On the Use of Partially Ordered Decision Graphs for Knowledge Compilation and Quantified Boolean Formulae*

Hélène Fargier
IRIT-CNRS, Toulouse
email: fargier@irit.fr

Pierre Marquis
CRIL-CNRS, Université d'Artois, Lens
email: marquis@cril.univ-artois.fr

Abstract

Decomposable Negation Normal Form formulae (DNNFs) form an interesting propositional fragment, both for efficiency and succinctness reasons. A famous subclass of the *DNNF* fragment is the *OBDD* fragment which offers many polytime queries and transformations, including quantifier eliminations (under some ordering restrictions). Nevertheless, the decomposable AND nodes at work in OBDDs enable only sequential decisions: clusters of variables are never assigned "in parallel" like in full DNNFs. This is an serious drawback since succinctness for the full *DNNF* fragment relies on such a "parallelization property". This is why we suggest to go a step further, from (sequentially) ordered decision diagrams to (partially) ordered, decomposable decision graphs, in which any decomposable AND node is allowed, and not only assignment ones. We show that, like the *OBDD* fragment, such a new class offers many tractable queries and transformations, including quantifier eliminations under some ordering restrictions. Furthermore, we show that this class is strictly more succinct than *OBDD*.

Introduction

Knowledge compilation is considered in many AI applications where short on-line response times are expected. It consists in turning (during an off-line phase) the initial data into a form that ensures the tractability of the requested queries and transformations. This principle is for instance used in many state-of-the-art approaches to product configuration where the set of possible products is compiled (Faltings & Weigel 1999)(Pargamin 2002)(Amilhastre, Fargier, & Marquis 2002).

The class of decomposable negation normal form formulae (DNNFs, (Darwiche 2001)) is a propositional fragment which is considered as an interesting target for knowledge compilation since it offers many tractable queries while being very succinct. A famous subclass of *DNNF* is the *OBDD* fragment, which offers more tractable queries and transformations, especially quantifier eliminations (under some ordering restrictions) (Coste-Marquis *et al.* 2005). The price to be paid concerns the succinctness issue: the *OBDD* fragment is strictly less succinct than the *DNNF* one, which means that some propositional formulae have *OBDD* representations exponentially larger than their *DNNF* representations.

OBDDs are Boolean decision diagrams in which decision nodes are sequentially ordered. This is the very property that allows the class to be polynomially closed for quantifier eliminations under some ordering restrictions. Other structures exist that are able to take advantage of the decomposability property, like DNNFs do, and that share with OBDDs a property of ordering. Let us mention cluster trees (Pargamin 2002), BDD trees (McMillan 1994) tree-driven automata (Fargier & Vilarem 2004), synthesis trees (Faltings & Weigel 1999) and AND/OR search graphs (Dechter 2004). Notice that the first two fragments are propositional ones while the last ones allow non Boolean domains. Note also that, in these five languages, decision variables are not linearly ordered.

The present paper aims at defining and investigating (partially) ordered, decomposable graphs in the NNF framework. The basic motivation behind this work was to define a proper superset of $OBDD$, in which decomposability is exploited in a less restricted way. In the following, we show that like *OBDD*, the class of such (partially) ordered, decomposable graphs enable many polytime queries and transformations, including quantifier eliminations under some restrictions, while being strictly more compact than *OBDD*. For space reasons, only some of the proofs are sketched.

DNNFs and OBDDs

Let X be a finite set of propositional variables. We denote by $\phi_{[x_i \leftarrow \psi]}$ the formula obtained by replacing in the propositional formula ϕ all the occurrences of the propositional variable x_i by the propositional formula ψ and call it the conditioning of ϕ by $x_i \leftarrow \psi$. In the following, conditioning is used under the form $\phi_{[x_i \leftarrow true]}$ and $\phi_{[x_i \leftarrow false]}$ (this corresponds to the instantiation of x_i to a truth value).

NNFs A formula in NNF is a rooted, finite directed acyclic graph (DAG) where each leaf node is labelled by

*We would like to thank the anonymous reviewers for many helpful comments. Pierre Marquis has been partly supported by the IUT de Lens, the Région Nord/Pas-de-Calais through the IRCICA Consortium, and the European Community FEDER program.

$true$, $false$, $\neg x$, or x with $x \in X$. Each internal node is labelled by $\wedge$ (*AND node*) or $\vee$ (*OR node*) and can have many children.

In NNFs, a leaf is thus either a constant or a literal – in the latter case, it represents the assignment of a variable.

Assignment An assignment node N is an AND node of the form $x \wedge \alpha$ (resp. of the form $\neg x \wedge \alpha$).

One can assume without loss of generality that α is not of the form y or $\neg y$ with $y \in X$ (if α is a literal, it can always be replaced by the assignment node $\alpha \wedge true$). With this proviso, the notations $var(N) = x$ and $tail(N) = \alpha$ are not ambiguous.

The term decision node is dedicated to OR nodes which give an alternative between possible assignments of the same variable (case analysis):

Decision node A decision node N is an OR node of the form $(\neg x \wedge \alpha) \vee (x \wedge \beta)$.[1] $var(N)$ denotes the variable x.

A binary decision diagram (BDD) is an NNF formula in which all the AND nodes are assignment nodes and all the OR nodes are decision nodes. The success of BDDs is due to two successive subclasses, free BDDs and ordered BDDs, which are BDDs respectively satisfying the read-once property and the ordering property w.r.t. a total order — the latter implying the former:

Read-once property An NNF formula ϕ is read-once iff for any assignment node N in ϕ, $var(N)$ does not occur in $tail(N)$.

Ordering Let $<$ be a strict order on X. An NNF formula ϕ is ordered w.r.t. $<$ iff for every pair of decision nodes M and N in ϕ, if M is an ancestor of N in ϕ, then $var(M) < var(N)$.[2]

Let us introduce some notations. Let BDD be the subset of NNF of formulae that are BDDs, $FBDD$ the subset of BDD formulae that are read-once, $OBDD_<$ the subset of $FBDD$ of formulae being ordered w.r.t. a *total* order $<$ and $OBDD$ the union of all $OBDD_<$ (for every $<$).

The success of the *OBDD* class is due to the fact it offers many tractable queries, especially consistency (CO), validity (VA), clausal entailment (CE), implicant check (IM), model counting (CT), model enumeration (ME). Moreover, equivalence (EQ) and entailment (SE) are tractable for $OBDD_<$: deciding whether two formulae from $OBDD_<$ are logically equivalent is in P, as well as deciding whether an element of $OBDD_<$ entails another one. This latter class is also polynomially closed for (SFO) (single variable forgetting), ($\wedge$ BC) (bounded conjunction) and ($\vee$ BC) (bounded disjunction) transformations and linearly closed for the conditioning transformation. This just means that computing a formula from the class which is equivalent to the result of the transformation given some formula(e) from the class can be achieved in polynomial (resp. linear) time.

The tractability of (EQ) for $OBDD_<$ is due to the fact that, once a total order $<$ is fixed, two formulae from $OBDD_<$ are equivalent whenever their reduced forms are identical, and the reduction transformation can be achieved in polynomial time.

(Darwiche & Marquis 2001) have shown that the tractability of $OBDD$ (and more generally of $FBDD$) for many queries is due to two general properties any $FBDD$ satisfies: decomposability and determinism.

Decomposability An NNF formula ϕ is decomposable iff for each AND node N in ϕ, the conjuncts of N do not share any variable.

Determinism An NNF formula ϕ is deterministic iff for each OR node N in ϕ, the disjuncts of N are pairwise logically inconsistent.

$DNNF$ (resp. $d-NNF$) denotes the subset of NNF of formulae satisfying decomposability (resp. determinism). $d-DNNF$ denotes the subset of NNF where both properties are satisfied.

(CO), (CE) and (ME) (resp. (VA), (IM), (CT)) have been proved to be tractable for $DNNF$ (resp. $d-DNNF$), and $OBDD$ is obviously a subclass of $d-DNNF$. Decision diagrams in their more general definition (i.e. BDDs) are not decomposable, since nothing forbids in the expression of $(x \wedge \alpha) \vee (\neg x \wedge \beta)$ that x occurs in α or β. Once the read-once property has been added, i.e. when one moves from BDD to $FBDD$ or $OBDD$, decomposability is recovered.

Observe that the decomposability property is extremely simple in $OBDD$ and $FBDD$: for formulae from those classes, an AND node has necessarily two children, a literal, and a decision node; so, decomposability implies that Boolean variables are necessarily assigned sequentially. Clusters of variables are never assigned "in parallel" (i.e. independently from each other), like in full DNNFs. This is an serious drawback, since the succinctness of $DNNF$ relies on such a "parallelization property".

Decision Graphs

In order to avoid this restriction, we need to relax one part of the basic requirement on decision diagrams, i.e. the fact that the tail of an assignment node is either a decision node or a constant. In the following *decomposable decision graphs*, the OR nodes will still be required to be decision nodes, but AND nodes can be of any form, provided that they are decomposable. For instance, a decomposable AND node $N \wedge M$, where N and M are decision nodes, can be the tail of an assignment node. Technically, this amounts to using a less demanding property than the one considered for BDDs.

Definition 1 (Simple decision) *An NNF formula ϕ satisfies the* simple decision *property iff for every node N in ϕ, if N is an OR node, then it is a decision node.*

Definition 2 (DGs and DDGs)
DG is the subset of NNF formulae satisfying the simple decision property.

[1] Such a node is sometimes noted by $ite(x, \alpha, \beta)$.

[2] Note that such ordering property is more general than the one considered in (Darwiche & Marquis 2001) since any NNF formula and any strict order (not necessarily total ones) are considered.

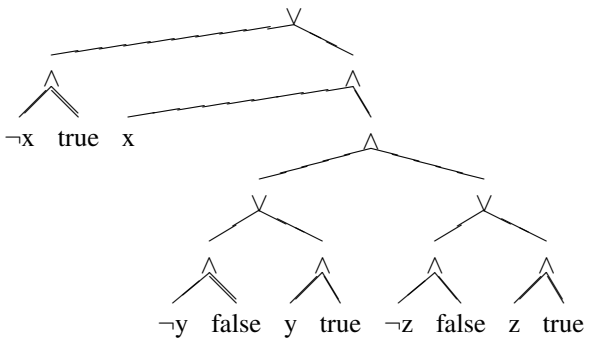

Figure 1: A DDG representing $x \Rightarrow (y \wedge z)$.

DDG is the subset of DG formulae satisfying decomposability.

DDGs combine the advantages of FBDDs and the "parallelization property" of DNNFs. Non decomposable decision graphs exist but do not have a great interest. An example of a DDG representing the formula $x \Rightarrow (y \wedge z)$ is given in Figure 1.

Clearly enough, BDDs are decisions graphs and FBDDs are decomposable decision graphs. To be more precise, we have:

Proposition 1
$BDD \subset DG$.
$OBDD \subset FBDD \subset DDG \subset d-DNNF$.

From (Darwiche & Marquis 2001), we immediately get:

Corollary 1 *(CO), (CE), (ME), (VA), (IM) and (CT) are tractable for DDG.*

Now, the previous definition of ordering applies to decision graphs:

Definition 3 (Ordered DDGs) *Let $<$ be a strict order on X (not necessarily a total one). $O-DDG_<$ is the set of all the DDG formulae ordered w.r.t. $<$. $O-DDG$ is the union of all $O-DDG_<$ (for every $<$).*

Unlike the $OBDD_<$ case, the definition does no longer imply that if $y < z$, then a given decision path either does not make any decision on y or makes a decision on z after the decision on y. Variables y and z can be assigned in parallel branches. This is because of the possible presence of some decomposable AND nodes.

Enabling partial orders $<$ in $O-DDG_<$ is a way to achieve more compact representations thanks to conditional independence (e.g. in the formula $x \Rightarrow (y \wedge z)$, y is independent from z given x and this is exploited in the representation depicted in Figure 1). The price to be paid is that no fragment $O-DDG_<$ is complete (i.e. enables the representation of every propositional formula), whenever $<$ is not total (just consider the formula $x \Leftrightarrow (y \Leftrightarrow z)$ in which no variable is independent from another one given the remaining variable: such a formula cannot be represented in $O-DDG_<$ when $<$ is not total).

Observe that $O-DDG$ and $FBDD$ cannot be compared w.r.t. set inclusion. For instance, $x \wedge y$ is an $O-DDG_<$ formula whatever $<$, but not an $FBDD$ formula. On the other hand, the $FBDD$ formula

$$ite(x, ite(y, ite(z, 0, 1), 1), ite(z, 0, ite(y, 0, 1)))$$

is not an $O-DDG$ formula.

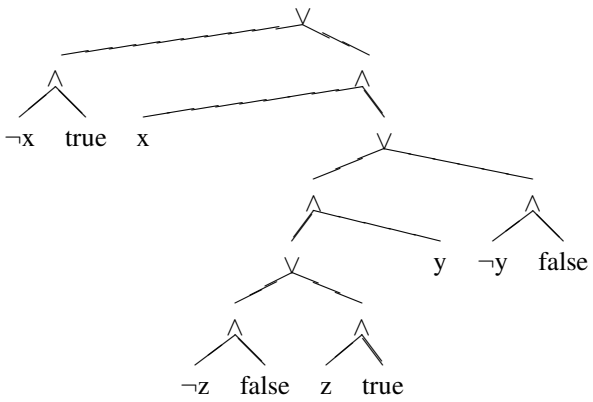

Figure 2: A second DDG (actually, an $OBDD_<$ formula) ordered w.r.t. $x < y < z$ and representing $x \Rightarrow (y \wedge z)$.

Observe also that, unlike $OBDD_<$, the canonicity of an $O-DDG_<$ formula is not guaranteed any longer since the way in which AND nodes partition the next variables is not constrained. For instance, the order $x < y < z$ allows several representations of the formula $x \Rightarrow (y \wedge z)$ (see Figures 1 and 2). As a consequence, the definition is not strong enough to ensure the tractability of (EQ).

In order to deal with this problem, we introduce a more restricted fragment. We first need to constrain further the admissible orders:

Definition 4 (Tree orders) *A strict order $<$ on X is a* tree order *iff $(X, <_{min})$ is a tree where $<_{min}$ is the smallest binary relation on X w.r.t. $\subseteq$ such that the transitive closure of $<_{min}$ is $<$.*

Obviously enough, every total order on X is a tree order.

We also need a stronger notion of ordering:

Definition 5 (Strong ordering) *Let $<$ be a tree order. A NNF formula ϕ is* strongly ordered *w.r.t. $<$ iff it is ordered w.r.t. $<$ and for every pair of variables y and z from X, if $z < y$, then for every decision node N in ϕ s.t. $var(N) = y$, there exists a decision node M in ϕ s.t. $var(M) = z$ and M is an ancestor of N.*

Definition 6 (Strongly ordered DDGs) *Let $<$ be a tree order. $SO-DDG_<$ is the set of all strongly ordered DDGs w.r.t. $<$. $SO-DDG$ is the union of all $SO-DDG_<$ (for every tree order $<$).*

Let us step back to the previous example as a matter of illustration. The DDG of Figure 1 is not strongly ordered w.r.t. $x < y < z$ but it is strongly ordered w.r.t. the tree order $x < y$ and $x < z$.

Ordered fragments can be compared as follows:

Proposition 2
$OBDD \subset SO-DDG \subset O-DDG$.
For each total order $<$ on X, $OBDD_< = SO-DDG_<$.

The strong ordering requirement being sufficient to fully specify both the sequencing of decision nodes and the way AND nodes separate the variables, it guarantees the existence of a unique reduced form. The reduction algorithm is roughly the same as the one applying to OBDDs (see e.g. (Sieling & Wegener 1993)). We have:

Proposition 3 *(EQ) is tractable for $SO-DDG_<$.*

It can also be shown that $SO-DDG_<$ is polynomially closed for variable forgetting (but it is not known to be polynomially closed for negation). So, going from $OBDD_<$ to $SO-DDG_<$, one does not lose much in efficiency. Interestingly, one can gain a lot in succinctness, as shown by the following theorem.

Theorem 1 *$SO-DDG$ (and thus $O-DDG$) is strictly more compact than $OBDD$.*

Sketch of proof: It can be shown that any $OBDD_<$ formula can be represented by a polyspace CSP automaton in the sense of (Vempaty 1992) and reciprocally. The two classes are equivalent in terms of spatial compactness. In (Fargier & Vilarem 2004) a tree-structured CSP is presented that cannot be represented by any polyspace CSP automaton. On the other hand, any tree-structured CSP can be represented by a polyspace $SO-DDG_<$ formula (with a log representation of its domains). So, there exists a formula whose representation is of polynomial size in $SO-DDG$ but not in $OBDD$. □

Quantifier Elimination

We now focus on two further transformations (quantifier eliminations) which prove useful for solving the validity problem for quantified boolean formulae, and can be achieved efficiently when quantified $O-DDG_<$ formulae are considered.

Quantified Boolean Formulae

A quantified Boolean formula (QBF) Φ is of the form $q_1x_1q_2x_2\ldots q_nx_n\phi$ where ϕ is a usual propositional formula built up from X and called the matrix of Φ.[3] The prefix $q_1x_1q_2x_2\ldots q_nx_n$ consists of universal $\forall$ and existential $\exists$ quantifiers and propositional variables x_i of X. In the following, we consider only polite and closed formulae, i.e. we assume that two distinct quantifiers do not bear on the same variable and that all the variables in ϕ are quantified. The strict order induced by the prefix $q_1x_1q_2x_2\ldots q_nx_n$ of such a formula is $x_1<\ldots<x_n$.

The semantics of a closed QBF formula Φ is a truth value, recursively defined as follows:

- if Φ does not contain variables (i.e. consists of connectives and the constants *true* and *false*), its truth is defined by the truth table for the connectives.
- A formula $\Phi=\exists x\Psi$ is true iff $\Psi_{[x\leftarrow true]}$ is true or $\Psi_{[x\leftarrow false]}$ is true.
- A formula $\Phi=\forall x\Psi$ is true iff $\Psi_{[x\leftarrow true]}$ is true and $\Psi_{[x\leftarrow false]}$ is true.

[3] In this paper, uppercase Greek letters $\Phi,\Psi,\ldots$ will denote QBFs, and lowercase letters $\phi,\psi,\ldots$ propositional formulae.

The decision problem QBF "Given a (prenex, closed, polite) QBF Φ, is Φ true ?" is the canonical PSPACE-complete problem. As such, many decision problems considered in AI (e.g. deciding inference relations – like circumscription or some paraconsistent inference relations, see (Egly *et al.* 2000) – or determining whether a plan exist for several kinds of plans, see (Rintanen 1999)) can be reduced to it.

The restriction of QBF to instances whose matrices are from $OBDD_<$ where $<$ is the total order induced by their prefixes has been shown tractable (Coste-Marquis *et al.* 2005). In the following, we show that this still holds for ordered DDGs.

Last Variable Elimination in QBFs

Complete algorithms for QBF are based on two different paradigms: branching, that is the key point of backtrack algorithms, and quantifier elimination. Roughly, branching methods act on outermost quantifiers first while variable elimination bear on the innermost ones first: they aim at transforming the formula Φ involving a variable x into another one that does not contain x but is equivalent to Φ.

Polytime quantifier elimination actually offers more than solving the decision problem QBF, but enables the polytime computation of solution policies for QBFs (roughly, one can compute in polynomial time the truth value that must be given to every existentially quantified variable y in Φ from the truth values given to all the universally quantified variables preceding y in the prefix, so as to make Φ true). See Proposition 3 in (Coste-Marquis *et al.* 2006).

When the variable to be eliminated is existentially quantified, this amounts to an operation of forgetting (sometimes also called "existential abstraction").

Definition 7 (Forgetting)
$forget(\phi,x)=\phi_{[x\leftarrow true]}\vee\phi_{[x\leftarrow false]}$.

Proposition 4 *Let Φ be a quantified Boolean formula of the form $q_1x_1\ldots q_{n-1}x_{n-1}\exists x_n\phi$. Φ is true iff $q_1x_1\ldots q_{n-1}x_{n-1}\ forget(\phi,x_n)$ is true.*

As such, the size of the formula $forget(\phi,x)$ is not significantly greater than the one of ϕ (about twice as big) but it can become huge after a repeated sequence of forgettings. Moreover, $forget(\phi,x)$ does not necessarily belong to the same class as ϕ. For instance, if ϕ is a CNF formula, $forget(\phi,x)$ is not. It should then be transformed into a CNF formula, replacing the n_1 clauses involving the literal x and the n_2 clauses involving the literal $\neg x$ by (at most) $n_1\times n_2$ clauses without x (obtained by resolution upon x). The CNF class is thus polynomially (but actually not linearly) closed for the forgetting operation of a single variable. A direct consequence is that forgetting all the variables one after the other can lead to an exponentially sized data structure. On the contrary, the DNF class as well as the Blake class (prime implicates formulae) and the $DNNF$ class are linearly closed for variable forgetting. So, one can say that these classes are tractable for variable forgetting while CNF is not (Darwiche & Marquis 2001).

It must be noticed that quantifier elimination is *not sound* in general for variables other than the most internal one: $\forall x_1\exists x_2\forall x_3\ \phi$ is equivalent to $\forall x_1\exists x_2\ \phi_{[x_3\leftarrow true]}\wedge$

$\phi_{[x_3 \leftarrow false]}$ but not to $\exists x_2 \forall x_3 \phi_{[x_1 \leftarrow true]} \wedge \phi_{[x_1 \leftarrow false]}$. The restrictions SAT and TAUT (the validity problem for propositional formulae) of QBF are very particular cases where any variable can be chosen for elimination (since all variables are quantified in the same way).

Obviously, Proposition 4 cannot be used would x_n be universally quantified. One cannot forget universally quantified variables but they must be "ensured" thanks to an operation dual to forgetting (this operation is sometimes called "universal abstraction").

Definition 8 (Ensuring)
$ensure(\phi, x) = \phi_{[x \leftarrow true]} \wedge \phi_{[x \leftarrow false]}$.

Proposition 5 *Let Φ be a quantified Boolean formula of the form $q_1 x_1 \ldots q_{n-1} x_{n-1} \forall x_n \phi$. Φ is true iff $q_1 x 1 \ldots q_{n-1} x_{n-1}$ $ensure(\phi, x_n)$ is true.*

CNF is linearly closed for variable ensuring: if ϕ is a conjunction of clauses, computing a formula equivalent to $ensure(\phi, x_i)$ just requires to shorten the (non valid) clauses of ϕ bearing on x_i. Most of the state-of-the-art QBF solvers dealing with *CNF* matrices take advantage of this property when eliminating the internal $\forall$ quantifiers. Contrastingly, neither *DNNF* nor *DNF* is polynomially closed for variable ensuring unless P = NP (see (Coste-Marquis *et al.* 2005)).

Last Variable Elimination in DDGs

Single variable forgetting or ensuring is generally not a linear transformation for OBDDs or for O-DDGs. However, such transformations can be achieved in linear time when the last variable of the structure is considered.

Definition 9 (Final assignment node) *An assignment node is* final *iff its tail is a constant.*

Definition 10 (Final variable) *A variable x is* final *in ϕ iff any assignment node in ϕ that bears on x is final.*

An important property is:

Proposition 6 *DDG, $OBDD_<$, $O-DDG_<$ and $SO-DDG_<$ are linearly closed for forgetting a final variable and ensuring a final variable.*

Sketch of proof: The key is that the forgetting/ensuring operations distribute over the binary connectives in DDGs, until reaching the decision nodes that bear on the variable under concern:

- If $N = \alpha \wedge \beta$ is a decomposable AND node, then $forget(\alpha \wedge \beta, x) \equiv forget(\alpha, x) \wedge forget(\beta, x)$ and is also a decomposable AND node. Indeed, if α bears on x, β does not (decomposability assumption), so
$$\begin{array}{rcl} forget(\alpha \wedge \beta, x) & \equiv & (\alpha_{[x \leftarrow true]} \wedge \beta) \vee (\alpha_{[x \leftarrow false]} \wedge \beta) \\ & \equiv & \beta \wedge (\alpha_{[x \leftarrow true]} \vee \alpha_{[x \leftarrow false]}) \\ & \equiv & \beta \wedge forget(\alpha, x). \end{array}$$
Similarly, when β bears on x we get $forget(\alpha \wedge \beta, x) \equiv \alpha \wedge forget(\beta, x)$. When neither α nor β bears on x, $forget(\alpha \wedge \beta, x) \equiv \alpha \wedge \beta$. The three cases are then summarized by $forget(\alpha \wedge \beta, x) \equiv forget(\alpha, x) \wedge forget(\beta, x)$.
- If $N = (y \wedge \alpha) \vee (\neg y \wedge \beta)$ is a decision node, and $x \neq y$, then $forget((y \wedge \alpha) \vee (\neg y \wedge \beta), x) \equiv (y \wedge forget(\alpha, x)) \vee (\neg y \wedge forget(\beta, x))$.
- If $N = \alpha \wedge \beta$ is a decomposable AND node, then $ensure(\alpha \wedge \beta, x) \equiv ensure(\alpha, x) \wedge ensure(\beta, x)$. It is still a decomposable AND node.
- If $N = (y \wedge \alpha) \vee (\neg y \wedge \beta)$ is a decision node, and $x \neq y$, then $ensure((y \wedge \alpha) \vee (\neg y \wedge \beta), x)$
$$\begin{array}{rcl} & \equiv & ((y \wedge \alpha) \vee (\neg y \wedge \beta))_{[x \leftarrow true]} \\ & & \wedge ((y \wedge \alpha) \vee (\neg y \wedge \beta))_{[x \leftarrow false]} \\ & \equiv & ((y \wedge \alpha)_{[x \leftarrow true]} \vee (\neg y \wedge \beta)_{[x \leftarrow true]}) \\ & & \wedge ((y \wedge \alpha)_{[x \leftarrow false]} \vee (\neg y \wedge \beta)_{[x \leftarrow false]}) \\ & \equiv & (y \wedge \alpha_{[x \leftarrow true]} \wedge y \wedge \alpha_{[x \leftarrow false]}) \\ & & \vee (y \wedge \alpha_{[x \leftarrow true]} \wedge \neg y \wedge \beta_{[x \leftarrow false]}) \\ & & \vee (\neg y \wedge \beta_{[x \leftarrow true]} \wedge \neg y \wedge \beta_{[x \leftarrow false]}) \\ & & \vee (\neg y \wedge \beta_{[x \leftarrow true]} \wedge y \wedge \alpha_{[x \leftarrow false]})) \\ & \equiv & (y \wedge \alpha_{[x \leftarrow true]} \wedge \alpha_{[x \leftarrow false]}) \\ & & \vee (\neg y \wedge \beta_{[x \leftarrow true]} \wedge \beta_{[x \leftarrow false]}) \\ & \equiv & (y \wedge ensure(\alpha, x)) \vee (\neg y \wedge ensure(\beta, x)). \end{array}$$

So, the forget (resp. ensure) operation for variable x distributes over the connectives until either reaching a leaf that does not bear on x (leaving it unchanged) or a decision node N on x – say $N = (x \wedge \alpha) \vee (\neg x \wedge \beta)$). But x is a final variable by hypothesis. Thus α and β are constants. Then $forget((x \wedge \alpha) \vee (\neg x \wedge \beta), x) \equiv \alpha \vee \beta$ and $ensure((x \wedge \alpha) \vee (\neg x \wedge \beta), x) \equiv \alpha \wedge \beta$. In other words, forgetting (resp. ensuring) x amounts to replacing the decision nodes on x by constants. These constants can then be propagated up in the graph so as to simplify it. The bottom up propagation is done in linear time with respect to the size of the graph, it does not increase its size (but may rather decrease it) and does not change its properties: the decision ordering is not changed, and the remaining nodes are either decomposable AND nodes or decision nodes. □

The importance of the $O-DDG$ fragment compared to the full DDG one as to QBF comes from the following lemma and theorem, based on Propositions 4, 5, and 6.

Lemma 1 *If x does not have any successors w.r.t. $<$, then it is a final variable for any formula from $O-DDG_<$.*

Theorem 2 *Let $<$ be any total, strict order on X. QBF. The restriction of* QBF *to formulae with matrices from $O-DDG_<$ and with prefixes inducing $<$ is in* P.

A Glimpse at Extended DDGs

In the previous sections, we have shown how $OBDD$ can be generalized, defining more compact but still efficient classes of propositional formulae, namely $O-DDG$ and $SO-DDG$.

Other characteristics of decision diagrams can be enlarged, without losing the tractability of the class for last variable eliminations – for instance, one can introduce non-decision OR nodes, provided that, like AND nodes, they are decomposable. In the sequel, we focus on such extended decision graphs and show how the tractability results for (CO), (ME), (CE), (VA), (CD) and last variable eliminations can be extended to them.

First, a new class of OR nodes can be straightforwardly considered without questioning the distributivity of ensuring over binary connectives offered by DDGs: decomposable OR nodes.

Definition 11 (Decomposable OR node) *An OR node is* decomposable *iff its disjuncts do not share any variable.*

Proposition 7 *If $\alpha \vee \beta$ is a decomposable OR node then $forget(\alpha \vee \beta, x) \equiv forget(\alpha, x) \vee forget(\beta, x)$ and $ensure(\alpha \vee \beta, x) \equiv ensure(\alpha, x) \vee ensure(\beta, x)$.*

Hence the fragment of NNFs that are both OR and AND decomposable gives rise immediately to a tractable restriction of QBF. However, this fragment is not very interesting since it contains only monotone formulae. A more interesting fragment is obtained by enabling both decomposable OR nodes and decision nodes:

Definition 12 (Extended DDGs) *An* extended, decomposable decision graph *is an NNF formula in which every node is either a constant, a decision node, a decomposable AND node or a decomposable OR node.*

Proposition 8 *The class of extended DDGs is linearly closed for final variable ensuring and for final variable forgetting.*

Sketch of proof: Propositions 6 and 7 show that both the ensuring operation and the forgetting operation for x distribute over binary connectives until a decision node on x is reached. Such a node does not have any successor since x is final. The decision node is then replaced by a constant. The simplification by backward propagation of the constants is then performed. It is linear in the size of the graph. Since this does not question the properties of decomposability or decision, we get a (possibly smaller) extended DDG. □

Now, a notion of ordered extended decision graph is obtained in a straightforward way by adding the ordering requirement. As a direct corollary of the previous proposition, we get:

Corollary 2 *Let $<$ be any total, strict order on X. The restriction of* QBF *to formulae with matrices from the class of extended DDGs that are ordered w.r.t. $<$ and with prefixes inducing $<$ is in* P.

Clearly enough, the class of extended DDGs is tractable for (CO), (ME), (CE) and linearly closed for conditioning since it is a subset of $DNNF$. It is also tractable for (VA) since it is linearly closed for final variable ensuring.

Other Related Work and Conclusion

If we make abstraction of the domains arities and of the representation of counter-models, it appears that $SO-DDGs$ are not completely unknown structures (but $O-DDGs$ are): they are equivalent to tree-driven automata (Fargier & Vilarem 2004) and to AND/OR search graphs (Dechter 2004). Although also based on a tree structure, cluster trees (Pargamin 2002) are slightly different: they use a hierarchy of arrays rather than a hierarchy of graphs. But it can be shown that, compiling each array into a small OBDD, each cluster tree can be turned into an equivalent polynomially sized $SO-DDG$ formula. The converse transformation is more expensive in the worst case (since an $OBDD$ formula can account for exponentially many models). Finally, it seems that tree BDDs (McMillan 1994) form a different kind of data structure, in which the tree of variables is used from the leaves to the root. This allows for a polytime transformation for negation that is unlikely for $O-DDG$ or $SO-DDG$. On the other hand, tree BDDs are not known as leading to a tractable restriction of QBF.

The contribution of the present paper is twofold. First, we have formally defined several classes of decision graphs, focusing on ordered and decomposable ones. We have proved that they enable several polytime queries and transformations, which shows them as interesting target classes for knowledge compilation. Second, we have shown that $OBDD$ is not the most general propositional fragment which is polynomially closed for final variable elimination: decomposable AND nodes are not required to be limited to assignment nodes and decomposable OR nodes can be allowed while preserving this property. All these results contribute to complete the knowledge compilation map from (Darwiche & Marquis 2001), which is probably far from being exhausted.

References

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

Coste-Marquis, S.; Le Berre, D.; Letombe, F.; and Marquis, P. 2005. Propositional fragments for knowledge compilation and quantified boolean formulae. In *Proceedings of AAAI-05*, 288–293.

Coste-Marquis, S.; Fargier, H.; Lang, J.; Le Berre, D.; ; and Marquis, P. 2006. Representing policies for quantified boolean formulae. In *Proceedings of KR-06*, to appear.

Darwiche, A., and Marquis, P. 2001. A perspective on knowledge compilation. In *Proceedings of IJCAI-01*, 175–182.

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

Dechter, R. 2004. And/or search spaces for graphical models. Technical report, ICS Technical Report.

Egly, U.; Eiter, T.; Tompits, H.; and Woltran, S. 2000. Solving advanced reasoning tasks using quantified boolean formulas. In *Proceedings of AAAI-00*, 417–422.

Faltings, B., and Weigel, R. 1999. Compiling constraint satisfaction problems. *Artificial Intelligence* 115(2):257–287.

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

McMillan, K. 1994. Hierarchical representation of discrete functions with application to model checking. In *Proceedings of CAV-94*, 41–54.

Pargamin, B. 2002. Vehicle sales configuration: the cluster tree approach. In *Proceedings of ECAI'02 Workshop on Configuration*, 35–40.

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research* 10:323–352.

Sieling, D., and Wegener, I. 1993. Reduction of OBDDs in linear time. *Information Processing Letters* 48(3):139–144.

Vempaty, N. 1992. Solving constraint satisfaction problems using finite state automata. In *Proceedings of AAAI-92*, 453–458.