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 Sym-FBDD$X,Y$ and 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 Sym-FBDD$X,Y$ and 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 Sym-FBDD$X,Y$ and 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 Sym-FBDD$X,Y$ and Sym-DDG$X,Y$ unless the polynomial hierarchy collapses. The price to be paid is that only restricted forms of conditioning and a restricted form of forgetting are offered by Sym-FBDD$X,Y$ and Sym-DDG$X,Y$. Nevertheless, this proves sufficient for a number of applications, including configuration and planning. We describe a compiler targeting Sym-FBDD$X,Y$ and 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 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 ∧-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 Sym-FBDD$X,Y$ and 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 Sym-FBDD$X,Y$ and Sym-DDG$X,Y$. Based on these investigations, it turns out that each of Sym-FBDD$X,Y$ and 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 Sym-FBDD$X,Y$ and 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 Sym-FBDD$X,Y$ and 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 Sym-FBDD$X,Y$ and 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 Sym-FBDD$X,Y$ and Sym-DDG$X,Y$ are defined and analyzed. A CNF-to-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 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 $\sigma$ over $L_{PS}$, the set of all literals over $PS$, is a bijective mapping from $L_{PS} = PS \cup \{\sim x \mid x \in PS\}$ to $L_{PS}$. Any permutation $\sigma$ 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 $\sigma$ under consideration in this paper is assumed to satisfy the following stability condition: for any pair of literals $\ell_1,\ell_2$, $\sigma(\ell_1) = \ell_2$ iff $\sigma(\sim \ell_1) = \sim \ell_2$ where $\sim \ell$ is the opposite of $\ell$, i.e., $\sim x = x$ and $\sim \sim x = x$. Any permutation $\sigma$ 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
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 \(\sigma\).

For instance, if \(PS = \{x_1, \ldots, x_6\}\), \((x_1 \ \bar{x}_3 \ x_4)(x_5 x_6)\) denotes the permutation \(\sigma\) associating \(x_1\) with \(\bar{x}_3\), \(\sim x_1\) with \(x_3\), \(x_3\) with \(\bar{x}_1\), \(x_4\) with \(x_1\), \(\bar{x}_4\) with \(x_5\), \(x_5\) with \(x_6\), \(\bar{x}_5\) with \(x_6\), \(x_6\) with \(\bar{x}_5\), and \(\bar{x}_2\) and \(x_2\) are left unchanged by \(\sigma\). The identity permutation is represented by the empty word using the simplified cycle notation.

By \(\Sigma\), we denote the set of all bijective mappings from \(L_{PS}\) to \(L_{PS}\) satisfying the stability condition. Clearly enough, \(\Sigma\) is closed by composition: if \(\sigma_1, \sigma_2 \in \Sigma\) then \(\sigma_1 \circ \sigma_2 \in \Sigma\). Since \(\Sigma\) 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), \(\Sigma\) is a permutation group. Clearly enough, applying a permutation \(\sigma \in \Sigma\) to a propositional formula \(\alpha\) does not change the number of models of the latter; especially, \(\alpha\) is satisfiable (resp. valid) if \(\sigma(\alpha)\) is satisfiable (resp. valid).

In the rest of the paper, we focus on subsets of \(\text{Sym-EDD}\), the language of symmetry-driven extended decision diagrams, where permutations are defined over \(\Sigma\). Basically, \(\text{Sym-EDD}\) generalizes the language of "extended" decision diagrams (i.e., binary decision diagrams in which \(\lambda\)-nodes are allowed) by associating some permutations to the arcs and to the root node. Diagrams from \(\text{Sym-EDD}\) 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:

\[
\begin{array}{c}
N \\
N_1 \sigma_1 \to \sigma_2 \\
N_2
\end{array}
\]

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

**Definition 1 (Sym-EDD).** \(\text{Sym-EDD}\) is the set of all finite, single-rooted multi-DAGs\(^2\) (also referred to as \"formulæ\") \(\alpha\) where:

- each leaf node of \(\alpha\) is either the \(T\)-node (a node labeled by the Boolean constant \(T\) — always true) or the \(\bot\)-node (a node labeled by the Boolean constant \(\bot\) — always false);
- each internal node of \(\alpha\) is labeled by \(\land\) and has a finite number of children (\(\geq 1\)), or it is a decision node labeled with a variable from \(PS\);
- each arc of \(\alpha\) is labeled with a permutation from \(\Sigma\);
- the root of \(\alpha\) is labeled with a permutation from \(\Sigma\).

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

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

Clearly enough, \(\text{Var}(\alpha)\) can be computed in time linear in \(|\alpha|\). Note that \(\text{Var}(\alpha)\) may easily differ from the set of variables occurring in \(\alpha\), 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 \(\text{Sym-EDD}\) formulæ. A simple way to do so consists in associating with every \(\text{Sym-EDD}\) formulæ \(\alpha\) a tree-shaped NNF formulæ \(T(\alpha)\) which is logically equivalent to \(\alpha\). Formally, \(T(\alpha)\) is given by \(\sigma_\alpha(T(N))\) where \(N\) is the root of \(\alpha\) and \(T(N)\) is defined inductively as follows:

- if \(N\) is a leaf node labeled by the Boolean constant \(T\) (resp. \(\bot\)), then \(T(N) = T\) (resp. \(\bot\));
- if \(N\) is a node labeled by \(\land\) and having \(k\) children \(N_1, \ldots, N_k\) such that \(\forall i \in 1, \ldots, k, \sigma_i\) is the label of the arc \((N_i, N)\), 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 \(\forall i \in 1, \ldots, 2, \sigma_i\) is the label of the arc \((N, N_i)\), then \(T(N) = (\sim x \land \sigma_1(T(N_1))) \lor (x \land \sigma_2(T(N_2)))\).

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

For space reasons, we assume the reader is familiar with the languages \(\text{BBDD}, \text{DDG}, \text{DNNF}, \text{CD}, \text{CDT}, \text{CT}\), and \(\text{FT}\). The basic queries considered in the \(\text{KC}\) map include tests for consistency \(\text{CO}\), validity \(\text{VA}\), implicates (clausal entailment) \(\text{CE}\), implicants \(\text{IM}\), sentential entailment \(\text{SE}\), model counting \(\text{CT}\), and model enumeration \(\text{ME}\). We add to them the model checking query \(\text{MC}\), which is not obvious for the languages we introduce in the paper. The basic transformations include conditioning (\(\text{CD}\)), (possibly bounded) closures under the connectives (\(\land, \lor, \neg\)), and forgetting (\(\text{FO}\)), or dually projection (\(\text{PR}\)). We add to them the restricted conditioning transformation, and the restricted projection transformation:

**Definition 2 (X-RCD).** Let \(L\) be a subset of \(\text{Sym-EDD}\), and \(X \subseteq PS\). \(L\) satisfies \(\text{X-RCD}\) iff there is a polynomial-time algorithm that maps every formulæ \(\alpha\) in \(L\) and every consistent term \(\gamma\) over some variables in \(X\) to a formulæ \(\gamma\) in \(L\) which is logically equivalent to the general logical consequence \(\beta\) of \(\alpha\land\gamma\), where \(\beta\) is independent from the variables occurring in \(\gamma\).

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

3 Or, in an equivalent way, to the forgetting of every variable in \(\alpha\) but those of \(Z\).

\(^2\) More than one arc between two nodes is allowed.
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 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 \( \alpha \) is an \( x \)-decision node iff there exists a path \( a_1, \ldots, a_m \) from the root of \( \alpha \) to \( N \), such that the corresponding permutation \( \sigma_1 \circ \cdots \circ \sigma_m \) satisfies \( \sigma_0 \circ \cdots \circ \sigma_m(y) = x \). \( N \) is a \( X \)-decision node where \( X \subseteq PS \) if it is an \( x \)-decision node for some \( x \in X \).
- A Sym-EDD formula \( \alpha \) satisfies the read-once property iff for every variable \( x \in PS \), in every path from the root of \( \alpha \) 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 \( \sigma_i(Var(N)) \cap \sigma_j(Var(N)) = \emptyset \), where \( \sigma_i \) (resp. \( \sigma_j \)) is the permutation labeling the arc \((N, N_i)\) (resp. \((N, N_j)\)).
- A Sym-EDD formula \( \alpha \) 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 \( \alpha \) which are both read-once and decomposable.
- Sym-FBDD is the subset of Sym-DDG consisting of formulae \( \alpha \) containing no \( \wedge \)-node.

Unlike what happens in the FBDD case, a node of \( \alpha \) 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.

\[
\begin{align*}
(x_1 x_2) & \quad (x_1 x_3) \\
\wedge & N \\
\downarrow & x_1 \\
\top & x_2 \\
\end{align*}
\]

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 \( \alpha \) satisfies the symmetry-freeness condition on \( X \) iff for any \( x \in X \) and any \( x \)-decision node \( N \) in \( \alpha \), if \( N \) is an \( x \)-decision node and a \( y \)-decision node (\( y \in PS \)), then \( y = x \).

Here, \( \sigma_0 \) is the permutation labeling the root of \( \alpha \) and, for all \( i \in \{1, \ldots, m\}, \sigma_i \) is the permutation labeling \( a_i \).

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

Definition 9 (Sym-DDG_{X,Y}, Sym-FBDD_{X,Y}).

Given two subsets \( X, Y \) of \( PS \):

- Sym-DDG_{X,Y} is the subset of Sym-DDG containing diagrams satisfying the symmetry-freeness condition on \( X \) and the precedence condition on \( Y \).
- 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, it is assumed that \( X \) and \( Y \) are “fixed” subsets of \( PS \). In other words, for any formula \( \alpha \) in Sym-DDG_{X,Y}, the \( X \)-decision nodes of \( \alpha \) and the \( Y \)-decision nodes of \( \alpha \) 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 Sym-DDG_{X,Y} \subseteq Sym-DDG_{X',Y}. \) When \( Y = PS \), i.e., when no precedence condition actually constrains the diagrams, two extreme cases are reached with \( X = \emptyset \) and \( X = PS \), respectively. Sym-DDG_{X,PS} coincides with Sym-DDG, while Sym-DDG_{PS,PS} coincides with DDG (of course, similar equalities hold for Sym-FBDD, mutatis mutandis). By definition, each Sym-FBDD_{X,Y} (resp. Sym-DDG_{X,Y}) formula is a Sym-FBDD (resp. Sym-DDG) formula. Conversely, every Sym-FBDD (resp. Sym-DDG) formula \( \alpha \) can also be viewed as a Sym-FBDD_{X,Y} (resp. Sym-DDG_{X,Y}) formula for every \( X \subseteq Y \subseteq PS \) such that \( X \cap Var(\alpha) = \emptyset \) and \( Var(\alpha) \subseteq Y \).

In what follows, a family of propositional languages \( L_X,Y \) 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 Sym-FBDD_{X,Y} (resp. 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 \( \gamma \) ) 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 \( \bot \)-node otherwise). Things are a bit more tricky for MC and ME (polynomial-time algorithms for these queries are presented in the extended version).
Consider the CNF formula \( \Delta_n \) containing every 3-clause \( \delta \) (i.e., a clause of size 3) over \( Y = \{y_1, \ldots, y_n\} \) augmented by an additional literal \( x_5 \) which identifies the clause. \( \Delta_n \) is thus a CNF over \( Y \cup \bar{X} \) containing \( 8 \times \binom{n}{3} \) 4-clauses. \( X \) contains \( 8 \times \binom{n}{3} \) variables \( x_i \). The proof of Proposition 2 relies on the fact that \( \Delta_n \) can be represented by an equivalent Sym-FBDD formula of size linear in the size of \( \Delta_n \). 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 \( \gamma_\alpha = \bigwedge_{i=1}^m \neg x_i \) such that \( \alpha \) is satisfiable iff \( \Delta_n \land \gamma_\alpha \) is satisfiable iff \( \Delta_n \) conditioned by \( \gamma_\alpha \) is satisfiable. Suppose now that \( \Delta_n \) has a polynomial-space representation \( comp(\Delta_n) \) in a propositional language \( L \) over \( X \cup Y \), where \( L \) 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 \( L \)-representation of \( comp(\Delta_n) \) conditioned by \( \gamma_\alpha \), and to determine in polynomial time whether it is consistent or not. We would therefore get \( NP \subseteq \mathsf{P/poly} \), hence \( \Sigma_2^p = \Pi_2^p \) (see e.g. [113] for details).

As a consequence, state-of-the-art languages for knowledge compilation, like DNF, are not more succinct than any of the two languages Sym-FBDD, and Sym-DDG. Especially, due to the obvious inclusions \( \text{Sym-DDG} \supseteq \text{DDG} \), and \( \text{Sym-FBDD} \supseteq \text{FBDD} \), we have that \( \text{Sym-DDG} \subseteq \text{DDG} \), and \( \text{Sym-FBDD} \subseteq \text{FBDD} \). This implies that:

**Proposition 3.** Sym-DDG \( \not< \) DDG and Sym-FBDD \( \not< \) FBDD.

### Table 1. Sym-DDG\(_{X,Y}\), Sym-FBDD\(_{X,Y}\), and the transformations CO, VA, CE, IM, SE, CT, ME, MC.

<table>
<thead>
<tr>
<th>Transformation</th>
<th>CO</th>
<th>VA</th>
<th>CE</th>
<th>IM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-DDG(_{X,Y})</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sym-FBDD(_{X,Y})</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DDG</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>FBDD</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-DDG(_{X,Y})</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>0</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Sym-FBDD(_{X,Y})</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DDG</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>FBDD</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

In a nutshell, it turns out that Sym-FBDD\(_{X,Y}\) and Sym-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 \( \alpha \) is any Sym-EDD formula, the Sym-EDD formula rooted at node \( N_3 \) on the figure below is equivalent to \( \exists x. \alpha \) while the formula rooted at node \( N_7 \) 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 \( \text{PSPACE}\)-hard!\(^{5}\)

The non-standard behavior of Sym-DDG\(_{X,Y}\) (and its subset Sym-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 Sym-DDG\(_{X,Y}\) and Sym-FBDD\(_{X,Y}\), are in some sense "very succinct":

**Proposition 2.** Let \( X, Y \) be two subsets of \( \mathbb{P} \). No propositional language \( L \) over \( X \cup Y \) satisfying CD and CO is at least as succinct as Sym-FBDD\(_{X,Y} \) unless \( \Sigma_2^p = \Pi_2^p \), i.e., we have \( L \not< \) Sym-FBDD\(_{X,Y} \).\(^{6}\)

\(^{5}\) Observe that, due to the read-once condition, none of these two formulae belongs to Sym-DDG, unless \( x \not\in \text{Var}(\alpha) \).

\(^{6}\) Removing lines 3 and 4 in the pseudo-code of symddg\(_{X,Y}\) leads to downsize it as a CNF-to-Sym-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 Sym-DDG\(_{X,Y}\) formula is generated, it is added to the cache when it is not already in it.
Algorithm 1: symddg\(_{X,Y}(\Delta)\)

input : a CNF formula \(\Delta\), a set \(X\) of symmetry-free variables, and a set \(Y\) of top variables

output: a Sym-DDG\(_{X,Y}\) formula equivalent to \(\Delta\)

1. if \(\Delta\) is empty then return leaf(T);
2. if \(\Delta\) contains an empty clause then return leaf(\(\perp\));
3. let \(\Delta_1, \ldots, \Delta_n\) be the connected components of \(\Delta\);
4. if \(k > 1\) then return 
   \(\text{ano}de(\text{symddg}_{X,Y}(\Delta_1), \ldots, \text{symddg}_{X,Y}(\Delta_k))\);
5. if \((\sigma \leftarrow \text{findSymmetry}(\Delta))\) such that
   \((\text{key} \leftarrow \text{inCache}(\sigma(\Delta)))\) \(\neq \text{nil}\) then return cache(key);
6. choose a variable \(x\) of \(\Delta\):
   return dnode\(x, \text{symddg}_{X,Y}(\Delta_{|x\leftarrow 0}), \text{symddg}_{X,Y}(\Delta_{|x\leftarrow 1})\)

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 symddg\(_{X,Y}\) compiler lies in an efficient implementation of the findSymmetry method for retrieving a formula \(\Delta'\) in the cache that is equivalent to the input formula \(\Delta\), modulo an admissible symmetry \(\sigma\). To this point, the problem of determining whether there exists a symmetry between two CNF formulae \(\Delta\) and \(\Delta'\) 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, 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 \(\Delta\) is given by two sorted vectors, which respectively encode the signature of the variables occurring in \(\Delta\) and the signature of the clauses occurring in \(\Delta\). 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 \(\Delta\). The signature of a clause \(\delta\) 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 \(\Delta\) and \(\Delta'\) 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 \(\Delta\) and \(\Delta'\) is cast as a constraint satisfaction problem (CSP). The set of variables of the CSP is given by the collection of variables occurring in \(\Delta\), and the signature of the clauses occurring in \(\Delta\). The signature of a variable \(x\) is formed by the set of all literals \(\ell\) occurring in \(\Delta'\), such that \(x\) and \(\ell\) have the same signature. Finally, a binary constraint \(x \neq y\) is associated with each pair of variables \(x, y\) occurring in the same clause \(\delta\) of \(\Delta\). Based on this representation, the space of candidate permutations between \(\Delta\) and \(\Delta'\) 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 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 \(\pi\) for \(P\) is a sequence \(\pi\) 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 \(\Delta_P\) over the set of variables \(PS = (\bigcup_{i=0}^{N} \{f_0 \mid f \in F\}) \cup \{a_1 \mid a \in O, i = 0, \ldots, N - 1\}\). \(\Delta_P\) 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 \(\Delta_P\) as soon as the truth values of the variables \(\{f_0 \mid f \in F\} \cup \{a_{i-1} \mid a \in A\}\) are fixed (i.e., as soon as the initial state and the plan under consideration are specified).

Based on this encoding, the formula \(\Delta_P\) is compiled into a 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 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 Sym-DDG\(_{X,Y}\) satisfies both X-RCD and CO, we can determine in polynomial time whether a plan \(\pi\) exists for any \(I\) and \(G\) given on-line; since Sym-DDG\(_{X,Y}\) satisfies CT, we can also count how many \(\pi\) 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 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 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 online queries and transformations is governed by the size of the comp-

---

8 In an obvious way, the signature of \(f\) is \((p_x, n_x)\) if \(\ell\) is the positive literal \(x\), and \((n_x, p_x)\) if \(\ell\) is the negative literal \(-x\).

9 c2d available at reasoning.cs.ucla.edu/c2d, was run using the command c2d -in file.cnf -dt_count 50 -smooth_all
Another perspective consists in studying the connections between \( \Sigma \) full symmetry group findSymmetry complete to lead to much longer off-line compilation times than our incom-
ted results show Sym-FBDD and Sym-DDG 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 ob-
tained results show Sym-FBDD and Sym-DDG as attractive; indeed, both languages offer sufficiently many queries and transfor-
mations for enabling efficient on-line reasoning for a number appli-
cations; 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 incom-
plete findSymmetry procedure, they are susceptible to explore the full symmetry group \( \Sigma \) hence to provide smaller representations. Another perspective consists in studying the connections between Sym-DDG and the language of first-order NNF circuits.

### 6 CONCLUSION

In this paper, we have shown how symmetries can be exploited for achieving significant space savings in a knowledge compilation per-
pective. We introduced two new languages Sym-FBDD and Sym-DDG which generalize respectively the languages FBDD and DDG of decision diagrams. We have analyzed Sym-FBDD and Sym-DDG 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 Sym-FBDD and Sym-DDG as attractive; indeed, both languages offer sufficiently many queries and transfor-
mations for enabling efficient on-line reasoning for a number appli-
cations; 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 incom-
plete findSymmetry procedure, they are susceptible to explore the full symmetry group \( \Sigma \) hence to provide smaller representations. Another perspective consists in studying the connections between Sym-DDG and the language of first-order NNF circuits.

### REFERENCES