basic transformations include conditioning (**CD**), (possibly bounded) closures under the connectives ($\wedge$**C**, $\wedge$**BC**, $\vee$**C**, $\vee$**BC**, $\neg$**C**), and forgetting (**FO**), or dually projection (**PR**). We add to them the *restricted conditioning transformation*, and the *restricted projection transformation*:

Definition 2 (X-**RCD**). *Let $\mathcal{L}$ be a subset of* Sym−EDD*, and $X \subseteq PS$. $\mathcal{L}$ satisfies X-**RCD** iff there is a polynomial-time algorithm that maps every formula α in $\mathcal{L}$ and every consistent term γ over some variables in X to a formula $\alpha \mid \gamma$ in $\mathcal{L}$ which is logically equivalent to the most general logical consequence β of $\alpha \wedge \gamma$, where β is independent from the variables occurring in γ.*

Definition 3 (Y-**RPR**). *Let $\mathcal{L}$ be a subset of* Sym−EDD*, and $Y \subseteq PS$. $\mathcal{L}$ satisfies Y-**RPR** iff there is a polynomial-time algorithm that maps every formula α in $\mathcal{L}$ and every $Z \subseteq Y$ to a formula in $\mathcal{L}$ which is logically equivalent to the projection $\exists PS \setminus Z.\alpha$ of α on Z.*[3]

Clearly enough, X-**RCD** (resp. Y-**RPR**) coincides with the usual conditioning transformation **CD** (resp. projection transformation **PR**) when $X = PS$ (resp. $Y = PS$).

The relative space efficiency of propositional languages is captured by a pre-order $\leq_s$, where $\mathcal{L}_1 \leq_s \mathcal{L}_2$ means that $\mathcal{L}_1$ is at least as succinct as $\mathcal{L}_2$, i.e., there exists a polynomial p such that for every formula $\alpha \in \mathcal{L}_2$, there exists an equivalent formula $\beta \in \mathcal{L}_1$ where $|\beta| \leq p(|\alpha|)$.

$\sim_s$ denotes the the symmetric part of $\leq_s$ defined by $\mathcal{L}_1 \sim_s \mathcal{L}_2$ iff $\mathcal{L}_1 \leq_s \mathcal{L}_2$ and $\mathcal{L}_2 \leq_s \mathcal{L}_1$. $<_s$ denotes the asymmetric part of $\leq_s$ defined by $\mathcal{L}_1 <_s \mathcal{L}_2$ iff $\mathcal{L}_1 \leq_s \mathcal{L}_2$ and $\mathcal{L}_2 \not\leq_s \mathcal{L}_1$. $\mathcal{L}_1 \not\leq_s^* \mathcal{L}_2$ means that $\mathcal{L}_1 \not\leq_s \mathcal{L}_2$ unless the polynomial hierarchy PH collapses (which is considered very unlikely in complexity theory). $\mathcal{L}_1 <_s^* \mathcal{L}_2$ is a short for $\mathcal{L}_1 \leq_s \mathcal{L}_2$ and $\mathcal{L}_2 \not\leq_s^* \mathcal{L}_1$.

[3] Or, in an equivalent way, to the forgetting of every variable in α but those of Z.

3 SYMMETRY-DRIVEN DIAGRAMS

Sym−EDD does not qualify as an interesting language for knowledge compilation. Especially, it contains BDD (not to be mixed with $\text{OBDD}_<$), as a subset, and BDD is highly intractable [4]. Accordingly, some restrictions must be put on the symmetry-based decision diagrams in order to get languages which are exploitable from the knowledge compilation point of view. Those two restrictions are formally expressed by two conditions, the read-once condition and the decomposability condition, which extend the eponymous conditions on decision graphs to symmetry-driven ones:

Definition 4 (read-once)**.**

- *A decision node N labeled with $y \in PS$ in a* Sym−EDD *formula α is an x-decision node iff there exists a path $a_1, \ldots, a_m$ from the root of α to N, such that the corresponding permutation $\sigma_0 \circ \sigma_1 \circ \ldots \circ \sigma_m$*[4] *satisfies $\sigma_0 \circ \sigma_1 \circ \ldots \circ \sigma_m(y) = x$. N is a X-decision node where $X \subseteq PS$ if it is a x-decision node for some $x \in X$.*
- *A* Sym−EDD *formula α satisfies the* read-once property *iff for every variable $x \in PS$, in every path from the root of α to a leaf, at most one x-decision node can be encountered.*

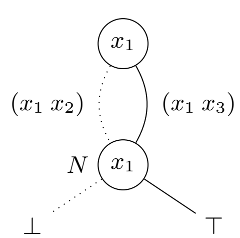
Definition 5 (decomposability)**.**

- *A $\wedge$-node N with children $N_1, \ldots, N_k$ is* decomposable *if and only if for all $i, j \in 1, \ldots, k$, if $i \neq j$, then $\sigma_i(Var(N_i)) \cap \sigma_j(Var(N_j)) = \varnothing$, where σ_i (resp. σ_j) is the permutation labeling the arc (N, N_i) (resp. (N, N_j)).*
- *A* Sym−EDD *formula α satisfies the* decomposability property *iff each $\wedge$-node of it is decomposable.*

Definition 6 (Sym−DDG, Sym−FBDD)**.**

- Sym−DDG *is the subset of* Sym−EDD *consisting of formulae α which are both read-once and decomposable.*
- Sym−FBDD *is the subset of* Sym−DDG *consisting of formulae α containing no $\wedge$-node.*

Unlike what happens in the FBDD case, a node of α can easily be an x-decision node for several variables x. For instance, the node N represented on the figure below is both an x_2-decision node and an x_3-decision node.



Clearly, DDG (resp. FBDD) is the subset of Sym−DDG (resp. Sym−FBDD) containing formulae where every permutation used in them is the identity permutation. In order to define the classes of interest in this study, two additional conditions need to be considered:

Definition 7 (symmetry-freeness condition on X)**.** *Let X be a subset of PS. A* Sym−DDG *formula α satisfies the* symmetry-freeness condition *on X iff for any $x \in X$ and any decision node N in α, if N is an x-decision node and a y-decision node ($y \in PS$), then $y = x$.*

[4] Here, σ_0 is the permutation labeling the root of α and, for all $i \in \{1, \cdots, m\}$, σ_i is the permutation labeling a_i.

Definition 8 (precedence condition on Y)**.** *Let Y be a subset of PS. A* Sym−DDG *formula α satisfies the* precedence condition *on Y iff for every z-decision node N of α such that $z \notin Y$ and for every y-decision node M with $y \in Y$, there is no path in α from N to M.*

Definition 9 ($\text{Sym−DDG}_{X,Y}$, $\text{Sym−FBDD}_{X,Y}$)**.**
Given two subsets X, Y of PS:

- $\text{Sym−DDG}_{X,Y}$ *is the subset of* Sym−DDG *containing diagrams satisfying the symmetry-freeness condition on X and the precedence condition on Y, and*
- $\text{Sym−FBDD}_{X,Y}$ *is the subset of* Sym−FBDD *containing diagrams satisfying the symmetry-freeness condition on X and the precedence condition on Y.*

Importantly, in the above languages, is it assumed that X and Y are "fixed" subsets of PS. In other words, for any formula α in $\text{Sym−DDG}_{X,Y}$, the X-decision nodes of α and the Y-decision nodes of α are known, and tagged as such.

Clearly, if X, X' are subsets of PS such that $X \supseteq X'$ then for any $Y \subseteq PS$, we have $\text{Sym−DDG}_{X,Y} \subseteq \text{Sym−DDG}_{X',Y}$. When $Y = PS$, i.e., when no precedence condition actually constrains the diagrams, two extreme cases are reached with $X = \varnothing$ and $X = PS$, respectively. $\text{Sym−DDG}_{\varnothing,PS}$ coincides with Sym−DDG, while $\text{Sym−DDG}_{PS,PS}$ coincides with DDG (of course, similar equalities hold for Sym−FBDD, *mutatis mutandis*).

By definition, each $\text{Sym−FBDD}_{X,Y}$ (resp. $\text{Sym−DDG}_{X,Y}$) formula is a Sym−FBDD (resp. Sym−DDG) formula. Conversely, every Sym−FBDD (resp. Sym−DDG) formula α can also be viewed as a $\text{Sym−FBDD}_{X,Y}$ (resp. $\text{Sym−DDG}_{X,Y}$) formula for every $X, Y \subseteq PS$ such that $X \cap Var(\alpha) = \varnothing$ and $Var(\alpha) \subseteq Y$.

In what follows, a family of propositional languages $\mathcal{L}_{X,Y}$ parameterized by two sets of variables X and Y, is said to satisfy a given query request **R** (i.e., a query or a transformation), if **R** is satisfied for each language obtained by fixing the values of X and of Y.

Concerning the queries and the transformations, we have obtained the following results:

Proposition 1. *The results in Table 1 and in Table 2 hold.*

In the two tables the results concerning DDG and FBDD [7, 5] are reported as base lines for the comparison matter. One can observe that $\text{Sym−FBDD}_{X,Y}$ (resp. $\text{Sym−DDG}_{X,Y}$) typically offers the same tractable transformations as FBDD (resp. DDG), but **CD**, which must be downsized to X-**RCD**, and **PR**, which must be downsized to Y-**RPR**. Concerning queries, **CT** and the related conditions **CO** and **VA** are preserved. Model checking (**MC**) and model enumeration (**ME**) are offered by each of the two languages. Contrastingly, other queries related to the conditioning transformation (**CE**, **IM**) are lost.

Interestingly, the polynomial-time algorithm at work for the **CT** query in the DDG case can be extended in a trivial way to the Sym−DDG case (just ignores the permutations since they do not change the number of models). This is also the case for the X-**RCD** transformation (for each $x \in X$, replace each arc reaching an x-decision node N by the arc from N to its right child or by the arc from N to its left child, depending on the polarity of x in the term γ) and the Y-**RPR** transformation (determine for each decision node N whether it is a "last" Y-decision node, i.e., no successor of it in a path from N to a leaf is a Y-decision node; then replace the arc (N, M) from each "last" Y-decision node N by an arc towards the $\top$-node if there is a path from M to the $\top$-node, and replace (N, M) by an arc towards the $\perp$-node otherwise). Things are a bit more tricky for **MC** and **ME** (polynomial-time algorithms for these queries are presented in the extended version).

	CO	**VA**	**CE**	**IM**
$\mathtt{Sym\text{-}DDG}_{X,Y}$	$\surd$	$\surd$	$\circ$	$\circ$
$\mathtt{Sym\text{-}FBDD}_{X,Y}$	$\surd$	$\surd$	$\circ$	$\circ$
DDG	$\surd$	$\surd$	$\surd$	$\surd$
FBDD	$\surd$	$\surd$	$\surd$	$\surd$

	SE	**CT**	**ME**	**MC**
$\mathtt{Sym\text{-}DDG}_{X,Y}$	$\circ$	$\surd$	$\surd$	$\surd$
$\mathtt{Sym\text{-}FBDD}_{X,Y}$	$\circ$	$\surd$	$\surd$	$\surd$
DDG	$\circ$	$\surd$	$\surd$	$\surd$
FBDD	$\circ$	$\surd$	$\surd$	$\surd$

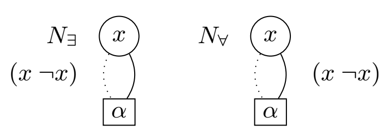
Table 1. $\mathtt{Sym\text{-}DDG}_{X,Y}$, $\mathtt{Sym\text{-}FBDD}_{X,Y}$, and the queries **CO**, **VA**, **CE**, **IM**, **SE**, **CT**, **ME**, **MC**. $\surd$ means "satisfies", and $\circ$ means "does not satisfy unless P = NP".

	CD	X-**RCD**	**PR**	Y-**RPR**
$\mathtt{Sym\text{-}DDG}_{X,Y}$	$\circ$	$\surd$	$\circ$	$\surd$
$\mathtt{Sym\text{-}FBDD}_{X,Y}$	$\circ$	$\surd$	$\circ$	$\surd$
DDG	$\surd$	$\surd$	$\circ$	$\circ$
FBDD	$\surd$	$\surd$	$\bullet$	$\bullet$

	$\wedge$**C**	$\wedge$**BC**	$\vee$**C**	$\vee$**BC**	$\neg$**C**
$\mathtt{Sym\text{-}DDG}_{X,Y}$	$\circ$	$\circ$	$\circ$	$\circ$	?
$\mathtt{Sym\text{-}FBDD}_{X,Y}$	$\circ$	$\circ$	$\circ$	$\circ$	$\surd$
DDG	$\circ$	$\circ$	$\circ$	$\circ$	?
FBDD	$\bullet$	$\circ$	$\bullet$	$\circ$	$\surd$

Table 2. $\mathtt{Sym\text{-}DDG}_{X,Y}$, $\mathtt{Sym\text{-}FBDD}_{X,Y}$, and the transformations **CD**, X-**RCD**, **PR**, Y-**RPR**, $\wedge$**C**, $\wedge$**BC**, $\vee$**C**, $\vee$**BC**, $\neg$**C**. $\surd$ means "satisfies", $\bullet$ means "does not satisfy", and $\circ$ means "does not satisfy unless P=NP".

In a nutshell, it turns out that $\mathtt{Sym\text{-}FBDD}_{X,Y}$ and $\mathtt{Sym\text{-}DDG}_{X,Y}$ exhibit quite non-standard properties as target languages for knowledge compilation. Indeed, **CT** is typically hard to be satisfied (a #P-complete problem) while **CD** is usually obvious. In the same vein, model checking **MC** which is a straightforward query for usual knowledge compilation languages, is far from being easy for symmetry-driven graph-based languages, due to their ability of encoding quantifications in a succinct way. Indeed, assuming that α is any Sym−EDD formula, the Sym−EDD formula rooted at node $N_\exists$ on the figure below is equivalent to $\exists x.\alpha$ while the formula rooted at node $N_\forall$ is equivalent to $\forall x.\alpha$.[5] As a consequence, we get that Sym−EDD satisfies **CD** but also that **MC** for Sym−EDD formulae is PSPACE-hard!



The non-standard behavior of $\mathtt{Sym\text{-}DDG}_{X,Y}$ (and its subset $\mathtt{Sym\text{-}FBDD}_{X,Y}$) w.r.t. unrestricted conditioning seems to be the price to be paid for an improved succinctness power. Indeed, the next proposition shows that the languages $\mathtt{Sym\text{-}DDG}_{X,Y}$ and $\mathtt{Sym\text{-}FBDD}_{X,Y}$, are in some sense "very succinct":

Proposition 2. *Let X, Y be two subsets of PS. No propositional language $\mathcal{L}$ over $X \cup Y$ satisfying* **CD** *and* **CO** *is at least as succinct as $\mathtt{Sym\text{-}FBDD}_{X,Y}$ unless $\Sigma_2^p = \Pi_2^p$, i.e., we have $\mathcal{L} \not\leq_s^* \mathtt{Sym\text{-}FBDD}_{X,Y}$.*

[5] Observe that, due to the read-once condition, none of these two formulae belongs to Sym−DDG, unless $x \notin Var(\alpha)$.

Consider the CNF formula Δ_n containing every 3-clause δ (i.e., a clause of size 3) over $Y = \{y_1, \ldots, y_n\}$ augmented by an additional literal x_δ which identifies the clause. Δ_n is thus a CNF over $Y \cup X$ containing $8 \times \binom{n}{3}$ 4-clauses. X contains $8 \times \binom{n}{3}$ variables x_δ. The proof of Proposition 2 relies on the fact that Δ_n can be represented by an equivalent Sym−FBDD formula of size linear in the size of Δ_n. However, this statement does not hold for propositional languages satisfying **CO** and **CD** unless the polynomial hierarchy collapses. Indeed, to every CNF formula $\alpha = \bigwedge_{i=1}^m \delta_i$ over $Y = \{y_1, \ldots, y_n\}$, we can associate a consistent term $\gamma_\alpha = \bigwedge_{i=1}^m \neg x_{\delta_i}$ such that α is satisfiable iff $\Delta_n \wedge \gamma_\alpha$ is satisfiable iff Δ_n conditioned by γ_α is satisfiable. Suppose now that Δ_n has a polynomial-space representation $comp(\Delta_n)$ in a propositional language $\mathcal{L}$ over $X \cup Y$, where $\mathcal{L}$ satisfies **CO** and **CD**. Then in order to check whether a CNF formula $\alpha = \bigwedge_{i=1}^m \delta_i$ over $\{y_1, \ldots, y_n\}$ is satisfiable it would be enough to compute in polynomial time a $\mathcal{L}$-representation of $comp(\Delta_n)$ conditioned by γ_α, and to determine in polynomial time whether it is consistent or not. We would therefore get NP $\subseteq$ P/poly, hence $\Sigma_2^p = \Pi_2^p$ (see e.g. [13] for details).

As a consequence, state-of-the-art languages for knowledge compilation, like DNNF, are not more succinct than any of the two languages Sym−FBDD, and Sym−DDG. Especially, due to the obvious inclusions Sym−DDG $\supseteq$ DDG, and Sym−FBDD $\supseteq$ FBDD, we have that Sym−DDG $\leq_s$ DDG, and Sym−FBDD $\leq_s$ FBDD. This implies that:

Proposition 3. Sym−DDG $<_s^*$ DDG *and* Sym−FBDD $<_s^*$ FBDD.

4 A CNF-TO-$\mathtt{Sym\text{-}DDG}_{X,Y}$ COMPILER

Our compiler $\mathsf{symddg}_{X,Y}$ (see Algorithm 1) is essentially a CNF-to-DDG compiler, enriched with some modules for symmetry handling.

The two base cases are when the input CNF formula Δ is the empty set of clauses (line 1) or contains the empty clause (line 2). In the first case, Δ is equivalent to $\top$ and a $\mathtt{Sym\text{-}DDG}_{X,Y}$ reduced to the $\top$-node is returned. In the second case, Δ is equivalent to $\bot$ and a $\mathtt{Sym\text{-}DDG}_{X,Y}$ reduced to the $\bot$-node is returned. Otherwise the connected components $\Delta_1, \cdots, \Delta_k$ of the constraint graph of Δ are looked for (line 3); if Δ has more than one connected component (line 4), then the root node of the resulting $\mathtt{Sym\text{-}DDG}_{X,Y}$ formula is a $\wedge$-node (generating using function anode) and its children are obtained by calling $\mathsf{symddg}_{X,Y}$ recursively on $\Delta_1, \cdots, \Delta_k$.[6] Then, using a method findSymmetry for symmetry detection, we look whether there is an admissible permutation σ w.r.t. Δ such that $\sigma(\Delta)$ has already been encountered and is in the cache (line 5); if so, the $\mathtt{Sym\text{-}DDG}_{X,Y}$ formula associated with $\sigma(\Delta)$ is returned, and σ is associated with the arc which has been followed to reach the root node of the current formula Δ or to the root node of the initial CNF formula at the first call.[7] σ is admissible w.r.t. Δ when all the requirements imposed by $\mathtt{Sym\text{-}DDG}_{X,Y}$ are met, i.e., σ satisfies the stability condition, $\sigma(\Delta)$ is read-once and satisfies the precedence condition on Y, and $\forall x \in X$, $\sigma(x) = x$ (which is enough for ensuring that $\sigma(\Delta)$ satisfies the symmetry-freeness condition on X). Finally, in the remaining case (line 6), one chooses a variable from Δ; the root node of the resulting $\mathtt{Sym\text{-}DDG}_{X,Y}$ formula is a decision node labeled with x, and its two children are obtained by calling $\mathsf{symddg}_{X,Y}$ recursively on Δ conditioned by $\neg x$, and Δ conditioned

[6] Removing lines 3 and 4 in the pseudo-code of $\mathsf{symddg}_{X,Y}$ leads to downsize it as a CNF-to-$\mathtt{Sym\text{-}FBDD}_{X,Y}$ compiler.

[7] For the sake of clarity, this is not detailed in the pseudo-code. Also, though not explicitly indicated in the algorithm, each time a $\mathtt{Sym\text{-}DDG}_{X,Y}$ formula is generated, it is added to the cache when it is not already in it.

Algorithm 1: $\mathsf{symddg}_{X,Y}(\Delta)$

input : a CNF formula Δ, a set X of symmetry-free variables, and a set Y of top variables
output: a $\texttt{Sym-DDG}_{X,Y}$ formula equivalent to Δ

if Δ *is empty* then return $\mathsf{leaf}(\top)$;
if Δ *contains an empty clause* then return $\mathsf{leaf}(\bot)$;
let $\Delta_1, \cdots, \Delta_k$ be the connected components of Δ;
if $k > 1$ then return
$\mathsf{anode}(\mathsf{symddg}_{X,Y}(\Delta_1), \cdots, \mathsf{symddg}_{X,Y}(\Delta_k))$;
if $(\sigma \leftarrow \mathsf{findSymmetry}(\Delta))$ *such that*
$(key \leftarrow \mathsf{inCache}(\sigma(\Delta))) \neq \mathsf{nil}$ then return $\mathsf{cache}(key)$;
choose a variable x of Δ;
return $\mathsf{dnode}(x, \mathsf{symddg}_{X,Y}(\Delta\mid_{x\leftarrow 0}), \mathsf{symddg}_{X,Y}(\Delta\mid_{x\leftarrow 1}))$

by x. Specifically, x is chosen thanks to the VSADS heuristic function [12], adapted to ensure that the constraint on top variables Y is satisfied.

The key issue in the design of our $\mathsf{symddg}_{X,Y}$ compiler lies in an efficient implementation of the $\mathsf{findSymmetry}$ method for retrieving a formula Δ' in the cache that is equivalent to the input formula Δ, modulo an admissible symmetry σ. To this point, the problem of determining whether there exists a symmetry between two CNF formulae Δ and Δ' can be reduced to a graph isomorphism problem for which, unfortunately, no polynomial-time algorithm is known [6]. In the present study, this computational issue is circumvented using an *incomplete* method for detecting symmetries.

Specifically, $\mathsf{findSymmetry}$ is based on a two-stage filtering technique followed by a greedy search in the filtered space of permutations. In order to rapidly explore the cache, the first stage compares formulae according to their canonical *signature*. The signature of a CNF expression Δ is given by two sorted vectors, which respectively encode the signature of the variables occurring in Δ and the signature of the clauses occurring in Δ. The signature of a variable x is given by a pair (p_x, n_x) where p_x (resp. n_x) is the number of literals that occur positively (resp. negatively) in Δ. The signature of a clause δ is simply given by its size (i.e., its number of literals). Both vectors are sorted in increasing order of their entries. Based on this encoding, two CNF formulae Δ and Δ' with different signatures cannot be equivalent modulo an admissible symmetry.

During the second stage, the task of identifying an admissible symmetry between two comparable formulae Δ and Δ' is cast as a constraint satisfaction problem (CSP). The set of variables of the CSP is given by the collection of variables occurring in Δ, which are renamed for convenience. The domain of each (renamed) variable x is formed by the set of all literals ℓ occurring in Δ', such that x and ℓ have the same signature.[8] Finally, a binary constraint $x \neq y$ is associated with each pair of variables x, y occurring in the same clause δ of Δ. Based on this representation, the space of candidate permutations between Δ and Δ' is refined by enforcing arc-consistency in the CSP. An admissible permutation is searched in a greedy way by iteratively pruning values from the variable with largest domain, and propagating these unary constraints in the network.

5 EXPERIMENTAL RESULTS

We focus on the application of $\texttt{Sym-DDG}_{X,Y}$ to planning, an area wherein symmetries naturally occur (see e.g. [11]). Given a time horizon N, our objective is to compile a deterministic planning problem $P = (F, O, I, G)$, where the initial state I and the goal G vary. Here, F is a finite set of fluents, O is a set of deterministic actions with possibly conditional effects, I is a complete truth assignment of initial fluents in F, and G is a partial assignment of final fluents in F representing the goal situation. A plan π for P is a sequence π of sets of actions, one per time point between 0 and $N-1$, which maps the initial state I to a goal state (i.e., a model of G).

In order to compile P, we first encode a description of P into a corresponding CNF theory Δ_P over the set of variables $PS = (\bigcup_{i=0}^{N}\{f_i \mid f \in F\}) \cup \{a_i \mid a \in O, i = 0, \ldots, N-1\}$. Δ_P can be viewed as a compact representation of the transition model associated with O. In this encoding, f_i is true if and only if fluent f holds at time point i, and a_i is true if and only if action a holds at time point i. Since only deterministic actions are considered in O, the truth value of every fluent f_i ($f \in F, i \in 1, \ldots, N$) is fully determined in Δ_P as soon as the truth values of the variables $\{f_0 \mid f \in F\} \cup \bigcup_{i=0}^{N-1}\{a_i \mid a \in A\}$ are fixed (i.e., as soon as the initial state and the plan under consideration are specified).

Based on this encoding, the formula Δ_P is compiled into a $\texttt{Sym-DDG}_{X,Y}$ formula where $X = \{f_0 \mid f \in F\} \cup \{f_N \mid f \in F\}$ and $Y = PS$. Thus, the permutation group for the target class is defined over all action variables and all fluent variables that exclude initial and goal descriptions. Once this compiled form has been computed, one can take advantage of the set of queries and transformations offered by $\texttt{Sym-DDG}_{X,Y}$ to address in a computationally efficient way a number of issues which are NP-hard in the general case. Thus, since $\texttt{Sym-DDG}_{X,Y}$ satisfies both X-**RCD** and **CO**, we can determine in polynomial time whether a plan π exists for any I and G given on-line; since $\texttt{Sym-DDG}_{X,Y}$ satisfies **CT**, we can also count how many π exist in polynomial time.

The instances we selected cover a range of different planning benchmarks, with varying horizon length. “blocks-n” refers to the famous blocks-world domain with n blocks. “bomb-m-n” is another popular domain involving m bombs, n toilets, and 2 actions. “comm-m-n” is an IPC5 problem about communication signals with m stages, n packets, and 5 actions. “emptyroom-n” is about navigating a robot in an $n \times n$ empty grid. Finally, “safe-n” is about opening a safe with n possible combinations.

All instances described in PDDL were translated into CNF theories using the DIMACS format, and then compiled according to three target languages: d-DNNF generated by the standard c2d compiler,[9] DDG generated by our $\mathsf{symddg}_{X,Y}$ compiler without symmetry detection, and Sym-DDG targeted by the full power of our compiler. Our experiments have been conducted on a Xeon E5-2643 (3.30GHz) with 7.6GB of memory. A time-out of one hour per instance has been considered for the compilation phase; for space reasons we do not provide detailed compilation times but it is worth noting that they remain reasonable: on the instances above, the mean (resp. maximum) compilation time of $\mathsf{symddg}_{X,Y}$ was 22.5 seconds (resp. 109.42 seconds, obtained for the bomb-20-05 instance).

Table 3 presents the compilation results obtained from instances for which the horizon N was fixed to 5. ”Nodes” (resp. ”arcs”) refer to the numbers of nodes (resp. arcs) in the compiled representations. The last two columns give the percentage of size reduction achieved by Sym-DDG compared with DDG. Figure 1 plots the size (nodes + arcs) of the compiled formula versus the horizon length (from 1 to 10) for three of the test instances. Since the running time of on-line queries and transformations is governed by the size of the com-

[8] In an obvious way, the signature of ℓ is (p_x, n_x) if ℓ is the positive literal x, and (n_x, p_x) if ℓ is the negative literal $\neg x$.

[9] c2d, available at `reasoning.cs.ucla.edu/c2d/`, was run using the command `c2d -in file.cnf -dt_count 50 -smooth_all`

Instance	CNF		d-DNNF		DDG		Sym-DDG		% reduction	
	vars	clauses	nodes	arcs	nodes	arcs	nodes	arcs	nodes	arcs
blocks-2	406	1901	4679	37892	7332	15130	6792	14044	7.4	7.2
blocks-3	804	4343	157292	2357558	1208463	2598572	950142	2039622	**21.4**	**21.5**
bomb-5-1	564	1086	2745	4639	501	1194	352	896	**29.7**	**25.0**
bomb-5-5	1340	2610	6987	12233	1433	4290	984	3392	**31.3**	**20.9**
bomb-10-5	4680	9220	15324	28679	3273	12435	2224	10337	**32.1**	**16.9**
bomb-10-10	2760	5415	26730	48736	5863	27080	3964	23282	**32.4**	**14.0**
bomb-20-05	7100	14025	41469	76308	9203	50100	6204	44102	**32.6**	**12.0**
comm-5-2	781	2289	28292	143118	48978	108252	33346	73221	**31.9**	**32.4**
comm-6-3	1275	4032	89904	502804	132007	295113	69415	156382	**47.4**	**47.0**
emptyroom-4	188	584	731	2017	147	292	143	284	2.7	2.7
emptyroom-8	396	1292	11069	126398	8661	17432	8513	17136	1.7	1.7
safe-5	86	171	433	1061	73	163	35	87	**52.1**	**46.6**
safe-10	166	356	869	2927	191	420	40	99	**79.1**	**76.4**
safe-30	486	1346	2530	11262	451	1048	100	259	**77.8**	**75.3**

Table 3. Results for instances with horizon $N = 5$. Reduction gains above 10% are shown in boldface.

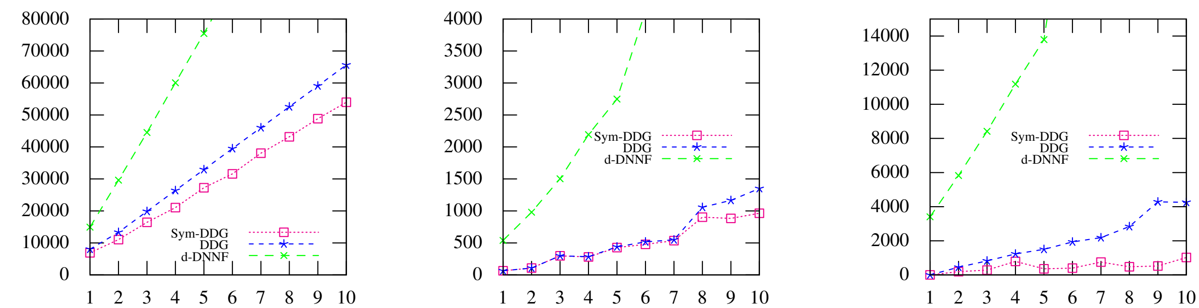


Figure 1. Results for Bomb-10-10 (left), Emptyroom-4 (middle), and Safe-30 (right) with varying horizon length.

piled representations, we can observe that both DDG and Sym-DDG, targeted by our compiler, are competitive with respect to d-DNNF, compiled using c2d. Furthermore, the experiments revealed that for many instances, a significant reduction of size in resulting diagrams is achieved by exploiting symmetries.

6 CONCLUSION

In this paper, we have shown how symmetries can be exploited for achieving significant space savings in a knowledge compilation perspective. We introduced two new languages $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ which generalize respectively the languages FBDD and DDG of decision diagrams. We have analyzed $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ both from a theoretical standpoint (following the lines of the knowledge compilation map) and from a practical standpoint (by compiling some planning benchmarks into them). The obtained results show $\texttt{Sym-FBDD}_{X,Y}$ and $\texttt{Sym-DDG}_{X,Y}$ as attractive; indeed, both languages offer sufficiently many queries and transformations for enabling efficient on-line reasoning for a number applications; furthermore, they achieve a high level of succinctness.

This work calls for many perspectives. One of them consists in taking advantage of complete methods for detecting symmetries, such as nauty [10, 1] and saucy [2]. While such methods are likely to lead to much longer off-line compilation times than our incomplete findSymmetry procedure, they are susceptible to explore the full symmetry group Σ, hence to provide smaller representations. Another perspective consists in studying the connections between $\texttt{Sym-DDG}_{X,Y}$ and the language of first-order NNF circuits.

REFERENCES

[1] F. Aloul, K. Sakallah, and I. Markov, 'Efficient symmetry breaking for boolean satisfiability', in *Proc. of IJCAI'03*, pp. 271–276, (2003).

[2] P. Darga, M. Liffiton, K. Sakallah, and I. Markov, 'Exploiting structure in symmetry detection for CNF', in *Proc. of DAC'04*, pp. 530–534, (2004).

[3] A. Darwiche, 'Decomposable negation normal form', *Journal of the ACM*, **48**(4), 608–647, (2001).

[4] A. Darwiche and P. Marquis, 'A knowledge compilation map', *Journal of Artificial Intelligence Research*, **17**, 229–264, (2002).

[5] H. Fargier and P. Marquis, 'On the use of partially ordered decision graphs in knowledge compilation and quantified Boolean formulae', in *Proc. of AAAI'06*, pp. 42–47, (2006).

[6] M.R. Garey and D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, 1979.

[7] J. Gergov and C. Meinel, 'Efficient analysis and manipulation of OBDDs can be extended to FBDDs', *IEEE Transactions on Computers*, **43**(10), 1197–1209, (1994).

[8] A. Haken, 'The intractability of resolution', *Theoretical Computer Science*, **39**, 297–308, (1985).

[9] B. Krishnamurthy, 'Short proofs for tricky formulas', *Acta Informatica*, **22**, 253–275, (1985).

[10] B. D. McKay, 'Practical graph isomorphism', *Congressus Numerantium*, **30**, 45–57, (1981).

[11] H. Palacios, B. Bonet, A. Darwiche, and H. Geffner, 'Pruning conformant plans by counting models on compiled d-DNNF representations', in *Proc. of ICAPS'05*, pp. 141–150, (2005).

[12] T. Sang, P. Beame, and H. Kautz, 'Heuristics for fast exact model counting', in *Proc. of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pp. 226–240, (2005).

[13] B. Selman and H.A. Kautz, 'Knowledge compilation and theory approximation', *Journal of the ACM*, **43**, 193–224, (1996).

[14] A. Urquhart, 'The symmetry rule in propositional logic', *Discrete Applied Mathematics*, **96/97**, 177–193, (1999).