Lattice-Valued Binary Decision Diagrams

G. Geeraerts, G. Kalyon, T. Le Gall, N. Maquet, and J.-F. Raskin

Université Libre de Bruxelles – Dept. d'Informatique (méthodes formelles et vérification)
{gigeerae, gkalyon, tlegall, nmaquet, jraskin}@ulb.ac.be

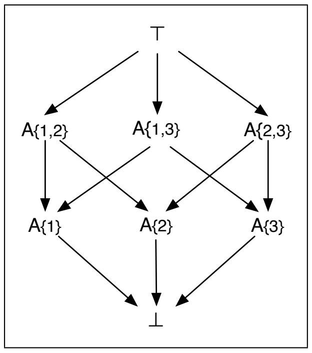
Abstract. This work introduces a new data structure, called Lattice-Valued Binary Decision Diagrams (or LVBDD for short), for the compact representation and manipulation of functions of the form $\theta : 2^{\mathsf{P}} \mapsto \mathcal{L}$, where P is a finite set of Boolean propositions and $\mathcal{L}$ is a finite distributive lattice. Such functions arise naturally in several verification problems. LVBDD are a natural generalisation of multi-terminal ROBDD which exploit the structure of the underlying lattice to achieve more compact representations. We introduce two canonical forms for LVBDD and present algorithms to symbolically compute their conjunction, disjunction and projection. We provide experimental evidence that this new data structure can outperform ROBDD for solving the finite-word LTL satisfiability problem.

1 Introduction

Efficient symbolic data structures are often the cornerstone of efficient implementations of model-checking algorithms [5]. Tools like SMV [6] and NuSMV [8] have been applied with success to industrial-strength verification problems. These tools exploit reduced ordered binary decision diagrams [10] (ROBDD for short) which is the reference data structure that has been designed to compactly encode and manipulate Boolean functions i.e., functions of the form $\theta : 2^P \rightarrow \{0, 1\}$. ROBDD are often regarded as the *basic toolbox* of verification, and are, indeed, a *very general symbolic data structure*.

The dramatic success of ROBDD in the field of computer aided verification has prompted researchers to introduce several variants of ROBDD [1, 2] or to extend them to represent other kinds of functions[3, 12]. For instance, MTBDD [12] have been successfully applied to the representation of functions of the form $2^P \mapsto D$, where D is an *arbitrary domain*. However, MTBDD work best when the set D is small, and make no assumptions about the structure of D. In this paper, we introduce a new data structure, called *lattice-valued binary decision diagrams* (or LVBDD for short) to efficiently handle *lattice-valued Boolean functions* (LVBF), i.e. functions of the form $\theta : 2^P \mapsto \mathcal{L}$, where P is a finite set of Boolean propositions and $\mathcal{L}$ is a *finite distributive lattice*.

In our opinion, such an efficient data structure has potentially many applications, as many examples of algorithms manipulating LVBF can be found in the literature. A first example is the *transition relation of an alternating finite state automaton* (AFA for short). It is well-known that the transition relation of an AFA is an LVBF of the form $2^P \mapsto \mathcal{L}_U$, where $\mathcal{L}_U$ is the *(finite distributive) lattice of upward-closed sets of sets of locations of the automaton*. Exploiting the particular structure of this transition relation is of crucial importance when analyzing AFA, as recently shown in the antichain line of research [13, 23]. This application is the case study we have retained in this paper. A second example, is the field of *multi-valued logics* where formulas have Boolean



c_1	c_2	c_3	$\theta(v)$
0	0	0	$A_{\{1,2\}}$
0	0	1	$A_{\{1,2\}}$
0	1	0	$\perp$
0	1	1	$A_{\{1\}}$
1	0	0	$A_{\{3\}}$
1	0	1	$A_{\{3\}}$
1	1	0	$A_{\{1,3\}}$
1	1	1	$A_{\{1,3\}}$

Fig. 1. The lattice $\mathcal{L}_A$ and the truth table of an LVBF $\theta : 2^{\{c_1,c_2,c_3\}} \mapsto \mathcal{L}_A$.

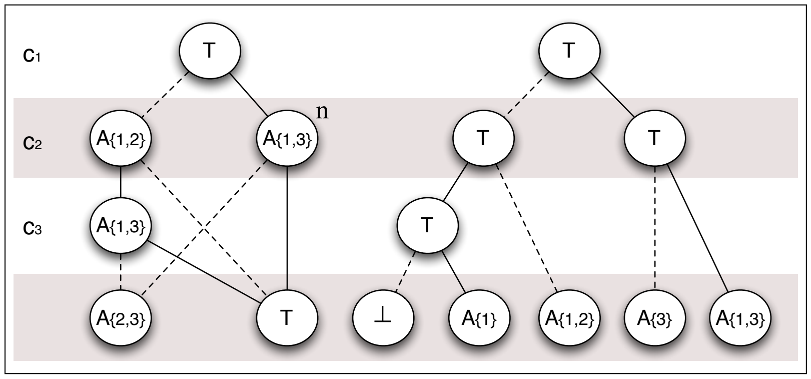


Fig. 2. Two examples of LVBDD resp. in SNF (left) and UNF (right) representing the LVBF θ.

variables but truth values are taken in an arbitrary finite set, usually a lattice. Such multi-valued formula occur in the field of *multi-valued model checking* [7]. Finally, LVBDD could also be useful in the field of *abstract interpretation* [4], where they would allow to manipulate more efficiently the abstract domains that mix Boolean variables (to encode some control state for instance), and some numerical abstract domain (to encode more fine-grained information about the numerical variables).

Syntactically, LVBDD are an extended form of MTBDD where each node is labeled with a lattice value. The lattice value associated to a path from the root of an LVBDD to a terminal node (which corresponds to a valuation of the Boolean variables) is computed by taking the *greatest lower bound* of the lattice values along the path. Let us provide a simple example; Fig. 1 illustrates a finite distributive lattice $\mathcal{L}_A$, along with an LVBF θ over the propositions c_1, c_2, c_3 and $\mathcal{L}_A$. Intuitively, the lattice $\mathcal{L}_A$ can be seen as the possible truth values of a logic that models the potential disagreement of three observers (for instance, $A_{\{1,2\}}$ represents the situation where the observers 1 and 2 say "true" while observer 3 says "false"). In that context, the function θ represents the "truth statement" of each of the three observers for each valuation. Two different LVBDD that represent θ are illustrated at Fig.2. By following the paths going from the root to a terminal node, one can easily check that both LVBDD represent the function θ.

The LVBDD of Fig. 2 illustrate the fact that several syntactically different LVBDD can represent the same LVBF. In practice, it is often desirable to have *canonical data structures* so we introduce two normal forms for LVBDD, which we respectively call *shared* (SNF) and *unshared* normal form (UNF). The unshared normal form is very similar to MTBDD as it requires that each non-terminal node be labelled with the largest value $\top$ of the lattice. On the other hand, the shared normal form can achieve a much more compact representation of LVBF by exploiting the structure of the lattice to share redundant information along its paths. We show that the SNF has the potential to be *exponentially more compact* than LVBDD in UNF at the price of worst-case-exponential algorithms for the disjunction and intersection. In Section 5, we provide experimental evidence that this exponential behavior is often avoided in practice.

Let us provide some intuition on the shared normal form of LVBDD, using the example of Fig 2. The main idea of the SNF is extremely simple: each node is always labeled with the *largest possible lattice value* that preserves the desired semantics. Let us denote by $\mathsf{paths}(n)$ the set of paths going from a node n to a terminal node, and let us denote by $\mathsf{value}(\pi)$ the greatest lower bound of the lattice values along a path π. Because of the semantics we have just sketched for LVBDD,one can see that the greatest possible label of a node n is exactly $\mathsf{lub}(n) \equiv \bigsqcup\{\mathsf{value}(\pi) \mid \pi \in \mathsf{paths}(n)\}$. In effect, the lattice value $\mathsf{lub}(n)$ *synthesizes the common information* shared by the subgraph rooted by n. This information can be further exploited by the nodes in the subgraph rooted by n in order to have a labeling that contains as few redundancies as possible. More formally, let ℓ be the label of n, and n' be a descendant of n labeled by ℓ'; it is easy to see that we can replace the label of n' by *the largest lattice value* ℓ'' such that $\ell \sqcap \ell'' = \ell'$. We show in Section 2 that this *factorization* operation is well-defined and corresponds to known lattice-theoretic notions. The reader can verify that by applying successive information-sharing and factorization steps repeatedly on the UNF LVBDD on the right of Fig. 2, we obtain the SNF LVBDD on the left. This SNF LVBDD is more compact because the information-sharing and factorization operations have allowed to increase the sharing in the resulting graph.

Related works While LVBDD represent functions of the form $\theta : 2^{\mathsf{P}} \mapsto \mathcal{L}$, other structures to represent similar but different kinds of functions have been studied in the literature. ROBDD [10], and some variants like ZBDD [1] and *Boolean Expression Diagrams* [2], encode purely Boolean functions $\theta : 2^{\mathsf{P}} \mapsto \{0, 1\}$. MTBDD [12] represent functions of the form $2^{\mathsf{P}} \mapsto D$, but do not exploit the structure of D when it is a lattice. *Edge-Shifted Decision Diagrams* [3] represent functions of the form $\mathcal{L}^{\mathsf{P}} \mapsto \mathcal{L}$ and have been applied to Multi-Valued Model Checking. Finally, *Lattice Automata* [11] represent functions of the form $\Sigma^* \mapsto \mathcal{L}$, where Σ is a finite alphabet.

Structure of the paper In Section 2, we the define some basic lattice-theoretic notions. In Section 3, we formally define LVBDD and their normal forms. In Section 4, we describe the algorithms to manipulate LVBDD and discuss their worst-case complexity. Finally Section 5 presents experimental evidence that LVBDD-based algorithms can outperform state-of-the-art tools in the context of finite-word LTL satisfiability. Due to lack of space, the proofs have been omitted. A technical report with the proofs and additional details is available at `http://www.antichains.be/atva2010/`.

2 Preliminaries

Boolean variables Let P be a finite set of Boolean propositions. A *valuation* (also called *truth assignment*) $v : \mathsf{P} \mapsto \{0,1\}$ is a function that associates a truth value to each proposition. We denote by 2^{P} the set of all valuations over P. For $v \in 2^{\mathsf{P}}$, $p \in \mathsf{P}$ and $i \in \{0,1\}$, we denote by $v|_{p=i}$ the valuation v' such that $v'(p) = i$ and $v'(p') = v(p')$ for all $p' \in \mathsf{P} \setminus \{p\}$.

Lattices A finite lattice is a tuple $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$ where L is a finite set of elements, $\sqsubseteq \subseteq L \times L$ is a partial order on L; $\top$ and $\bot$ are two elements from L such that: for all $x \in L : \bot \sqsubseteq x \sqsubseteq \top$; and for all x, y there exists a unique greatest lower bound denoted by $x \sqcap y$ and a unique least upper bound denoted by $x \sqcup y$. A finite lattice $\langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap, \rangle$ is *distributive* (FDL for short) iff, for all x, y, z in L: $(x \sqcap y) \sqcup z = (x \sqcup z) \sqcap (y \sqcup z)$ and $(x \sqcup y) \sqcap z = (x \sqcap z) \sqcup (y \sqcap z)$.

Example 1. The lattice of the introduction is $\langle L_A, \sqsubseteq_A, \top = A_{\{1,2,3\}}, \bot = A_\emptyset, \sqcup_A, \sqcap_A \rangle$ where $L_A = \{A_I \mid I \subseteq \{1,2,3\}\}$, $A_I \sqsubseteq_A A_K$ iff $I \subseteq K$, $A_I \sqcup_A A_K = A_{I \cup K}$ and $A_I \sqcap_A A_K = A_{I \cap K}$. Another example that will be useful in the sequel is the lattice $\mathcal{L}_{\mathsf{UC}(S)}$ (for a finite set S) of *upward-closed sets of cells of* S. We call a *cell* any finite subset of S. A set of cells U is *upward-closed* iff for any $c \in U$, for any c' s.t. $c \subseteq c'$: $c' \in U$ too. Given a finite set of cells C, we denote by $\uparrow C$ the upward-closure of C, i.e., the set $\{c' \mid \exists c \in C : c \subseteq c'\}$. We denote by $\mathsf{UC}(S)$ the set of all upward-closed sets of cells of S. Then, $\mathcal{L}_{\mathsf{UC}(S)} = \langle \mathsf{UC}(S), \subseteq, \{\emptyset\}, \{S\}, \cup, \cap \rangle$. These two lattices are clearly finite and distributive.

Lattice-valued Boolean functions Now, let us formalise the notion of *lattice-valued Boolean function*, which is the type of functions that we want to be able to represent and manipulate thanks to LVBDD. For a set of Boolean propositions P and an FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, a *lattice-valued Boolean function* (LVBF for short) over P and $\mathcal{L}$ is a function $\theta : 2^{\mathsf{P}} \mapsto L$. We denote by $\mathsf{LVBF}(\mathsf{P}, \mathcal{L})$ the set of all LVBF over Boolean propositions P and FDL $\mathcal{L}$. We use the shorthands $\theta_1 \sqcap \theta_2$ for $\lambda v \cdot \theta_1(v) \sqcap \theta_2(v)$; $\theta_1 \sqcup \theta_2$ for $\lambda v \cdot \theta_1(v) \sqcup \theta_2(v)$, d for $\lambda v \cdot d$ (with $d \in L$), p for $\lambda v \cdot$ if $v(p) = 1$ then $\top$ else $\bot$ and $\neg p$ for $\lambda v \cdot$ if $v(p) = 0$ then $\top$ else $\bot$ (with $p \in \mathsf{P}$). Given an LVBF θ over P, we denote its *existential quantification* by $\exists \mathsf{P} \cdot \theta \equiv \bigsqcup \{\theta(v) \mid v \in 2^{\mathsf{P}}\}$. For $p \in \mathsf{P}$ and $i \in \{0,1\}$, we define $\theta|_{p=i}$ as the LVBF $\theta' : 2^{\mathsf{P}} \mapsto L$ such that $\theta'(v) = \theta(v|_{p=i})$ for all $v \in 2^{\mathsf{P}}$. Finally, the *dependency set* $I_\theta \subseteq \mathsf{P}$ of an LVBF θ is the set of propositions over which θ depends; formally $I_\theta = \{p \in \mathsf{P} \mid \theta|_{p=0} \neq \theta|_{p=1}\}$.

Example 2. We consider again the lattice $\mathcal{L}_A$ (see Example 1). An example (using the shorthands defined above) of LVBF of the form $2^{\{c_1,c_2,c_3\}} \mapsto \mathcal{L}_A$ is $\theta' \equiv A_{\{1,3\}} \sqcap \big(c_2 \sqcup (\neg c_2 \sqcap A_{\{2,3\}})\big)$. For instance, $\theta'(\langle 1,0,0 \rangle) = \theta'(\langle 1,0,1 \rangle) = A_{\{1,3\}} \sqcap \big(\bot \sqcup (\top \sqcap A_{\{2,3\}})\big) = A_{\{1,3\}} \sqcap A_{\{2,3\}} = A_{\{3\}}$. Also, $\exists \{c_1, c_2, c_3\} \cdot \theta' = A_{\{1,3\}} \sqcup A_{\{3\}} = A_{\{1,3\}}$.

Recall from the introduction that LVBDD in shared normal form attempt to make the representation of LVBF more compact by *common information-sharing* and *factorization*. Let us formalize these intuitive notions. The idea of *common information-sharing* between lattice values $\ell_1, \ldots, \ell_n$ corresponds to the *least upper bound* $\ell_1 \sqcup \cdots \sqcup \ell_n$ of these values. This can be seen on node n in Fig. 2. All paths containing n correspond to valuations where $c_1 = 1$. In that case, the LVBF θ (see Fig. 1) returns either $A_{\{3\}}$ or $A_{\{1,3\}}$. Hence, n is labelled by the common information $A_{\{3\}} \sqcup A_{\{1,3\}} = A_{\{1,3\}}$. To formalize the idea of *factorization*, we need an additional lattice operator, namely the *relative pseudocomplement* [17]. In an FDL, the *pseudocomplement of x relative to y*, denoted by $x \rightarrow y$, is the *largest* lattice value z s.t. $z \sqcap x = y$. For instance, in $\mathcal{L}_A$, $A_{\{1,3\}} \rightarrow A_{\{3\}} = A_{\{2,3\}}$ (see Fig. 1). Thus, once n has been labelled by $A_{\{1,3\}}$, the label $A_{\{3\}}$ of its left child can be replaced by $A_{\{1,3\}} \rightarrow A_{\{3\}}$, i.e. $A_{\{2,3\}}$.

Relative pseudocomplement Let $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$ be a lattice. For any x, y in L we consider the set $\{z \mid z \sqcap x \sqsubseteq y\}$. If this set has a *unique* maximal element, we call this element the *pseudocomplement of x relative to y* and denote it by $x \rightarrow y$, otherwise $x \rightarrow y$ is undefined. We extend this notion to $\mathsf{LVBF}\,(\mathsf{P}, \mathcal{L})$ as follows. Let $x \in L$; if $x \rightarrow \theta(v)$ is defined for all $v \in 2^{\mathsf{P}}$, then $x \rightarrow \theta$ is defined as $\lambda v \cdot x \rightarrow \theta(v)$. Otherwise, $x \rightarrow \theta$ is undefined. In the case where $\mathcal{L}$ is an FDL, one can easily show that $x \rightarrow y$ is defined for any pair x, y. Moreover, when $x \sqsupseteq y$, $x \rightarrow y$ is the greatest element z such that $z \sqcap x = y$.

Lemma 1. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all $x, y \in L$: $x \rightarrow y$ is defined. Moreover, $y \sqsubseteq x$ implies that (i) $(x \rightarrow y) \sqcap x = y$ and that (ii) for all z such that $z \sqcap x = y$: $z \sqsubseteq (x \rightarrow y)$.*

3 Lattice-Valued Binary Decision Diagrams

In this section, we formally define the *lattice-valued binary decision diagrams* (LVBDD for short) data structure. An LVBDD is a symbolic representation of an LVBF. Syntactically speaking, each LVBDD is a directed, rooted, acyclic graph, whose nodes are labeled by two pieces of information: an index, and a lattice value. LVBDD are thus a strict generalization of *reduced ordered binary decision diagrams* [10] (ROBDD). LVBDD are also closely related to *multi-terminal binary decision diagrams* [12] (MTBDD) but do not generalize them since MTBDD can have arbitrary co-domains.

Definition 1. *Given a set of Boolean propositions $\mathsf{P} = \{p_1, \ldots, p_k\}$ and a finite distributive lattice $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, an* LVBDD *$n$ over P and $\mathcal{L}$ is: (i) either a* terminal LVBDD $\langle \mathsf{index}(n), \mathsf{val}(n) \rangle$ *where* $\mathsf{index}(n) = k + 1$ *and* $\mathsf{val}(n) \in L$*; or (ii) a* non-terminal LVBDD $\langle \mathsf{index}(n), \mathsf{val}(n), \mathsf{lo}(n), \mathsf{hi}(n) \rangle$*, where* $1 \leq \mathsf{index}(n) \leq k$, $\mathsf{val}(n) \in L$ *and* $\mathsf{lo}(n)$ *and* $\mathsf{hi}(n)$ *are (terminal or non-terminal)* LVBDD *such that* $\mathsf{index}(\mathsf{hi}(n)) > \mathsf{index}(n)$ *and* $\mathsf{index}(\mathsf{lo}(n)) > \mathsf{index}(n)$.

In the sequel, we refer to LVBDD also as *"*LVBDD *node"*, or simply *"node"*. For any non-terminal node n, we call $\mathsf{hi}(n)$ (resp. $\mathsf{lo}(n)$) the *high-child* (*low-child*) of n. We denote by $\mathsf{LVBDD}(\mathsf{P}, \mathcal{L})$ the set of all LVBDD over P and $\mathcal{L}$. Finally, the set $\mathsf{nodes}(n)$ of an LVBDD n is defined recursively as follows. If n is terminal, then $\mathsf{nodes}(n) = \{n\}$. Otherwise $\mathsf{nodes}(n) = \{n\} \cup \mathsf{nodes}(\mathsf{lo}(n)) \cup \mathsf{nodes}(\mathsf{hi}(n))$. The *number of nodes* of an LVBDD is denoted by $|n|$.

Semantics of LVBDD The semantics of LVBDD is an LVBF, as sketched in the introduction. Formally, the semantics is defined by the unary function $[\![\cdot]\!] : \mathsf{LVBDD}(\mathsf{P}, \mathcal{L}) \mapsto \mathsf{LVBF}\,(\mathsf{P}, \mathcal{L})$ such that for any $n \in \mathsf{LVBDD}(\mathsf{P}, \mathcal{L})$, $[\![n]\!] = \mathsf{val}(n)$ if n is terminal, and $[\![n]\!] = \mathsf{val}(n) \sqcap \Big((\neg p_{\mathsf{index}(n)} \sqcap [\![\mathsf{lo}(n)]\!]) \sqcup (p_{\mathsf{index}(n)} \sqcap [\![\mathsf{hi}(n)]\!]) \Big)$ otherwise.

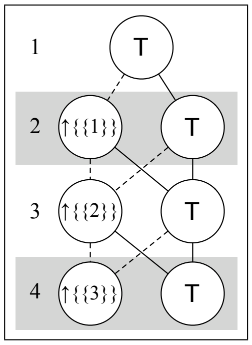
Isomorphisms and reduced LVBDD In order to share common subgraphs in LVBDD, we define a notion of *isomorphism* between LVBDD nodes. Let $n_1, n_2 \in \mathsf{LVBDD}(\mathsf{P}, \mathcal{L})$. We say that n_1 and n_2 are *isomorphic*, denoted by $n_1 \equiv n_2$, iff either (i) n_1 and n_2 are both terminal and $\mathsf{val}(n_1) = \mathsf{val}(n_2)$, or (ii) n_1 and n_2 are both non-terminal and $\mathsf{val}(n_1) = \mathsf{val}(n_2)$, $\mathsf{index}(n_1) = \mathsf{index}(n_2)$, $\mathsf{lo}(n_1) \equiv \mathsf{lo}(n_2)$ and $\mathsf{hi}(n_1) \equiv \mathsf{hi}(n_2)$. An LVBDD n is *reduced* iff (i) for all $n \in \mathsf{nodes}(n)$: either n is terminal or (i) $\mathsf{lo}(n) \neq \mathsf{hi}(n)$ and (ii) for all $n_1, n_2 \in \mathsf{nodes}(n)$: $n_1 \equiv n_2$ implies $n_1 = n_2$.

Normal Forms It is easy to see that there are LVBF θ for which one can find at least two different *reduced* LVBDD n_1 and n_2 s.t. $[\![n_1]\!] = [\![n_2]\!] = \theta$. For instance, the two LVBDD of Fig. 2 both represent the LVBF θ of Fig. 1. In order to obtain efficient algorithms to manipulate LVBDD, we define *normal forms* that associate to each LVBF a unique LVBDD representing it, up to isomorphism and order of the Boolean propositions. In this work, we define two normal forms for LVBDD: (i) the *unshared normal form* (UNF for short) which is similar to MTBDD, and (ii) the *shared normal form* (SNF for short) in which common lattice values along paths are shared. We associate to each LVBF θ, a unique LVBDD $\mathcal{D}^U(\theta)$ and a unique LVBDD $\mathcal{D}^S(\theta)$ which are respectively the UNF and SNF LVBDD representing θ:

Definition 2 (Unshared normal form). *Let* P *be a set of Boolean propositions and* $\mathcal{L}$ *be an FDL. Then, for all* $\theta \in \mathsf{LVBF}(\mathsf{P}, \mathcal{L})$*, the UNF* LVBDD $\mathcal{D}^U(\theta)$ *is the* reduced LVBDD *defined recursively as follows. If* $I_\theta = \emptyset$*, then* $\theta(v) = d$ *for some* $d \in L$*. In this case,* $\mathcal{D}^U(\theta)$ *is the terminal* LVBDD $\langle k+1, d\rangle$*. Otherwise, let* p_i *be the proposition of lowest index in* I_θ*. Then,* $\mathcal{D}^U(\theta)$ *is the non-terminal* LVBDD $\langle i, \top, \mathcal{D}^U(\theta|_{p_i=0}), \mathcal{D}^U(\theta|_{p_i=1})\rangle$.

Definition 3 (Shared normal form). *Let* P *be a set of Boolean propositions and let* $\mathcal{L}$ *be an FDL. Then, for all* $\theta \in \mathsf{LVBF}(\mathsf{P}, \mathcal{L})$*, the SNF* LVBDD $\mathcal{D}^S(\theta)$ *is the* reduced LVBDD *defined recursively as follows. If* $I_\theta = \emptyset$*, then* $\theta(v) = d$ *for some* $d \in L$*. In this case,* $\mathcal{D}^S(\theta)$ *is the terminal* LVBDD $\langle k+1, d\rangle$*. Otherwise, let* p_i *be the proposition of lowest index in* I_θ*. Then,* $\mathcal{D}^S(\theta)$ *is the non-terminal* LVBDD $\langle i, \exists \mathsf{P}\cdot\theta, \mathcal{D}^S((\exists \mathsf{P}\cdot\theta) \rightarrow (\theta|_{p_i=0})), \mathcal{D}^S((\exists \mathsf{P}\cdot\theta) \rightarrow (\theta|_{p_i=1}))\rangle$.

It is not difficult to see that for all LVBF θ, Definition 2 and 3 each yield a unique LVBDD $\mathcal{D}^U(\theta)$ and $\mathcal{D}^S(\theta)$. Then, an LVBDD n is in UNF iff $n = \mathcal{D}^U([\![n]\!])$. Similarly, n is in SNF iff $n = \mathcal{D}^S([\![n]\!])$. We denote by $\mathsf{LVBDD}^U(\mathsf{P}, \mathcal{L})$ (resp. $\mathsf{LVBDD}^S(\mathsf{P}, \mathcal{L})$) the set of all LVBDD in UNF (resp. SNF) on set P of Boolean propositions and FDL $\mathcal{L}$.



Example 3. Consider the family of lattice-valued Boolean functions $\theta_i : 2^{\{p_1,\dots p_i\}} \mapsto \mathcal{L}_{\mathsf{UC}(\{1,\dots,i\})}$ for $i \geq 1$, defined as $\theta_i \equiv \sqcap_{1\leq j\leq i}\big(p_j \sqcup \uparrow\{\{j\}\}\big)$. It is easy to see that, for any $i \geq 1$, the SNF LVBDD $\mathcal{D}^S(\theta_i)$ has $2 \times i + 1$ nodes. For instance, $\theta_3 = (p_1 \sqcup \uparrow\{\{1\}\}) \sqcap (p_2 \sqcup \uparrow\{\{2\}\}) \sqcap (p_3 \sqcup \uparrow\{\{3\}\})$ and $\mathcal{D}^S(\theta_3)$ is shown on the left. However, the corresponding MTBDD (or UNF LVBDD) is of exponential size, as for any $v \neq v'$, we have $\theta_i(v) \neq \theta_i(v')$.

Algorithm 1: Relative pseudocomplementation for LVBDD in SNF

```
1 begin PseudoCompSNF$(n, d)$
2     if index$(n) = k+1$ then return MK$(k+1, d \rightarrow$ val$(n)$, nil, nil$)$ ;
3     else
4         $d' := (d \rightarrow \mathsf{val}(n)) \sqcap (\mathsf{val}(\mathsf{lo}(n)) \sqcup (\mathsf{val}(\mathsf{hi}(n)))$ ;
5         return MK(index$(n), d'$, lo$(n)$, hi$(n))$ ;
6 end
```

4 Algorithms on LVBDD

In this section, we discuss *symbolic* algorithms to manipulate LVBF *via* their LVBDD representation. The proofs of these algorithms have been omitted here, but they can be found in the technical report. Remark however that we have managed to prove the correctness of the algorithms *for any finite distributive lattice*. This general result can be obtained by exploiting Birkhoff's representation theorem [17], a classical result in lattice theory. Birkhoff's theorem says that any FDL $\mathcal{L}$ is *isomorphic* to a lattice $\hat{\mathcal{L}}$ that has a special structure: its elements are sets of incomparable cells of certain elements of the original lattice $\mathcal{L}$ (the *meet-irreducible elements*). We can thus exploit this structure to prove the algorithms in the special case of $\hat{\mathcal{L}}$ and deduce the correctness of the algorithm on $\mathcal{L}$, thanks to the isomorphism. The interested reader is referred to the technical report for the complete details.

Throughout this section, we assume that we manipulate LVBDD ranging over the set of propositions $\mathsf{P} = \{p_1, \dots, p_k\}$ and the FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$. We present algorithms to compute the least upper bound $\sqcup$, the greatest lower bound $\sqcap$, the test for equality, the existential quantification of *all* the Boolean variables and the relative pseudocomplement with a lattice value for LVBDD in SNF. Then, we briefly sketch these operations for UNF LVBDD as they can be easily obtained from the definition, and are very similar to those for MTBDD.

Memory management and memoization The creation of LVBDD nodes in memory is carried out by function MK: calling $\mathsf{MK}(i, d, \ell, h)$ returns the LVBDD node $\langle i, d, \ell, h \rangle$. As in most BDD packages (see for instance [18]), our implementation exploits caching techniques to ensure that each unique LVBDD is stored only once, even across multiple diagrams. The implementation maintains a global cache that maps each tuple $\langle i, d, \ell, h \rangle$ to a memory address storing the corresponding LVBDD node (if it exists). A call to $\mathsf{MK}(i, d, \ell, h)$ first queries the cache and allocates fresh memory space for $\langle i, d, \ell, h \rangle$ in the case of a cache miss. Thus, MK guarantees that two isomorphic LVBDD always occupy the same memory address, but does not guarantee that any particular normal form is enforced. We assume that cache queries and updates take $\mathcal{O}(1)$ which is what is observed in practice when using a good hash map. We implemented a simple reference counting scheme to automatically free unreferenced nodes from the cache.

We use the standard *memoization* technique in graph traversal algorithms. Each algorithm has access to its own pair of memo! and memo? functions; $\mathsf{memo!}(key, value)$ stores a computed value and associates it to a key; $\mathsf{memo?}(key)$ returns the previously stored value, or nil if none was found. Both memo! and memo? run in $\mathcal{O}(1)$.

Algorithm 2: Meet of a LVBDD in SNF with a lattice value

```
begin ConstMeetSNF$(n, d)$
    $n' := \mathsf{memo?}(\langle n, d\rangle)$ ;
    if $n' \neq \mathsf{nil}$ then return $n'$ ;
    else if $\mathsf{val}(n) \sqsubseteq d$ then $n' := n$ ;
    else if $\mathsf{val}(n) \sqcap d = \bot$ then $n' := \mathsf{MK}(k+1, \bot, \mathsf{nil}, \mathsf{nil})$ ;
    else if $\mathsf{index}(n) = k+1$ then $n' := \mathsf{MK}(k+1, \mathsf{val}(n) \sqcap d, \mathsf{nil}, \mathsf{nil})$ ;
    else
        $\ell := \mathsf{ConstMeetSNF}(\mathsf{lo}(n), d)$ ;
        $h := \mathsf{ConstMeetSNF}(\mathsf{hi}(n), d)$ ;
        if $\ell = h$ then $n' := \mathsf{MK}(\mathsf{index}(\ell), \mathsf{val}(\ell) \sqcap \mathsf{val}(n), \mathsf{lo}(\ell), \mathsf{hi}(\ell))$ ;
        else
            $\ell' := \mathsf{PseudoCompSNF}(\ell, d)$ ; $h' := \mathsf{PseudoCompSNF}(h, d)$ ;
            $n' := \mathsf{MK}(\mathsf{index}(n), \mathsf{val}(n) \sqcap d, \ell', h')$ ;
    $\mathsf{memo!}(\langle n, d\rangle, n')$ ;
    return $n'$ ;
end
```

Operations on LVBDD in SNF The operations on SNF LVBDD and their complexities are summarized in Table 4. The procedure PseudoCompSNF(n, d) (see Algorithm 1) takes an LVBDD in SNF n and a lattice value $d \sqsupseteq \mathsf{val}(v)$, and computes the LVBDD n' in SNF such that $[\![n']\!] = d \rightarrow [\![n]\!]$. It runs in $\mathcal{O}(1)$, since it is sufficient to modify the label of the root. The resulting LVBDD is guaranteed to be still in SNF. This procedure will be invoked by the other algorithms to enforce canonicity.

The procedure ConstMeetSNF(n, d) (Algorithm 2), returns the SNF LVBDD representing $[\![n]\!] \sqcap d$, where d is a lattice value. ConstMeetSNF consists in recursively traversing the graph (the two recursive calls at lines 8 and 9 return the new subgraphs ℓ and h), and to call PseudoCompSNF on ℓ and h to enforce canonicity. This procedure runs in $\mathcal{O}(|n|)$, thanks to memoization. The procedure MeetSNF(n_1, n_2) (Algorithm 3) returns the SNF LVBDD representing $[\![n_1]\!] \sqcap [\![n_2]\!]$. Its execution is sketched in Fig. 3, when $\mathsf{index}(n_1) = \mathsf{index}(n_2)$ (the case $\mathsf{index}(n_1) \neq \mathsf{index}(n_2)$ is similar). The algorithm first performs two recursive calls on n_1 and n_2's respective lo- and hi- sons, which produces LVBDD ℓ and h (not shown on the figure), and computes the value $d = \mathsf{val}(n_1) \sqcap \mathsf{val}(n_2)$ (i.e., $d_1 \sqcap d_2$ on the figure), which will be the label of the root in the result. Then, canonicity of the result is enforced in two steps. First, the conjunction of ℓ and h with d is computed (second step in the figure). Second, the subgraphs returned by the recursive calls are *factorized* w.r.t. d, thanks to PseudoCompSNF (third step on the figure). Procedure JoinSNF(n_1, n_2) (Algorithm 4) returns the SNF LVBDD representing $[\![n_1]\!] \sqcup [\![n_2]\!]$. Its principle is similar to MeetSNF.

Both ConstMeetSNF and JoinSNF run in $2^{\mathcal{O}(|n_1|+|n_2|)}$ in the worst case. As an example of worst case for JoinSNF, consider, for any $n \geq 1$, the lattice $\mathcal{L}_{\mathsf{UC}(\{1,\ldots,2n\})}$ and the two LVBF $\theta_n \equiv \sqcap_{1 \leq j \leq n}\big(p_j \sqcup \uparrow\{\{j\}\}\big)$ and $\theta'_n \equiv \sqcap_{1 \leq j \leq n}\big(p_j \sqcup \uparrow\{\{n+j\}\}\big)$. It is easy to check that, for any $n \geq 1$, $|\mathcal{D}^S(\theta_n)| = |\mathcal{D}^S(\theta'_n)| = 2\,n+1$ (see Example 3), but that $\mathcal{D}^S(\theta_n \sqcup \theta'_n)$ has $2^{\mathcal{O}(|n_1|+|n_2|)}$ nodes. A similar example can be built for ConstMeetSNF.

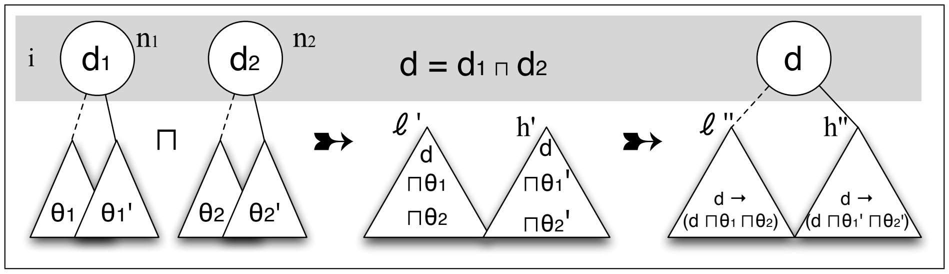


Fig. 3. The main steps of the computation of MeetSNF.

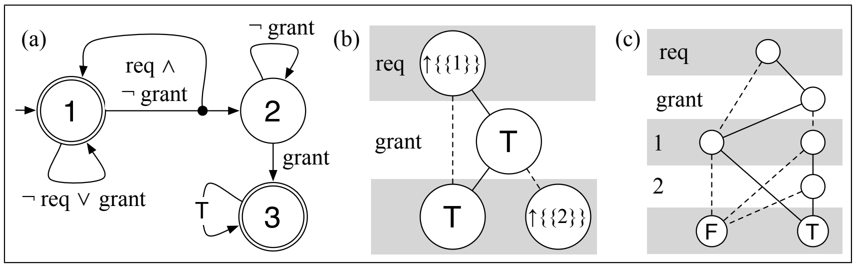


Fig. 4. An AFA (a) with the LVBDD (b) and ROBDD (c) encoding the transitions of location 1.

Operations on LVBDD in UNF For an UNF LVBDD n, $\exists \mathsf{P} : [\![n]\!] = \sqcup_v [\![n]\!](v)$ amounts to $\sqcup_{n' \in N} \mathsf{val}(n')$, where N is the set of terminal nodes of n. This operation is thus in $\mathcal{O}(|n|)$. The computation of the $\sqcap$ and $\sqcup$ operators is done by computing the synchronised product of the two diagrams, similarly to MTBDD [12]. For instance, when n_1 and n_2 are terminal LVBDD in UNF, the UNF LVBDD that represents $[\![n_1]\!] \sqcup [\![n_2]\!]$ is $\langle k+1, \mathsf{val}(n_1) \sqcup \mathsf{val}(n_2) \rangle$. When n_1 and n_2 are non-terminal LVBDD in UNF with the same index, the UNF LVBDD that represents $[\![n_1]\!] \sqcup [\![n_2]\!]$ is obtained by building recursively the LVBDD representing $[\![\mathsf{lo}(n_1)]\!] \sqcup [\![\mathsf{lo}(n_2)]\!]$ and $[\![\mathsf{hi}(n_1)]\!] \sqcup [\![\mathsf{hi}(n_2)]\!]$, and adding a root labelled by $\top$. With memoization, we can achieve polynomial complexity.

5 Empirical Evaluation.

In this section, we apply our new data structure to the *satisfiability problem* for the *finite-word linear temporal logic* (LTL for short). In recent works [22, 23], it has been shown that algorithms based on both *antichains* and ROBDD, can outperform purely ROBDD-based techniques like the ones implemented in the tools SMV and NuSMV. Our experiments show that an approach based on a combination of antichains and LVBDD in SNF can be even more efficient.

We solve the LTL satisfiability problem by the classical reduction to the language emptiness for *alternating automata* (AFA for short). AFA are a natural generalization of both *non-deterministic* and *universal* automata, as they use both conjunctive and disjunctive constraints to encode the transition relation. Due to lack of space, we do not define AFA formally but we illustrate their semantics on the example AFA of Fig. 4; this AFA has three locations $1, 2, 3$, with 1 being the *initial* location and $1, 3$ being

Algorithm 3: Meet of two LVBDD in SNF

```
begin MeetSNF($n_1, n_2$)
    $n'$ := memo?($\langle n_1, n_2\rangle$) ;
    if $n' \neq$ nil then return $n'$ ;
    else if index($n_1$) $= k+1$ then $n'$ := ConstMeetSNF($n_2$, val($n_1$));
    else if index($n_2$) $= k+1$ then $n'$ := ConstMeetSNF($n_1$, val($n_2$));
    else
        if index($n_1$) = index($n_2$) then
            $\ell$ := MeetSNF(lo($n_1$), lo($n_2$)) ; $h$ := MeetSNF(hi($n_1$), hi($n_2$)) ;
            $d$ := val($n_1$) $\sqcap$ val($n_2$) ;
        else
            if index($n_1$) > index($n_2$) then swap($n_1, n_2$) ;
            $\ell$ := MeetSNF(lo($n_1$), $n_2$) ; $h$ := MeetSNF(hi($n_1$), $n_2$) ;
            $d$ := val($n_1$) ;
        $\ell'$ := ConstMeetSNF($\ell, d$) ; $h'$ := ConstMeetSNF($h, d$) ;
        if $\ell' = h'$ then $n'$ := $\ell'$ ;
        else
            $d'$ := val($\ell'$) $\sqcup$ val($h'$) ;
            $\ell''$ := PseudoCompSNF($\ell', d'$) ; $h''$ := PseudoCompSNF($h', d'$) ;
            $n'$ := MK(index($n_1$), $d', \ell'', h''$) ;
    memo!($\langle n_1, n_2\rangle, n'$) ; memo!($\langle n_2, n_1\rangle, n'$) ;
    return $n'$ ;
end
```

the *accepting* locations. A *run* of an AFA is a sequence of sets of locations (called *configurations*); a run is accepting if it ends in a subset of the accepting locations, and is initial if it begins with a configuration that contains the initial location. The language of an AFA is defined as the set of finite words for which the automaton admits an initial accepting run. In the figure, the forked arrows depict the AFA's transitions. Reading a valuation v, the AFA can move from configuration c_1 to c_2 iff for each $\ell \in c_1$ there exists a transition from ℓ labeled by φ such that all target locations are in c_2 and $v \models \varphi$. For example, if the AFA reads the valuation $\mathrm{grant} = 0\ \mathrm{req} = 1$ from $\{1, 3\}$, it must go to $\{1, 2, 3\}$. Alternating automata enjoy the following useful property: the set of successor configurations of any configuration is always *upward-closed* for subset inclusion; this observation is the basis for the antichain-based approach to AFA analysis. We do not recall the framework of antichains here, but it can be found in [14].

The translation from LTL to AFA yields automata which have an alphabet equal to the set of *valuations* of the Boolean propositions of the LTL formula. It is easy to see that, in that case, the transition function of an AFA can be encoded with an LVBF over the set of propositions of the formula and the lattice of upward-closed sets of configurations of the automaton. For example, the LTL formula $\Box(\mathrm{req} \rightarrow \Diamond\mathrm{grant})$ translates to the automaton of Fig. 4 (a), and the LVBF corresponding to the outgoing transitions of location 1 is $\big(\uparrow\{\{1\}\} \sqcap (\neg\mathrm{req} \sqcup \mathrm{grant})\big) \sqcup \big(\uparrow\{\{1, 2\}\} \sqcap \mathrm{req} \sqcap \neg\mathrm{grant}\big)$.

We consider two encodings for the LVBF of AFA transitions. The first encoding uses LVBDD in shared normal form, while the second uses traditional ROBDD. The LVBDD encoding uses one decision variable per proposition of the formula. Each node

Algorithm 4: Join of two LVBDD in SNF

```
1  begin ReJoinSNF($n_1, n_2, d_1, d_2$)
2      $n'$ := memo?($\langle n1, n2, d_1, d_2\rangle$) ;
3      if $n' \neq$ nil then return $n'$;
4      $d'_1 := d_1 \sqcap$ val($n_1$) ; $d'_2 := d_2 \sqcap$ val($n_2$) ;
5      if index($n_1$) = index($n_2$) = $k + 1$ then $n'$ := MK($k + 1, d'_1 \sqcup d'_2$, nil, nil) ;
6      else
7          if index($n_1$) = index($n_2$) then
8              $\ell$ := ReJoinSNF(lo($n_1$), lo($n_2$), $d'_1, d'_2$) ;
               $h$ := ReJoinSNF(hi($n_1$), hi($n_2$), $d'_1, d'_2$) ;
9          else
10             if index($n_1$) > index($n_2$) then
11                 swap($n_1, n_2$) ; swap($d_1, d_2$) ; swap($d'_1, d'_2$) ;
12             $\ell$:= ReJoinSNF(lo($n_1$), $n_2, d'_1, d'_2$) ; $h$:= ReJoinSNF(hi($n_1$), $n_2, d'_1, d'_2$) ;
13         if $\ell = h$ then $n'$ := $\ell$ ;
14         else
15             $d$ := val($l$) $\sqcup$ val($h$) ;
16             $\ell$ := PseudoCompSNF($\ell, d$) ; $h$ := PseudoCompSNF($h, d$) ;
17             $n'$ := MK(index($n_1$), $d, \ell, h$) ;
18     memo!($\langle n_1, n_2, d_1, d_2\rangle, n'$) ; memo!($\langle n_2, n_1, d_2, d_1\rangle, n'$) ;
19     return $n'$ ;
20 end
21 begin JoinSNF($n_1, n_2$)
22     return ReJoinSNF($n_1, n_2, \top, \top$);
23 end
```

of these LVBDD is labeled with a lattice value, here an upward-closed set of configurations of the automaton. In this work, we encode these upward-closed sets with ROBDD; other encodings are possible (e.g., covering sharing trees [9]) but we do not discuss them here as this is orthogonal to our work. For the ROBDD encoding of LVBF of AFA transitions, we use one variable per proposition of the LTL formula and location of the automaton. Both encodings are illustrated at Fig. 4 (b) and (c).

We consider three series of parametric scalable LTL formulas: *mutex* formulas, *lift* formulas and *pattern* formulas. The mutex and lift formulas have been used previously as LTL benchmark formulas in [22], and the pattern formulas were used as benchmark formulas by Vardi *et al.* in [15] and previously in [16]. The presentation (given below) of the mutex and lift formulas has been slightly simplified due to space limitations; all the benchmarks in their complete form are available for download at `http://www.antichains.be/atva2010/`.

For each set of formulas, we supply an *initial ordering* of the propositions. Providing a sensible initial ordering is critical to the fair evaluation of BDD-like structures, as these are known to be very sensitive to variable ordering.

The mutex formulas describe the behavior of n concurrent processes involved in a mutual exclusion protocol. The proposition c_i indicates that process i is in its critical section, r_i that it would like to enter the critical section, and d_i that it has completed its execution. The initial ordering on the propositions is $r_1, c_1, d_1, \ldots, r_n, c_n, d_n$. We

Operation	LVBDD **type**	**Procedure**	**Time Complexity**	**Maximum Size**
$[\![n_1]\!] \sqcap [\![n_2]\!]$	UNF	similar to MTBDD	$\mathcal{O}(\lvert n_1\rvert\, \lvert n_2\rvert)$	$\lvert n_1\rvert\, \lvert n_2\rvert$
$[\![n_1]\!] \sqcup [\![n_2]\!]$	UNF	similar to MTBDD	$\mathcal{O}(\lvert n_1\rvert\, \lvert n_2\rvert)$	$\lvert n_1\rvert\, \lvert n_2\rvert$
$\exists \mathsf{P} : [\![n]\!]$	UNF	bottom-up propagation	$\mathcal{O}(\lvert n\rvert)$	-
$[\![n_1]\!] \sqcap [\![n_2]\!]$	SNF	$\mathsf{MeetSNF}(n_1, n_2)$	$2^{\mathcal{O}(\lvert n_1\rvert + \lvert n_2\rvert)}$	$2^{(\lvert I_{[\![n_1]\!]} \cup I_{[\![n_2]\!]}\rvert + 1)} - 1$
$[\![n_1]\!] \sqcup [\![n_2]\!]$	SNF	$\mathsf{JoinSNF}(n_1, n_2)$	$2^{\mathcal{O}(\lvert n_1\rvert + \lvert n_2\rvert)}$	$2^{(\lvert I_{[\![n_1]\!]} \cup I_{[\![n_2]\!]}\rvert + 1)} - 1$
$d \rightarrow [\![n]\!]$	SNF	$\mathsf{PseudoCompSNF}(n, d)$	$\mathcal{O}(1)$	$\lvert n\rvert$
$\exists \mathsf{P} : [\![n]\!]$	SNF	root inspection	$\mathcal{O}(1)$	-

Table 1. Time complexity and maximum output size of LVBDD algorithms.

check that $\mu(n) \wedge \neg(\Diamond r_1 \rightarrow \Diamond d_1)$ is unsatisfiable.

$$\mu(n) \equiv \bigwedge_{i=1}^{n} \big(\Box(c_i \rightarrow \bigwedge_{j \neq i} \neg c_j) \wedge \Box(r_i \rightarrow \Diamond c_i) \wedge \Box(c_i \rightarrow \Diamond \neg c_i) \wedge \neg d_i \wedge \Box((c_i \wedge X \neg c_i) \rightarrow d_i)\big)$$

The lift formula describes the behavior of a lift system with n floors. The proposition b_i indicates that the button is lit at floor i, and f_i that the lift is currently at floor i. The initial variable ordering is $b_1, f_1, \ldots, b_n, f_n$. We check that $\lambda(n) \wedge \neg(b_{n-1} \rightarrow (\neg f_{n-1}\, U f_n))$ is unsatisfiable.

$$\lambda(n) \equiv f_1 \wedge \bigwedge_{i=1}^{n} \Big(\Box(b_i \rightarrow (b_i\, U f_i) \wedge \Box\big(f_i \rightarrow (\neg b_i \wedge \bigwedge_{j \neq i}^{n} \neg f_j \wedge \neg X f_j) \wedge (\bigvee_{i-1 \leq j \leq i+1} X X f_j)\big)\Big)$$

The pattern formulas of [15] are found below, and their initial proposition ordering is set to $p_1, \ldots, p_n$.

$$E(n) = \textstyle\bigwedge_{i=1}^{n} \Diamond p_i \qquad Q(n) = \textstyle\bigwedge_{i=1}^{n-1} (\Diamond p_i \vee \Box p_{i+1})$$
$$U(n) = (\ldots(p_i\, U p_2)\, U \ldots)\, U p_n \qquad S(n) = \textstyle\bigwedge_{i=1}^{n} \Box p_i$$
$$R(n) = \textstyle\bigwedge_{i=1}^{n} (\Box\Diamond p_i \vee \Diamond\Box p_{i+1})$$

In order to evaluate the practical performances of LVBDD, we have implemented two nearly identical C++ prototypes, which implement a simple antichain-based forward fixpoint computation [22, 23] to solve the satisfiability problem for finite word LTL. These two prototypes differ only in the encoding of the LVBF of the AFA transitions: one uses LVBDD while the other uses the BuDDy [18] implementation of ROBDD with the SIFT reordering method enabled. On the other hand, a recent survey by Vardi and Rozier [15] identifies NuSMV [8] as one of the best tools available to solve this problem [1]. NuSMV implements many optimization such as *conjunctive-clustering* of the transition relation and *dynamic reordering* of the BDD variables, so we believe that NuSMV provides an excellent point of comparison for our purpose. The use of NuSMV for finite-word LTL satisfiabilty is straightforward: we translate the formula into an AFA, which is then encoded into an SMV module with one input variable (`IVAR`) per proposition and one state variable (`VAR`) per location of the automaton. To check for satisfiability, we ask NuSMV to verify the property "`CTLSPEC AG !accepting`" where `accepting` is a formula which denotes the set of accepting configurations of

[1] Vardi et al. considered *infinite-word* LTL, but this applies just as well to finite-word LTL.

the automaton. We invoke NuSMV with the "`-AG`" and "`-dynamic`" command-line options which respectively enable a single forward reachability computation and dynamic reordering of BDD variables. We now present two sets of experiments which illustrate the practical efficiency of LVBDD in terms of *running time* and *compactness*.

Running time comparison In our first set of experiments, we have compared the respective running times of our prototypes and NuSMV on the benchmarks described above. These results are reported in Table 2, where we highlight the best running times in bold.

It is well-known that ordered decision diagrams in general are very sensitive to the ordering of the variables. In practice, ROBDD packages implement *dynamic variable reordering techniques* to automatically avoid bad variable orderings. However, these techniques are known to be sensitive to the *initial variable ordering*, so a sensible initial ordering is a necessary component to the fair evaluation of ordered decision diagrams. In our experiments, we have two sets of variables which respectively encode the LTL propositions and the AFA locations. We provide an initial sensible ordering for both sets of variables; the LTL propositions are initially ordered as described previously, and the AFA locations are ordered by following a topological sort[2]. Finally, for the ROBDD-based tools, we provide an initial ordering such that the LTL propositions variables *precede* the AFA location variables. In Table 2, the "NuSMV + ord" and "NuSMV" columns respectively contain the running times of NuSMV when provided with our initial ordering, or without any initial ordering.

On most of the examples, the LVBDD-based prototype performs better than NuSMV and the ROBDD prototype. For the *mutex* and *lift* benchmarks, LVBDD seem to scale much better than ROBDD. We have investigated the scalability of ROBDD on these instances with profiling tools, which revealed that a huge proportion of the run time is spent on variable reordering. Disabling dynamic reordering for either the ROBDD-based prototype or NuSMV on these instances made matters even worse, with neither NuSMV nor our ROBDD-based prototype being able to solve them for parameter values beyond 30. These observations shed light on one of the key strengths of LVBDD in the context of LVBF representation. While ROBDD-based encodings must find a *suitable interleaving* of the domain and co-domain variables, which can be very costly, LVBDD avoid this issue altogether, even when co-domain values are encoded using ROBDD.

Finally, the results for the pattern formulas confirm earlier research [23] by showing that the antichain approach (i.e., columns ROBDD, LVBDD) and the fully-symbolic approach (NuSMV in our case) exhibit performance behaviors that are incomparable in general; in the Q benchmark, the antichains grow exponentially in length, while the S benchmark makes the ROBDD reordering-time grow exponentially.

Compactness comparison In this set of experiments, we compare the compactness of LVBDD and ROBDD when encoding LVBF occurring along the computation of the fixed point that solves the satisfiability for the *lift* formulas. These experiments are reported in Table 3. We report on the largest and average structure sizes encountered along the fixed point. We performed the experiments for ROBDD both with and without dynamic reordering enabled, and for two different reordering techniques provided in the BuDDy package: SIFT and WIN2. The sizes reported for LVBDD is equal to the

[2] This ordering is sensible because the translation from LTL produces AFA that are *very weak*.

number of decision nodes of the LVBDD *plus* the number of unique ROBDD nodes that are used to encode the lattice values labelling the LVBDD. This metric is thus an *accurate* representation of the total memory footprint of LVBDD and is *fair* for the comparison with ROBDD. These experiments show that, as expected, LVBDD are more compact than ROBDD in the context of LVBF representation, although ROBDD can achieve sizes that are comparable with LVBDD, but at the price of a potentially very large reordering overhead. This increased compactness explains the better running times of the LVBDD prototype reported in Table 2.

All experiments were performed with a timeout of 1000 seconds on an Intel Core i7 3.2 Ghz CPU with 12 GB of RAM. A preliminary version of our C++ LVBDD library is freely available at `http://www.ulb.ac.be/di/ssd/nmaquet/#research`.

References

1. Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In DAC'93, ACM, 1993.
2. Reif Andersen, H. and Hulgaard, H.: Boolean Expression Diagrams. In LICS, IEEE, 1997.
3. Devereux, B. and Chechik, M.: Edge-Shifted Decision Diagrams for Multiple-Valued Logic. In JMVLSC, Old City Publishing, 2003.
4. Cousot, P. and Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In POPL'77, ACM, 1977.
5. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000).
6. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, J.: Symbolic Model Checking: 10^{20} States and Beyond. In LICS'90, IEEE, 1990.
7. Chechik, M., Devereux, B., Easterbrook, S., Lai, A. and Petrovykh, V. Efficient Multiple-Valued Model-Checking Using Lattice Representations. In CONCUR'01, LNCS 2154, Springer, 2001.
8. Cimatti, A., Clarke, E.M., Giunchiglia, F., Roveri, M.: NuSMV: A new symbolic model verifier. In CAV'99, LNCS 1633, Springer, 1999.
9. Delzanno, G., Raskin, J-.F., Van Begin, L.: Covering sharing trees: a compact data structure for parameterized verification. STTT vol. 5, num. 2-3, Springer, 2003.
10. Bryant, R.: Graph-based Algorithms for Boolean Function Manipulation. IEEE Trans. on Comp. C-35(8) (1986).
11. Kupferman, O., Lustig, Y.: Lattice Automata. In VMCAI'07, LNCS 4349, Springer, 2007.
12. Fujita, M., McGeer, P.C., Yang, J.C.Y.: Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. Form. Methods Syst. Des. **10**(2-3) (1997).
13. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.F.: Antichains: A new algorithm for checking universality of finite automata. In CAV'06, LNCS 4144, Springer, 2006.
14. Doyen, L., Raskin, J.F.: Antichain Algorithms for Finite Automata. In TACAS'10, LNCS 6015, Springer, 2010.
15. Rozier, K., Vardi, M.: LTL Satisfiability Checking. In SPIN'07, LNCS 4595, Springer, 2007.
16. Geldenhuys J., Hansen H.: Larger automata and less work for LTL model checking. In MCS'06, LNCS 3925, Springer, 2006.
17. Birkhoff, G.: Lattice Theory. Colloquim Publications. Am. Math. Soc. (1999)
18. Lind-Nielsen, J.: Buddy: BDD package. `http://www.itu.dk/research/buddy`
19. NuSMV Model-checker. `http://nusmv.irst.itc.it/`
20. SMV Model-checker. `http://www.cs.cmu.edu/~modelcheck/smv.html`
21. Somenzi, F.: BDD package CUDD. `http://vlsi.colorado.edu/~fabio/CUDD/`
22. Ganty, P., Maquet, N., Raskin, J.F.: Fixpoint Guided Abstraction Refinements for Alternating Automata. In of CIAA'09, LNCS 5642, Springer, 2009.
23. De Wulf, M., Doyen, L., Maquet, N., Raskin, J.F.: Antichains: Alternative Algorithms for LTL Satisfiability. In of TACAS'08, LNCS 4963, Springer, 2008.

Table 2. Running times in seconds. 'n' is the parameter of the formula, 'locs' the number of locations of the AFA, 'props' the number of propositions of the LTL formula and 'iters' the number of iterations of the fixpoint. Time out set at 1,000 seconds.

	n	locs	props	iters	ROBDD	LVBDD	NuSMV + ord	NuSMV
Mutex	40	328	121	3	12	**2**	12	11
	80	648	241	3	74	**12**	31	34
	120	968	361	3	284	**37**	87	180
	160	1288	481	3	t.o.	**79**	206	325
	200	1608	601	3	-	**132**	t.o.	t.o.
Lift	20	98	42	41	1	**1**	1	1
	40	178	82	81	10	**3**	6	10
	60	258	122	121	36	**8**	24	44
	80	338	162	161	83	**17**	53	162
	100	418	202	201	188	**31**	185	581
	120	498	242	241	330	**51**	341	t.o.
	140	578	282	281	t.o.	**81**	t.o.	-
E	100	103	101	2	12	**0.1**	1	1
	200	203	201	2	110	**0.3**	4	4
	300	303	301	2	392	**0.6**	19	19
U	100	102	101	2	**1**	1	2	2
	200	202	201	2	**7**	12	23	19
	300	302	301	2	**39**	44	117	116
R	6	27	8	2	0.2	0.4	**0.1**	**0.1**
	8	35	10	2	20	21	**0.1**	**0.1**
	10	43	12	2	t.o.	t.o.	**0.1**	**0.1**
Q	10	23	12	2	1	1	**0.1**	**0.1**
	15	33	17	2	205	234	**0.1**	**0.1**
	20	43	22	2	t.o.	t.o.	**0.2**	**0.1**
S	200	203	201	2	**0.1**	**0.1**	3	4
	300	303	301	2	**0.1**	**0.3**	6	11
	400	403	401	2	**0.1**	**0.4**	16	25

Table 3. Comparison of ROBDD and LVBDD sizes on the lift example.

						ROBDD					
				LVBDD		no reord.		SIFT		WIN2	
n	locs	props	iters	avg.	max.	avg.	max.	avg.	max.	avg.	max.
10	58	22	21	58	284	2,374	8,190	2,374	8,190	2,374	8,190
12	66	26	25	65	300	9,573	32,766	682	32,766	695	32,766
14	74	30	29	75	306	t.o.	t.o.	269	33,812	1,676	131,070
50	218	102	101	201	654	t.o.	t.o.	270	1,925	t.o.	t.o.
100	418	202	201	376	1,304	t.o.	t.o.	469	1,811	t.o.	t.o.

A Relative Pseudo-Complement $\rightarrow$

In this appendix, we detail the algebraic properties of $\rightarrow$. We first need to introduce some definitions which were not given in the preliminaries.

A.1 Additional definitions

Meet-irreducible element An element x of a lattice $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$ is *meet-irreducible* iff it is not $\top$ and for all y, z if $x = y \sqcap z$ then $x = y$ or $x = z$. We denote by $\mathsf{MI}(\mathcal{L})$ the set of meet-irreducible elements of $\mathcal{L}$.

Closed sets and antichains Let S be a finite set and $\sqsubseteq$ a partial order on S. The *upward-closure* of a set $X \subseteq S$, denoted by $\uparrow X$, is the set of elements of S which are $\sqsubseteq$-greater than some element of X, or formally: $\uparrow X = \{s \in S \mid \exists x \in X : x \sqsubseteq s\}$. Similarly, the *downward-closure* is defined as: $\downarrow X = \{s \in S \mid \exists x \in X : x \sqsupseteq s\}$. A set $X \subseteq S$ is *upward-closed* iff $X = \uparrow X$ and *downward-closed* iff $X = \downarrow X$. We denote by $\mathsf{UC}\,(S)$ the set of all the upward-closed sets $X \subseteq S$, and by $\mathsf{DC}(S)$ the set of all the downward-closed sets $X \subseteq S$. We define the *lattice of upward-closed sets of elements of S* as $\mathcal{L}_{\mathsf{UC}(S)} = \langle \mathsf{UC}\,(S), \subseteq, S, \emptyset, \cup, \cap \rangle$. It is easy to see that $\mathcal{L}_{\mathsf{UC}(S)}$ respects the definition of a lattice, and that it is an FDL. For any set $X \subseteq S$, X is an $\sqsubseteq$*-antichain* iff X contains only pairwise incomparable elements, i.e., for all $x, y \in X$: $x \not\sqsubseteq y$. We denote by $\mathsf{ATC}^{\sqsubseteq}(S)$ the set of all the $\sqsubseteq$-antichains of elements from S. Given an upward-closed set $X \subseteq S$, we denote by $\lfloor X \rfloor$ the (unique) antichain $\{x \in X \mid \nexists x' \in X : x' \sqsubset x\}$. Clearly, $\uparrow\lfloor X \rfloor = X$. Moreover, $\lfloor X \rfloor$ is a canonical representation of X. Similarly, for a downward-closed set $X \subseteq S$, we let $\lceil X \rceil$ be the (unique) antichain $\{x \in X \mid \nexists x' \in X : x' \sqsupset x\}$. Again, $\downarrow\lceil X \rceil = X$ and $\lceil X \rceil$ is a canonical representation of X.

A.2 Algebraic properties of $\rightarrow$ in finite distributive lattices

The algebraic properties of $\rightarrow$ needed to prove the correctness of our algorithms are given hereunder in Proposition 1. At first glance, these properties seem hard to establish given the definition of the $\rightarrow$ operator. Fortunately, we only need properties of $x \rightarrow y$ in the special case where $x \sqsupseteq y$, because all of our algorithms make that assumption. Moreover, there exists a classical result in lattice theory, namely *Birkhoff's representation theorem*, which allows to build for any FDL $\mathcal{L}$ an FDL $\hat{\mathcal{L}}$ which is isomorphic to $\mathcal{L}$, and for which $x \rightarrow y$ admits a very simple definition when $x \sqsupseteq y$. Indeed we show that computing $\rightarrow$ amounts to a simple *set-wise difference* in the Birkhoff lattice. This powerful result allows us to prove the desired properties of $\rightarrow$ in a most general and elegant fashion: since $\hat{\mathcal{L}}$ is isomorphic to $\mathcal{L}$, all properties which can be proved true in the Birkhoff lattice are true for any FDL. We first recall Birkhoff's theorem, then prove that $\rightarrow$ can be defined in terms of a set-wise difference in the isomorphic lattice, and finally proceed the proofs of the properties of $\rightarrow$.

Birkhoff's representation theorem Let $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$ be an FDL. The Birkhoff representation of $\mathcal{L}$ is denoted by $\hat{\mathcal{L}}$ and is defined as the lattice $\hat{\mathcal{L}} = \langle \hat{L}, \hat{\sqsubseteq}, \hat{\top}, \hat{\bot}, \hat{\sqcup}, \hat{\sqcap} \rangle$ where: $\hat{L} = \mathsf{ATC}^{\sqsubseteq}(\mathsf{MI}(\mathcal{L}))$; for all $X, Y \in \hat{L}$: $X \hat{\sqsubseteq} Y$ iff for all $y \in Y$, there exists $x \in X$: $x \sqsubseteq y$; $\hat{\bot} = \mathsf{MI}(\mathcal{L})$; $\hat{\top} = \emptyset$; for all $X, Y \in \hat{L}$: $X \hat{\sqcap} Y = \lfloor X \cup Y \rfloor$ and

$X \mathbin{\hat{\sqcup}} Y = \lfloor\{x \sqcup y \mid x \in X \text{ and } y \in Y\}\rfloor$. It is easy to check that $\hat{\mathcal{L}}$ is an FDL [17]. Given an FDL $\mathcal{L}$ and its Birkhoff representation $\hat{\mathcal{L}}$, we define $\sigma_{\mathcal{L}} : L \mapsto \hat{L}$ as the function s.t. for any $x \in L$: $\sigma_{\mathcal{L}}(x) = \lfloor\{z \in \mathsf{MI}(\mathcal{L}) \mid z \sqsupseteq x\}\rfloor$. It is easy to check that $\sigma_{\mathcal{L}}$ is in fact a bijection, and that for all $X \in \hat{L}$: $\sigma_{\mathcal{L}}^{-1}(X) = \bigsqcap X$. Birkhoff's representation theorem states that $\sigma_{\mathcal{L}}$ defines an isomorphism between $\mathcal{L}$ and $\hat{\mathcal{L}}$ in the following sense:

Theorem 1 ([17]). *Let $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap\rangle$ be an FDL and let $\hat{\mathcal{L}} = \langle \hat{L}, \hat{\sqsubseteq}, \hat{\top}, \hat{\bot}, \hat{\sqcup}, \hat{\sqcap}\rangle$ be its Birkhoff representation. Then: (i) $\sigma_{\mathcal{L}}(\top) = \hat{\top}$; (ii) $\sigma_{\mathcal{L}}(\bot) = \hat{\bot}$; (iii) for all $x, y \in L$: $\sigma_{\mathcal{L}}(x \sqcup y) = \sigma_{\mathcal{L}}(x) \mathbin{\hat{\sqcup}} \sigma_{\mathcal{L}}(y)$; (iv) for any $x, y \in L$: $\sigma_{\mathcal{L}}(x \sqcap y) = \sigma_{\mathcal{L}}(x) \mathbin{\hat{\sqcap}} \sigma_{\mathcal{L}}(y)$; (v) for all $x, y \in L$: $x \sqsubseteq y$ iff $\sigma_{\mathcal{L}}(x) \mathrel{\hat{\sqsubseteq}} \sigma_{\mathcal{L}}(y)$.*

Lemma 2. *For any FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap\rangle$, and for any $x, y \in L$ with $x \sqsupseteq y$: $\sigma_{\mathcal{L}}(x \rightarrow y) = \sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x)$.*

Proof. Since $\sigma_{\mathcal{L}}$ is a bijection, we can show equivalently that $x \rightarrow y = \sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x))$. By definition of $\rightarrow$ we have to show that $\sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x))$ is the $\sqsubseteq$-maximal element of $\{z \mid z \sqcap x \sqsubseteq y\}$ (which exists and is unique by Lemma 1). For that purpose, we first show that $\sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x))$ belongs to $\{z \mid z \sqcap x \sqsubseteq y\}$, i.e. that $\sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x)) \sqcap x \sqsubseteq y$. This is obtained as follows:

$$\begin{array}{lr}
\quad \sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x)) \sqcap x & \\
= \sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x) \mathbin{\hat{\sqcap}} \sigma_{\mathcal{L}}(x)) & \text{By Theorem 1} \\
= \sigma_{\mathcal{L}}^{-1}(\lfloor\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x) \cup \sigma_{\mathcal{L}}(x)\rfloor) & \text{By def. of } \hat{\sqcap} \\
= \sigma_{\mathcal{L}}^{-1}(\lfloor\sigma_{\mathcal{L}}(y) \cup \sigma_{\mathcal{L}}(x)\rfloor) & \\
= \sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \mathbin{\hat{\sqcap}} \sigma_{\mathcal{L}}(x)) & \text{By Def. of } \hat{\sqcap} \\
= y \sqcap x & \text{By Theorem 1} \\
= y & \text{Since } y \sqsubseteq x
\end{array}$$

Then, we show that $\sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x))$ is the maximal element of $\{z \mid z \sqcap x \sqsubseteq y\}$, i.e. that, for all $z \in L$ s.t. $z \sqcap x = y$: $z \sqsubseteq \sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x))$. This is done by contradiction: assume that there is $z \in L$ s.t. $z \sqcap x = y$ and $z \not\sqsubseteq \sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x))$. Thus, z is s.t.:

$$\begin{array}{lr}
\quad z \sqcap x = y \text{ and } z \not\sqsubseteq \sigma_{\mathcal{L}}^{-1}(\sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x))) & \\
\Leftrightarrow \sigma_{\mathcal{L}}(z) \mathbin{\hat{\sqcap}} \sigma_{\mathcal{L}}(x) = \sigma_{\mathcal{L}}(y) \text{ and } \sigma_{\mathcal{L}}(z) \mathrel{\hat{\not\sqsubseteq}} \sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x) & (a) \\
\Leftrightarrow \lfloor\sigma_{\mathcal{L}}(z) \cup \sigma_{\mathcal{L}}(x)\rfloor = \sigma_{\mathcal{L}}(y) \text{ and } \exists e \in \sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x) : \forall e' \in \sigma_{\mathcal{L}}(z) : e \not\sqsubseteq e' & (b) \\
\Rightarrow \lfloor\sigma_{\mathcal{L}}(z) \cup \sigma_{\mathcal{L}}(x)\rfloor = \sigma_{\mathcal{L}}(y) \text{ and } \exists e \in \sigma_{\mathcal{L}}(y) \setminus \sigma_{\mathcal{L}}(x) : e \notin \sigma_{\mathcal{L}}(z) & \\
\Rightarrow \sigma_{\mathcal{L}}(z) \cup \sigma_{\mathcal{L}}(x) \supseteq \sigma_{\mathcal{L}}(y) \text{ and } \sigma_{\mathcal{L}}(y) \not\subseteq \sigma_{\mathcal{L}}(z) \cup \sigma_{\mathcal{L}}(x) & (c)
\end{array}$$

Point (a) is shown by Theorem 1, (b) by definition of $\hat{\sqcap}$, and (c) by definition of $\lfloor\cdot\rfloor$ and $\subseteq$. The last line is clearly a contradiction, and such a z cannot exist. $\square$

We now use Lemma 2 to prove the algebraic properties on which our algorithms rely.

Proposition 1. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, and set of propositions* P*, for all* x*,* y *in* L*, for all* $\theta \in \mathsf{LVBF}(\mathsf{P}, \mathcal{L})$*:*

$$x \rightarrow (x \sqcap \theta) = \theta \textit{ implies } (x \sqcap y) \rightarrow (x \sqcap y \sqcap \theta) = y \rightarrow (y \sqcap \theta) \quad (1)$$

$$x \rightarrow (x \sqcap \theta) = \theta \textit{ and } y \sqsupseteq x \textit{ implies } y \rightarrow (x \sqcap \theta) = (y \rightarrow x) \sqcap \theta \quad (2)$$

$$x \rightarrow (x \sqcap \theta) = \theta \textit{ and } y \sqsupseteq x \textit{ implies } y \rightarrow (y \sqcap \theta) = \theta \quad (3)$$

$$x \rightarrow (y \rightarrow (x \sqcap y \sqcap \theta)) \textit{ equals } (x \sqcap y) \rightarrow (x \sqcap y \sqcap \theta) \quad (4)$$

Proof. We prove these three properties independentely in Corollary 1, Corollay 2 and Corollay 3, hereunder.

Lemma 3. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all x, y, z in L,* if $x \rightarrow (x \sqcap y) = y$ then $(x \sqcap z) \rightarrow (x \sqcap y \sqcap z) = z \rightarrow (y \sqcap z)$.

Proof. Let us assume that $\hat{\mathcal{L}} = \langle \hat{L}, \hat{\sqsubseteq}, \hat{\top}, \hat{\bot}, \hat{\sqcup}, \hat{\sqcap} \rangle$, and let x, y and z be three elements from L. Throughout the proof, we will denote, $\sigma_{\mathcal{L}}(x)$, $\sigma_{\mathcal{L}}(y)$ and $\sigma_{\mathcal{L}}(z)$ respectively by $\hat{x}$, $\hat{y}$ and $\hat{z}$, in order to alleviate the notations. Let us first rephrase the statement of the Lemma by exploiting the bijection $\sigma_{\mathcal{L}}$:

$$\begin{aligned}
&\text{if } x \rightarrow (x \sqcap y) = y \text{ then } (x \sqcap z) \rightarrow (x \sqcap y \sqcap z) = z \rightarrow (y \sqcap z)\\
\Leftrightarrow\ &\text{if } \sigma_{\mathcal{L}}(x \rightarrow (x \sqcap y)) = \hat{y} \text{ then } \sigma_{\mathcal{L}}((x \sqcap z) \rightarrow (x \sqcap y \sqcap z)) = \sigma_{\mathcal{L}}(z \rightarrow (y \sqcap z))\\
&\hfill \sigma_{\mathcal{L}} \text{ is a bijection}\\
\Leftrightarrow\ &\text{if } \sigma_{\mathcal{L}}(x \sqcap y) \setminus \hat{x} = \hat{y} \text{ then } \sigma_{\mathcal{L}}(x \sqcap y \sqcap z) \setminus \sigma_{\mathcal{L}}(x \sqcap z) = \sigma_{\mathcal{L}}(y \sqcap z) \setminus \hat{z}\\
&\hfill \text{By Lemma 2}\\
\Leftrightarrow\ &\text{if } \left(\hat{x} \mathbin{\hat{\sqcap}} \hat{y}\right) \setminus \hat{x} = \hat{y} \text{ then } \left(\hat{x} \mathbin{\hat{\sqcap}} \hat{y} \mathbin{\hat{\sqcap}} \hat{z}\right) \setminus \left(\hat{x} \mathbin{\hat{\sqcap}} \hat{z}\right) = \left(\hat{y} \mathbin{\hat{\sqcap}} \hat{z}\right) \setminus \hat{z}\\
&\hfill \text{By Theorem 1}\\
\Leftrightarrow\ &\text{if } \lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{x} = \hat{y} \text{ then } \lfloor \hat{x} \cup \hat{y} \cup \hat{z} \rfloor \setminus \lfloor \hat{x} \cup \hat{z} \rfloor = \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}\\
&\hfill \text{By Def of } \hat{\sqcap}
\end{aligned}$$

Now let us prove that this last line holds. Recall that, by definition, $\hat{x}$, $\hat{y}$ and $\hat{z}$ are antichains of elements of $\mathsf{MI}(L)$. First, observe that the hypothesis $\lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{x} = \hat{y}$ implies that:

$$\forall e_y \in \hat{y} : \nexists e_x \in \hat{x} : e_x \sqsubseteq e_y. \quad (5)$$

Indeed, if there were $e_y \in \hat{y}$ and $e_x \in \hat{x}$ with $e_x \sqsubseteq e_y$, then $e_y \notin \lfloor \hat{x} \cup \hat{y} \rfloor$, and thus, $e_y \in \hat{y}$ but $e_y \notin \lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{x}$, which contradicts $\lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{x} = \hat{y}$. Under the hypothesis (5), let us now prove that (i) $\lfloor \hat{x} \cup \hat{y} \cup \hat{z} \rfloor \setminus \lfloor \hat{x} \cup \hat{z} \rfloor \supseteq \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}$ and that (ii) $\lfloor \hat{x} \cup \hat{y} \cup \hat{z} \rfloor \setminus \lfloor \hat{x} \cup \hat{z} \rfloor \subseteq \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}$.

For point (i), we consider $e \in \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}$. Observe that $\lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z} \subseteq \lfloor \hat{y} \cup \hat{z} \rfloor \subseteq \lfloor \hat{y} \rfloor \cup \lfloor \hat{z} \rfloor$. Thus, either $e \in \lfloor \hat{y} \rfloor$ or $e \in \lfloor \hat{z} \rfloor$. However, if $e \in \lfloor \hat{z} \rfloor$, we have $e \in \hat{z}$, and it is thus not possible to have $e \in \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}$. We conclude that $e \in \lfloor \hat{y} \rfloor$. Moreover, there is no $e_z \in \hat{z}$ s.t. $e_z \sqsubseteq e$, otherwise e would not belong to $\lfloor \hat{y} \cup \hat{z} \rfloor$. Since $e \in \hat{y}$, there is also no $e_x \in \hat{x}$ s.t. $e_x \sqsubseteq e$, by (5). Thus, $e \in \lfloor \hat{x} \cup \hat{y} \cup \hat{z} \rfloor$, but $e \notin \lfloor \hat{x} \cup \hat{z} \rfloor$. Hence, $e \in \lfloor \hat{x} \cup \hat{y} \cup \hat{z} \rfloor \setminus \lfloor \hat{x} \cup \hat{z} \rfloor$. Since this is valid for all $e \in \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}$ we conclude that $\lfloor \hat{x} \cup \hat{y} \cup \hat{z} \rfloor \setminus \lfloor \hat{x} \cup \hat{z} \rfloor \supseteq \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}$.

For point (ii), we consider $e \in \lfloor \hat{x} \cup \hat{y} \cup \hat{z} \rfloor \setminus \lfloor \hat{x} \cup \hat{z} \rfloor$. By the same reasoning as above, we conclude that $e \in \lfloor \hat{y} \rfloor$ and that there is no $e_z \in \hat{z}$ s.t. $e_z \sqsubseteq e$. Thus, $e \in \lfloor \hat{y} \cup \hat{z} \rfloor$, but $e \notin \hat{z}$. Hence, $e \in \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}$. Since this is valid for all $e \in \lfloor \hat{x} \cup \hat{y} \cup \hat{z} \rfloor \setminus \lfloor \hat{x} \cup \hat{z} \rfloor$ we conclude that $\lfloor \hat{x} \cup \hat{y} \cup \hat{z} \rfloor \setminus \lfloor \hat{x} \cup \hat{z} \rfloor \subseteq \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}$. □

Corollary 1. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all x, y in L, for all $\theta \in$* $\mathsf{LVBF}(\mathsf{P}, \mathcal{L})$*:* if $x \to (x \sqcap \theta) = \theta$ then $(x \sqcap y) \to (x \sqcap \theta \sqcap y) = y \to (\theta \sqcap y)$.

Lemma 4. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all x, y, z in L with $z \sqsupseteq x$:* if $x \to (x \sqcap y) = y$ then $z \to (x \sqcap y) = (z \to x) \sqcap y$.

Proof. Let us assume that $\hat{\mathcal{L}} = \langle \hat{L}, \hat{\sqsubseteq}, \hat{\top}, \hat{\bot}, \hat{\sqcup}, \hat{\sqcap} \rangle$, and let x, y and z be three elements from L. Throughout the proof, we will denote, $\sigma_{\mathcal{L}}(x)$, $\sigma_{\mathcal{L}}(y)$ and $\sigma_{\mathcal{L}}(z)$ respectively by $\hat{x}$, $\hat{y}$ and $\hat{z}$, in order to alleviate the notations. Let us first rephrase the statement of the Lemma by exploiting the bijection $\sigma_{\mathcal{L}}$:

$$\begin{array}{lr}
\text{if } x \to (x \sqcap y) = y \text{ then } z \to (x \sqcap y) = (z \to x) \sqcap y & \\
\Leftrightarrow \text{if } \sigma_{\mathcal{L}}(x \to (x \sqcap y)) = \hat{y} \text{ then } \sigma_{\mathcal{L}}(z \to (x \sqcap y)) = \sigma_{\mathcal{L}}((z \to x) \sqcap y) & \sigma_{\mathcal{L}} \text{ is a bij.} \\
\Leftrightarrow \text{if } \sigma_{\mathcal{L}}(x \sqcap y) \setminus \hat{x} = \hat{y} \text{ then } \sigma_{\mathcal{L}}(x \sqcap y) \setminus \hat{z} = \sigma_{\mathcal{L}}((z \to x) \sqcap y) & \text{Lemma 2} \\
\Leftrightarrow \text{if } (\hat{x} \hat{\sqcap} \hat{y}) \setminus \hat{x} = \hat{y} \text{ then } (\hat{x} \hat{\sqcap} \hat{y}) \setminus \hat{z} = \sigma_{\mathcal{L}}(z \to x) \hat{\sqcap} \hat{y} & \text{Theorem 1} \\
\Leftrightarrow \text{if } (\hat{x} \hat{\sqcap} \hat{y}) \setminus \hat{x} = \hat{y} \text{ then } (\hat{x} \hat{\sqcap} \hat{y}) \setminus \hat{z} = (\hat{x} \setminus \hat{z}) \hat{\sqcap} \hat{y} & \text{Lemma 2} \\
\Leftrightarrow \text{if } \lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{x} = \hat{y} \text{ then } \lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{z} = \lfloor (\hat{x} \setminus \hat{z}) \cup \hat{y} \rfloor & \text{Def. of } \hat{\sqcap}
\end{array}$$

Now let us prove that this last line holds. This part of the proof is very similar to the end of the proof of Lemma 3. Recall that, by definition, $\hat{x}$, $\hat{y}$ and $\hat{z}$ are antichains of elements of $\mathsf{MI}(L)$. First, observe that, as in the proof of Lemma 3, the hypothesis $\lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{x} = \hat{y}$ implies that:

$$\forall e_y \in \hat{y} : \nexists e_x \in \hat{x} : e_x \sqsubseteq e_y. \tag{6}$$

Moreover, we know that $x \sqsubseteq z$. Hence, by Theorem 1, $\hat{x} \mathrel{\hat{\sqsubseteq}} \hat{z}$. Thus, by definition of $\hat{\sqsubseteq}$, this means that for all $e_z \in \hat{z}$, there is $e_x \in \hat{x}$ s.t. $e_x \sqsubseteq e_z$. As a consequence, for all $e_y \in \hat{y}$, there is no $e_z \in \hat{z}$ s.t. $e_z \sqsubseteq e_y$. Indeed, if there were $e_z \in \hat{z}$ and $e_y \in \hat{y}$ s.t. $e_z \sqsubseteq e_y$, we know that there is $e_x \in \hat{x}$ with $e_x \sqsubseteq e_z$, and thus $e_x \sqsubseteq e_y$, which is not possible by (6). We thus conclude that:

$$\forall e_y \in \hat{y} : \big(\nexists e_x \in \hat{x} : e_x \sqsubseteq e_y \text{ and } \nexists e_z \in \hat{z} : e_z \sqsubseteq e_y \big) \tag{7}$$

Under the hypothesis (7), let us now prove that (i) $\lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{z} \supseteq \lfloor (\hat{x} \setminus \hat{z}) \cup \hat{y} \rfloor$ and (ii) $\lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{z} \subseteq \lfloor (\hat{x} \setminus \hat{z}) \cup \hat{y} \rfloor$.

For point (i), we consider $e \in \lfloor (\hat{x} \setminus \hat{z}) \cup \hat{y} \rfloor$. We have two possibilities. Either (a) $e \in \lfloor \hat{x} \setminus \hat{z} \rfloor$ and there is no $e_y \in \hat{y}$ with $e_y \sqsubseteq e$, or (b) $e \in \lfloor \hat{y} \rfloor$. In the first case (a), it is clear that $e \in \lfloor \hat{x} \rfloor$. Since there is no $e_y \in \hat{y}$ with $e_y \sqsubseteq e$, e is in $\lfloor \hat{x} \cup \hat{y} \rfloor$ too. Moreover, $e \in \lfloor \hat{x} \setminus \hat{z} \rfloor \subseteq \hat{x} \setminus \hat{z}$ implies that $e \notin \hat{z}$. We conclude thus that $e \in \lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{z}$. In the former case (b), we can invoke (7), since $e \in \lfloor \hat{y} \rfloor$ implies that $e \in \hat{y}$. From $e \in \lfloor \hat{y} \rfloor$ and the fact that there is no $e_x \in \hat{x}$ s.t. $e_x \sqsubseteq e$, we obtain $e \in \lfloor \hat{x} \cup \hat{y} \rfloor$. From the fact that there is no $e_z \in \hat{z}$ s.t. $e_z \sqsubseteq e$ we obtain $e \notin \hat{z}$. Thus, $e \in \lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{z}$. Since this is valid for all $e \in \lfloor (\hat{x} \setminus \hat{z}) \cup \hat{y} \rfloor$ we conclude that $\lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{z} \supseteq \lfloor (\hat{x} \setminus \hat{z}) \cup \hat{y} \rfloor$.

For point (ii), we consider $e \in \lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{z}$. Clearly, $e \notin \hat{z}$, and either (a) $e \in \lfloor \hat{x} \rfloor$ and there is no $e_y \in \hat{y}$: $e_y \sqsubseteq e$, or (b) $e \in \lfloor \hat{y} \rfloor$ and there is no $e_x \in \hat{x}$: $e_x \sqsubseteq e$. In the former case (a), $e \in \hat{x}$ but $e \notin \hat{z}$. Hence, $e \in \hat{x} \setminus \hat{z}$. Moreover, since there is no $e_y \in \hat{y}$ with $e_y \sqsubseteq e$, we conclude that $e \in \lfloor (\hat{x} \setminus \hat{z}) \cup \hat{y} \rfloor$. In the latter case (b), $e \in \hat{y}$, and, since there is no $e_x \in \hat{x}$ with $e_x \sqsubseteq e$, we have, in particular, that there is no $e' \in \hat{x} \setminus \hat{z}$ s.t. $e' \sqsubseteq e$. Thus, we obtain: $e \in \lfloor \hat{y} \cup (\hat{x} \setminus \hat{z}) \rfloor$. Since this is valid for all $e \in \lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{z}$ we conclude that $\lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{z} \subseteq \lfloor (\hat{x} \setminus \hat{z}) \cup \hat{y} \rfloor$. □

Corollary 2. *for all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all x, y in L with $y \sqsupseteq x$, for all $\theta \in \mathsf{LVBF}(\mathsf{P}, \mathcal{L})$:* if $x \rightarrow (x \sqcap \theta) = \theta$ then $y \rightarrow (x \sqcap \theta) = (y \rightarrow x) \sqcap \theta$.

Lemma 5. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all x, y, z in L with $z \sqsupseteq x$:* if $x \rightarrow (x \sqcap y) = y$ then $z \rightarrow (z \sqcap y) = y$.

Proof. Let us assume that $\hat{\mathcal{L}} = \langle \hat{L}, \hat{\sqsubseteq}, \hat{\top}, \hat{\bot}, \hat{\sqcup}, \hat{\sqcap} \rangle$, and let x, y and z be three elements from L. Throughout the proof, we will denote, $\sigma_{\mathcal{L}}(x)$, $\sigma_{\mathcal{L}}(y)$ and $\sigma_{\mathcal{L}}(z)$ respectively by $\hat{x}$, $\hat{y}$ and $\hat{z}$, in order to alleviate the notations. Let us first rephrase the statement of the Lemma by exploiting the bijection $\sigma_{\mathcal{L}}$:

$$\begin{array}{llr}
& \text{if } x \rightarrow (x \sqcap y) = y \text{ then } z \rightarrow (z \sqcap y) = y & \\
\Leftrightarrow & \text{if } \sigma_{\mathcal{L}}(x \rightarrow (x \sqcap y)) = \hat{y} \text{ then } \sigma_{\mathcal{L}}(z \rightarrow (z \sqcap y)) = \hat{y} & \sigma_{\mathcal{L}} \text{ is a bijection} \\
\Leftrightarrow & \text{if } \sigma_{\mathcal{L}}(x \sqcap y) \setminus \hat{x} = \hat{y} \text{ then } \sigma_{\mathcal{L}}(z \sqcap y) \setminus \hat{z} = \hat{y} & \text{By Lemma 2} \\
\Leftrightarrow & \text{if } (\hat{x} \hat{\sqcap} \hat{y}) \setminus \hat{x} = \hat{y} \text{ then } (\hat{z} \hat{\sqcap} \hat{y}) \setminus \hat{z} = \hat{y} & \text{By Theorem 1} \\
\Leftrightarrow & \text{if } \lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{x} = \hat{y} \text{ then } \lfloor \hat{z} \cup \hat{y} \rfloor \setminus \hat{z} = \hat{y} & \text{By Theorem 1}
\end{array}$$

Now let us prove that this last line holds. Recall that, by definition, $\hat{x}$, $\hat{y}$ and $\hat{z}$ are antichains of elements of $\mathsf{MI}(L)$. First, observe that the hypothesis $\lfloor \hat{x} \cup \hat{y} \rfloor \setminus \hat{x} = \hat{y}$ together with the fact that $z \sqsupseteq x$ imply that:

$$\forall e_y \in \hat{y} : \left(\nexists e_x \in \hat{x} : e_x \sqsubseteq e_y \text{ and } \nexists e_z \in \hat{z} : e_z \sqsubseteq e_y \right) \tag{8}$$

as established in the proof of Lemma 4. Under the hypothesis (8), let us now prove that (i) $\lfloor \hat{z} \cup \hat{y} \rfloor \setminus \hat{z} \subseteq \hat{y}$ and that (ii) $\lfloor \hat{z} \cup \hat{y} \rfloor \setminus \hat{z} \supseteq \hat{y}$.

For point (i), we consider $e \in \lfloor \hat{z} \cup \hat{y} \rfloor \setminus \hat{z}$. Since $\lfloor \hat{z} \cup \hat{y} \rfloor \setminus \hat{z} \subseteq \lfloor \hat{z} \cup \hat{y} \rfloor \subseteq \lfloor \hat{z} \rfloor \cup \lfloor \hat{y} \rfloor$, e is either in $\lfloor \hat{y} \rfloor$ or in $\lfloor \hat{z} \rfloor$. However, since $e \in \lfloor \hat{z} \cup \hat{y} \rfloor \setminus \hat{z}$, e cannot by in $\hat{z}$, neither in $\lfloor \hat{z} \rfloor \subseteq \hat{z}$. Thus, $e \in \lfloor \hat{y} \rfloor$. Since $\lfloor \hat{y} \rfloor \subseteq \hat{y}$, we have $e \in \hat{y}$. Since this is valid for all $e \in \lfloor \hat{z} \cup \hat{y} \rfloor \setminus \hat{z}$, we conclude that $\lfloor \hat{z} \cup \hat{y} \rfloor \setminus \hat{z} \subseteq \hat{y}$.

For point (ii), we consider $e \in \hat{y}$. Since $\hat{y}$ is an antichain, by definition, we have $\hat{y} = \lfloor \hat{y} \rfloor$, and thus $e \in \lfloor \hat{y} \rfloor$. Since there is no $e_z \in \hat{z}$ s.t. $e_z \sqsubseteq e$, by (8), we obtain that $e \in \lfloor \hat{y} \cup \hat{z} \rfloor$, and that $e \notin \hat{z}$. As a consequence $e \in \lfloor \hat{y} \cup \hat{z} \rfloor \setminus \hat{z}$. Since this is valid for all $e \in \hat{y}$, we conclude that $\lfloor \hat{z} \cup \hat{y} \rfloor \setminus \hat{z} \supseteq \hat{y}$. □

Corollary 3. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all x, y in L with $y \sqsupseteq x$, for all $\theta \in \mathsf{LVBF}(\mathsf{P}, \mathcal{L})$:* if $x \rightarrow (x \sqcap \theta) = \theta$ then $y \rightarrow (y \sqcap \theta) = \theta$.

B Omitted Proofs

B.1 Proofs of Section 2

Lemma 1. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all $x, y \in L$: $x \rightarrow y$ is defined. Moreover, $y \sqsubseteq x$ implies that (i) $(x \rightarrow y) \sqcap x = y$ and that (ii) for all z such that $z \sqcap x = y$: $z \sqsubseteq (x \rightarrow y)$.*

Proof. The proof is split in two parts. We first show that $x \to y$ is defined in Lemma 6 hereunder. Then, we show that, when $y \sqsubseteq x$, then $(x \to y) \sqcap x = y$ and for all z such that $z \sqcap x = y$: $z \sqsubseteq (x \to y)$ in Lemma 7 hereunder.

Lemma 6. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all x, y in L: $x \to y$ is defined.*

Proof. By defintion of $x \to y$, we have to show that $\{z \mid z \sqcap x \sqsubseteq y\}$ has a unique maximal element. We proceed by contradiction and assume that it is not the case. There are two cases. Either $\{z \mid z \sqcap x \sqsubseteq y\}$ is empty, or $\lceil \{z \mid z \sqcap x \sqsubseteq y\} \rceil = \{e_1, \ldots, e_n\}$ with $n > 1$. In the first case, we observe that $y \sqcap x \sqsubseteq y$ for any $x, y \in L$. Hence $\{z \mid z \sqcap x \sqsubseteq y\}$ cannot be empty. In the second case we consider $e = \bigsqcup \{e_i \mid 1 \leq i \leq n\}$. Remark that, for all $1 \leq i \leq n$, $e_i \sqsubset e$. Indeed, if $e_j = e$ holds for some $1 \leq j \leq n$, then $e_i \sqsubseteq e_j$ for all $1 \leq i \leq n$. This is not possible because $n > 1$ and all the elements in $\{e_1, \ldots, e_n\}$ are pairwise incomparable, since $\{e_1, \ldots, e_n\}$ is the set of maximal elements of $\{z \mid z \sqcap x \sqsubseteq y\}$. By distributivity of the lattice, $e \sqcap x = \bigsqcup \{e_i \sqcap x \mid 1 \leq i \leq n\}$. Since, for all $e_i \sqcap x \sqsubseteq y$ for any $1 \leq i \leq n$, we have: $\bigsqcup \{e_i \sqcap x \mid 1 \leq i \leq n\} \sqsubseteq y$, and thus: $e \sqcap x \sqsubseteq y$. Finally, $e \in \{z \mid z \sqcap x \sqsubseteq y\}$, and, for all $1 \leq i \leq n$, $e_i \sqsubset e$. As a consequence, $\{e_1, \ldots, e_n\} \neq \lceil \{z \mid z \sqcap x \sqsubseteq y\} \rceil$. Contradiction. □

Lemma 7. *For all FDL $\mathcal{L} = \langle L, \sqsubseteq, \top, \bot, \sqcup, \sqcap \rangle$, for all x, y in L such that $y \sqsubseteq x$: (i) $(x \to y) \sqcap x = y$ and (ii) for all z such that $z \sqcap x = y$: $z \sqsubseteq (x \to y)$.*

Proof. Let x, y be two elements of L s.t. $y \sqsubseteq x$. Since $\mathcal{L}$ is an FDL, we know, by Lemma 1 that $x \to y$ exists and is thus, by definition, the unique $\sqsubseteq$-largest element of the set $\{z \mid z \sqcap x \sqsubseteq y\}$.

To establish point (i), observe that $y \sqcap x = y$ since $y \sqsubseteq x$. Thus, $y \in \{z \mid z \sqcap x \sqsubseteq y\}$. As a consequence, $x \to y \sqsupseteq y$, because $x \to y$ is the unique $\sqsubseteq$-largest element of $\{z \mid z \sqcap x \sqsubseteq y\}$. Thus, $(x \to y) \sqcap x \sqsupseteq y \sqcap x = y$, and we conclude that $(x \to y) \sqcap x \sqsupseteq y$. However, we know that $(x \to y) \sqcap x \sqsubseteq y$, by definition of $\to$. Hence, $(x \to y) \sqcap x = y$.

To establish point (ii), we invoke again the fact that $x \to y$ is the unique $\sqsubseteq$-largest element of the set $\{z \mid z \sqcap x \sqsubseteq y\}$. Otherwise stated, for all z s.t. $z \sqcap x \sqsubseteq y$: $z \sqsubseteq x \to y$. Thus, in particular, for all z s.t. $z \sqcap x = y$, we have $z \sqsubseteq x \to y$. □

B.2 Proofs of section 4

In this section, we provide the lemmas that establish the correctness of the algorithms from Section 4.

Lemma 8. *For all non-terminal* LVBDD $n \in \mathsf{LVBDD}^S(\mathsf{P}, \mathcal{L})$*:* $\mathsf{val}(n) \sqsubseteq (\mathsf{val}(\mathsf{lo}(n)) \sqcup \mathsf{val}(\mathsf{hi}(n)))$.

Proof. From the definition of SNF, we know that $\mathsf{val}(n) = \exists \mathsf{P} : [\![n]\!]$. By the semantics of LVBDD we have: $\exists \mathsf{P} : [\![n]\!] = \mathsf{val}(n) \sqcap \big((\exists \mathsf{P} : [\![\mathsf{lo}(n)]\!]) \sqcup (\exists \mathsf{P} : [\![\mathsf{lo}(n)]\!])\big)$. Since $\mathsf{val}(\mathsf{lo}(v)) = \exists \mathsf{P} : [\![\mathsf{lo}(n)]\!]$ and $\mathsf{val}(\mathsf{hi}(v)) = \exists \mathsf{P} : [\![\mathsf{hi}(n)]\!]$, we have that $\mathsf{val}(n) = \mathsf{val}(n) \sqcap \big(\mathsf{val}(\mathsf{lo}(n)) \sqcup \mathsf{val}(\mathsf{hi}(n))\big)$, so $\mathsf{val}(n) \sqsubseteq \mathsf{val}(\mathsf{lo}(n)) \sqcup \mathsf{val}(\mathsf{hi}(n))$. □

Lemma 9. *For all* $n \in \mathsf{LVBDD}^S(\mathsf{P}, \mathcal{L})$ *and* $d \in L$ *with* $d \sqsupseteq \mathsf{val}(n)$*:* $\mathcal{D}^S(d \to [\![n]\!]) = \mathsf{PseudoCompSNF}(n, d)$.

Proof (soundness of Algorithm 1). If n is terminal, the result is immediate. Assume that n is non-terminal and let $i = \mathsf{index}(n)$. Clearly, the algorithm returns a non-terminal node $n' = \langle i, d \rightarrow \mathsf{val}(n) \sqcap (\mathsf{val}(\mathsf{lo}(n)) \sqcup \mathsf{val}(\mathsf{hi}(n))), \mathsf{lo}(n), \mathsf{hi}(n)\rangle$.

We start by showing that $[\![n']\!] = d \rightarrow [\![n]\!]$. From the semantics definition of LVBDD we have that $d \rightarrow [\![n]\!] = d \rightarrow \big(\mathsf{val}(n) \sqcap (\neg p_i \sqcap [\![\mathsf{lo}(n)]\!]) \sqcup (p_i \sqcap [\![\mathsf{hi}(n)]\!])\big)$. Let $v \in 2^{\mathsf{P}}$, and recall that $d \rightarrow \theta \equiv \lambda v \cdot d \rightarrow \theta(v)$; we see that for any valuation v:

$$(d \rightarrow [\![n]\!])(v) = \begin{cases} d \rightarrow \big(\mathsf{val}(n) \sqcap [\![\mathsf{lo}(n)]\!](v)\big) \text{ if } v(p_i) = 0 \\ d \rightarrow \big(\mathsf{val}(n) \sqcap [\![\mathsf{hi}(n)]\!](v)\big) \text{ if } v(p_i) = 1 \end{cases}$$

From the definition of LVBDD in SNF, we know that $[\![\mathsf{lo}(n)]\!] = \mathsf{val}(n) \rightarrow ([\![n]\!]|_{p_i=0})$. It is easy to see that $[\![n]\!]|_{p_i=0} = \mathsf{val}(n) \sqcap [\![\mathsf{lo}(n)]\!]$, thus we have that $[\![\mathsf{lo}(n)]\!] = \mathsf{val}(n) \rightarrow (\mathsf{val}(n) \sqcap [\![\mathsf{lo}(n)]\!])$. With similar reasoning we obtain: $[\![\mathsf{hi}(n)]\!] = \mathsf{val}(n) \rightarrow (\mathsf{val}(n) \sqcap [\![\mathsf{hi}(n)]\!])$. By using Proposition 1, point (2) and the decomposition of $(d \rightarrow [\![n]\!])(v)$ above, we now have, for any valuation v:

$$(d \rightarrow [\![n]\!])(v) = \begin{cases} (d \rightarrow \mathsf{val}(n)) \sqcap [\![\mathsf{lo}(n)]\!](v) \text{ if } v(p_i) = 0 \\ (d \rightarrow \mathsf{val}(n)) \sqcap [\![\mathsf{hi}(n)]\!](v) \text{ if } v(p_i) = 1 \end{cases}$$

We thus now have that: $d \rightarrow [\![n]\!] = (d \rightarrow \mathsf{val}(n)) \sqcap \big((\neg p_i \sqcap [\![\mathsf{lo}(n)]\!]) \sqcup (p_i \sqcap [\![\mathsf{hi}(n)]\!])\big)$. It is now easy to see that $d \rightarrow [\![n]\!] = d' \sqcap \big((\neg p_i \sqcap [\![\mathsf{lo}(n)]\!]) \sqcup (p_i \sqcap [\![\mathsf{hi}(n)]\!])\big) = [\![n']\!]$, since $(\mathsf{val}(\mathsf{lo}(n)) \sqcup \mathsf{val}(\mathsf{hi}(n))) \sqcap [\![\mathsf{lo}(n)]\!] = [\![\mathsf{lo}(n)]\!]$ and likewise for $[\![\mathsf{hi}(n)]\!]$.

We have shown that n' has the correct semantics so it remains to prove that n' is in SNF. We begin by showing that n' has the correct label, i.e. we need to show that $\exists \mathsf{P} : [\![n']\!] = \mathsf{val}(n') = d'$. We know that $[\![n']\!] = d \rightarrow [\![n]\!]$ so by reusing the results above we have: $\exists \mathsf{P} : [\![n']\!] = \exists \mathsf{P} : (d \rightarrow \mathsf{val}(n)) \sqcap \big((\neg p_i \sqcap [\![\mathsf{lo}(n)]\!]) \sqcup (p_i \sqcap [\![\mathsf{hi}(n)]\!])\big) = (d \rightarrow \mathsf{val}(n)) \sqcap \big(\mathsf{val}(\mathsf{lo}(n)) \sqcup \mathsf{val}(\mathsf{hi}(n))\big) = d'$. It remains to show that n' has the correct low- and high-child, i.e. $[\![\mathsf{lo}(n)]\!] = \mathsf{val}(n') \rightarrow ([\![n']\!]|_{p_i=0})$ and $[\![\mathsf{hi}(n)]\!] = \mathsf{val}(n') \rightarrow ([\![n']\!]|_{p_i=1})$. We know that $[\![n']\!]|_{p_i=0} = d' \sqcap [\![\mathsf{lo}(n)]\!]$, and since $\mathsf{val}(n') = d'$ we have that $\mathsf{val}(n') \rightarrow ([\![n']\!]|_{p_i=0}) = d' \rightarrow (d' \sqcap [\![\mathsf{lo}(n)]\!])$. By definition of $\rightarrow$ and by Lemma 8, it is easy to see that $\mathsf{val}(n) \sqsubseteq d'$. We have already shown that $[\![\mathsf{lo}(n)]\!] = \mathsf{val}(n) \rightarrow (\mathsf{val}(n) \sqcap [\![\mathsf{lo}(n)]\!])$, thus by using Proposition 1, point (3), we have that $[\![\mathsf{lo}(n)]\!] = d' \rightarrow (d' \sqcap [\![\mathsf{lo}(n)]\!]) = d' \rightarrow ([\![n']\!]|_{p_i=0})$. Since $\mathsf{lo}(n)$ is assumed in SNF, we have proved that it is the correct low-child of n'; by similar reasoning we can show that $\mathsf{hi}(n)$ is the correct high-child of n'. Finally, we show that p_i is the lowest-index variable in $I_{d \rightarrow [\![n]\!]}$. We know by hypothesis that p_i is the lowest-index variable in $I_{[\![n]\!]}$, because n is in SNF. Hence, it is sufficient to show that $p_i \in I_{d \rightarrow [\![n]\!]}$. We have shown above that $[\![\mathsf{lo}(n)]\!] = d' \rightarrow ([\![n']\!]|_{p_i=0})$ and that $[\![\mathsf{hi}(n)]\!] = d' \rightarrow ([\![n']\!]|_{p_i=1})$, with $\mathsf{lo}(n) \neq \mathsf{hi}(n)$, by hypothesis that n is in SNF. Since $\mathsf{lo}(n)$ and $\mathsf{hi}(n)$ are in SNF, we have: $d' \rightarrow ([\![n']\!]|_{p_i=0}) \neq d' \rightarrow ([\![n']\!]|_{p_i=1})$ Hence, we deduce that $[\![n']\!]|_{p_i=0} \neq [\![n']\!]|_{p_i=1}$, and thus $p_i \in I_{d \rightarrow [\![n]\!]}$. □

Lemma 10. *For all $n \in \mathsf{LVBDD}^S(\mathsf{P}, \mathcal{L})$, $d \in L$: $\mathsf{ConstMeetSNF}(n, d) = \mathcal{D}^S([\![n]\!] \sqcap d)$.*

Proof. Since Algorithm 2 is recursive, the proof is by induction on the structure of the LVBDD n. We thus show that the LVBDD returned by Algorithm 2 is $\mathcal{D}^S([\![n]\!] \sqcap d)$.

Base cases The first base case at line 4 occurs when $\mathsf{val}(n) \sqsubseteq d$; from the definition of SNF we have that $\mathsf{val}(n) = \exists \mathsf{P} : [\![\mathsf{val}(n)]\!]$, thus $[\![\mathsf{val}(n)]\!] \sqcap d = [\![\mathsf{val}(n)]\!]$, thus $\mathcal{D}^S([\![n]\!] \sqcap d) = n$. The second base case at line 5 occurs when $\mathsf{val}(n) \sqcap d = \bot$, which implies that $[\![n]\!] \sqcap d = \bot$ and $\mathcal{D}^S([\![n]\!] \sqcap d) = \langle k+1, \bot, \mathsf{nil}, \mathsf{nil}\rangle$. The final base case of line 6 occurs when n is a terminal node, thus clearly $\mathcal{D}^S([\![n]\!] \sqcap d) = \langle k+1, \mathsf{val}(n) \sqcap d, \mathsf{nil}, \mathsf{nil}\rangle$.

Inductive case Let $i = \mathsf{index}(n)$; since we know that n is a non-terminal node we have that $[\![n]\!] = \mathsf{val}(n) \sqcap \big((\neg p_i \sqcap [\![\mathsf{lo}(n)]\!]) \sqcup (p_i \sqcap [\![\mathsf{hi}(n)]\!])\big)$, thus $[\![n]\!] \sqcap d = \mathsf{val}(n) \sqcap \big((\neg p_i \sqcap [\![\mathsf{lo}(n)]\!] \sqcap d) \sqcup (p_i \sqcap [\![\mathsf{hi}(n)]\!] \sqcap d)\big)$. By the induction hypothesis, we know that that $\ell = \mathcal{D}^S([\![\mathsf{lo}(n)]\!] \sqcap d)$ and $h = \mathcal{D}^S([\![\mathsf{hi}(n))]\!] \sqcap d)$, which are computed at line 8.

Subcase 1 If $\ell = h$, the algorithm computes, at line 10, the LVBDD $n' = \langle \mathsf{index}(\ell), \mathsf{val}(\ell) \sqcap \mathsf{val}(n), \mathsf{lo}(\ell), \mathsf{hi}(\ell)\rangle$. Let us show that n' has the correct semantics i.e., $[\![n']\!] = [\![n]\!] \sqcap d$. We know by the semantics of LVBDD that $[\![n]\!] \sqcap d = \mathsf{val}(n) \sqcap (\neg p_i \sqcap [\![\mathsf{lo}(n)]\!] \sqcap d) \sqcup (p_i \sqcap [\![\mathsf{hi}(n)]\!] \sqcap d)) = \mathsf{val}(n) \sqcap [\![\mathsf{lo}(n)]\!] \sqcap d = \mathsf{val}(n) \sqcap [\![\ell]\!]$, since $[\![\mathsf{lo}(n)]\!] \sqcap d = [\![\mathsf{hi}(n)]\!] \sqcap d$ and $[\![\mathsf{lo}(n)]\!] \sqcap d = [\![\ell]\!]$. Clearly we also have that $[\![n']\!] = \mathsf{val}(n) \sqcap [\![\ell]\!]$ hence $[\![n']\!] = [\![n]\!] \sqcap d$. Next, we need to ensure that n' has the appropriate label i.e., $\mathsf{val}(n') = \exists \mathsf{P} : [\![n']\!]$. This is easy because $\exists \mathsf{P} : [\![n']\!] = \exists \mathsf{P} : ([\![\ell]\!] \sqcap \mathsf{val}(n)) = (\exists \mathsf{P} : [\![\ell]\!]) \sqcap \mathsf{val}(n) = \mathsf{val}(\ell) \sqcap \mathsf{val}(n) = \mathsf{val}(n')$. Let $j = \mathsf{index}(n') = \mathsf{index}(\ell)$; we now need to prove that n' has the appropriate low- and high-child. If ℓ is a terminal node this easy since n' must then be terminal also, which means that both $\mathsf{lo}(n')$ and $\mathsf{hi}(n')$ must be nil and $\mathsf{lo}(\ell), \mathsf{hi}(\ell)$ are nil if ℓ is terminal. We thus assume that ℓ is nonterminal and show that $[\![\mathsf{lo}(n')]\!] = \mathsf{val}(n') \to ([\![n']\!]|_{p_j=0})$ and $[\![\mathsf{hi}(n')]\!] = \mathsf{val}(n') \to ([\![n']\!]|_{p_j=1})$. We show this for $\mathsf{lo}(n')$. By substituting $\mathsf{lo}(n')$ and $\mathsf{val}(n')$ by their assigned values of line 10, we see that we must show the following equivalent expression:

$$[\![\mathsf{lo}(\ell)]\!] = (\mathsf{val}(\ell) \sqcap \mathsf{val}(n)) \to (\mathsf{val}(\ell) \sqcap \mathsf{val}(n) \sqcap [\![\mathsf{lo}(\ell)]\!])$$

Moreover, we can easily see that $\mathsf{val}(\ell) = \mathsf{val}(\mathsf{lo}(n)) \sqcap d$ and thus obtain the following:

$$(*) \quad [\![\mathsf{lo}(\ell)]\!] = (\mathsf{val}(\mathsf{lo}(n)) \sqcap d \sqcap \mathsf{val}(n)) \to (\mathsf{val}(\mathsf{lo}(n)) \sqcap d \sqcap \mathsf{val}(n) \sqcap [\![\mathsf{lo}(\ell)]\!])$$

By unfolding the definition of LVBDD in SNF twice we can see that:

$$[\![\mathsf{lo}(\mathsf{lo}(n))]\!] = \mathsf{val}(\mathsf{lo}(n)) \to \big(\mathsf{val}(n) \to (\mathsf{val}(\mathsf{lo}(n)) \sqcap \mathsf{val}(n) \sqcap [\![\mathsf{lo}(\mathsf{lo}(n))]\!])\big)$$

which by application of Proposition 1, point (4), can be rewritten as:

$$(**) \quad [\![\mathsf{lo}(\mathsf{lo}(n))]\!] = \big(\mathsf{val}(\mathsf{lo}(n)) \sqcap \mathsf{val}(n)\big) \to \big(\mathsf{val}(\mathsf{lo}(n)) \sqcap \mathsf{val}(n) \sqcap [\![\mathsf{lo}(\mathsf{lo}(n))]\!]\big)$$

We now make the following assumption, which for the sake of clarity is justified later:

$$(***) \quad d \to (d \sqcap [\![\mathsf{lo}(\mathsf{lo}(n))]\!]) = [\![\mathsf{lo}(\ell)]\!]$$

By Proposition 1, point (1), using $(**)$ and $(***)$, we obtain:

$$[\![\mathsf{lo}(\ell)]\!] = (\mathsf{val}(\mathsf{lo}(n)) \sqcap d \sqcap \mathsf{val}(n)) \to (\mathsf{val}(\mathsf{lo}(n)) \sqcap d \sqcap \mathsf{val}(n) \sqcap [\![\mathsf{lo}(\mathsf{lo}(n))]\!])$$

Finally, by using $(***)$ again, we can substitute $d \sqcap [\![\mathsf{lo}(\mathsf{lo}(n))]\!]$ by $d \sqcap [\![\mathsf{lo}(\ell)]\!]$ in the previous expression and thus show that $(*)$ is true. To complete our argument, we now prove that the assumption $(***)$ is correct. By the definition of LVBDD in SNF, and the fact that $\mathsf{val}(\ell) = \mathsf{val}(\mathsf{lo}(n)) \sqcap d$, we can see that:

$$[\![\mathsf{lo}(\ell)]\!] = (\mathsf{val}(\mathsf{lo}(n)) \sqcap d) \rightarrow \big(\mathsf{val}(\mathsf{lo}(n)) \sqcap d \sqcap [\![\ell]\!]|_{p_j=0}\big)$$

Using the induction hypothesis, we know that $[\![\ell]\!] = [\![\mathsf{lo}(n)]\!] \sqcap d$, thus:

$$\begin{aligned}[\![\mathsf{lo}(\ell)]\!] &= (\mathsf{val}(\mathsf{lo}(n)) \sqcap d) \rightarrow \big(\mathsf{val}(\mathsf{lo}(n)) \sqcap d \sqcap ([\![\mathsf{lo}(n)]\!] \sqcap d)|_{p_j=0}\big) \\ &= (\mathsf{val}(\mathsf{lo}(n)) \sqcap d) \rightarrow \big(\mathsf{val}(\mathsf{lo}(n)) \sqcap d \sqcap [\![\mathsf{lo}(\mathsf{lo}(n))]\!]\big)\end{aligned}$$

Moreover, by definition of LVBDD in SNF (this time applied to $\mathsf{lo}(\mathsf{lo}(n))$):

$$[\![\mathsf{lo}(\mathsf{lo}(n))]\!] = \mathsf{val}(\mathsf{lo}(n)) \rightarrow \big(\mathsf{val}(\mathsf{lo}(n)) \sqcap [\![\mathsf{lo}(\mathsf{lo}(n))]\!]\big)$$

Finally, $(***)$ is obtained by the two previous expressions and Proposition 1, point (1).

Subcase 2 If $\ell \neq h$ the algorithm computes ℓ' and h' at line 12, which by soundness of Algorithm 1 are such that $\ell' = \mathcal{D}^S(d \rightarrow [\![\ell]\!])$ and $h' = \mathcal{D}^S(d \rightarrow [\![h]\!])$.
We begin by showing that n' has the correct semantics, i.e. $[\![n']\!] = [\![n]\!] \sqcap d$. Indeed, $[\![n']\!] = d \sqcap \mathsf{val}(n) \sqcap \big((\neg p_i \sqcap [\![\ell']\!]) \sqcup (p_i \sqcap [\![h']\!])\big) = d \sqcap \mathsf{val}(n) \sqcap \big((\neg p_i \sqcap [\![\mathsf{lo}(n)]\!]) \sqcup (p_i \sqcap [\![\mathsf{hi}(n)]\!])\big) = d \sqcap [\![n]\!]$, since by definition of $\rightarrow$ we know that $[\![\ell']\!] \sqcap d = [\![\mathsf{lo}(n)]\!] \sqcap d$ and $[\![h']\!] \sqcap d = [\![\mathsf{hi}(n)]\!] \sqcap d$.
We now show that $n' = \mathcal{D}^S([\![n]\!] \sqcap d)$, which we know is equivalent to $n' = \mathcal{D}^S([\![n']\!])$. We begin by showing that $\mathsf{val}(n') = \exists \mathsf{P} : [\![n']\!]$; this is easy because $\exists \mathsf{P} : [\![n']\!] = \exists \mathsf{P} : ([\![n]\!] \sqcap d) = (\exists \mathsf{P} : [\![n]\!]) \sqcap d = \mathsf{val}(n) \sqcap d$, by distributivity. Next, we must show that n' has the correct low- and high-child, i.e. $\mathsf{lo}(n') = \mathcal{D}^S(\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=0})$ and $\mathsf{hi}(n') = \mathcal{D}^S(\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=1})$. We show this for $\mathsf{lo}(n')$; from the definition of LVBDD semantics we know that $[\![n']\!]|_{p_i=0} = \mathsf{val}(n') \sqcap [\![\ell']\!]$. Since $\mathsf{val}(n') = \mathsf{val}(n) \sqcap d$ we have that:

$$\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=0} = (\mathsf{val}(n) \sqcap d) \rightarrow (\mathsf{val}(n) \sqcap d \sqcap [\![\ell']\!])$$

Since $[\![\ell']\!] \sqcap d = [\![\mathsf{lo}(n)]\!] \sqcap d$ we can see that:

$$\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=0} = (\mathsf{val}(n) \sqcap d) \rightarrow (\mathsf{val}(n) \sqcap d \sqcap [\![\mathsf{lo}(n)]\!])$$

Furthermore, we know from the definition of SNF that $\mathsf{lo}(n) = \mathcal{D}^S(\mathsf{val}(n) \rightarrow [\![n]\!]|_{p_i=0})$, thus $[\![\mathsf{lo}(n)]\!] = \mathsf{val}(n) \rightarrow (\mathsf{val}(n) \sqcap [\![\mathsf{lo}(n)]\!])$. By Proposition 1, point (1), we obtain that:

$$\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=0} = d \rightarrow (d \sqcap [\![\mathsf{lo}(n)]\!]) = d \rightarrow [\![\ell]\!] = [\![\ell']\!]$$

By soundness of Alg. 1, $\ell' = \mathcal{D}^S(d \rightarrow [\![\ell]\!])$, thus $\ell' = \mathcal{D}^S(\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=0})$. Similar reasoning shows that $h' = \mathcal{D}^S(\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=1})$.
Finally, we must show that n' has the correct index i.e., that p_i is the proposition of lowest index in $I_{[\![n']\!]}$. Clearly $I_{[\![n']\!]} \subseteq I_{[\![n]\!]}$ so either $p_i \notin I_{[\![n']\!]}$ or p_i is the proposition of lowest index in $I_{[\![n']\!]}$, since we know that p_i is the proposition of lowest index in $I_{[\![n]\!]}$. Let us show that $p_i \in I_{[\![n']\!]}$ or equivalently that $[\![n']\!]|_{p_i=0} \neq [\![n']\!]|_{p_i=1}$. By the semantics definition of LVBDD we know that $[\![n']\!]|_{p_i=0} = \mathsf{val}(n) \sqcap d \sqcap [\![\ell']\!]$ and $[\![n']\!]|_{p_i=1} = \mathsf{val}(n) \sqcap d \sqcap [\![h']\!]$. We also know that $d \sqcap [\![\ell']\!] = [\![\ell]\!]$ and that $d \sqcap [\![h']\!] = [\![h]\!]$, by definition of $\rightarrow$ and soundness of Algorithm 1. Furthermore, we know that $[\![\ell]\!] = [\![\mathsf{lo}(n)]\!] \sqcap d$

and $[\![h]\!] = [\![\mathsf{hi}(n)]\!] \sqcap d$, by soundness of Algorithm 2. We must therefore show that $d \sqcap \mathsf{val}(n) \sqcap [\![\mathsf{lo}(n)]\!] \neq d \sqcap \mathsf{val}(n) \sqcap [\![\mathsf{hi}(n)]\!]$. We already know that $d \sqcap [\![\mathsf{lo}(n)]\!] \neq d \sqcap [\![\mathsf{hi}(n)]\!]$ since $\ell' \neq h'$, which implies that $d \rightarrow (d \sqcap [\![\mathsf{lo}(n)]\!]) \neq d \rightarrow (d \sqcap [\![\mathsf{hi}(n)]\!])$. From the definition of SNF, we know that $[\![\mathsf{lo}(n)]\!] = \mathsf{val}(n) \rightarrow (\mathsf{val}(n) \sqcap [\![\mathsf{lo}(n)]\!])$ and $[\![\mathsf{hi}(n)]\!] = \mathsf{val}(n) \rightarrow (\mathsf{val}(n) \sqcap [\![\mathsf{hi}(n)]\!])$; by Proposition 1, point (1) we thus obtain:

$$d \rightarrow (d \sqcap [\![\mathsf{lo}(n)]\!]) = (\mathsf{val}(n) \sqcap d) \rightarrow (\mathsf{val}(n) \sqcap d \sqcap [\![\mathsf{lo}(n)]\!])$$
$$d \rightarrow (d \sqcap [\![\mathsf{hi}(n)]\!]) = (\mathsf{val}(n) \sqcap d) \rightarrow (\mathsf{val}(n) \sqcap d \sqcap [\![\mathsf{hi}(n)]\!])$$

We can now see that $\mathsf{val}(n) \sqcap d \sqcap [\![\mathsf{lo}(n)]\!] \neq \mathsf{val}(n) \sqcap d \sqcap [\![\mathsf{hi}(n)]\!]$. □

Lemma 11. *For all $n_1, n_2 \in \mathsf{LVBDD}^S(\mathsf{P}, \mathcal{L})$: $\mathcal{D}^S([\![n_1]\!] \sqcap [\![c_2]\!]) = \mathsf{MeetSNF}(n_1, n_2)$.*

Proof (soundness of Algorithm 3). Since Algorithm 3 is recursive, the proof is by induction on the structure of the LVBDD n_1 and n_2. Throughout the proof we shall let $\theta = [\![n_1]\!] \sqcap [\![n_2]\!]$; we must now show that the LVBDD n' returned by Algorithm 3 is such that $n' = \mathcal{D}^S(\theta)$.

Base cases Lines 4 and 5 detect whether n_1 or n_2 are terminal nodes. In either case, we can use the procedure ConstMeetSNF of Algorithm 2 to compute n'.

Inductive case For the inductive cases, we know that neither n_1 nor n_2 are terminal nodes. We consider the following cases:

1. If $\mathsf{index}(n_1) = \mathsf{index}(n_2) = i$, we see from the semantics of LVBDD that:

   $$\theta = \mathsf{val}(n_1) \sqcap \mathsf{val}(n_2) \sqcap \big((\neg p_i \sqcap [\![\mathsf{lo}(n_1)]\!] \sqcap [\![\mathsf{lo}(n_2)]\!]) \sqcup (p_i \sqcap [\![\mathsf{hi}(n_1)]\!] \sqcap [\![\mathsf{hi}(n_2)]\!])\big)$$

   By the induction hypothesis we have that $\ell = \mathcal{D}^S([\![\mathsf{lo}(n_1)]\!] \sqcap [\![\mathsf{lo}(n_2)]\!])$ and $h = \mathcal{D}^S([\![\mathsf{hi}(n_1)]\!] \sqcap [\![\mathsf{hi}(n_2)]\!])$, which are both computed at line 8. Furthermore, by soundness of Algorithm 2 we have that $\ell' = \mathcal{D}^S([\![\ell]\!] \sqcap \mathsf{val}(n_1) \sqcap \mathsf{val}(n_2))$ and $h' = \mathcal{D}^S([\![h]\!] \sqcap \mathsf{val}(n_1) \sqcap \mathsf{val}(n_2))$, which are computed at line 14. From here on, we must again consider two cases:

   (a) If $\ell' = h'$ then we claim that $n' = \mathcal{D}^S(\theta)$. By soundness of Algorithm 2 we already know that $\ell' = \mathcal{D}^S([\![\ell']\!])$ so we just need to show that $[\![\ell']\!] = \theta$. Indeed, if $\ell' = h'$, then $\mathsf{val}(n_1) \sqcap \mathsf{val}(n_2) \sqcap [\![\mathsf{lo}(n_1)]\!] \sqcap [\![\mathsf{lo}(n_2)]\!] = \mathsf{val}(n_1) \sqcap \mathsf{val}(n_2) \sqcap [\![\mathsf{hi}(n_1)]\!] \sqcap [\![\mathsf{hi}(n_2)]\!]$, so we have:

   $$\theta = \mathsf{val}(n_1) \sqcap \mathsf{val}(n_2) \sqcap [\![\mathsf{lo}(n_1)]\!] \sqcap [\![\mathsf{lo}(n_2)]\!]$$

   which we know is equal to $[\![\ell']\!]$.

   (b) If $\ell' \neq h'$ then the algorithm proceeds to line 17 and computes ℓ'' and h'' which by soundness of Algorithm 1 are such that:

   $$\ell'' = \mathcal{D}^S((\mathsf{val}(\ell') \sqcup \mathsf{val}(h')) \rightarrow [\![\ell']\!]) \; ; \; h'' = \mathcal{D}^S((\mathsf{val}(\ell') \sqcup \mathsf{val}(h')) \rightarrow [\![h']\!])$$

Let us now show that $[\![n']\!] = \theta$, for n' computed at line 19. By the semantics of LVBDD we have:

$$[\![n']\!] = d' \sqcap \big((\neg p_i \sqcap [\![\ell'']\!]) \sqcup (p_i \sqcap [\![h'']\!])\big) = (\neg p_i \sqcap [\![\ell'']\!] \sqcap d') \sqcup (p_i \sqcap [\![h'']\!] \sqcap d')$$

By definition of $\rightarrow$ we know that $[\![\ell'']\!] \sqcap d' = [\![\ell']\!]$ and $[\![h'']\!] \sqcap d' = [\![h']\!]$. Furthermore, we have seen earlier that $[\![\ell']\!] = \mathsf{val}(n_1) \sqcap \mathsf{val}(n_2) \sqcap [\![\ell]\!]$ and $[\![h']\!] = \mathsf{val}(n_1) \sqcap \mathsf{val}(n_2) \sqcap [\![h]\!]$. By substitution and distributivity we have:

$$[\![n']\!] = \mathsf{val}(n_1) \sqcap \mathsf{val}(n_2) \sqcap \big((\neg p_i \sqcap [\![\ell]\!]) \sqcup (p_i \sqcap [\![h]\!])\big) = \theta$$

Let us now show that n' is in SNF, i.e. $n' = \mathcal{D}^S([\![n']\!])$. We first show that $\mathsf{val}(n') = \exists \mathsf{P} : [\![n']\!]$. By the semantics definition of LVBDD we have:

$$\exists \mathsf{P} : [\![n']\!] = \exists \mathsf{P} : \theta = d \sqcap (\exists \mathsf{P} : [\![\ell]\!] \sqcup \exists \mathsf{P} : [\![h]\!])$$

Since we know that $[\![\ell]\!] \sqcap d = [\![\ell']\!]$ and $[\![h]\!] \sqcap d = [\![h']\!]$ we see that:

$$\exists \mathsf{P} : [\![n']\!] = \exists \mathsf{P} : [\![\ell']\!] \sqcup \exists \mathsf{P} : [\![h']\!] = d' = \mathsf{val}(n')$$

Next, we must show that ℓ'' and h'' are the correct low- and high-child of n', i.e. $\ell'' = \mathcal{D}^S(\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=0})$ and $h'' = \mathcal{D}^S(\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=1})$. We show this for ℓ''. From the semantics definition of LVBDD we have that $[\![n']\!]|_{p_i=0} = d' \sqcap [\![\ell'']\!]$, and since $\mathsf{val}(n') = d'$ we must show that $\ell'' = \mathcal{D}^S(d' \rightarrow d' \sqcap [\![\ell'']\!])$. We already know that $d' \sqcap [\![\ell'']\!] = [\![\ell']\!]$ and that $\ell' = \mathcal{D}^S(d' \rightarrow [\![\ell']\!])$ so we have shown that $\ell'' = \mathcal{D}^S(\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=0})$. Similar reasoning shows that $h'' = \mathcal{D}^S(\mathsf{val}(n') \rightarrow [\![n']\!]|_{p_i=1})$.
Finally, we must show that n' has the correct index i.e., that p_i is the proposition of lowest index such that $p_i \in I_\theta$. Since $i = \mathsf{index}(n_1) = \mathsf{index}(n_2)$, p_i is the proposition of lowest index in both $I_{[\![n_1]\!]}$ and $I_{[\![n_2]\!]}$. Clearly we have that $I_\theta \subseteq I_{[\![n_1]\!]} \cup I_{[\![n_2]\!]}$, so either $p_i \notin I_\theta$ or p_i is the proposition of lowest index in I_θ. Let us show that $p_i \in \theta$, which is equivalent to $\theta_{p_i=0} \neq \theta_{p_i=1}$. We need to show that $d' \sqcap [\![\ell'']\!] \neq d' \sqcap [\![h'']\!]$, thus that $[\![\ell']\!] \neq [\![h']\!]$, which we know is true thanks to the **if** statement of line 15 and by canonicity of SNF.

2. If $\mathsf{index}(n_1) \neq \mathsf{index}(n_2)$, we assume that $\mathsf{index}(n_1) < \mathsf{index}(n_2)$ which translates into the swap of line 11. By the semantics of LVBDD we have that:

$$\theta = \mathsf{val}(n_1) \sqcap \big((\neg p_i \sqcap [\![\mathsf{lo}(n_1)]\!] \sqcap [\![n_2]\!]) \sqcup (p_i \sqcap [\![\mathsf{hi}(n_1)]\!] \sqcap [\![n_2]\!])\big)$$

By the induction hypothesis we know that: $\ell = \mathcal{D}^S([\![\mathsf{lo}(n_1)]\!] \sqcap [\![n_2]\!])$ and that $h = \mathcal{D}^S([\![\mathsf{hi}(n_1)]\!] \sqcap [\![n_2]\!])$. The remainder of the proof is similar to the case where $\mathsf{index}(n_1) = \mathsf{index}(n_2)$. □

C Algorithm to compute the meet or the join of UNF LVBDD

Let n_1 and n_2 be two LVBDD in UNF and $\mathsf{op} \in \{\sqcap, \sqcup\}$. Then, $\mathsf{ApplyUNF}(n_1, n_2, \mathsf{op})$ (see Algorithm 5) computes a LVBDD n' in UNF such that $[\![n']\!] = [\![n_1]\!] \,\mathsf{op}\, [\![n_2]\!]$. To determine the complexity of this algorithm, we can remark that it only generates recursive

Algorithm 5: Meet or Join of two LVBDD in UNF

```
1  begin ApplyUNF($n_1, n_2$, op)
2      $n' :=$ memo?($\langle n_1, n_2, \mathsf{op}\rangle$) ;
3      if $n' \neq$ nil then return $n'$ ;
4      else if index($n_1$) = index($n_2$) = $k+1$ then
5          $n' :=$ MK($k+1$, val($n_1$) op val($n_2$), nil, nil) ;
6      else
7          if index($n_1$) = index($n_2$) then
8              $\ell :=$ ApplyUNF(lo($n_1$), lo($n_2$), op) ; $h :=$ ApplyUNF(hi($n_1$), hi($n_2$), op) ;
9          else
10             if index($n_1$) > index($n_2$) then swap($n_1, n_2$) ;
11             $\ell :=$ ApplyUNF(lo($n_1$), $n_2$) ; $h :=$ ApplyUNF(hi($n_1$), $n_2$) ;
12         if $\ell = h$ then $n' := \ell$ ;
13         else $n' :=$ MK(index($n_1$), $\top$, $\ell$, $h$) ;
14     memo!($\langle n_1, n_2, \mathsf{op}\rangle, n'$) ; memo!($\langle n_2, n_1, \mathsf{op}\rangle, n'$) ;
15     return $n'$ ;
16 end
```

calls the first time it is invoked on a given pair of nodes (n_1, n_2), thanks to memoization. Thus, the total number of recursive operations does not exceed $\mathcal{O}(|n_1|.|n_2|)$. Moreover, all the other operations within a single call require constant time. Thus, the procedure $\mathsf{ApplyUNF}(n_1, n_2, \mathsf{op})$ runs in time $\mathcal{O}(|n_1|.|n_2|)$ in the worst case.

Lemma 12. *for all* $n_1, n_2 \in \mathsf{LVBDD}^U(\mathsf{P}, \mathcal{L})$*, and* $\mathsf{op} \in \{\sqcap, \sqcup\}$*:* $\mathcal{D}^U(\llbracket n_1 \rrbracket \, \mathsf{op} \llbracket n_2 \rrbracket) = \mathsf{ApplyUNF}(n_1, n_2, \mathsf{op})$.

Proof (soundness of Algorithm 5). Since Algorithm 5 is recursive, the proof is by induction on the structure of the LVBDD n_1 and n_2. Throughout the proof, we denote by θ the function $\llbracket n_1 \rrbracket \, \mathsf{op} \llbracket n_2 \rrbracket$. We thus establish that the LVBDD returned by Algorithm 5 is $\mathcal{D}^U(\theta)$.

Base Case The base case is when n_1 and n_2 are terminal, i.e., $\mathsf{index}(n_1) = \mathsf{index}(n_2) = k+1$. In this case, $\theta = d \in L$, and thus $I_\theta = \emptyset$. Thus, $\mathcal{D}^U(\theta)$ is the terminal LVBDD $\langle k+1, d\rangle$. It is easy to see that, in this case, Algorithm 5 computes and returns $\mathcal{D}^U(\theta)$, since it enters the enters the **if** at line 4.

Inductive Case The induction hypothesis is that every recursive call performed by Algorithm 5 respects the Lemma. Under this hypothesis, let us prove that the LVBDD returned by Algorithm 5 is $\mathcal{D}^U(\theta)$. First, since n_1 and n_2 are in UNF, and by definition of $\llbracket n_1 \rrbracket$ and $\llbracket n_2 \rrbracket$, we know that the following equalities hold:

$$\begin{array}{ll} \llbracket n_1 \rrbracket|_{p_{\mathsf{index}(n_1)}=0} = \llbracket \mathsf{lo}(n_1) \rrbracket & \llbracket n_2 \rrbracket|_{p_{\mathsf{index}(n_2)}=0} = \llbracket \mathsf{lo}(n_2) \rrbracket \\ \llbracket n_1 \rrbracket|_{p_{\mathsf{index}(n_1)}=1} = \llbracket \mathsf{hi}(n_1) \rrbracket & \llbracket n_2 \rrbracket|_{p_{\mathsf{index}(n_2)}=1} = \llbracket \mathsf{hi}(n_2) \rrbracket \end{array} \tag{9}$$

Then, we consider several cases:

1. Either $\mathsf{index}(n_1) = \mathsf{index}(n_2) = i$. In this case, by (9), we obtain:

$$\theta = \lambda v. \begin{cases} [\![n_1]\!]|_{p_i=0} \,\mathsf{op} [\![n_2]\!]|_{p_i=0} = [\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![\mathsf{lo}(n_2)]\!] & \text{If } v(p_i) = 0 \\ [\![n_1]\!]|_{p_i=1} \,\mathsf{op} [\![n_2]\!]|_{p_i=1} = [\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![\mathsf{hi}(n_2)]\!] & \text{If } v(p_i) = 1 \end{cases} \quad (10)$$

by definition of the application of $\mathsf{op} \in \{\sqcap, \sqcup\}$ on two LVBF. Moreover, we observe that, since $\mathsf{index}(n_1) = \mathsf{index}(n_2) = i$, Algorithm 5 enters the **if** at line 7, and thus compute the LVBDD $\ell = D^U([\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![\mathsf{lo}(n_2)]\!])$, and $h = D^U([\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![\mathsf{hi}(n_2)]\!])$, by induction hypothesis. Then, we consider two further cases:
   (a) Either $p_i \in I_\theta$. In this case, by Definition 2, $\mathcal{D}^U(\theta)$ is the non-terminal LVBDD $\langle i, \top, \mathcal{D}^U(\theta|_{p_i=0}), \mathcal{D}^U(\theta|_{p_i=1})\rangle$. Thus, by (10) and Definition 2, $\mathcal{D}^U(\theta)$ is the LVBDD $\langle i, \top, \mathcal{D}^U([\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![\mathsf{lo}(n_2)]\!]), \mathcal{D}^U([\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![\mathsf{hi}(n_2)]\!])\rangle$. Moreover, $\ell \neq h$ because $p_i \in I_\theta$, since otherwise we would have $\theta|_{p_i=0} = \theta|_{p_i=1}$, by (10), which contradicts $p_i \in I_\theta$. Thus, Algorithm 5 enters the **else** at line 13 and returns the LVBDD

$$\begin{aligned} &\langle \mathsf{index}(n_1), \top, \ell, h\rangle \\ &= \langle i, \top, D^U([\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![\mathsf{lo}(n_2)]\!]), D^U([\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![\mathsf{hi}(n_2)]\!])\rangle \end{aligned}$$

   which is the expected result.
   (b) Or $p_i \notin I_\theta$. Since $p_i \notin I_\theta$, $\theta = \theta|_{p_i=0} = \theta|_{p_i=1}$ by definition of I_θ. Thus, by (10), $D^U(\theta) = D^U([\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![\mathsf{lo}(n_2)]\!])$. On the other hand, since $p_i \notin I_\theta$, we know that $\theta|_{p_i=0} = \theta|_{p_i=1}$, and thus that

$$[\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![\mathsf{lo}(n_2)]\!] = [\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![\mathsf{hi}(n_2)]\!]$$

   by (10). Hence,

$$\ell = D^U([\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![\mathsf{lo}(n_2)]\!]) = D^U([\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![\mathsf{hi}(n_2)]\!]) = h.$$

   Since $\ell = h$, Algorithm (10) enters the **if** at line 12 and returns

$$\ell = D^U([\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![\mathsf{lo}(n_2)]\!])$$

   which is the expected result.
2. Or $i = \mathsf{index}(n_1) < \mathsf{index}(n_2)$. In this case, by (9), we obtain:

$$\theta = \lambda v. \begin{cases} [\![n_1]\!]|_{p_i=0} \,\mathsf{op} [\![n_2]\!] = [\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![n_2]\!] & \text{If } v(p_i) = 0 \\ [\![n_1]\!]|_{p_i=1} \,\mathsf{op} [\![n_2]\!] = [\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![n_2]\!] & \text{If } v(p_i) = 1 \end{cases} \quad (11)$$

by definition of the application of $\mathsf{op} \in \{\sqcap, \sqcup\}$ on two LVBF. Moreover, we observe that, since $i = \mathsf{index}(n_1) < \mathsf{index}(n_2)$, Algorithm 5 enters the **else** at line 9, and computes $\ell = D^U([\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![n_2]\!])$, and $h = D^U([\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![n_2]\!])$, by induction hypothesis. Then, we consider two further cases:
   (a) Either $p_i \in I_\theta$. By similar arguments as above, we conclude that $\mathcal{D}^U(\theta)$ is the non-terminal LVBDD $\langle i, \top, \mathcal{D}^U([\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![n_2]\!]), \mathcal{D}^U([\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![n_2]\!])\rangle$, and that $h \neq \ell$. On the other hand, Algorithm 5 enters the **else** at line 13 and returns the LVBDD

$$\begin{aligned} &\langle \mathsf{index}(n_1), \top, \ell, h\rangle \\ &= \langle i, \top, D^U([\![\mathsf{lo}(n_1)]\!] \,\mathsf{op} [\![n_2]\!]), D^U([\![\mathsf{hi}(n_1)]\!] \,\mathsf{op} [\![n_2]\!])\rangle \end{aligned}$$

   which is the expected result.

(b) Or $p_i \not\in I_\theta$. By similar arguments as above, we conclude that $D^U(\theta) = D^U(\llbracket \mathsf{lo}(n_1) \rrbracket \mathbin{\mathsf{op}} \llbracket n_2 \rrbracket)$ and that $h = \ell$. On the other hand, Algorithm 5 enters the **if** at line 12 and returns $\ell = D^U(\llbracket \mathsf{lo}(n_1) \rrbracket \mathbin{\mathsf{op}} \llbracket n_2 \rrbracket)$, which is the expected result.

3. Or $\mathsf{index}(n_1) > \mathsf{index}(n_2)$. This case is symmetrical to the former and is treated by a swap at line 10, which reduces it to the former case. $\square$