Nucleus-Satellites Systems of OMDDs for Reducing the Size of Compiled Forms

Hélène Fargier, Jérôme Mengin, Nicolas Schmidt

Nucleus-Satellites Systems of OMDDs for Reducing the Size of Compiled Forms

Hélène Fargier
IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, France

Jérôme Mengin
IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, France

Nicolas Schmidt
IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, France

Abstract

In order to reduce the size of compiled forms in knowledge compilation, we propose a new approach based on a splitting of the main representation into a nucleus representation and satellite representations. Nucleus representation is the projection of the original representation onto the "main" variables and satellite representations define the other variables according to the nucleus. We propose a language and a method, aimed at OBDD/OMDD representations, to compile into this split form. Our experimental study shows major size reductions on configuration- and diagnosis-oriented benchmarks.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Constraint and logic programming

Keywords and phrases Knowledge representation, knowledge compilation, ordered multivalued decision diagram

Digital Object Identifier 10.4230/LIPIcs.CP.2022.23

Funding The authors gratefully acknowledge the support of the Artificial and Natural Intelligence Toulouse Institute – ANITI. ANITI is funded by the French "Investing for the Future – PIA3" program under grant agreement ANR-19-PI3A-0004.

Acknowledgements We thank anonymous reviewers for the CPAIOR and CP conferences for their numerous comments and suggestions that helped improve the paper.

1 Introduction

Knowledge compilation aims at translating (off line) a problem expressed in some language into a target language in which operations which are important for the application targeted can be performed efficiently [4, 8]. Decision diagrams for instance (OBDDs [3], OMDDs [17, 12], ordered MDDGs [23, 18]) have shown to be a good target language for many problems expressed as constraint satisfaction problem or as CNF, and in particular for product configuration problems [26, 1, 13].

The size of the compiled form being a main criterion in knowledge compilation, its reduction is a major issue in the field. This is magnified by the fact that many information redundancies can be observed. Indeed, caching and the detection of isomorphic nodes allow the detection of equivalent sub-graphs, but not of all redundant information. This last aspect is why, in this paper, we propose a method for reducing the size of the compiled form, up to an additional (but polynomial) computational cost for the handling of queries and transformations. Because targetting interactive configuration problems, we focus our work on OBDD/OMDD [3, 28, 17, 12] representation languages, and show that the method proposed allows a quickest compilation of the original problem, leads to a much smaller compiled form and above all to an important saving in time when the compiled form is exploited.

This article is structured as follows. The next section introduces necessary background and notations. The "satellite system" approach we propose is developed in Section 3. We then evaluate the succinctness of this language from an experimental point of view in Section 4. We relate in Section 5 the theoretical concept on which our approach is based to the notion of "definability", as it has been studied in propositional logic.

2 Background and notations

2.1 Representation languages

Consider a finite set $\mathcal{X}$ of variables, each variable x ranging over a finite domain D_x. For any set $X \subseteq \mathcal{X}$, $\vec{x}$ denotes an assignment to the variables from X. D_X is the set of all assignments of X (the Cartesian product of the domains of the variables in X). The concatenation of two assignments $\vec{x}$ and $\vec{y}$ of disjoint subsets X and Y is an assignment to $X \cup Y$ denoted $\vec{x} \cdot \vec{y}$.

We consider functions f of variables from a subset $\mathrm{Scope}(f) \subseteq \mathcal{X}$ to a set $\mathcal{V}$. We write D_f to denote the domain of f, i.e. $D_f = D_{\mathrm{Scope}(f)}$. For any $Z \subseteq \mathrm{Scope}(f)$, $f_{\vec{z}}$ denotes the *restriction* (or *semantic conditioning*) of f by $\vec{z}$, that is, the function on $\mathrm{Scope}(f) \setminus Z$ such that for any $\vec{x} \in D_{\mathrm{Scope}(f)\setminus Z}$, $f_{\vec{z}}(\vec{x}) = f(\vec{z} \cdot \vec{x})$. Slightly abusing notations, if X and Y are two disjoint sets of variables, f is a function such that $\mathrm{Scope}(f) = X$, and $\vec{x} \cdot \vec{y}$ is an assignment of a super set $X \cup Y$ of X, then we write $f(\vec{x} \cdot \vec{y})$ for $f(\vec{x})$.

A representation language over $\mathcal{X}$ w.r.t a valuation set $\mathcal{V}$ is a set of data structures equipped with an interpretation function that associates with each data structure a mapping from $D_{\mathcal{X}}$ to $\mathcal{V}$. This mapping is called the *semantics* of the data structure, and the data structure is a *representation* of the mapping.

▶ **Definition 1** (representation language; inspired by [11]). *A representation language* $\mathtt{L}$ *over* $\mathcal{X}$ *w.r.t* $\mathcal{V}$*, is a 4-tuple* $\langle C_{\mathtt{L}}, \mathrm{Scope}_{\mathtt{L}}, f^{\mathtt{L}}, ||_{\mathtt{L}} \rangle$*, where:*

- $C_{\mathtt{L}}$ *is a set of data structures* ϕ *(also referred to as* $\mathtt{L}$ *representations or "formulæ"),*
- $\mathrm{Scope}_{\mathtt{L}} : C_{\mathtt{L}} \to 2^{\mathcal{X}}$ *is a scope function associating with each* $\mathtt{L}$ *representation the subset of* $\mathcal{X}$ *it depends on,*
- $f^{\mathtt{L}}$ *is an interpretation function associating with each* $\mathtt{L}$ *representation* ϕ *a mapping* $f^{\mathtt{L}}_{\phi}$ *from the set of all assignments over* $\mathrm{Scope}_{\mathtt{L}}(\phi)$ *to* $\mathcal{V}$*,*
- $||_{\mathtt{L}}$ *is a size function from* $C_{\mathtt{L}}$ *to* $\mathbb{N}$ *that provides the size* $|\phi|_{\mathtt{L}}$ *of any* $\mathtt{L}$ *representation* ϕ*.*

Two formulæ ϕ *and* ψ *(possibly from different languages) are equivalent iff they have the same scope and semantics; this is denoted* $\phi \equiv \psi$*.*

In the following, $\mathcal{X}$ is a set of discrete variables and $\mathcal{V} = \{\top, \bot\}$. Given two functions f and g, the assignments $\vec{x}$ of $\mathcal{X}$ such that $f(\vec{x}) = \top$ are said to be "models" (or "solutions") of f and g is said to be a consequence of f (denoted $f \models g$) if any model $\vec{x}$ of ϕ can be extended to a model of ψ: $f(\vec{x}) = \top$ implies $\exists \vec{y} \in D_{\mathrm{Scope}(g)\setminus\mathrm{Scope}(f)}$ such that $g(\vec{x}.\vec{y}) = \top$. Given two $\mathtt{L}$ representations ϕ and ψ, $\vec{x}$ is a model (or "solution") of ϕ iff it is a model of $f^{\mathtt{L}}_{\phi}$, $\vec{x}$ is then said to be consistent with f. ψ is said to be a consequence of ϕ (denoted $\phi \models \psi$) iff any model of ϕ can be extended to a model of ψ (i.e. $f^{\mathtt{L}}_{\phi} \models f^{\mathtt{L}}_{\psi}$).

For any function f from (a subset of) $\mathcal{X}$ to $\mathcal{V}$ and any partition (Y, Z) of $\mathrm{Scope}(f)$, $\exists Z.f$ is the function on Y which maps $\vec{y}$ to $\top$ iff there exist a $\vec{z}$ such that $f(\vec{y}.\vec{z}) = \top$. $\exists Z.f$ is the projection of f on $\mathcal{X} \setminus Z$. Finally, $f \wedge g$ denotes the conjunction of f and g: $(f \wedge g)(\vec{x}) = \top$ iff $f(\vec{x}) = \top$ and $g(\vec{x}) = \top$; and $f \vee g$ denotes their disjunction: $(f \vee g)(\vec{x}) = \top$ iff $f(\vec{x}) = \top$ or $g(\vec{x}) = \top$.

2.2 CSPs

A CSP formula is a set $\phi = \{f_1, \ldots, f_{|\phi|}\}$ of functions f_i (also called "constraints") mapping assignments of subsets of $\mathcal{X}$ to $\{\top, \bot\}$. For any $\vec{x}$ in $D_{Scope(f_i)}$, $f_i(\vec{x}) = \top$ means that $\vec{x}$ satisfies the constraint. A model (a solution) of the CSP is an assignment of $\mathcal{X}$ satisfying all the constraints – ϕ is a representation of the function $f^{CSP}(\phi) = \bigwedge_{f_i \in \phi} f_i$. No assumption is made on the way constraints are represented – it is simply assumed that $f_i(\vec{x})$ can be computed quickly (generally, in linear or constant time).

2.3 Propositional Logic, CNFs

Given a set of *Boolean* variables $\mathcal{X}$, a *literal* over $\mathcal{X}$ is either a variable of $\mathcal{X}$ or the negation of a variable of $\mathcal{X}$. Well formed logical formulae are defined as usual using the logical connectors $\neg$, $\vee$, $\wedge$. For any well formed formula ϕ, f^{PROP}_{ϕ} obeys the classical semantic. For instance a *clause* is a disjunction $cl_i = l_1 \vee \cdots \vee l_{|cl_i|}$ of literals; Scope(cl_i) is the set of variables on which the literals of cl_i bear; the semantics of cl_i is the Boolean function $f^{Clause}_{cl_i}$ from $D_{Scope(cl_i)}$ to $\{\top, \bot\}$ defined by $f^{PROP}_{cl_i}(\vec{x}) = \top$ iff $\vec{x}$ maps value $\top$ to at least one positive literal of ϕ or value $\bot$ to at least one negative literal of ϕ. Likewise, a CNF over $\mathcal{X}$ is a conjunction $\phi = cl_1 \wedge \cdots \wedge cl_{|\phi|}$ of clauses; then Scope(ϕ) $= \bigcup_{cl_i \in \phi}$ Scope(cl_i) and ϕ is a representation of the function $f^{CNF}(\phi) = \bigwedge_{cl_i \in \phi} f^{Clause}_{cl_i}$ – it is satisfied iff all the clauses of ϕ are satisfied.

2.4 Decision diagrams

A decision diagram (DD) is a directed acyclic graph with a single root node denoted $root(\phi)$ and two leaf nodes labelled with $\top$ and $\bot$ respectively. Non-leaf nodes can be of two kinds, "AND" nodes and decision nodes.

- *Decision* nodes are labelled with variables of $\mathcal{X}$; if v is a decision node labelled with $x \in \mathcal{X}$, then v has as many children w as there are values in D_x, and the edges (v, w) are univocally labelled with the values in D_x. We write $a = label(v, w)$ to indicate that edge (v, w) is labelled with value a. For every $a \in D_x$, $next(v, a)$ will denote the child of v selected by value a, i.e. $next(v, a) = w$ iff edge (v, w) is labelled with value a. $next(v)$ denotes the set of children of v.
  The paths of the DD are often assumed to satisfy the read-once property: no path from the root to the $\top$ leaf node contains a given variable label more than once.
- "AND" nodes are labelled with $\wedge$. An AND node v can have any number of children, and if w is one of them then the edge (v, w) is not labelled;

The scope of a decision diagram is naturally defined as the set of variables that label its decision nodes.

The interpretation function of decision diagrams is defined as follows. Let v be the root node of ϕ. If ϕ contains only one node (v is aleaf), it is necessarily labelled with a constant $c \in \{\top, \bot\}$; then $f^{DD}_{\phi}(\vec{x}) = c$. If v is an AND node, then $f^{DD}_{\phi}(\vec{x}) = \bigwedge_{v' \in next(v)} f^{DD}_{\phi(v')}(\vec{x})$. If v is a decision node with label x_i, then $f^{DD}_{\phi}(\vec{x}) = f^{DD}_{next(v,a)}(\vec{x})$ where a is the value assigned to x_i by $\vec{x}$.

A decision diagram is in reduced form iff all isomorphic subgraphs are merged. The reduction of a decision diagram can always be performed in linear time. We assume in the following that the decision diagrams are in reduced form. Several valuable categories of decision diagrams have been identified:

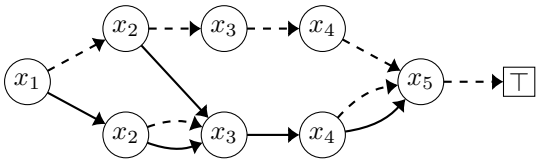


Figure 1 An OBDD equivalent to the logical formula $\phi = \big[(x_1 \wedge x_3) \vee (\neg x_1 \wedge ((x_2 \wedge x_3) \vee (\neg x_2 \wedge \neg x_3 \wedge \neg x_4)))\big] \wedge \neg x_5$; dashed edges are implicitly labelled with value 0, plain edges with value 1.

- A *MDDG* is a decision diagram the AND nodes of which are *decomposable*, meaning that if w and w' are two distinct children of AND node v, then the two sets of variables that appear in the two decision diagrams rooted at w and w' respectively must be disjoint.
- Let $<$ be total order on $\mathcal{X}$. A DD is ordered by $<$ iff for any pair (v, v') of decision nodes in ϕ, if v' can be reached from v, then $label(v) < label(v')$ (on a path, the nodes are encountered according to $<$) – as a consequence the DD does not contain any AND node. Such graphs are called *Ordered Multivalued Decision Diagrams* (OMDD). They constitute a generalization of well-known Ordered Binary Decision Diagrams (OBDD), and allow Boolean functions of discrete variables, instead of Boolean variables only.

In the "logical" definition of OMDDs given above, there are two sink nodes labelled with $\top$ and $\bot$, and every node has an outgoing edge for every value of the domain of the variable that labels the node; but when implementing OMDDs, and when drawing them, it is sufficient to implement / draw only the paths that lead to $\top$ – every "dead-end" then corresponds to a path to $\bot$; the sink node labelled $\bot$ is implicit. We adopt this convention for OMDDs throughout. This means that for every node v, if $\text{next}(v) \neq \emptyset$ then v has $\top$ as one of its descendants; if there is no node $next(v, a)$ for a value a in the domain of the variable labeling v, this is equivalent to having $next(v, a) = \bot$.

▶ **Example 2.** Figure 1 depicts an OBDD over $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5\}$ that is equivalent to the following formula of propositional logic:

$$\phi = \big[(x_1 \wedge x_3) \vee (\neg x_1 \wedge ((x_2 \wedge x_3) \vee (\neg x_2 \wedge \neg x_3 \wedge \neg x_4)))\big] \wedge \neg x_5$$

2.5 Operations

In the following, we will use several basic operations on formulae. Let ϕ and ψ be two representations in a language L.

CD: The *conditioning* of ϕ by the assignment $z = a$ of variable z computes a L representation of $f(\vec{x}) = f^{\mathtt{L}}_{\phi}(\vec{x}.a)$

FO: the *forgetting* of a set of variables $X \subseteq \mathcal{X}$ in ϕ computes a L representation of $\exists \vec{x}.f^{L}_{\phi}$

$\bigwedge$BC: the *conjunction* of two L representations ϕ and ψ computes a L representation of the function $f^{L}_{\phi} \wedge f^{L}_{\psi}$

$\bigvee$BC: the *disjunction* of two L representations ϕ and ψ computes a L representation of the function $f^{L}_{\phi} \vee f^{L}_{\psi}$

GIC: A value $a \in D_x$ is Globally Inversely Consistent for some L representation ϕ if there exist at least an assignment $\vec{x}$ for which the value of x is a and $f_{\phi}(\vec{x}) = \top$. Ensuring *Global Inverse Consistency* of a formula consists in computing, for each variable, the set of all its globally inversely consistent values.

Operations CD and GIC are particularly useful in the domain of interactive product configuration, where ϕ represent a configurable product (each model is a feasible product): at each step, the user chooses a variable and the system then computes the set of its globally consistent values. The user then chooses one of these values and the corresponding conditioning is performed.

These two operations can be performed in linear time on decision diagrams – hence their attractiveness as a target language for compilation. Moreover, the forgetting, bounded conjunction and bounded disjunction are tractable when considering OMDD ($O(|\phi|)$ for the former, $O(|\phi| \times |\psi|)$ for the latter two).

2.6 Compilation of CSP/CNF into OMDD/OBDDs

Knowledge compilation aims at translating a formula ϕ expressed in some language into a language in which operations which are important for the application targeted can be performed efficiently – of course, there is no free lunch: there may exist some instances for which this process in not tractable.

In this paper, we consider the compilation of Constraint Satisfaction Problems (resp. CNF) into Ordered Multivalued (resp. Boolean) Decision Diagrams. Several compilers exist in this context, that preserve the set of models of the original representation; that is, if ϕ is the CSP representation of a problem, and ψ the Decision Diagram representation computed, any model of ϕ is a model of ψ and reciprocally.

Existing compilers are either *top-down* compilers or *bottom-up* compilers. Roughly, *top-down* compilers [18, 7, 27, 20, 24] perform a backtrack search of the models, adding the (set of) models reached to the current compiled form. These compilers make use of a SAT or CSP solver.

On the other hand, *bottom-up* compilers [28, 10, 5] start by separately compiling the constraints of the CSP representation, using a common order of the variables; the conjunction of these compiled constraints is then computed, using the $Apply_\wedge$ algorithm [3]. Each bounded conjunction is realized in polytime – the resulting form can be smaller than the former one, but it may also grow (polynomially) at each step. There is thus a risk of explosion when the number of conjunctions is not limited.

3 Nucleus-Satellites System of OMDD

The overall approach that we propose in this paper consists in reducing the size of one OMDD ϕ by extracting the information about some variable y. This information is represented in another formula ϕ_y, and y is deleted from ϕ (ϕ becomes $\exists y\phi$). This can induce a direct and moderate gain on the depth of the OMDD, and an indirect and bigger gain in the width of the structure. In many cases, repeated applications of this process can significantly reduce the size of the overall structure, if the information about the variables deleted from the original OMDD depends on a limited number of variables (in which case, ϕ_y will be small). ϕ_y is called a satellite and the reduced OMDD is a nucleus of the formula. This is illustrated on Figure 2.

Formally, we propose a new language, which we call Nucleus-Satellites System of OMDDs:

3.1 Definition

▶ **Definition 3.** *A Nucleus-Satellites System (NSS) of OMDDs is a triple* $\Phi = (\phi_n, Y_\phi, \{(y, \phi_y) \mid y \in Y_\phi\})$ *where* Y_ϕ *is a set of variables,* ϕ_n *and the* ϕ_y*'s are OMDDs on subsets of* $\mathcal{X}$*, such that:*

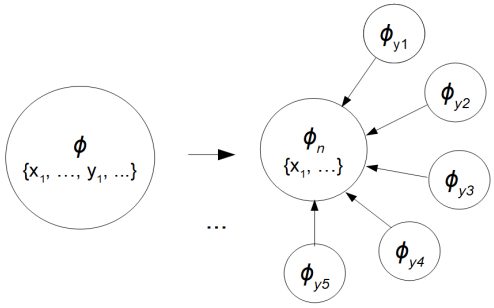


Figure 2 From an OMDD ϕ to a Nucleus-Satellites System.

1. *ϕ_n does not involve any of the y's in Y_ϕ and each $y \in Y_\phi$ appears only in ϕ_y ($\forall y \in Y_\phi$, $y \notin \text{Scope}(\phi_n)$ and $\forall y' \neq y \in Y_\phi, y' \notin \text{Scope}(\phi_y)$;*
2. *ϕ_n and the ϕ_y's obey the same order on $\mathcal{X} \setminus \{y\}$ and y labels the root of ϕ_y;*
3. *for any model $\vec{x}$ of ϕ_n, there exists an assignment $\vec{y}$ of the y's such that $\vec{x}.\vec{y}$ is a model of each ϕ_y.*

ϕ_n is called the nucleus of the system, and the ϕ_y's are its satellites.

A satellite system represents the conjunction of all its element, i.e. for any $\vec{x} \in D_{X \setminus Y_\phi}$ and any $\vec{y} = \vec{y_1}.\ldots.\vec{y_m} \in D_{Y_\phi}$:

- $f^{NSS}_\phi(\vec{x}.\vec{y}) = f^{OMDD}_{\phi_n}(\vec{x}) \wedge f^{OMDD}_{\phi_1}(\vec{x}.\vec{y_1}) \wedge \cdots \wedge f^{OMDD}_{\phi_m}(\vec{x}.\vec{y_m})$
- $\text{Scope}(\Phi) = Scope(\phi_n) \cup (\bigcup_{y \in Y_\phi} \text{Scope}(\phi_y))$
- $size(\Phi) = size(\phi) + \Sigma_{y_i \in Y_\phi} size(\phi_{y_i})$.

A satellite system can be viewed as a tree of OMDDs [9] of depth 1 where each edge ϕ_{y_i} uniquely defines the value of a variable y_i (not present in the nucleus nor in the other satellites).

Given some input L representation ϕ, we want to compute a Nucleus-Satellites System $\Phi = (\phi_n, Y_\phi, \{(y, \phi_y) \mid y \in Y_\phi\})$ that is much smaller than ϕ but equivalent to it: that is, we want that $f^L_\phi \equiv f^{NSS}_\Phi$. As we shall see shortly:

- Y_ϕ will be a set of variables such that $\exists Y_\phi.\phi$ is easy to compute;
- $\phi_n \equiv \exists Y_\phi.\phi$;
- for each $y \in Y_\phi$, ϕ_y retains enough information about y in order to be able to answer some queries of interest.

Because ϕ and the ϕ_y's are OMDDs, the fact that the nucleus ϕ_n and each satellite ϕ_y obey the same variable ordering (condition 2 in Definition 3) guarantees that the conjunction of these two OMDDs can be performed in quadratic time – this will be important for the online exploitation of the data structure.

In the next section, we describe a method for computing an NSS that is equivalent to, and hopefully much smaller than, an initial OMDD.

3.2 Computing a satellite system

We first show how it is possible to easily recognise, in an OMDD ϕ, some variables that will turn out to be easily forget from ϕ, and are therefore good candidates to be in the satellites.

▶ **Definition 4.** *A node v in an OMDD ϕ is passive iff it has at most one child different from $\bot$. Variable y is passive in ϕ iff each node labelled with y is passive.*

Algorithm 1 Satellisation.

Input: OMDD ϕ;
Output: satellite system equivalent to ϕ
1. $Y_\phi = \emptyset$; $\mathcal{X}_N = \emptyset$;
2. for all $y \in \mathcal{X}$:
   a. if y is passive in ϕ: add y to Y_ϕ;
   b. else: add y to $\mathcal{X}_N$;
3. for all $y \in Y_\phi$:
   a. compute weak definition ϕ_y of y in ϕ
   b. compute a representation of $\exists y.\phi$
   c. $\phi \leftarrow \exists y.\phi$
4. return$((\phi, Y_\phi, \{(y, \phi_y) \mid y \in Y_\phi\}))$

In other words, node v labelled by y is passive iff for every pair $a, b \in D_y$, if $next(v, a) \neq \bot$ and $next(v, b) \neq \bot$, then $next(v, a) = next(v, b)$. For instance, x_3 and x_4 are passive in the OMDD of Figure 1.

The set of passive variables in a given OMDD ϕ can be computed with a single traversal of ϕ (simply checking, for each variable y whether all the nodes labelled by y have at most one child different from $\bot$).

▶ **Proposition 5.** *If y is a passive variable of an OMDD ϕ, an OMDD ψ that represents $\exists y.f_\phi$ can be computed in linear time and $size(\psi) \leq size(\phi)$.*

Proof. Let us apply the classical algorithm processing the forgetting of one variable. When forgetting a node labelled by some y, this algorithm performs the disjunction (by a pass of the $Apply_\vee$ algorithm on all the children of this node, except on the $\bot$ node).

By definition if y is passive each node v labelled by y has at most two children: $\bot$ and another node u. So there is no need to perform the $apply_\vee$ stage and node v is directly replaced by node u (all the edges pointing at v now point at u).

So, ϕ can be transformed into a representation of $\exists y.f_\phi$ by replacing every node m labelled with y, by its unique child different from $\bot$. ◀

So we have a way to identify variables that can be easily forgotten in an OMDD of interest. Our next step is to provide a way to retain enough information about such a variable y , in another, hopefully small, OMDD, in order to be able to reason about y.

▶ **Definition 6.** *Given a formula ϕ of some representation language $\mathtt{L}$ over $\mathcal{X}$, and some variable $y \in \text{Scope}(\phi)$, a $\mathtt{L}$ formula ϕ_y over some subset $Z \subseteq \mathcal{X} \setminus \{y\}$* weakly defines [1] *$y$ in ϕ iff $f^{\mathtt{L}}_\phi = (\exists y.f^{\mathtt{L}}_\phi) \wedge f^{\mathtt{L}}_{\phi_y}$.*

We are now ready to describe the computation of the Nucleus-Satellites System that correspond to some input formula ϕ. It is formalized in Algorithm 1. The set of passive variables of the input OMDD ϕ is computed at step 2. Then, at step 3, for every passive variable y, a weak definition ϕ_y for y in ϕ is computed with Algorithm 2, described below; y is then forgotten in ϕ.

[1] As we shall see in section 5, this notion is close to the notion of *definability* as it has been studied in propositional logic.

Note that at the end of Algorithm 1 the main OMDD ϕ only bears on the variables in $\mathcal{X} \setminus Y_\phi$, since all variables in Y_ϕ have been forgotten in ϕ. The succession of forgetting at step 3b of this algorithm never increases the size of the nucleus (Proposition 5), and can significantly reduce the size of ϕ: the height of ϕ is lowered by 1 every time a variable is forgotten; and the recovery of a reduced form (the fusion of isomorphic nodes) that follows can lower its breadth.

After Algorithm 1 has been executed on some OMDD , the "satellite" variables – i.e. those in Y_ϕ – do not appear in the nucleus anymore. Moreover, it can easily be checked that if y, y' are two satellite variables, then y' does not appear in ϕ_y: either y' is below y in the variable ordering (and will thus not appear in the satellite) or the value of y is independent of that of y' (because y' is passive): it will never appear in the y satellite.

We now turn to the computation of the satellites. The main idea is that when a small set Z of variables defines some y in ϕ, ϕ_y should be small. Importantly the size of ϕ_y is bounded by $|D_y| \cdot size(\phi)$, because, as we shall see below, ϕ_y is a disjunction of $|D_y|$ formulas, each of which being no larger than ϕ.

The main loop of Algorithm 2, at step 2, iterates over all values $a \in D_y$: it computes the part of a weak definition of y in ϕ that pertains to value a. For every value $a \in D_y$, ϕ is simplified so as to retain just enough information to decide when an instantiation of the variables in ϕ_n is consistent with $y = a$. A fresh copy of ϕ is made at step 2a. Non-sink nodes below y-nodes are bypassed at step 2b, since they do not influence the consistency of value a for y. Then the ancestors of y nodes, and that are not relevant w.r.t. y and a, are bypassed at step 2c; they are called *undecisive*, see definition 7 below, their parents are redirected to one of their children (function redirect); finally y nodes are bypassed at steps 2d and 2e.

▶ **Definition 7.** *Given an OMDD ϕ over $\mathcal{X}$, given $y \in \mathcal{X}$ and $a \in D_y$, we say that a node v of ϕ is undecisive w.r.t. variable $y \in \mathcal{X}$ and $a \in D_y$ in ϕ if v is not labelled with y and:*
1. $|\,\mathrm{next}(v)| = 1$*; or*
2. *for all $w \in \mathrm{next}(v)$, w is labelled with y and $\mathrm{next}(w, a) \neq \bot$; or*
3. *for all $w \in \mathrm{next}(v)$, w is labelled with y and $\mathrm{next}(w, a) = \bot$.*

▶ **Example 8.** Figure 3 describes the computation of a satellite system for the OBDD of Figure 1, with $Y_\phi = \{x_3, x_4\}$. Figure 3a describes in details step 2 for $y = x_4$, $a = 1$: at step 2b, the plain edge $(x_4) \rightarrow (x_5)$ is redirected to $\boxed{\top}$; at step 2c, there are 3 passive nodes : the two nodes labelled x_3, and the one labelled x_2 on the bottom path, they are bypassed ; finally, at steps 2e and 2d, we bypass variable x_4, keeping only the paths that go through an edge where $x_4 = 1$. Note that x_5 is passive too: the "bottom" variable of an OMDD is always passive, according to our definition. However, in practice, all variables do not have to be "sent into orbit", passive variables disappear from the satellites anyway.

▶ **Proposition 9.** *Given some OMDD ϕ over $\mathcal{X}$, $y \in \mathcal{X}$ passive in ϕ, let ψ_y be the OMDD returned by Algorithm 2 when called with ϕ, y.Then $f_\phi = (\exists y.f_\phi) \wedge f_{\psi_y}$.*

The proof of the proposition is based on the following lemma. Its proof is in the appendix, and shows that equation (I) below is an invariant of the main loop in Algorithm 2.

▶ **Lemma 10.** *Given OMDD ϕ, y passive in ϕ, $a \in D_y$, if ψ_a is the OMDD computed at steps 2a to 2e in Algorithm 2 then*

$$\phi \wedge (y = a) \models \psi_a \quad \textit{and} \quad \psi_a \wedge (y = a) \wedge \exists y.\phi \models \phi \tag{I}$$

Algorithm 2 Computation of a weak definition.

Input : OMDD ϕ; y passive in ϕ and s.t. every path from root to $\top$ has a y-node;
Output : OMDD ψ s.t. $\phi \wedge (y = a) \models \psi$ and $\psi \wedge (y = a) \wedge \exists y.\phi \models \phi$.

1. $\phi_y \leftarrow \bot$;
2. for every $a \in D_y$ do:
   a. $\psi_a \leftarrow$ a fresh copy of ϕ; *// nodes below $y = a$ edges are bypassed*
   b. for every node v labelled with y, if $\text{next}(v, a) \neq \bot$: $\text{next}(v, a) \leftarrow \top$;
   c. while there is some undecisive node w.r.t. y, a in ψ_a do:
   //nodes not "relevan" w.r.t. which models are possible when $y = a$ are bypassed;
      i. $v \leftarrow$ a node of ψ_a undecisive w.r.t. y, a;
      ii. $w \leftarrow$ some node $\in \text{next}(v)$;
      iii. $\text{redirect}(v, w)$;
   //y nodes are bypassed, keeping only information pertaining to value a
   d. for every node v labelled with y s.t. $\text{next}(v, a) = \top$: $\text{redirect}(v, \top)$;
   e. for every node v labelled with y s.t. $\text{next}(v, a) = \bot$: $\text{redirect}(v, \bot)$;
   f. delete from ψ_a every node that is not accessible from the root anymore;
   g. $\phi_y \leftarrow \phi_y \vee (y = a \wedge \psi_a)$; *//compute disjunction of diagrams computed for all y values*
3. return ϕ_y.

Uses **function** $\text{redirect}(v, w)$: for every u, b such that $\text{next}(u, b) = v$: $\text{next}(u, b) \leftarrow w$.

Proof of the proposition. Let $\psi_y = \bigvee_{a \in D_y} (y = a \wedge \psi_a)$. We must prove that $\phi \models \psi_y$ and that $\psi_y \wedge (\exists y.\phi) \models \phi$. Consider some assignment $\vec{x}$ of $\mathcal{X}$, and let a be the value assigned to y in $\vec{x}$. We know, from lemma 10, that $\phi \wedge (y = a) \models \psi_a$ and $\psi_a \wedge (y = a) \wedge \exists y.\phi \models \phi$.

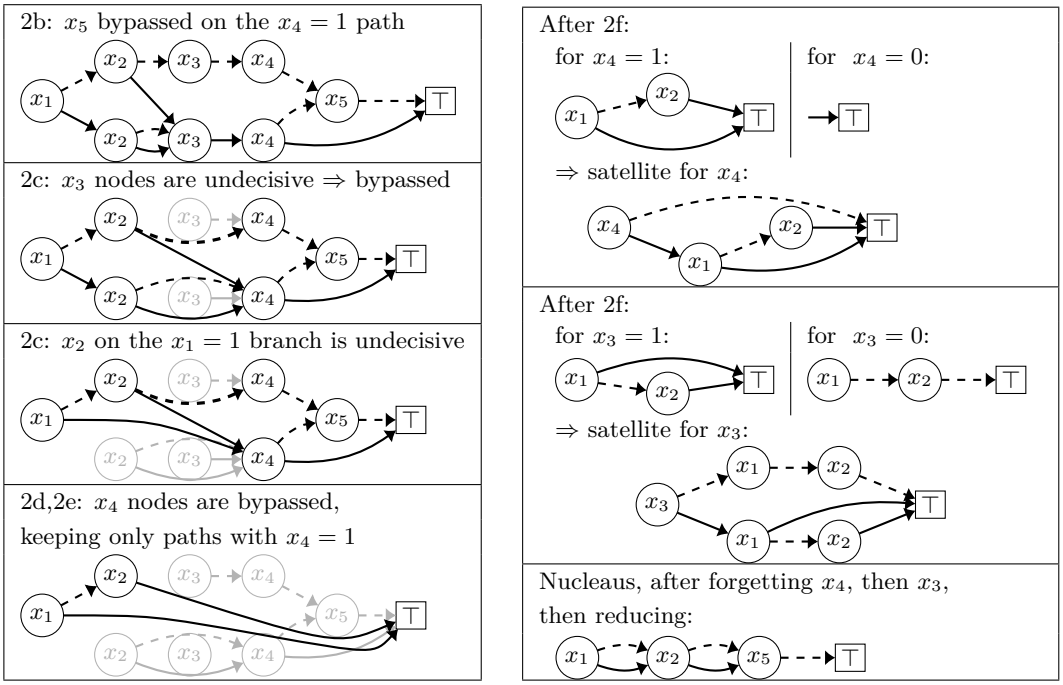
Suppose first that $f_\phi(\vec{x}) = \top$: $\phi \wedge (y = a) \models \psi_a$, $f_{\psi_a}(\vec{x}) = \top$, thus $f_{\psi_y}(\vec{x}) = \top$. For the converse, suppose that $f_{\psi_y}(\vec{x}) = f_{\exists y.\phi}(\vec{x}) = \top$. Then $f_{y=a'}(\vec{x}) = \bot$ for every $a' \in D_y$ with $a' \neq a$, thus it must be the case that $f_{\psi_a}(\vec{x}) = \top$, hence, because of equation (I), $f_\phi(\vec{x}) = \top$. ◀

In practice, making a fresh copy of the main OMDD ϕ at step 2a of Algorithm 2 is not efficient (nodes are created in the unique tables that will be destroyed immediately). Our implementation, used for the experiments described in Section 4, starts from an empty OMDD for ϕ_a, performs a bottom-up traversal of ϕ, starting at the y nodes, and adds decisive nodes (the ones that are not undecisive) to ϕ_a as they are encountered. Decisive nodes are recognised with some colouring scheme applied during this bottom-up traversal of ϕ.

3.3 Conditioning and maintaining GIC of an NSS of OMDDs

The application we target, interactive product configuration, mainly relies on two operations, the conditioning of one variable by the user and the maintaining of the global inverse consistency (GIC): if ϕ represents the current set of possible products, and if value a is chosen by the user for some currently unassigned variable y, then some representation of $\phi \wedge (y = a)$ must be computed that satisfies global inverse consistency (so that the user cannot choose, for the next assignment, a value that cannot lead to a feasible product). In the case of *partial conditioning*, the set of admissible values for y is restricted to, say, $\{a_1, \ldots, a_k\}$, and some representation of $\phi \wedge (y = a_1 \vee \ldots \vee y = a_k)$ must be computed and GIC restored.

By definition the GIC property holds for OMDDs. When constructing an OMDD, or when applying some transformation on it ($\bigwedge$BC, CD, etc), GIC is ensured during the reduction phase, by suppressing inconsistent values from the domains of their respective variables.



(a) Step by step exec. of step 2 for $x_4 = 1$.

(b) Satellites for x_3, x_4 and nucleus.

Figure 3 Execution of Algorithm 2 on the OBDD of figure 1, with $Y_\phi = \{x_3, x_4\}$.

A satellite system $(\phi, Y_\phi, \{(y, \phi_y) | y \in Y_\phi\})$ returned by Algorithm 1 has the GIC property, because it is logically equivalent to the input OMDD, which has the GIC property, and no new value has been introduced for any variable during satellisation.

Now, consider a satellite system that satisfies GIC, and a variable y on which a conditioning (possibly partial) must be performed :

- if $y \in Y_\phi$: conditioning is first performed on the satellite ϕ_y; then a new nucleus must be computed which is a representation of $\phi_n \wedge \phi_y$.
- if $y \notin Y_\phi$: conditioning is only performed on the nucleus ϕ_n.

In both cases, the nucleus has been modified, so GIC may then be lost for the satellite variables; in order to restore it, one can perform a "blank" computation of $\phi_y \wedge \phi_n$ for each ϕ_y (where ϕ_n is now the conditioned nucleus), without returning a new OMDD but just to check which values of y are not "GIC" anymore, in order to remove them from ϕ_y.

Now, recall that the computation of the conjunction of two OMDDs takes time at most quadratic in their size [3, 8]. As a consequence, the worst-time complexity of the conditioning of a satellite system is not linear in its size but quadratic. However, if the satellisation significantly reduces the size of the nucleus and leads to small satellites, then the effective time taken to perform conditioning is reduced too. Furthermore, recall that the size of a nucleus is necessarily smaller than the one of the OMDD equivalent to the full NSS. So the conjunction of a nucleus and a satellite leads to a new nucleus, and the maximal space taken by the conjunction of the original nucleus and the satellite cannot be higher than the one of the OMDD obtained if no satellisation were to be performed. This is because the nucleus is identical to the original OMDD except for the nodes corresponding to the satellites variables, that are passive in the original OMDD and can be forgotten by just by-passing them; the satellites are also obtained by bypassing irrelevant nodes.

Table 1 Configuration benchmarks – size (number of nodes) of the OMDD, nucleus-satellites system of OMDD and MDDG representation of the configuration instances. Column "Biggest" provide the maximal size reached during compilation process.

CSP	OMDD			nucleus-satellites system of OMDD					MDDG
	final	biggest	time	Nucleus	Satellites	sum	biggest	time	size
Small	**321**	328	0.05s	13	58	**71**	73	0.04s	**22**
Medium	**829**	941	0.6s	81	118	**199**	299	0.5s	**64**
Big	**13,916**	14,312	10s	1,475	220	**1,695**	1,723	8s	**2,552**
Master	**41,190**	42,343	13s	2,495	397	**2,892**	4,601	10s	**4,129**
Megane	**146,295**	150,506	18s	1,576	753	**2,329**	5,002	4s	**10,922**

4 Experimental results

In order to evaluate the efficiency of the approach, we have implemented a bottom-up OMDD compiler and an NSS bottom-up compiler which computes directly a nucleus-satellites system from a CSP given as input. The passive variables are detected on the fly, as early as possible during the compilation process, that is to say, as soon as they do not appear in any remaining uncompiled constraint. This method reduces the size of the maximal memory needed (recall that in a bottom-up approach, the size of the current data structure may increase and decrease with the addition of new constraints) – and as a side effect, the compilation time. This method leads to the same final NSS as the naive one (building the full OMDD first and satellizing the passive variables in a second step). Only the compilation time and maximal memory occupation may differ.

The following experiments are based on two families of benchmarks, configuration benchmarks[2], one the one hand, and diagnosis benchmarks[3], on the other hand. More precisely, we have compiled each instance (i) as an NSS and (ii) as an OMDD, and we have measured the sizes in terms of number of nodes, as this number is representative of the size of the diagram. We also measured the CPU time used by each compilation. Each instance has also be given to the CN2MDDG top-down compiler [18, 21] as a base line for the evaluation in terms of size spatial evaluation of the compiled form. Compilation times with CN2MDDG seem irrelevant here (different programming language C++ vs Java, valued vs non valued compilation, different programmers...) The experiments were performed on an Intel(R) Core(TM) i5-8265U CPU 1.60GHz 1.80 GHz, with 32Go of RAM.

4.1 Product configuration benchmarks

Product configuration benchmarks are CSPs representing real products (car) provided by the french car manufacturer Renault.

Table 1 gives the results on configuration instances: it shows a good spatial efficiency of nucleus-satellites systems. The bigger the problem gets, the more efficient they seem to be compared to OMDD. For example the size is divided by 5 on smaller instances and by 50 on bigger ones. This reduction in size makes it competitive with the MDDG language and even smaller on some benchmarks.

[2] `https://www.irit.fr/~Helene.Fargier/BR4CP/benches.html`
[3] `http://www.cril.univ-artois.fr/KC/benchmarks/cnf/circuit.tgz`

Table 2 Diagnosis benchmarks – size (number of nodes) of the OMDD, nucleus-satellites system of OMDD and MDDG representation of the configuration instances. Column "biggest" provides the maximal size reached during compilation process.

CNF	OMDD			nucleus-satellites system of OMDD					MDDG
	final	biggest	time	nucleus	satellites	sum	biggest	time	size
s344	**197,284**	293,972	59s	3,838	1,620	**5,458**	168,611	33s	**215**
s400	**23,014**	70,830	13s	1,340	2,341	**3,681**	40,235	9s	**429**
s444	**20,485**	27,081	5s	1,969	1,885	**3,854**	17,666	3s	**325**
s420	**35,900**	1,040,643	155s	5,697	1,653	**7,350**	348,317	78s	**316**
c499	**2,117,382**	2,117,382	101s	62,746	56,749	**119,495**	167,674	52s	23,424,571
s938	**Memory out**	>3,000,000		3,668	6,617	**10,285**	237,570	182s	**669**

4.2 Diagnosis benchmarks

As to diagnosis benchmarks (see Table 2), the NSS approach leads to a huge gain in time and space (one order of magnitude) with respect to the pure OMDD approach. Moreover, the NSS compiler makes it possible to compile a benchmark that cannot be compiled under the OMDD form (out of memory) – it should be noticed that the NSS representation obtained on this instance is much smaller (several orders of magnitude) than the OMDD built when the OMDD compiler ran out of memory.

4.3 Exploitation of the compiled form

Finally, since the goal of compilation is to be able to perform some operations on the compiled form, the present section compares the performances of OMDD and nucleus-satellites systems on a protocol of product configuration [2]. Namely, the compiled form is submitted to a sequence of succession of variable conditioning, each followed by a GIC closure. Each conditioning represents a choice made by a user on the product : at each step, the user chooses whichever variable and assigns a value to this variable. For each variable, the GIC closure then suppresses every non-consistent value of its domain. Since this kind of operation is done online by a user, a quick answer is necessary.

For the experiment[4], at every step, variables were chosen randomly among variables that still have at least 2 consistent values. Time differences between various sequences of user choices where very small. We experimented with the configuration protocol on all car instances, and obtained similar results in terms of comparison between OMDDs and NSSs. The results reported in Figure 4 for the "*big*" instance show that despite the necessity of additional computations induced by nucleus-satellites representation when processing a CD operation, there is a global gain in CPU time. The reduction in size obtained by the use of an NSS largely compensates the additional processing.

Note that our compiler can handle partial conditioning. We ran experiments on this point, and did not notice sizeable difference in the response time.

5 Related work

The idea of extracting, from some initial formula ϕ, information about a particular y has been studied in propositional logic with the notions of *functional dependency* [14, 15, 16] and *definability* [22]. Definability has recently been used by [19] to facilitate model counting in propositional logic.

[4] We did not experiment the configuration protocol with MDDGs since CN2MDDG is not a solver.

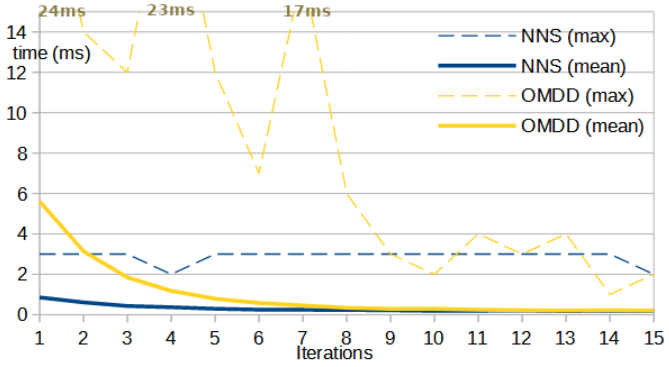


Figure 4 Processing time (in ms) at each iteration (conditioning + GIC closure) during a product configuration protocol (on 1000 tries) with the instance "*big*", compiled as an OMDD (in yellow) or as an nucleus-satellites system (in blue).

▶ **Definition 11** ([22]). *Let ϕ be a formula of propositional logic, $Z \subseteq \mathcal{X}$, $y \in \mathcal{X} \setminus Z$, then ϕ (explicitly) defines*[5] *y in terms of Z if and only if there is a formula ψ of propositional logic with* $\mathrm{Scope}(\psi) \subseteq Z$ *such that $\phi \models \psi \leftrightarrow y$. ψ is then called a definition of y on Z in ϕ.*

The following result shows that definability in the sense of [22] is a sufficient condition to build a Nucleus-Satellites System.

▶ **Proposition 12.** *If ϕ is a propositional formula on $\mathcal{X}$, and if ψ is a definition of y on Z in ϕ, then $\phi \equiv \exists y.\phi \wedge (y \leftrightarrow \psi)$.*

Proof. It is well-known that $\phi \models \exists V.\phi$ for any set of variables V, thus $\phi \models \exists y.\phi$. And $\phi \models y \leftrightarrow \psi$ by definition of a "definition". Suppose now that $m \models \exists y.\phi \wedge y \leftrightarrow \psi$. Let m' be the interpretation identical to m except that $m' \models \neg y$ if and only if $m \models y$. Since $m \models y \leftrightarrow \psi$, $m \models y$ if and only if $m \models \psi$, if and only if $m' \models \psi$ since m and m' give the same interpretation to all variables that appear in ψ. Thus $m' \models \neg y$ iff $m' \models \psi$, or, equivalently, $m' \models \neg y \leftrightarrow \psi$, or, equivalently, $m' \models \neg(y \leftrightarrow \psi)$. But by assumption $\phi \models y \leftrightarrow \psi$, thus $m' \models \neg\phi$. But, since $m \models \exists y.\phi$, it must be the case that $m \models \phi$ or $m' \models \phi$. Thus $m \models \phi$. ◀

However, definability, as studied in propositional logic, is not guaranteed; whereas it is always possible to extract enough information about a variable from a given formula in order to "satellize" it. The next example illustrates this.

▶ **Example 13.** Consider the propositional logic formula ϕ of Example 2:

$$\phi = \big[(x_1 \wedge x_3) \vee (\neg x_1 \wedge ((x_2 \wedge x_3) \vee (\neg x_2 \wedge \neg x_3 \wedge \neg x_4)))\big] \wedge \neg x_5$$

Considering the OBDD of Figure 1 equivalent to ϕ, it is easy to check that x_4 is not definable in the sense of [22] in ϕ: when $x_1 = \text{false}$ and $x_2 = \text{true}$, x_4 can be true but can also be false.

On the other hand, consider the formulas $\phi_y = (x_4 \rightarrow (x_1 \vee x_2)) \wedge (\neg x_4 \rightarrow \top)$ and $\psi = \big[(x_1 \wedge x_3) \vee (\neg x_1 \wedge ((x_2 \wedge x_3) \vee (\neg x_2 \wedge \neg x_3)))\big] \wedge \neg x_5$. It is easy to check that $\phi \equiv \phi_y \wedge \psi$.

[29] propose a similar approach, called *macro extraction and expansion*, for optimising the size of BDDs used for symbolic model checking. Their experimental results also indicate important gains when using this optimisation.

[5] [22] also introduce a notion of *implicit definability*, but they prove that, because of the projective Beth's theorem, both notions are equivalent in proposition logic.

6 Conclusion

This paper has proposed a method that allows to detect, on an OMDD, a set of variables that can be defined apart in several satellites, according to another common set of variables called the nucleus. We experimentally observe that it can lead to huge reductions of size and allows the compilation of benchmarks that could not be compiled as classical OMDDs.

Nucleus-satellites systems have to be studied further. First, satellites could also define a variable only partially (for example define and forget only passive nodes of a variable, and keep active nodes in the nucleus; the satellite would only be used when the variable is missing on path). A satellite could also represent a group of variables. With additional online computing, satellites could use variables defined in an other satellite to define a new variable, and create a satellite of "higher degree".

Finally, the approach can directly apply to any sub-languages of the d-DNNF family for which the operation of bounded conjunction is tractable under some conditions, e.g. structured d-DNNFs [25] or Sentential Decision Diagrams [6] – this is possible when (i) the order is computed on the basis on the original CSP and (ii) the operations targeted (here, the conditioning) do not modify the constraint graph. The question of the efficiency of satellite system based on other languages – and in particular of satellite systems of MDDG – is less easy to address; the question of the on line conditioning indeed becomes more tricky, although such structures can be easily defined and satellisation algorithms could be developed (the definition remain quasi unchanged and the notion of passive variable still applies).

References

1 Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic CSPs - Application to configuration. *Artificial Intellligence*, 135(1-2):199–234, 2002.

2 Jean-Marc Astesana, Laurent Cosserat, and Hélène Fargier. Constraint-based vehicle configuration: A case study. In *ICTAI 2010*, pages 68–75. IEEE Computer Society, 2010.

3 Randall E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers (TC)*, 38.8:677–691, 1986.

4 Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4):137–150, 1997.

5 Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In *AAAI'2013*, pages 187–194, 2013.

6 A. Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI'2011*, pages 819–826, 2011.

7 Adnan Darwiche. New Advances in Compiling CNF into Decomposable Negation Normal Form. In *ECAI 2004*, pages 328–332, 2004.

8 Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

9 Hélène Fargier and Pierre Marquis. Knowledge Compilation Properties of Trees-of-BDDs, Revisited. In *IJCAI 2009*, pages 772–777, 2009.

10 Hélène Fargier, Pierre Marquis, and Nicolas Schmidt. Semiring Labelled Decision Diagrams, Revisited: Canonicity and Spatial Efficiency Issues. In *IJCAI 2013*, pages 884–890, 2013. URL: `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6623`.

11 Goran Gogic, Henry Kautz, Christos Papadimitriou, and Bart Selman. The Comparative Linguistics of Knowledge Representation. In *IJCAI'1995*, pages 862–869, 1995.

12 Tarik Hadzic, Henrik Reif Andersen, John N. Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *CP 2007*, pages 118–132, 2007.

13 Tarik Hadzic, Rune Jensen, and Henrik Reif Andersen. Calculating Valid Domains for BDD-Based Interactive Configuration. *Computing Research Repository (CoRR)*, 0704.1394, 2007. `arXiv:0704.1394`.

14 Toshihide Ibaraki, Alexander Kogan, and Kazuhisa Makino. Functional dependencies in horn theories. *Artificial Intelligence*, 108(1-2):1–30, 1999.

15 Toshihide Ibaraki, Alexander Kogan, and Kazuhisa Makino. On functional dependencies in q-horn theories. *Artificial Intelligence*, 131(1-2):171–187, 2001.

16 Toshihide Ibaraki, Alexander Kogan, and Kazuhisa Makino. Inferring minimal functional dependencies in horn and q-horn theories. *Annals of Mathematics and Artificial Intelligence*, 38(4):233–255, 2003.

17 T. Kam, T.Villa, R.K. Brayton, and A.L Sangiovanni-Vincentelli. Multi-valued decision diagrams: Theory and applications. *International Journal of Multiple-Valued Logic*, 4:9–12, 1998.

18 Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Compiling constraint networks into multivalued decomposable decision graphs. In *IJCAI 2015*, 2015.

19 Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Definability for model counting. *Artificial Intelligence*, 281:103229, 2020.

20 Jean-Marie Lagniez and Pierre Marquis. An improved decision-DNNF compiler. In *IJCAI'2017*, volume 17, pages 667–673, 2017.

21 Jean-Marie Lagniez, Pierre Marquis, and Anastasia Paparrizou. Defining and evaluating heuristics for the compilation of constraint networks. In *CP 2017*, pages 172–188. Springer, 2017.

22 Jérôme Lang and Pierre Marquis. On propositional definability. *Artificial Intelligence*, 172(8):991–1017, 2008.

23 Robert Mateescu and Rina Dechter. Compiling Constraint Networks into AND/OR Multi-valued Decision Diagrams (AOMDDs). In *CP 2006*, pages 329–343, 2006.

24 Christian Muise, Sheila A McIlraith, J Christopher Beck, and Eric I Hsu. D sharp: fast d-DNNF compilation with sharpsat. In *CCAI'12*, pages 356–361, 2012.

25 Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI 2008*, pages 517–522, 2008.

26 Carsten Sinz. Knowledge compilation for product configuration. In *Proceedings of the Workshop on Configuration at the 15th European Conference on Artificial Intelligence (ECAI)*, pages 23–26, 2002.

27 Takahisa Toda and Takehide Soh. Implementing efficient all solutions SAT solvers. *ACM Journal of Experimental Algorithmics*, 21(1):1.12:1–1.12:44, 2016.

28 Nageshwara Rao Vempaty. Solving Constraint Satisfaction Problems Using Finite State Automata. In *AAAI'92*, pages 453–458, 1992.

29 Bwolen Yang, Reid G. Simmons, Randal E. Bryant, and David R. O'Hallaron. Optimizing symbolic model checking for constraint-rich models. In Nicolas Halbwachs and Doron A. Peled, editors, *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 328–340. Springer, 1999. `doi:10.1007/3-540-48683-6_29`.

A Appendix: Proof of the main lemma

▶ **Lemma 10.** *Given OMDD ϕ, y passive in ϕ, $a \in D_y$, if ψ_a is the OMDD computed at steps 2a to 2e in Algorithm 2 then*

$$\phi \wedge (y = a) \models \psi_a \quad \textit{and} \quad \psi_a \wedge (y = a) \wedge \exists y.\phi \models \phi \tag{I}$$

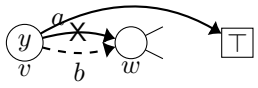
We will show that equation (I) is an invariant of the loop main loop of the algorithm, at step 2. For $\vec{x} \in D_{\mathcal{X}}$, variable z and $b \in D_z$, we write $\vec{x}[z] = b$ when $\vec{x}$ assigns value b to z.

Step 2a: That (I) holds just after ψ has been created is trivial, since at that point $\psi = \phi$.

For the remainder of the proof, we define three predicates, for OMDD ψ and $\vec{x} \in D_{\mathcal{X}}$ (w.r.t. some fixed OMDD ϕ):
$\boldsymbol{I_1(\psi,\vec{x})}$ is true iff $\vec{x}[y] \neq a$ or $f_\phi(\vec{x}) = \bot$ or $f_\psi(\vec{x}) = \top$;
$\boldsymbol{I_2(\psi,\vec{x})}$ is true iff $\vec{x}[y] \neq a$ or $f_\psi(\vec{x}) = \bot$ or $\exists y.f_\phi(\vec{x}) = \bot$ or $f_\phi(\vec{x}) = \top$.
$\boldsymbol{I_3(\psi,\vec{x})}$ is true iff $\vec{x}[y] \neq a$ or $\exists y.f_\phi(\vec{x}) = \bot$ or $\exists y.f_\psi(\vec{x}) = \top$.
We will prove that after every transformation that happen at steps 2b, 2c, 2d or 2e, ψ is such that for every $\vec{x} \in D_{\mathcal{X}}$, $I_1(\psi,\vec{x})$, $I_2(\psi,\vec{x})$ and $I_3(\psi,\vec{x})$ hold; and that $I_1(\psi,\vec{x})$ and $I_2(\psi,\vec{x})$ still hold after steps 2d and 2e.

In the remainder of the proof, $\vec{x} \in D_{\mathcal{X}}$ denotes a model such that $\vec{x}[y] = a$. We write that $\vec{x}$ "*passes through*" some node v in some OMDD ψ if the path in that diagram from the root to either $\top$ or $\bot$ that corresponds to the assignments in $\vec{x}$ contains v.

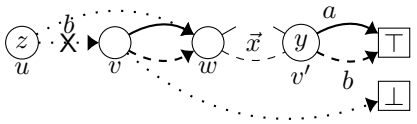
Step 2b: Let ψ denote the OMDD that is obtained after step 2a is executed: for every node v such that $label(v) = y$ and $\text{next}_\phi(v,a) \neq \bot$, $\text{next}_\psi(v,a) = \top$.



Suppose that $\vec{x}$ passes through such a node v in ϕ. (Otherwise, trivially $f_\psi(\vec{x}) = f_\phi(\vec{x})$). Note that $\vec{x}$ passes through v in ψ too, since the transformation applied here does not change the OMDD above y nodes).
$\boldsymbol{I_1(\psi,\vec{x})}$**:** by construction, $f_\psi(\vec{x}) = \top$. (Recall that we assume that $\vec{x}[y] = a$.)
$\boldsymbol{I_2(\psi,\vec{x})}$**:** if $\exists y.f_\phi(\vec{x}) = \top$, there must be some $b \in D_y$ such that $f_\phi(\vec{x}') = \top$, where $\vec{x}'$ is obtained by replacing with b the value, a, assigned to y in $\vec{x}$. But y is passive in ϕ, and $\text{next}_\phi(v,a) \neq \bot$, $\text{next}_\phi(v,b) \neq \bot$, so $\text{next}_\phi(v,a) = \text{next}(v,b)$ so necessarily $f_\phi(\vec{x}) = \top$.
$\boldsymbol{I_3(\psi,\vec{x})}$**:** that $\exists y.f_\psi(\vec{x}) = \top$ follows from the fact that $f_\psi(\vec{x}) = \top$.

We now turn to the transformations that take place at steps 2c. ψ will denote the current OMDD that is being built before such a transformation takes place, and ψ' will denote the OMDD just after the transformation has been applied. At every iteration of step 2c, an undecisive node v of ψ is picked, a child $w \in \text{next}(v)$ is picked, and for every parent u of v in ψ labelled with variable z, for every $b \in D_z$ such that $\text{next}_\psi(u,b) = v$, $\text{next}_{\psi'}(u,b) = w$. We must prove that for every $\vec{x} \in D_{\mathcal{X}}$ that passes through u and v (and thus such that $\vec{x}[z] = b$), if $I_1(\psi,\vec{x})$, $I_2(\psi,\vec{x})$ and $I_3(\psi,\vec{x})$ hold, then $I_1(\psi',\vec{x})$, $I_2(\psi',\vec{x})$ and $I_3(\psi',\vec{x})$ hold too.

Step 2c, undecisive node of type 1: $label(v) \neq y$, and $|\text{next}(v)| = 1$.

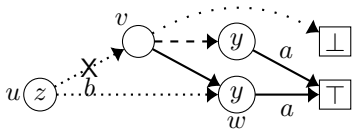


$\boldsymbol{I_1(\psi',\vec{x})}$**:** if $f_\phi(\vec{x}) = \top$, since $I_1(\psi,\vec{x})$ holds it must be the case that $f_\psi(\vec{x}) = \top$, so the path in ψ from w to a leaf that corresponds to $\vec{x}$ ends at $\top$, and this path is unchanged in ψ'; so $f_{\psi'}(\vec{x}) = \top$.
$\boldsymbol{I_2(\psi',\vec{x})}$**:** suppose that $f_{\psi'}(\vec{x}) = \top$; then the path in ψ' from w to a leaf that corresponds to $\vec{x}$ ends at $\top$, passing through à node v' labelled by y. Suppose too that $\exists y.f_\phi(\vec{x}) = \top$; then, since $I_3(\psi,\vec{x})$ is true, $\exists y.f_\psi(\vec{x}) = \top$: there is some $\vec{x}'$ identical to $\vec{x}$ except possibly that

$\vec{x}'[y] = b$ for some other $b \in D_y$, such that $f_\psi(\vec{x}') = \top$. Since y is passive, and Step 2 has been executed, it implies that $\text{next}_\psi(v',a) = \top$, and that $f_\psi(\vec{x}) = \top$. But then, since $I_2(\psi,\vec{x})$ holds, it must be the case that $f_\phi(\vec{x}) = \top$.

$\boldsymbol{I_3(\psi',\vec{x})}$: suppose that $\exists y.f_\phi(\vec{x}) = \top$; it implies that $\exists y.f_\psi(\vec{x}) = \top$, because $I_3(\psi,\vec{x})$ holds; but then $\exists y.f_{\psi'}(\vec{x}) = \top$ must also be true because ψ' creates a simple "shortcut" for $\vec{x}$.

Step 2c, undecisive node of type 2: v is a node of ψ, whose children, except $\bot$ if it is a child of v, are all y-nodes that are consistent with $y = a$.



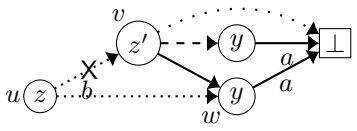
Note that $f_{\psi'}(\vec{x}) = \top$.

$\boldsymbol{I_1(\psi',\vec{x})}$: $f_{\psi'}(\vec{x}) = \top$ is true by construction.

$\boldsymbol{I_2(\psi',\vec{x})}$ Suppose that $\exists y.f_\phi(\vec{x}) = \top$. Since $I_3(\psi,\vec{x})$ holds, t cannot be the case that $\text{next}_\psi(v,a) = \bot$ because $\exists y.f_\psi(\vec{x}) = \top$, so next$\psi(v,a)$ is one of the y node consistent with $y = a$, hence $f_\psi(\vec{x}) = \top$. As a consequence, since $I_2(\psi,\vec{x})$ holds, $f_\phi(\vec{x}) = \top$.

$\boldsymbol{I_3(\psi',\vec{x})}$ That $\exists y.f_{\psi'}(\vec{x}) = \top$ is a simple consequence of the fact that $f_{\psi'}(\vec{x}) = \top$.

Step 2c, undecisive node of type 3: Let v be a node of ψ such that for all $w \in \text{next}(v)$, w is labelled with y and $\text{next}(w,a) = \bot$.



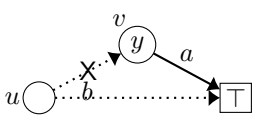
Note that $f_\psi(\vec{x}) = f_{\psi'}(\vec{x}) = \bot$.

$\boldsymbol{I_1(\psi',\vec{x})}$: since $I_1(\psi,\vec{x})$ holds and $f_\psi(\vec{x}) = \bot$, it must be the case that $f_\phi(\vec{x}) = \bot$.

$\boldsymbol{I_2(\psi',\vec{x})}$: holds because $f_{\psi'}(\vec{x}) = \bot$.

$\boldsymbol{I_3(\psi',\vec{x})}$: Let z' be the variable at v. If $\text{next}_\psi(v,\vec{x}[z']) = \bot$, then $\exists y.f_\psi(\vec{x}) = \bot$, thus, since $I_3(\psi,\vec{x})$, $\exists y.f_\phi(\vec{x}) = \bot$. Otherwise, $\text{next}_\psi(v,\vec{x}[z'])$ is a y node, and we assume that the input OMDD is "normalized", so every node has at least one successor different from $\bot$, so w has a successor $\neq \bot$; thus $\exists y.f_{\psi'}(\vec{x}) = \top$.

Step 2d: Let v be a node of ψ labelled with y such that $\text{next}(v,a) = \top$. Let u be a parent of v in ψ labelled with variable z, let $b \in D_z$ such that $\text{next}_\psi(u,b) = v$, then $\text{next}_{\psi'}(u,b) = \top$.

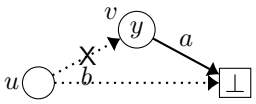


Suppose that $\vec{x}$ passes through u and v in ψ (otherwise, trivially $f_\psi(\vec{x}) = f_{\psi'}(\vec{x})$), so that $\vec{x}$ directly goes from u to $\top$ in ψ'. Note that $f_\psi(\vec{x}) = f_{\psi'}(\vec{x}) = \top$.

$\boldsymbol{I_1(\psi',\vec{x})}$: holds because $f_{\psi'}(\vec{x}) = \top$.

$\boldsymbol{I_2(\psi',\vec{x})}$: since $I_2(\psi,\vec{x})$ holds and $f_\psi(\vec{x}) = \top$, $\exists y.f_\phi(\vec{x}) = \bot$ or $f_\phi(\vec{x}) = \top$.

Step 2e: Let v be a node of ψ labelled with y such that $\text{next}(v,a) = \bot$. Let u be a parent of v in ψ labelled with variable z, let $b \in D_z$ such that $\text{next}_{\psi}(u,b) = v$, then $\text{next}_{\psi'}(u,b) = \bot$.



Suppose that $\vec{x}$ passes through u and v in ψ (otherwise, trivially $f_{\psi}(\vec{x}) = f_{\psi'}(\vec{x})$). Note that $f_{\psi}(\vec{x}) = f_{\psi'}(\vec{x}) = \bot$.

$\boldsymbol{I_1(\psi',\vec{x})}$**:** since $I_1(\psi,\vec{x})$ holds and $f_{\psi}(\vec{x}) = \bot$, it must be the case that $f_{\phi}(\vec{x}) = \bot$.
$\boldsymbol{I_2(\psi',\vec{x})}$**:** true because $f_{\psi'}(\vec{x}) = \bot$.