On Boolean Functions Encodable as a Single Linear Pseudo-Boolean Constraint

Jan-Georg Smaus

Institut für Informatik, Universität Freiburg, Georges-Köhler-Allee 52, 79110 Freiburg im Breisgau, Germany, smaus@informatik.uni-freiburg.de

March 6, 2007

Technical Report of Universität Freiburg No. 230
AVACS Technical Report No. 13

Abstract. A *linear pseudo-Boolean constraint* (LPB) is an expression of the form $a_1 \cdot \ell_1 + \ldots + a_m \cdot \ell_m \geq d$, where each ℓ_i is a *literal* (it assumes the value 1 or 0 depending on whether a propositional variable x_i is true or false) and $a_1, \ldots, a_m, d$ are natural numbers. An LPB is a generalisation of a propositional clause, on the other hand it is a restriction of *integer linear programming*. LPBs can be used to represent Boolean functions more compactly than the well-known *conjunctive* or *disjunctive* normal forms. In this paper, we address the question: *how much* more compactly? We compare the expressiveness of a single LPB to that of related formalisms, and give an algorithm for computing an LPB representation of a given formula if this is possible.
Note: This report is the long version of [18] and contains the proofs omitted there for space reasons.

1 Introduction

A *linear pseudo-Boolean constraint* (LPB) [1, 3, 5–8] is an expression of the form $a_1\ell_1 + \ldots + a_m\ell_m \geq d$. Here each ℓ_i is a *literal* of the form x_i or $\bar{x}_i \equiv 1 - x_i$, i.e. x_i becomes 0 if x_i is false and 1 if x_i is true, and vice versa for $\bar{x}_i$. Moreover, $a_1, \ldots, a_m, d$ are natural numbers.

An LPB can be used to represent a Boolean[1] function; e.g. $x_1 + \bar{x}_2 + x_3 \geq 3$ represents the same function as the propositional formula $x_1 \wedge \neg x_2 \wedge x_3$ (we identify propositional formulae with functions). It has been observed that a function can be often represented more compactly as a set of LPBs than as a *conjunctive* or *disjunctive* normal form (CNF or DNF) [5–8]. E.g. the LPB $2x_1+\bar{x}_2+x_3+x_4 \geq 2$ corresponds to the DNF $x_1 \vee (\neg x_2 \wedge x_3) \vee (\neg x_2 \wedge x_4) \vee (x_3 \wedge x_4)$.

The interest in Boolean functions, or propositional logic, comes from countless applications in verification, (symbolic) model checking and design automation concerning finite state systems [1, 3–8, 12, 13, 21].

Previous works on LPBs [1, 5–8] have focused on generalising techniques applied in CNF-based propositional satisfiability solving [12, 13, 21] to LPBs, emphasising that this is beneficial because of the compactness of LPB representations. Dixon and Ginsberg show that since LPBs are a special case of *integer programming*, the *cutting planes* proof system, a standard technique in operations research (OR), can be applied to LPBs. Cutting planes is a generalisation of *resolution*, a standard technique in artificial intelligence (AI). Cutting planes proofs can be exponentially shorter than resolution proofs [6].

But where do the LPBs come from? One possibility is that for an application domain, one gives a direct representation of a problem as a *set* of LPBs (usually

[1] Whenever we say "function" we mean "Boolean function".

interpreted as conjunction but also a disjunction is thinkable) and argues that the alternative representation as CNF would be less compact [1, 6, 8]. Another possibility is that one considers problem representations given as a CNF or DNF and transforms these into compact LPB representations. We are not aware that the latter has ever been proposed. In addition, except [8] (discussed in Subsec. 6.1), all the arguments that we found in favour of LPBs were not *strictly* about LPBs but about *cardinality constraints*, which are a subclass of LPBs. This raises the question: how can a propositional formula be transformed into an LPB representation that is as compact as possible? As a first but crucial step towards this aim, we believe that one should study the question which functions can be expressed by *a single* LPB, i.e. whether or not a given CNF or DNF represents a *threshold function* [15]. This is the topic of this paper.

In Sec. 3 we show that there is an inclusion chain from clauses to cardinality constraints to LPBs to the *monotone* functions (functions represented by a formula where each variable occurs only in one polarity).

In Sec. 4 we recall the difficulty of determining the number of monotone functions, and give some results on the cardinality of classes of functions. We give an upper bound for the blowup of using a DNF instead of an LPB encoding.

In Sec. 5, we show that if a DNF can be expressed by an LPB, then the dual CNF can be expressed by a very similar LPB, and vice versa.

In Sec. 6 we give a theorem that states that ϕ can be represented as an LPB if and only if ϕ can be decomposed into several smaller formulae, each of which can be represented by an LPB, and all these LPBs are in a certain sense very similar. Based on this theorem we give an algorithm for converting a DNF ϕ to an LPB if possible.

2 Preliminaries

We assume the reader to be familiar with the basic notions of propositional logic.

An m**-dimensional Boolean function** f is a function $Bool^m \to Bool$. We say that f **properly depends** on the ith argument if there exist $\beta \in Bool^{i-1}$, $\beta' \in Bool^{m-i}$ with $f(\beta, 0, \beta') \neq f(\beta, 1, \beta')$.

We follow [5]. A **0-1 ILP constraint** is an inequality of the form

$$a_1 x_1 + \ldots + a_m x_m \geq d \qquad a_i, d \in \mathbb{R}, x_i \in Bool \ (Bool \equiv \{0, 1\}). \tag{1}$$

We identify 0 with *false* and 1 with *true*. We call the a_i **coefficients** and d the **degree** [9].

Using the relation $\bar{x}_i \equiv 1 - x_i$ and noting that it is sufficient to consider integer coefficients, one can rewrite a 0-1 ILP constraint as a **linear pseudo-Boolean constraint** (LPB)

$$a_1 \ell_1 + \ldots + a_m \ell_m \geq d \qquad a_i \in \mathbb{N}, d \in \mathbb{Z}, \ell_i \in \{x_i, \bar{x}_i\}. \tag{2}$$

For example, $x_1 - 0.5x_2 - 0.5x_3 \geq 0$ can be written as $2x_1 + \bar{x}_2 + \bar{x}_3 \geq 2$. An occurrence of a **literal** x_i (resp., $\bar{x}_i$) is called an occurrence of x_i in **positive** (resp., **negative**) polarity. Note that if $d \leq 0$, then the LPB is a tautology. The reason for allowing for negative d will become apparent in Subsec. 6.2.

An LPB where $a_i = 1$ or $a_i = 0$ for all $i \in [1..m]$ is called a **cardinality constraint** (e.g. for $m = 4$: $1x_1 + 0x_2 + 1x_3 + 0x_4 \geq 1$, in short $x_1 + x_3 \geq 1$). Note that $\sum_{i \in J} \ell_i \geq 1$ (resp., $\sum_{i \in J} \ell_i \geq |J|$) corresponds to $\bigvee_{i \in J} \ell_i$ (resp., $\bigwedge_{i \in J} \ell_i$).

A **CNF** is a formula of the form $c_1 \wedge \ldots \wedge c_n$ where each **clause** c_j is a disjunction of literals. A **DNF** is defined dually; a conjunction of literals is called a **(dual) clause**. Formally, CNFs and DNFs are sets of sets of literals, i.e. the order of clauses and the order of literals within a clause are insignificant. For

CNFs and DNFs, we assume without loss of generality that no clause is a subset of another clause (the latter clause would be redundant since it is *absorbed*). Given a CNF, the **dual** DNF is obtained by swapping $\wedge$ and $\vee$. Any Boolean function can be represented by a CNF or DNF [20].

An **assignment** σ is a mapping $\{x_1, \ldots, x_m\} \to \mathit{Bool}$. The notion '$\sigma$ **satisfies** an LPB I' is defined as expected [7].

3 Inclusion Results

The results of this section are not difficult but provide some useful insights into the expressiveness of an LPB or cardinality constraint.

Following [19], we define **monotone** functions as follows.

Definition 3.1. A function is **monotone** (or *unate* [5]) if it can be written as $\vee, \wedge$-combination of literals, where each variable occurs in only one polarity. A monotone function is **isotone** if all variables occur in positive polarity.

We will now show that the functions representable by a single LPB are a subset of the monotone functions, and related results. It turns out that the polarity of a particular variable is an issue that is orthogonal to these results: each monotone function has 2^m variants obtained by modifying the polarity of each variable. Thus in this section, we assume without loss of generality that each variable has positive polarity.

We say that assignment σ **minimally** satisfies the LPB I if σ satisfies I and any assignment obtained from σ by changing any variable occurring in I from *true* to *false* does not satisfy I. We say that a dual clause **corresponds** to an assignment if it consists of the variables assigned *true* by σ.

Proposition 3.2. An LPB I represents the DNF that consists of exactly those dual clauses that correspond to assignments that minimally fulfill I.

We now give an inclusion result between the functions representable as a single LPB and monotone functions.

Lemma 3.3. Every LPB represents a monotone function. For $m \geq 4$, there is at least one monotone function not represented by any LPB. For $m \leq 3$, each monotone function can be represented as LPB.

Proof. The first statement is a consequence of Prop. 3.2 (the dual clauses mentioned there contain no negated variables).

The second statement is witnessed by the example $(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$: Suppose $I \equiv a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 \geq d$ is equivalent to $(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$. Then some straightforward considerations about when I must be true imply that $a_1 + a_3 < d$, $a_2 + a_4 < d$, $a_1 + a_2 \geq d$, $a_3 + a_4 \geq d$, which is a contradiction.

Concerning the third statement: We show that there are exactly 9 *iso*tone functions *properly* depending on 3 variables. The remaining generalisations are straightforward. The following enumeration shows all CNFs representing isotone functions, and the respective LPB representations:

$$\begin{array}{rll}
x_1 \vee x_2 \vee x_3 & x_1 + x_2 + x_3 \geq 1 & \\
x_1 \wedge (x_2 \vee x_3) & 2x_1 + x_2 + x_3 \geq 3 & (+2 \text{ symmetric ones}) \\
(x_1 \vee x_2) \wedge (x_1 \vee x_3) & 2x_1 + x_2 + x_3 \geq 2 & (+2 \text{ symmetric ones}) \\
(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3) & x_1 + x_2 + x_3 \geq 2 & \\
x_1 \wedge x_2 \wedge x_3 & x_1 + x_2 + x_3 \geq 3 &
\end{array}$$

□

We now give an inclusion result between LPBs and cardinality constraints.

Lemma 3.4. Every cardinality constraint is an LPB. For $m \geq 3$, there is at least one LPB not expressible as cardinality constraint. For $m \leq 2$, each monotone function can be represented as a cardinality constraint.

Proof. The first statement holds by definition. The second is witnessed by the example $2x_1 + x_2 + x_3 \geq 2$.

Concerning the third statement: By inspection it can be seen that there are 14 monotone functions (6 of which are *iso*tone) of which 10 can be represented as a usual clause, and another 4 can be represented as cardinality constraint. □

We now give an inclusion result between cardinality constraints and clauses.

Lemma 3.5. Every clause is a cardinality constraint. For every $m \geq 0$, there is at least one cardinality constraint not expressible as clause.

Proof. A cardinality constraint with $d = 1$ corresponds to a clause. On the other hand, *true* is represented by the cardinality constraint $\sum_{i=1}^{m} x_i \geq 0$ (even for $m = 0$) but by no clause (the empty clause corresponds to *false*). □

Summarising, we have "clauses" $\subseteq$ "cardinality constraints" $\subseteq$ "LPBs" $\subseteq$ "monotone functions", where these inclusions are strict except for very small dimensions.

4 Counting Boolean Functions

For comparing the expressiveness of formalisms for Boolean functions, it is of interest to compare the cardinalities of the function classes that can be expressed by the formalisms. Note, however, that from such comparisons we cannot infer how much blowup there is when translating from one formalism to another. We will come back to this point at the end of this section.

Proposition 4.1. There are $2^{(2^m)}$ m-dimensional Boolean functions [20].

The question of how many monotone functions there are is called *Dedekind's problem*, unsolved for more than a century. To be precise, Dedekind's problem is to determine the number of *iso*tone m-dimensional functions (*Dedekind numbers*). Confusingly, what we call *iso*tone is sometimes called *mono*tone, but we use the terminology of [19]. Nobody has found yet a closed form expression for the Dedekind numbers. In 1999, they have been calculated for up to $m = 8$, where the value is 56130437228687557907788. The Dedekind numbers are Sequence A000372 of [16]. Although the number of isotone functions is large, it is a small fraction of the number $2^{(2^m)}$ of Boolean functions [19]. The best known bound for the Dedekind numbers is given by [11].

We show that the number of *mono*tone functions is related to the number of isotone functions, so that finding a closed form expression for the former cannot be easier than for the latter. We need the following notations.

Definition 4.2. We denote by $\mathfrak{I}^{\leq}(m)$ the number of m-dimensional isotone functions (Dedekind numbers); by $\mathfrak{I}^{=}(m)$ the number of isotone functions that *properly* depend on m variables; and by $\mathfrak{M}^{\leq}(m)$, $\mathfrak{M}^{=}(m)$ the corresponding numbers of *mono*tone functions.

Lemma 4.3. The following identities hold:

$$\mathfrak{I}^{\leq}(m) = \sum_{i=0}^{m} \binom{m}{i} \mathfrak{I}^{=}(i) \tag{3}$$

$$\mathfrak{M}^{=}(m) = 2^m \mathfrak{I}^{=}(m) \tag{4}$$

$$\mathfrak{M}^{\leq}(m) = \sum_{i=0}^{m} \binom{m}{i} \mathfrak{M}^{=}(i) \;=\; \sum_{i=0}^{m} \binom{m}{i} 2^i \mathfrak{I}^{=}(i) \tag{5}$$

Proof. For (3), any m-dimensional isotone function that depends *properly* on i variables is given by choosing i out of m variables (there are $\binom{m}{i}$ ways of doing this), and then by choosing one of the $\mathfrak{I}^{=}(i)$ i-dimensional isotone functions that properly depend on i variables. Since $0 \leq i \leq m$, we get the given sum.

Each monotone function properly depending on m variables can be obtained by taking a uniquely determined isotone function properly depending on m variables and changing the polarity for some of the variables; there are clearly 2^m ways of doing this, giving (4).

The reasoning for (5) is as for (3), where the second equality follows from (4). □

The following table shows some of the values. Note that $\mathfrak{I}^{=}(3) = 9$ and $\mathfrak{M}^{\leq}(2) = 14$ have already been mentioned in Sec. 3.

m	$\mathfrak{I}^{=}(m)$	$\mathfrak{I}^{\leq}(m)$	$\mathfrak{M}^{=}(m)$	$\mathfrak{M}^{\leq}(m)$
0	2	2	2	2
1	1	3	2	4
2	2	6	8	**14**
3	**9**	20	72	104
4	114	168	1824	2170

The number of LPBs describing distinct m-dimensional functions will probably not be easier to describe than the Dedekind numbers [14, p. 64][15]. It is not difficult though to make a statement about cardinality constraints.

Lemma 4.4. There are $2+\sum_{k=1}^{m}\binom{m}{k}\cdot 2^k \cdot k$ cardinality constraints representing distinct m-dimensional functions.

Proof. It is easy to see that there are many ways of representing *true* and *false* as cardinality constraints. We show that there are $\sum_{k=1}^{m}\binom{m}{k}\cdot 2^k \cdot k$ cardinality constraints representing distinct non-trivial (i.e., other than *true* and *false*) Boolean functions.

The cardinality constraint can involve between 1 and m variables. Given $k \in [1..m]$, there are $\binom{m}{k}$ ways of choosing those variables. Each variable can be positive or negative, giving 2^k. The degree can be between 1 and k. □

Also it is not difficult to make a statement about (dual) clauses.

Proposition 4.5. There are 3^m m-dimensional functions expressible as clauses, and likewise for dual clauses.

Proof. Each variable is either not in the clause or positive or negative. □

Arguing as in Prop. 4.5, one can give a loose upper bound $3^m \cdot m$ for the number of cardinality constraints, since the degree can be between 1 and m. So the number of cardinality constraints is at most a linear factor above that of usual clauses. However, encoding one cardinality constraint as CNF can entail an exponential blowup in formula size (not considering encodings involving auxiliary variables, encodings which are not equivalence preserving). More precisely, encoding $x_1+\ldots+x_m \geq k$ requires $\binom{m}{(m-k)+1} = \binom{m}{k-1}$ clauses of length $m-k+1$ as CNF [3] and $\binom{m}{k}$ dual clauses of length k as DNF (in [7], this is said for CNF but in fact it should be DNF). Note that $\binom{m}{\lfloor m/2 \rfloor} \geq 2^{m/2}$.

The blowup when encoding an *LPB* as CNF or DNF is not worse however.

Lemma 4.6. Let $I \equiv \sum_{i=1}^{m} a_i x_i \geq d$ be an LPB. The DNF (CNF) ϕ represented by I has at most $\binom{m}{\lfloor m/2 \rfloor}$ clauses.

Proof. By Sec. 2, formally ϕ must be a subset of $\wp(\{x_1, \ldots, x_m\})$ such that no clause of ϕ is a subset of another clause of ϕ. Thus, for any chain $\emptyset = c_0 \subset \ldots \subset c_m = \{x_1, \ldots, x_m\}$ where $c_k \subseteq \{x_1, \ldots, x_m\}$ and $|c_k| = k$ for all $k \in [0..m]$, ϕ must contain at most one clause from $c_0, \ldots, c_m$. For each $k \in [0..m]$, there are $\binom{m}{k}$ different subsets of $\{x_1, \ldots, x_m\}$ having k elements. Now $\binom{m}{k}$ is maximal for $k = \lfloor m/2 \rfloor$, and thus ϕ can contain at most $\binom{m}{\lfloor m/2 \rfloor}$ clauses. □

Thus, an LPB can represent *more* DNFs than a cardinality constraint but not *bigger* DNFs. For example, $3x_1 + 2x_2 + 2x_3 + x_4 \geq 4$ represents a DNF of 4 dual clauses, while $2x_1 + 2x_2 + 2x_3 + 2x_4 \geq 4$ (which is effectively a cardinality constraint) represents a DNF of 6 dual clauses.

Note that the CNF or DNF corresponding to an LPB must be distinguished from translations of an LPB that introduce additional variables [2].

5 Duality

We show that if a DNF can be represented as an LPB, then the dual CNF can also be represented as an LPB, and the two LPBs are closely related. As in Sec. 3, we assume that each variable has positive polarity.

Theorem 5.1. If a DNF is represented by an LPB $I \equiv \sum_{i=1}^{m} a_i x_i \geq d$, then the dual CNF is represented by $\sum_{i=1}^{m} a_i x_i \geq \sum_{i=1}^{m} a_i + 1 - d$, and vice versa.

Proof. For a CNF $\phi \equiv c_1 \wedge \ldots \wedge c_n$ or a DNF $\phi \equiv c_1 \vee \ldots \vee c_n$, for any c_j, we call the set of variable indices occurring in c_j a *horizontal index set* of ϕ. Moreover, we call any set $V \subseteq [1..m]$ such that $\forall j \in [1..n].\ \exists x_i \in c_j.\ i \in V$ holds a *vertical index set* of ϕ.

Assume now the DNF $\phi \equiv c_1 \vee \ldots \vee c_n$ is represented by $I \equiv \sum_{i=1}^{m} a_i x_i \geq d$. If we make all the variables in one c_j true, then ϕ must be true. If we make all the variables with indices in a vertical index set false, then ϕ must be false. Hence for all horizontal index sets H and all vertical index sets V, it must hold that:

$$\sum_{i \in H} a_i \geq d \qquad \text{and} \qquad \sum_{i \notin V} a_i < d \tag{6}$$

Let ϕ' be the CNF dual to ϕ. Note that ϕ, ϕ' have the same horizontal and vertical index sets. If we make all the variables in one c_j false, then ϕ' must be false. If we make all the variables with indices in a vertical index set true, then ϕ' must be true. So if $I' \equiv \sum_{i=1}^{m} a'_i x_i \geq d'$ is an LPB representing ϕ', then for all horizontal index sets H and all vertical index sets V, it must hold that:

$$\sum_{i \notin H} a'_i < d' \qquad \text{and} \qquad \sum_{i \in V} a'_i \geq d' \tag{7}$$

We show that by setting $a'_i = a_i$ for $i \in [1..m]$ and $d' = \sum_{i=1}^{m} a_i + 1 - d$, (7) is fulfilled and thus I' is indeed an LPB representing ϕ'.

Let H be an arbitrary horizontal index set of ϕ. Then we have

$$\sum_{i \in H} a_i \geq d \Rightarrow \sum_{i \in H} a_i > d - 1 \Rightarrow 0 < \sum_{i \in H} a_i + 1 - d \Rightarrow \sum_{i \notin H} a_i < \sum_{i=1}^{m} a_i + 1 - d,$$

thus the first inequality of (7) holds. Now let V be an arbitrary vertical index set of ϕ. Then we have

$$\sum_{i \notin V} a_i < d \Rightarrow \sum_{i \notin V} a_i \leq d - 1 \Rightarrow 0 \geq \sum_{i \notin V} a_i + 1 - d \Rightarrow \sum_{i \in V} a_i \geq \sum_{i=1}^{m} a_i + 1 - d,$$

thus the second inequality of (7) holds.

The proof of the converse is analogous. □

Note the border cases: $\sum_{i=1}^{m} a_i x_i \geq \sum_{i=1}^{m} a_i$ represents a conjunction (of variables), $\sum_{i=1}^{m} a_i x_i \geq 1$ represents a disjunction.

Example 5.2. Consider $5x_1 + 2x_2 + 2x_3 + 2x_4 \geq i$ for $i \in [1..11]$. Note first that for $i = 1, 2$ the represented function is the same, and the dual of that function is represented by setting $i = 11, 10$. Similarly one has $i = 3, 4$ vs. $i = 9, 8$. For $i = 5$, the DNF is $x_1 \vee (x_2 \wedge x_3 \wedge x_4)$, and the dual CNF $x_1 \wedge (x_2 \vee x_3 \vee x_4)$ is represented by setting $i = 7$. For $i = 6$, the LPB represents $(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3 \wedge x_4)$. According to Thm. 5.1, since $12 - 6 = 6$, the dual CNF is represented by the same LPB, which means that the CNF is equivalent to its dual. This can easily be confirmed.

6 Representing a DNF as LPB

In this section we present an algorithm for the problem of converting a DNF to an equivalent LPB if possible.[2] Any results of this section can be applied to CNFs rather than DNFs using Sec. 5. In this section, by a *clause* we always mean a *dual* clause. As before, we assume that each variable has positive polarity.

6.1 Determining the Order of Coefficients

Given a DNF ϕ, one can determine a size order of the potential coefficients of an LPB representing ϕ. That is to say, if ϕ can be represented as an LPB at all, then the coefficients must respect this order.

The following notion is useful for reasoning about the structure of a formula.

Definition 6.1. Variables x and y are **symmetric** in ϕ if ϕ is equivalent to the formula obtained by exchanging x and y. A set of variables Y is **symmetric** in ϕ if each pair in Y is symmetric in ϕ.

Since the clause order and the order within a clause of a DNF or CNF is insignificant, symmetry is a straightforward syntactic property.

The following lemma relates symmetric variables to identical coefficients.

Lemma 6.2. Let $\sum_{i=1}^{m} a_i x_i \geq d$ be an LPB representing the DNF ϕ. For any $i, k \in [1..m]$, if $a_i = a_k$ then x_i, x_k are symmetric in ϕ; moreover, there exists an LPB $\sum_{i=1}^{m} a'_i x_i \geq d'$ representing ϕ such that if x_i, x_k are symmetric in ϕ then $a'_i = a'_k$.

Proof. For the first statement, for any clause that corresponds to an assignment that minimally satisfies I, the clause obtained by swapping x_i and x_k also corresponds to an assignment that minimally satisfies I.

For the second statement, assume that Φ can be represented by I and consider symmetric variables x_i, x_k. For any assignment that minimally satisfies I, the symmetry of x_i, x_k in Φ implies that the assignment obtained by swapping the truth values of x_i, x_k also minimally satisfies I. But then one can see that by defining $a'_i = a'_k = a_i + a_k$, $a'_l = 2a_l$ for all $l \in [1..m] \setminus \{i, k\}$, and $d' = 2d$, the LPB I' represents Φ and meets the requirement for x_i, x_k (intuitively, one replaces a_i and a_k by their average). The argument must be repeated for any pair of symmetric variables. □

[2] By Prop. 3.2, there is of course a naïve semi-decision procedure for this problem, involving enumeration of all LPBs.

For example, $x_1 \vee x_2$ can be represented by $2x_1 + x_2 \geq 1$ or $x_1 + x_2 \geq 1$.

We want to measure how often a variable occurs in a DNF, taking the length of the clauses into account. Intuitively, a variable is "important" if it occurs in *many* clauses and if it occurs in *short* clauses. To formalise this, we consider multisets of natural numbers. We represent multisets as strings of numbers in ascending order, written, e.g. $\{\!\{1, 1, 2\}\!\}$.

Definition 6.3. Let A, B be two multisets of numbers. We write $B \preceq A$ if B is obtained from A as follows: for each occurrence of a number n in A, either leave this occurrence in B, or replace it by an arbitrary (possibly 0) number of occurrences of numbers $> n$. We write $B \prec A$ if $B \preceq A$ and $A \not\preceq B$.

Example 6.4. We have $\{\!\{2, 2, 2, 2\}\!\} \succ \{\!\{2, 2, 2\}\!\} \succ \{\!\{2, 2, 3\}\!\} \succ \{\!\{2, 3\}\!\}$.

Note that $\{\!\{2, 2, 3\}\!\} \succ \{\!\{2, 3\}\!\}$ can be established in two ways: removing one 2 from $\{\!\{2, 2, 3\}\!\}$, or removing the 3 from $\{\!\{2, 2, 3\}\!\}$ and then replacing one 2 from $\{\!\{2, 2\}\!\}$ by one 3. $\preceq$ is a total order on multisets of natural numbers. In our representation, to determine whether $A \succeq B$, one must simply cut off the longest common prefix of A and B. If the remainder of A starts with a smaller number than that of B, or if the remainder of B is empty, then $A \succeq B$.

Definition 6.5. For a DNF ϕ, define $OP(\phi, x)$ as the multiset having one occurrence of n for each clause of length n in ϕ that contains x. We call $OP(\phi, x)$ the **occurrence pattern** of x.

Example 6.6. Consider $\phi \equiv (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_1 \wedge x_5) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4) \vee (x_3 \wedge x_4 \wedge x_5)$. The occurrence patterns are $OP(\phi, x_1) = \{\!\{2, 2, 2, 2\}\!\}$, $OP(\phi, x_2) = \{\!\{2, 2, 2\}\!\}$, $OP(\phi, x_3) = OP(\phi, x_4) = \{\!\{2, 2, 3\}\!\}$, and $OP(\phi, x_5) = \{\!\{2, 3\}\!\}$. ϕ can be represented by $4x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 5$.

To give some more intuition, compare x_1 and x_2, say. For clause $x_1 \wedge x_4$, replacing x_1 by x_2 yields another clause of ϕ, but for $x_1 \wedge x_5$ this is not the case. $OP(\phi, x_1)$ therefore has one more occurrence of 2 than $OP(\phi, x_2)$. The fact that replacing x_1 by x_2 in $x_1 \wedge x_5$ does not yield another clause of ϕ means that x_1 must have a bigger coefficient than x_2, in any LPB representing ϕ.

Computing the set of occurrence patterns for all variables in ϕ can be done in time linear in $|\phi|$. In fact, the number of elements of all occurrence patterns is exactly the number of literals in ϕ. Thus sorting the variables w.r.t. the occurrence patterns can be done in time polynomial in $|\phi|$.

The next lemma says that the coefficients of an LPB representing a DNF must correspond to the order given by the occurrence patterns. Thus, given a DNF, we know that *if* it can be represented as an LPB, then the coefficients of this LPB are ordered in a certain way.

Lemma 6.7. Let ϕ be a DNF represented by the LPB $\sum_{i=1}^{m} a_i x_i \geq d$. Then $a_i \geq a_k$ implies $OP(\phi, x_i) \succeq OP(\phi, x_k)$; moreover, there exists an LPB $\sum_{i=1}^{m} a'_i x_i \geq d'$ representing ϕ such that $OP(\phi, x_i) = OP(\phi, x_k)$ implies $a'_i = a'_k$.

Proof. Without loss of generality, assume $a_1 \geq \ldots \geq a_m$, and consider some x_i, x_k with $i < k$, i.e. $a_i \geq a_k$. We compare $OP(\phi, x_i)$ and $OP(\phi, x_k)$.

For each clause of length n in ϕ that contains x_i and x_k, we have an occurrence of n in $OP(\phi, x_i)$ and a uniquely matching occurrence of n in $OP(\phi, x_k)$.

For each clause of length n in ϕ that contains x_i, and for which replacing x_i with x_k gives another clause in ϕ, we have an occurrence of n in $OP(\phi, x_i)$ and a uniquely matching occurrence of n in $OP(\phi, x_k)$.

Now consider a clause of length n in ϕ that contains x_i, and for which replacing x_i with x_k does not give another clause in ϕ (the reason for this must

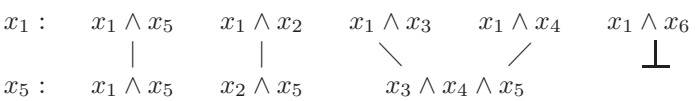
be that the corresponding assignment does not fulfill I). For such a clause, replacing x_i with x_k plus additional variables may give a clause in ϕ, in which case the occurrence of n in $OP(\phi, x_i)$ is mapped to one or more occurrences of numbers $> n$ in $OP(\phi, x_k)$ (note however that conversely, an occurrence of such a number in $OP(\phi, x_k)$ is not necessarily *uniquely* mapped to the occurrence of n in $OP(\phi, x_i)$). Or it may be the case that no way of replacing x_i with x_k plus additional variables gives a clause in ϕ, in which case the occurrence of n in $OP(\phi, x_i)$ is mapped to 0 occurrences of numbers $> n$ in $OP(\phi, x_k)$.

Summarising, we have $OP(\phi, x_i) \succeq OP(\phi, x_k)$, showing the first statement.

In particular, if there exists a clause of length n in ϕ that contains x_i, and for which replacing x_i with x_k does not give another clause in ϕ, then $OP(\phi, x_i) \succ OP(\phi, x_k)$. Therefore $OP(\phi, x_i) = OP(\phi, x_k)$ implies that for each clause of length n in ϕ that contains x_i, swapping x_i with x_k also gives a clause in ϕ. I.e. x_i and x_k are symmetric. Hence an LPB I' as required exists by Lemma 6.2. □

The proof is illustrated by the following example.

Example 6.8. Consider $(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_1 \wedge x_5) \vee (x_1 \wedge x_6) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4) \vee (x_2 \wedge x_5) \vee (x_2 \wedge x_6) \vee (x_3 \wedge x_4 \wedge x_5) \vee (x_3 \wedge x_4 \wedge x_6)$. The following picture illustrates the occurrence patterns of x_1 and x_5, and thereby all four cases of the proof of Lemma 6.7:



The crucial point is the following: it is not possible that a clause containing x_1 is mapped to a shorter clause containing x_5, since $a_1 \geq a_5$.

The following is a corollary of Lemmas 6.2 and 6.7.

Corollary 6.9. If the DNF ϕ is represented by an LPB I, then x_i, x_k are symmetric in ϕ iff x_i, x_k have identical occurrence patterns.

The results so far can be used to make statements about which DNFs can definitely not be represented by a single LPB. For example, it has been said that a single LPB can express an implication [7]. In [8], implications of the form $y \to (x_1 \wedge x_2)$ are expressed as LPB. In fact, we have a very limited result about implications that can be expressed by one LPB, but one can show that even a simple form of implications is beyond what can be expressed by a single LPB.

Lemma 6.10. $\sum_{i=1}^{m} x_i + m\,\bar{y} \geq m$ expresses $y \to (x_1 \wedge \ldots \wedge x_m)$ and $m\,y + \sum_{i=1}^{m} \bar{x}_i \geq m$ expresses $(x_1 \vee \ldots \vee x_m) \to y$.

An implication of the form $(x_1 \vee \ldots \vee x_m) \to (y_1 \wedge \ldots \wedge y_l)$, where $m, l \geq 2$ and $x_i \not\equiv y_k$ for all $i \in [1..m]$ and $k \in [1..l]$, cannot be expressed by a single LPB.

Proof. The first sentence is straightforward.

Concerning an implication of the form $(x_1 \vee \ldots \vee x_m) \to (y_1 \wedge \ldots \wedge y_l)$, note first that it is equivalent to the DNF $(\neg x_1 \wedge \ldots \wedge \neg x_m) \vee (y_1 \wedge \ldots \wedge y_l)$. To simplify, we flip the polarity of the x_i so that all variables have positive polarity: So suppose $(x_1 \wedge \ldots \wedge x_m) \vee (y_1 \wedge \ldots \wedge y_l)$ is represented by $I \equiv \sum_{i=1}^{m} a_i x_i + \sum_{i=1}^{l} b_i y_i \geq d$. By Cor. 6.9, $a_1 = \ldots = a_m$ and $b_1 = \ldots = b_l$ and $a_1 \neq b_1$. But then by Prop. 3.2 and straightforward arithmetic considerations, the DNF represented by I must contain at least one clause containing variables from $x_1, \ldots, x_m$ as well as $y_1, \ldots, y_l$, which is a contradiction. □

This result has also been shown in [10].

As another example of a DNF that is not representable as LPB, consider $\phi \equiv (x_1 \wedge x_2 \wedge x_5) \vee (x_1 \wedge x_4) \vee (x_3 \wedge x_4 \wedge x_5) \vee (x_2 \wedge x_3)$. We have $OP(\phi, x_i) = \{\!\{2, 3\}\!\}$ for $i \in [1..4]$, and yet $x_1, \ldots, x_4$ are not symmetric, and thus ϕ is not representable as LPB.

6.2 Decomposing a DNF

We want to find an LPB representing ϕ if possible. Using Lemma 6.7, we can establish the order of the coefficients. Assume the numbering is such that we have $OP(\phi, x_1) \succeq \ldots \succeq OP(\phi, x_m)$. Consider now the maximal set $X = \{x_1, \ldots, x_l\}$ such that $OP(\phi, x_1) = \ldots = OP(\phi, x_l)$ (=: $OP(\phi, X)$). If X is not symmetric in ϕ, then by Cor. 6.9, ϕ cannot be represented by an LPB and we can stop. Otherwise, we partition ϕ according to how many variables from X each clause contains. We then remove the variables from X from each clause, which gives $l+1$ subproblems. Theorem 6.15 states under which conditions solutions to these subproblems can be combined to an LPB for ϕ. However, since the solutions have to be similar in a certain sense, it turns out that we cannot simply solve the subproblems independently and *then* combine the solutions, but we must solve the subproblems in parallel, as will be shown in Subsec. 6.3.

The following statements do not require X to be *maximal*, e.g. if $\{x_1, \ldots, x_5\}$ is the maximal set such that $OP(\phi, x_1) = \ldots = OP(\phi, x_5)$, then the statements will also hold for $X = \{x_1, x_2, x_3\}$.

Note that our formalism bears a certain resemblance with [2], where one considers LPBs obtained from a certain given LPB by removing some variables.

Definition 6.11. Let ϕ be a DNF and X a subset of its variables with $|X| = l$. If ϕ contains a clause $c \subseteq X$, then let $k_{\max}$ be the length of the longest such clause; otherwise let $k_{\max} := \infty$. For $0 \leq k \leq l$, we define $S(\phi, X, k)$ as the disjunction of clauses from ϕ containing exactly $\min\{k, k_{\max}\}$ variables from X, with those variables removed.

When constructing the $S(\phi, X, k)$ from ϕ, we say that we *split away* the variables in X from ϕ.

Example 6.12. Let $\phi \equiv (x_1) \vee (x_2) \vee (x_3 \wedge x_4)$ and $X = \{x_1, x_2\}$. We have $k_{\max} = 1$. Then $S(\phi, X, 0) = (x_3 \wedge x_4)$, $S(\phi, X, 1) = \mathit{true}$ (i.e. the disjunction of twice the empty conjunction), and $S(\phi, X, 2) = \mathit{true}$.

We must solve the $l + 1$ subproblems in such a way that the resulting LPBs agree in all coefficients, and that the degree difference of neighbouring LPBs is always the same. Before giving the theorem, we give two examples for illustration.

Example 6.13. Consider $\phi \equiv (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3 \wedge x_4)$ and $X = \{x_1\}$. Then $S(\phi, X, 0) = x_2 \wedge x_3 \wedge x_4$, represented by $x_2 + x_3 + x_4 \geq 3$. Moreover, $S(\phi, X, 1) = x_2 \vee x_3 \vee x_4$, represented by $x_2 + x_3 + x_4 \geq 1$.

Since the coefficients of the two LPBs agree, it turns out that ϕ can be represented by $2x_1 + x_2 + x_3 + x_4 \geq 3$. The coefficient of x_1 is given by the difference of the two degrees, i.e. $3 - 1$.

Example 6.14. Consider $\phi \equiv (x_1 \wedge x_2) \vee (x_1 \wedge x_3 \wedge x_4) \vee (x_2 \wedge x_3 \wedge x_4)$ and $X = \{x_1, x_2\}$. We have $S(\phi, X, 0) = \mathit{false}$, represented by $x_3 + x_4 \geq 4$, $S(\phi, X, 1) = x_3 \wedge x_4$, represented by $x_3 + x_4 \geq 2$, and $S(\phi, X, 2) = \mathit{true}$, represented by $x_3 + x_4 \geq 0$. The DNF ϕ is represented by $2x_1 + 2x_2 + x_3 + x_4 \geq 4$. The coefficient of x_1, x_2 is given by $4 - 2 = 2 - 0 = 2$ (the degrees are "equidistant").

Theorem 6.15. Let ϕ be a DNF in variables $x_1, \ldots, x_m$ and suppose $X = \{x_1, \ldots, x_l\}$ are symmetric variables such that $OP(\phi, X)$ is maximal w.r.t. $\preceq$ in ϕ. Then ϕ is represented by an LPB $\sum_{i=1}^{m} a_i x_i \geq d$, where $a_1 = \ldots = a_l$, iff for all $k \in [0..l]$, the DNF $S(\phi, X, k)$ is represented by $\sum_{i=l+1}^{m} a_i x_i \geq d - k \cdot a_1$.

Proof. For an assignment σ and a set of variables V, we denote by $\sigma \setminus V$ the assignment that is undefined on V and else equal to σ. We denote by $\sigma \cup \{V \mapsto \mathit{true}\}$ the assignment that maps all variables of V to *true* and is else equal to

σ. For a clause c, we denote by $c \setminus V$ the clause obtained from c by deleting the variables in V.

Let n be the greatest number such that ϕ contains a clause $c \subseteq X$ with $|c| = n$; set n to $l+1$ if no such clause exists. Since the variables in X are symmetric, by Def. 6.11 the following holds for all $k \leq n$:

$$\text{If } c \in S(\phi, X, k), \text{ then for all } V \subseteq X \text{ with } |V| = k, \text{ we have } c \cup V \in \phi. \quad (8)$$

Throughout, we use Prop. 3.2 and use I as "macro" for $\sum_{i=1}^{m} a_i x_i \geq d$ and I' as "macro" for $\sum_{i=l+1}^{m} a_i x_i \geq d - k \cdot a_1$.

"$\Leftarrow$": We assume that each $S(\phi, X, k)$ is represented by I', and show that the assignments minimally fulfilling I correspond exactly to the clauses of ϕ.

a) Consider an assignment σ that minimally fulfils I and makes exactly k variables from X true, say, the set $V \subseteq X$. Note that $k \leq n$, since if making n variables from X true fulfils I, then an assignment that makes *more* than n variables from X true cannot *minimally* fulfil I. Then $\sigma \setminus V$ minimally fulfils I', and hence $c \setminus V \in S(\phi, X, k)$, where c is the clause corresponding to σ. By (8) this implies $c \in \phi$.
b) Conversely, consider a clause c in ϕ that contains exactly k variables from X, say the set $V \subseteq X$. Then $c \setminus V \in S(\phi, X, k)$ by Def. 6.11 and thus $\sigma \setminus V$ minimally fulfils I', where σ is the assignment corresponding to c, and thus σ minimally fulfils I.

"$\Rightarrow$": We assume that ϕ is represented by I and show that for each k the assignments minimally fulfilling I' correspond exactly to the clauses of $S(\phi, X, k)$.

c) Consider an arbitrary $k \in [1..l]$.
If $n < k \leq l$, then $n < l$, and since I represents ϕ, it follows that $n \cdot a_1 \geq d$ and hence $d - k \cdot a_1 \leq 0$, thus only the empty assignment fulfils I' *minimally*. The empty assignment corresponds to the empty clause, which is the only clause in $S(\phi, X, k)$ by Def. 6.11.
Otherwise, let σ be an assignment that minimally fulfils I' and c the clause corresponding to σ. Then for any $V \subseteq X$ with $|V| = k$, the assignment $\sigma \cup \{V \mapsto \mathit{true}\}$ fulfils I. By definition of n we have $(n-1) \cdot a_1 < d$ (making $n-1$ variables from X true is not sufficient to fulfill I) and so since $k \leq n$, we have $d - (k-1) \cdot a_1 > 0$. This, together with, $a_1 = \ldots = a_l > a_{l+1} \geq \ldots \geq a_m$, implies that the assignment $\sigma \cup \{V \mapsto \mathit{true}\}$ fulfils I *minimally* and hence $c \cup V \in \phi$ and hence $c \in S(\phi, X, k)$.
d) Conversely, consider a clause $c \in S(\phi, X, k)$ and let σ be the assignment corresponding to c.
If $n < k \leq l$, then $S(\phi, X, k) = \mathit{true}$ and σ is the empty assignment. Since $d - k \cdot a_1 \leq 0$ as in point c, σ minimally fulfils I'.
Otherwise by (8), for any $V \subseteq X$ with $|V| = k$, we have that $c \cup V \in \phi$ and thus $\sigma \cup \{V \mapsto \mathit{true}\}$ is an assignment that minimally fulfils I, and thus σ minimally fulfils I'. □

The remaining problem is that a DNF might be represented by various LPBs, and so even if the LPBs computed recursively do not have agreeing coefficients and equidistant degrees, one might find alternative LPBs (such as the non-obvious LPB for *false* in Ex. 6.14) so that Thm. 6.15 can be applied.

Before addressing this problem, we generalise LPBs by recording to what extent degrees can be shifted without changing the meaning. To formulate this, we temporarily lift the restriction that coefficients and degrees must be integers. How to obtain integers in the end is explained at the end of Subsec. 6.3.

Definition 6.16. Given an LPB $I \equiv \sum_{i=1}^{m} a_i x_i \geq d$, we call s the **minimum degree** of I if s is the smallest number (possibly $-\infty$) such that for any $s' \in (s, d]$, the LPB $\sum_{i=1}^{m} a_i x_i \geq s'$ represents the same function as I. We call b the **maximum degree** if b is the biggest number (possibly ∞) such that $\sum_{i=1}^{m} a_i x_i \geq b$ represents the same function as I.

Note that the minimum degree of I is not a possible degree of I. Since the minimum and maximum degrees of an LPB are more informative than its actual degree, we introduce the notation $\sum_{i=1}^{m} a_i x_i \geq (s, b]$ for denoting an LPB with minimum degree s and maximum degree b.

The next lemma strengthens Thm. 6.15, stating that information about minimum and maximum degrees can be maintained with little overhead.

Lemma 6.17. Make the same assumptions as in Thm. 6.15, and assume that for all $k \in [0..l]$, the DNF $S(\phi, X, k)$ is represented by $I^k \equiv \sum_{i=l+1}^{m} a_i x_i \geq d - k \cdot a_1$. Moreover, for all $k \in [0..l]$, let s_k, b_k be minimum and maximum degrees of I^k, respectively. Then $s := \max_{k \in [0..l]}(s_k + k \cdot a_1)$, $b := \min_{k \in [0..l]}(b_k + k \cdot a_k)$ are the minimum and maximum degrees of $\sum_{i=1}^{m} a_i x_i \geq d$.

Proof. In the first part, we first prove that s is $\geq$, and secondly that it is $\leq$ the minimum degree of $\sum_{i=1}^{m} a_i x_i \geq d$.

We show that for any d' with $s < d' \leq d$, any assignment σ satisfying $\sum_{i=1}^{m} a_i x_i \geq d'$ also satisfies $\sum_{i=1}^{m} a_i x_i \geq d$. So let σ be such an assignment, and let k be the number of variables from $x_1, \ldots, x_l$ that are made true by σ. Then σ satisfies

$$\sum_{i=l+1}^{m} a_i x_i \geq d' - k \cdot a_1 \tag{9}$$

for any $s < d' \leq d$, and since $s_k + k \cdot a_1 \leq s$ by definition of s, the assignment σ satisfies (9) for any $s_k + k \cdot a_1 < d' \leq d$, or equivalently, σ satisfies $\sum_{i=l+1}^{m} a_i x_i \geq d'$ for any $s_k < d' \leq d - k \cdot a_1$. Thus by definition of s_k, σ satisfies $\sum_{i=l+1}^{m} a_i x_i \geq d - k \cdot a_1$, and so σ satisfies $\sum_{i=1}^{m} a_i x_i \geq d$.

We now show that there exists an assignment σ satisfying $\sum_{i=1}^{m} a_i x_i \geq s$ which does not satisfy $\sum_{i=1}^{m} a_i x_i \geq d$. Let k be the number for which $s_k + k \cdot a_1$ is maximal, i.e. $s = s_k + k \cdot a_1$. Then by definition of s_k, there exists an assignment σ' on $x_{l+1}, \ldots, x_m$ satisfying $\sum_{i=l+1}^{m} a_i x_i \geq s_k$ but not satisfying $\sum_{i=l+1}^{m} a_i x_i \geq d - k \cdot a_1$. If we extend σ' to an assignment σ making k of the variables $x_1, \ldots, x_l$ true, then σ satisfies $\sum_{i=1}^{m} a_i x_i \geq s$ but not $\sum_{i=1}^{m} a_i x_i \geq d$.

In the second part, we first prove that b is $\leq$, and secondly that it is $\geq$ the maximum degree of $\sum_{i=1}^{m} a_i x_i \geq d$.

Let σ be an assignment satisfying $\sum_{i=1}^{m} a_i x_i \geq d$. We show that σ satisfies $\sum_{i=1}^{m} a_i x_i \geq b$. Let k be the number of variables from $x_1, \ldots, x_l$ that are made true by σ. Then σ satisfies $\sum_{i=l+1}^{m} a_i x_i \geq d - k \cdot a_1$, and by the definition of b_k, σ satisfies $\sum_{i=l+1}^{m} a_i x_i \geq b_k$, and so since $b_k \geq b - k \cdot a_1$ by definition of b, the assignment σ satisfies $\sum_{i=l+1}^{m} a_i x_i \geq b - k \cdot a_1$, and so σ satisfies $\sum_{i=1}^{m} a_i x_i \geq b$.

We now show that there exists an assignment σ satisfying $a_1 x_1 + \ldots + a_m x_m \geq d$ which does not satisfy $\sum_{i=1}^{m} a_i x_i \geq b'$, for any $b' > b$. Let k be the number for which $b_k + k \cdot a_k$ is minimal, i.e. $b = b_k + k \cdot a_k$. Then by definition of b_k there exists an assignment σ' on $x_{l+1}, \ldots, x_m$ satisfying $\sum_{i=l+1}^{m} a_i x_i \geq b_k = b - k \cdot a_k$ but not satisfying $\sum_{i=l+1}^{m} a_i x_i \geq b' - k \cdot a_k$ for any $b' > b$. If we extend σ' to an assignment σ making k of the variables $x_1, \ldots, x_l$ true, then σ satisfies $\sum_{i=1}^{m} a_i x_i \geq b$ but not $\sum_{i=1}^{m} a_i x_i \geq b'$. □

6.3 Composing LPBs

Theorem 6.15 suggests a recursive algorithm where, at least conceptually, in the base case we have at most 2^m trivial problems of determining an LPB, trivial since the formula for which we must find an LPB is either *true* or *false*.

ϕ					
$S(\cdot, x_1, 0)$ $\equiv (x_2 \wedge x_3) \vee$ $(x_2 \wedge x_4) \vee$ $(x_3 \wedge x_4 \wedge x_5)$	$S(\cdot, x_2, 0) \equiv$ $(x_3 \wedge x_4 \wedge x_5)$	$S(\cdot, x_3, 0) \equiv f$ $S(\cdot, x_3, 1)$ $\equiv (x_4 \wedge x_5)$	$S(\cdot, x_{3..4}, 0) \equiv f$ $S(\cdot, x_{3..4}, 1) \equiv f$ $S(\cdot, x_{3..4}, 2) \equiv x_5$	$S(\cdot, x_{3..5}, 0) \equiv f$ $S(\cdot, x_{3..5}, 1) \equiv f$ $S(\cdot, x_{3..5}, 2) \equiv f$ $S(\cdot, x_{3..5}, 3) \equiv t$
	$S(\cdot, x_2, 1) \equiv$ $x_3 \vee x_4$	$S(\cdot, x_3, 0) \equiv x_4$ $S(\cdot, x_3, 1) \equiv t$	$S(\cdot, x_{3..4}, 0) \equiv f$ $S(\cdot, x_{3..4}, 1) \equiv t$ $S(\cdot, x_{3..4}, 2) \equiv t$	
$S(\cdot, x_1, 1)$ $\equiv x_2 \vee x_3$ $\vee x_4 \vee x_5$	$S(\cdot, x_2, 0) \equiv$ $x_3 \vee x_4 \vee x_5$ $S(\cdot, x_2, 1) \equiv t$	$S(\cdot, x_3, 0) \equiv$ $x_4 \vee x_5$ $S(\cdot, x_3, 1) \equiv t$ $S(\cdot, x_3, 2) \equiv t$	$S(\cdot, x_{3..4}, 0) \equiv x_5$ $S(\cdot, x_{3..4}, 1) \equiv t$ $S(\cdot, x_{3..4}, 2) \equiv t$ $S(\cdot, x_{3..4}, 3) \equiv t$	$S(\cdot, x_{3..5}, 0) \equiv f$ $S(\cdot, x_{3..5}, 1) \equiv t$ $S(\cdot, x_{3..5}, 2) \equiv t$ $S(\cdot, x_{3..5}, 3) \equiv t$ $S(\cdot, x_{3..5}, 4) \equiv t$

Table 2. The recursive problems of Ex. 6.18

Example 6.18. Consider Ex. 6.6. To find an LPB for ϕ, we must find LPBs for $S(\phi, \{x_1\}, 0)$ and $S(\phi, \{x_1\}, 1)$. To find an LPB for $S(\phi, \{x_1\}, 0)$, we must find LPBs for $S(S(\phi, \{x_1\}, 0), \{x_2\}, 0)$ and $S(S(\phi, \{x_1\}, 0), \{x_2\}, 1)$, and so forth. Table 2 gives all the formulae for which we must find LPBs. For a concise notation we use some abbreviations which we explain using $S(\cdot, x_{3..5}, 0) \equiv f$ in the top-right corner: it stands for $S((x_3 \wedge x_4 \wedge x_5), \{x_3, x_4, x_5\}, 0) \equiv \mathit{false}$, i.e. the '$\cdot$' stands for the nearest *non-shaded* formula to the left, here $(x_3 \wedge x_4 \wedge x_5)$. Note how we arranged the subproblem formulae in the table: e.g. $(x_3 \wedge x_4 \wedge x_5)$ has *three* symmetric variables that are split away to obtain the subproblems to be solved, so these subproblems are located *three* columns to the right of $(x_3 \wedge x_4 \wedge x_5)$. The two shaded boxes in between contain the subproblems obtained by splitting away only $\{x_3\}$, $\{x_3, x_4\}$, resp.

The algorithm we propose is not a purely recursive one, since the subproblems at each level must be solved in parallel. Explained using the example, we first find LPBs for the formulae in the rightmost column, which have 0 variables and hence we must determine 0 coefficients. Next to the left, we have formulae that contain (at most) x_5, and we determine LPBs representing these, where we use the same a_5 for all formulae! Then we determine a_4, and so forth.

Taking $(x_3 \wedge x_4 \wedge x_5)$ in Table 2 as an example, Thm. 6.15 suggests that a_3, a_4, a_5 should be equal (x_3, x_4, x_5 are symmetric) and determined in one go. However, since a_3, a_4, a_5 also have to represent other subproblem formulae where x_3, x_4, x_5 are not necessarily symmetric, one cannot determine a_3, a_4, a_5 in one go, but rather first a_5, then a_4, then a_3. Therefore, it is necessary to define and interpret formulae obtained by splitting away fewer variables than one could split away, in the sense of Thm. 6.15. These are the shaded formulae.

We call the formulae in column $l+1$ the *l-successors*. Shaded formulae are called *auxiliary*, the others are called *main*. Formulae that have no further formulae to the right are called *final*. The following definition formalises these notions.

Definition 6.19. Let ϕ be a DNF in m variables. Then ϕ is the 0-**successor** of ϕ. Furthermore, ϕ is a **main** successor of ϕ. Moreover, if ϕ' is a main n-successor of ϕ, and l is maximal so that $x_{n+1}, \ldots, x_{n+l}$ are symmetric in ϕ', then for all l', k with $1 \leq l' \leq l$ and $0 \leq k \leq l'$, we say that $S(\phi', \{x_{n+1}, \ldots, x_{n+l'}\}, k)$ is an $(n+l')$-successor of ϕ. The $(n+l)$-successors are called **main**, and for $l' < l$, the $(n+l')$-successors are called **auxiliary**. If $x_{n+1}, \ldots, x_{n+l}$ are the only variables of ϕ', then we call the $(n+l)$-successors **final**.

Note in particular $x_3 \vee x_4$ in column 3 in Table 2. It does not contain x_5, and so we obtain final 4-successors in the last-*but-one* column. Clearly, a final successor of ϕ is either *true* or *false*.

Proposition 6.20. Assume ϕ, ϕ', n, l as in Def. 6.19. For $0 < l' < l$ and $0 \leq k \leq l'$, we have

$$S(S(\phi', \{x_{n+1}, \ldots, x_{n+l'}\}, k), \{x_{n+l'+1}\}, 0) \equiv S(\phi', \{x_{n+1}, \ldots, x_{n+l'+1}\}, k)$$
$$S(S(\phi', \{x_{n+1}, \ldots, x_{n+l'}\}, k), \{x_{n+l'+1}\}, 1) \equiv S(\phi', \{x_{n+1}, \ldots, x_{n+l'+1}\}, k+1)$$

For example, consider $S((x_3 \wedge x_4 \wedge x_5), \{x_3\}, 1) \equiv (x_4 \wedge x_5)$ in Table 2. We have $S((x_4 \wedge x_5), \{x_4\}, 0) \equiv S((x_3 \wedge x_4 \wedge x_5), \{x_3, x_4\}, 1)$ and $S((x_4 \wedge x_5), \{x_4\}, 1) \equiv S((x_3 \wedge x_4 \wedge x_5), \{x_3, x_4\}, 2)$. More generally, each non-final successor is associated with two formulae in the column right next to it, one slightly up and one slightly down, obtained by splitting away the variable with the smallest index. This is not surprising per se and corresponds to a naïve approach where we ignore symmetries and always only split away one variable at a time (for applying Thm. 6.15), thereby constructing 2^m formulae in the rightmost column. The point of Prop. 6.20 is that we can usually construct fewer formulae since

$$S(S(\phi, \{x_{n+1}, \ldots, x_{n+l'}\}, k), \{x_{n+l'+1}\}, 1)$$

and

$$S(S(\phi, \{x_{n+1}, \ldots, x_{n+l'}\}, k+1), \{x_{n+l'+1}\}, 0)$$

coincide. In Table 2, we have 12 final formulae rather than $2^5 = 32$.

The following theorem states if and how one can find the next coefficient and degrees for representing all k-successors of ϕ provided one has coefficients and degrees for representing all $(k+1)$-successors.

Theorem 6.21. Assume ϕ as in Thm. 6.15 and some k with $0 \leq k \leq m-1$, and let Φ_k be the set of k-successors of ϕ. For every non-final $\phi' \in \Phi_k$, suppose we have two LPBs $\sum_{i=k+2}^{m} a_i x_i \geq (s_{\phi'0}, b_{\phi'0}]$ and $\sum_{i=k+2}^{m} a_i x_i \geq (s_{\phi'1}, b_{\phi'1}]$, representing $S(\phi', \{x_{k+1}\}, 0)$ and $S(\phi', \{x_{k+1}\}, 1)$, respectively.

If it is possible to choose a_{k+1} such that

$$\max_{\phi' \in \Phi_k} (s_{\phi'0} - b_{\phi'1}) < a_{k+1} < \min_{\phi' \in \Phi_k} (b_{\phi'0} - s_{\phi'1}), \tag{10}$$

then for all $\phi' \in \Phi_k$, the LPB $\sum_{i=k+1}^{m} a_i x_i \geq (s_{\phi'}, b_{\phi'}]$ represents ϕ', where

$$s_{\phi'} = \max\{s_{\phi'0}, s_{\phi'1} + a_{k+1}\},\ b'_{\phi} = \min\{b_{\phi'0}, b_{\phi'1} + a_{k+1}\} \text{ for non-final } \phi' \tag{11}$$
$$s_{\phi'} = -\infty,\ b_{\phi'} = 0 \text{ for } \phi' \equiv \mathit{true},\ s_{\phi'} = \textstyle\sum_{i=k+1}^{m} a_i,\ b_{\phi'} = \infty \text{ for } \phi' \equiv \mathit{false} \tag{12}$$

If $\max_{\phi' \in \Phi_k}(s_{\phi'0} - b_{\phi'1}) \geq \min_{\phi' \in \Phi_k}(b_{\phi'0} - s_{\phi'1})$, then no a_{k+1}, $s_{\phi'}$, $b_{\phi'}$ exist such that $\sum_{i=k+1}^{m} a_i x_i \geq (s_{\phi'}, b_{\phi'}]$ represents ϕ' for all $\phi' \in \Phi_k$.

Proof. First consider an arbitrary non-final $\phi' \in \Phi_k$. By Thm. 6.15, if we choose any $d_{\phi'0}$, $d_{\phi'1}$ such that $s_{\phi'0} < d_{\phi'0} \leq b_{\phi'0}$ and $s_{\phi'1} < d_{\phi'1} \leq b_{\phi'1}$, then we can set $a_{k+1} := d_{\phi'0} - d_{\phi'1}$, and $I_{\phi'} = \sum_{i=k+1}^{m} a_i x_i \geq d_{\phi'0}$ represents ϕ'. This however means that $s_{\phi'0} - b_{\phi'1} < a_{k+1} < b_{\phi'0} - s_{\phi'1}$, i.e. a_{k+1} can obtain any desired value in this range by choosing $d_{\phi'0}$, $d_{\phi'1}$ accordingly.

Clearly, an a_{k+1} fulfilling $s_{\phi'0} - b_{\phi'1} < a_{k+1} < b_{\phi'0} - s_{\phi'1}$ for *all* $\phi' \in \Phi_k$ exists iff (10) holds.

Now consider a final $\phi' \in \Phi_k$. It can only be *true* or *false*, and the degrees in (12) are straightforward.

It remains to show that if (10) holds, the minimum and maximum of the new LPBs are as stated in (11). But this follows from Lemma 6.17. □

The m-successors of ϕ are represented by LPBs with an empty sum as l.h.s.: $\sum_{i=m+1}^{m} a_i x_i \geq (0, \infty]$ for *false*, $\sum_{i=m+1}^{m} a_i x_i \geq (-\infty, 0]$ for *true*. Then we proceed using Thm. 6.21, in each step choosing an arbitrary a_{k+1} fulfilling (10).

$4x_1+3x_2+$ $2x_3+2x_4+$ $x_5 \geq \ldots$	$3x_2+$ $2x_3+2x_4+$ $x_5 \geq \ldots$	$2x_3+2x_4+$ $x_5 \geq \ldots$	$2x_4+$ $x_5 \geq \ldots$	$x_5 \geq \ldots$	$\sum_{i=6}^{5} a_i x_i$ $\geq \ldots$
				$(1,\infty]$	$(0,\infty]$
			$(3,\infty]$	$(1,\infty]$	$(0,\infty]$
		$(4,5]$	$(2,3]$	$(0,1]$	$(0,\infty]$
	$(4,5]$				$(-\infty,0]$
				$(1,\infty]$	
			$(1,2]$	$(-\infty,0]$	
$(4,5]$		$(1,2]$	$(-\infty,0]$	$(-\infty,0]$	
				$(0,1]$	$(0,\infty]$
			$(0,1]$	$(-\infty,0]$	$(-\infty,0]$
	$(0,1]$	$(0,1]$	$(-\infty,0]$	$(-\infty,0]$	$(-\infty,0]$
		$(-\infty,0]$	$(-\infty,0]$	$(-\infty,0]$	$(-\infty,0]$
					$(-\infty,0]$

Table 3. LPBs for Ex. 6.18

Example 6.22. Consider again Ex. 6.18. Table 3 is arranged in strict correspondence to Table 2 and shows LPBs for all successors of Φ. In the top line we give the l.h.s. of the LPBs, which is of course the same for each LPB in a column. In the main table, we list the minimum and maximum degree of each formula.

In the first step, applying (10), we have to choose a_5 so that

$$\max\{0-\infty, 0-\infty, 0-0, \quad 0-0, -\infty-0, -\infty-0, -\infty-0\} < a_5 < \\ \min\{\infty-0, \infty-0, \infty--\infty, \quad \infty--\infty, 0--\infty, 0--\infty, 0--\infty\}.$$

Choosing $a_5 = 1$ will do. The minimum and maximum degrees in column 5 are computed using (11); e.g. the topmost $(1,\infty]$ is $(\max\{0, 0+1\}, \min\{\infty, \infty+1\}]$.

In the next step, we have to choose a_4 so that

$$\max\{1-\infty, 1-1, \quad 1-0, -\infty-0, \quad 0-0, -\infty-0, -\infty-0\} < a_4 < \\ \min\{\infty-1, \infty-0, \quad \infty--\infty, 0--\infty, \quad 1--\infty, 0--\infty, 0--\infty\}.$$

Choosing $a_4 = 2$ will do. Note that the bound $1-0 < a_4$ comes from the middle box of the fifth column and thus ultimately from $x_3 \vee x_4$. Our algorithm enforces that $a_4 > a_5$, which must hold for an LPB representing $x_3 \vee x_4$.

In the next step, a_3 can also be chosen to be any number > 1 so we choose 2 again.[3] In the next step, $2 < a_2 < 4$ must hold so we choose $a_2 = 3$. Finally, $3 < a_1 < 5$ must hold so we choose $a_1 = 4$. We obtain the LPB $4x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq (4,5]$ given in Ex. 6.6.

We have seen in the example how our algorithm works. However, since the choice of a_{k+1} is not unique in general, one might be worried that a bad choice of a_{k+1} might later lead to non-applicability of Thm. 6.21.

The following lemma says that we do not have to worry about this. It states that every vector of coefficients $(a_1, \ldots, a_m)$ that is suitable for representing ϕ corresponds, for every $k \in [1..m-1]$, to a vector $(a_{k+1}, \ldots, a_m)$ that is suitable for simultaneously representing all k-successors of ϕ, and more importantly, vice versa: every vector $(a_{k+1}, \ldots, a_m)$ that is suitable for simultaneously representing all k-successors of ϕ corresponds to a vector $(a_1, \ldots, a_m)$ that is suitable for representing ϕ. Since in the intermediate steps of the algorithm, we have a vector $(a_{k+1}, \ldots, a_m)$ suitable for representing all k-successors of ϕ, we know that this vector can be completed.

[3] The algorithm could be improved by determining a_3 and a_4 in one go since x_3, x_4 are symmetric in Φ. We refrain from spelling this out to avoid further complication.

Lemma 6.23. For any DNF ϕ' of m variables, let

$$\mathcal{A}_{\phi'} := \{(a_1, \ldots a_m) \mid \exists d \text{ such that } \sum_{i=1}^{m} a_i x_i \geq d \text{ represents } \phi'\}.$$

Moreover, for any set $\mathcal{A}$ of m-tuples and any $k \in [1..m-1]$, define $chop(\mathcal{A}, k) = \{(a_{k+1}, \ldots a_m) \mid (a_1, \ldots a_m) \in \mathcal{A}\}$, and let Φ_k be the set of k-successors of ϕ, where ϕ is a DNF that can be represented as an LPB. Then for all $k \in [1..m-1]$

$$chop(\mathcal{A}_\phi, k) = \bigcap_{\phi' \in \Phi_k} \mathcal{A}_{\phi'}$$

Proof. Since *true* and *false* can be represented using arbitrary coefficients, we disregard them here.

We show the statement by induction on k. For $k = 1$, the statement follows from Thm. 6.15. Now suppose the statement holds for some k. Again by Thm. 6.15, for every $\phi' \in \Phi_k$ we have

$$chop(\mathcal{A}_{\phi'}, 1) = \mathcal{A}_{S(\phi', \{x_k\}, 0)} \cap \mathcal{A}_{S(\phi', \{x_k\}, 1)} \tag{13}$$

But then (using the induction hypothesis, (13) and some straightforward transformations)

$$chop(\mathcal{A}_\phi, k+1) = chop(chop(\mathcal{A}_\phi, k), 1) \overset{\text{hyp.}}{=} chop(\textstyle\bigcap_{\phi' \in \Phi_k} \mathcal{A}_{\phi'}, 1) =$$
$$\textstyle\bigcap_{\phi' \in \Phi_k} chop(\mathcal{A}_{\phi'}, 1) \overset{(13)}{=} \bigcap_{\phi' \in \Phi_k} (\mathcal{A}_{S(\phi', \{x_k\}, 0)} \cap \mathcal{A}_{S(\phi', \{x_k\}, 1)}) = \bigcap_{\phi' \in \Phi_{k+1}} \mathcal{A}_{\phi'}.$$

□

However, there are some pragmatic choices. As stated in the example, to obtain an LPB with small coefficients, one might always choose a_{k+1} as the smallest possible integer value. It might also occur, though not in the above example, that a_{k+1} is forced to be between neighbouring integers, in which case it cannot be an integer itself. In this case, one can multiply all LPBs of the current system by 2 (this obviously preserves the meaning of the LPBs) before proceeding so that a_{k+1} can be chosen to be an integer.

From the construction of the successors (see Table 2) it follows that all formulae in a column together have size less than all formulae in the column to the left of it, so that the entire table has size less than $|\phi| \cdot (m+1)$. One can thus show that the complexity of the algorithm is polynomial in the size of ϕ, while the size of ϕ itself can be exponential in m. In fact, this is the most interesting case, because in this case an LPB representation may yield an exponential saving.

A thorougher analysis of the complexity of the algorithm will be due once it is embedded into a more complete algorithm which converts an *arbitrary* DNF (or CNF) into a *set* of LPBs. It is clear that such an algorithm would first have to partition the DNF according to the polarity of each variable, which is straightforward. The next step would be to partition a DNF where each variable occurs in only one polarity into sub-DNFs each of which can be represented by a single LPB. This step is nontrivial and the main topic for future work.

7 Conclusion

Linear pseudo-Boolean constraints have attracted interest because they can often be used to represent Boolean functions more compactly than CNFs or DNFs, and because techniques applied in CNF-based propositional satisfiability solving can be generalised to LPBs, which can be more efficient than solving a problem based on a CNF representation [1, 5–8]. This generalisation is essentially an

application of a technique known from OR to the field of AI, or more specifically, propositional logic [6].

It is assumed here that the problems, as they arise in an application domain, have a natural encoding as LPB, and that the CNF encoding would be larger. Our work was initially motivated by three main issues, which were not addressed in previous works.

Firstly, several authors have emphasised that an LPB representation of a function can be exponentially more compact than a CNF representation [1, 5–8]. However, it is shown in fact that *cardinality constraints* can be exponentially more compact than a CNF. Thus no evidence is given that the additional expressive power that LPBs have compared to cardinality constraints is useful.

Secondly, it has been noted *en passant* that a single LPB can be used to express an implication [7], but it remains unclear what kind of implications can or cannot be expressed. In fact, our Lemma 6.10 shows that the power of an LPB for expressing implications is very limited.

Most importantly, since an LPB representation can be more compact than a CNF representation, one might use LPB encodings even in cases where they do not arise naturally from the application domain. That is, one might convert a CNF to a (small) set of LPBs and then apply LPB solving [1, 5–8]. Here we see the potential for practical application of our work.

As a further comment on the first point, Barth [3] mentions that LPBs arise in AI applications [4]. Since he used a solver that could only deal with cardinality constraints, he proposes a transformation of LPBs to cardinality constraints. Note that this transformation goes in the opposite direction compared to ours, from a more concise to a less concise representation.

In [8], LPBs are used for bounded model checking. At one point, an LPB of the form $x_1 + x_2 + 2\bar{y} \geq 2$ (which is not a cardinality constraint) is used.

Apart from that, the above works say little about where the problem instances come from, and if anything, then these are in fact cardinality constraints rather than LPBs. In [1], problems Min-Cover, Max-SAT, and MAX-ONEs are mentioned. E.g. Max-SAT is the problem of finding a variable assignment that maximises the number of satisfied clauses of an unsatisfiable SAT instance. Furthermore, applications from design automation [5], the pigeonhole problem [6], and gate level netlists [7] are mentioned as applications.

However, we are not suggesting that our approach of converting a CNF or DNF to an LPB is the only way to go. If for a problem domain, there is a natural direct encoding as an LPB not going via CNF or DNF, then this should definitely be considered.

Hooker has proposed an algorithm for generating the strongest 0-1 ILP constraints, within a candidate set T, that are implied by a set S of 0-1 ILP constraints [9]. Letting T be the set of all LPBs, the algorithm can be used to transform a CNF to an LPB. However, the algorithm is practical only for certain restrictions of T. In the general case, which we need here, it is unclear if the algorithm is any better than enumerating and checking all LPBs. This is however an interesting topic for future work.

Complementary to this paper, we have also obtained results about Boolean functions that can definitely *not* be represented compactly as a set of LPBs. More precisely, there is a class of *monotone* functions for which the DNF representation is exponential and the LPB representation saves nothing [17].

We summarise our contributions to the understanding of LPBs. We demonstrated that the functions expressible as one LPB constraint are a strict subset of the monotone functions. We gave some results about the cardinality of various classes of Boolean functions, and showed that the blowup when encoding an LPB as CNF or DNF is not worse than when encoding a cardinality constraint. We showed that the problems of encoding a DNF or a CNF as LPB have a very sim-

ple duality. Finally and most importantly, we gave an algorithm for computing an LPB representation for a DNF whenever this is possible.

Acknowledgements This work was supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB TR14/AVACS). I would like to thank Markus Behle, Martin Fränzle, Marc Herbstritt, Christian Herde, Felix Klaedtke, Bernhard Nebel, and the other AVACS colleagues, for useful discussions.

References

1. Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Generic ILP versus specialized 0-1 ILP: an update. In Lawrence T. Pileggi and Andreas Kuehlmann, editors, *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pages 450–457. ACM, 2002.
2. Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. A translation of pseudo Boolean constraints to SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:191–200, 2006.
3. Peter Barth. Linear 0-1 inequalities and extended clauses. In Andrei Voronkov, editor, *Proceedings of the 4th International Conference on Logic Programming and Automated Reasoning*, volume 698 of *LNCS*, pages 40–51. Springer-Verlag, 1993.
4. Peter Barth and Alexander Bockmayr. Solving 0-1 problems in CLP(PB). In *Proceedings of the 9th Conference on Artificial Intelligence for Applications*. IEEE, 1993.
5. Donald Chai and Andreas Kuehlmann. A fast pseudo-Boolean constraint solver. In *Proceedings of the 40th Design Automation Conference*, pages 830–835. ACM, 2003.
6. Heidi E. Dixon and Matthew L. Ginsberg. Combining satisfiability techniques from AI and OR. *The Knowledge Engineering Review*, 15:31–45, 2000.
7. Martin Fränzle and Christian Herde. Efficient SAT engines for concise logics: Accelerating proof search for zero-one linear constraint systems. In Moshe Y. Vardi and Andrei Voronkov, editors, *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2850 of *LNCS*, pages 302–316. Springer-Verlag, 2003.
8. Martin Fränzle and Christian Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 2006. Online version, http://dx.doi.org/10.1007/s10703-006-0031-0. Print version in press.
9. John N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6(1-3):271–286, 1992.
10. John N. Hooker and Hong Yan. Tight representations of logical constraints as cardinality rules. *Mathematical Programming*, 85(2):363–377, 1999.
11. D. Kleitman and G. Markowsky. On Dedekind's problem: the number of isotone Boolean functions. II. *Transactions of the American Mathematical Society*, 213:373–390, 1975.
12. João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.
13. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, pages 530–535. ACM, 2001.
14. Raúl Rojas. *Neural Networks. A Systematic Introduction*. Springer-Verlag, 1996.
15. Ching Lai Sheng. *Threshold Logic*. Academic Press, 1969.
16. Neil J. A. Sloane. On-line encyclopedia of integer sequences. http://www.research.att.com/~njas/sequences/Seis.html.
17. Jan-Georg Smaus. Representing Boolean functions as linear pseudo-Boolean constraints. In Youssef Hamadi, editor, *Proceedings of the CP 2006 Workshop on the Integration of SAT and CP techniques*, 2006.

18. Jan-Georg Smaus. On Boolean functions encodable as a single linear pseudo-Boolean constraint. In Pascal Van Hentenryck and Laurence Wolsey, editors, *Proceedings of the 4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, LNCS. Springer-Verlag, 2007.
19. Vetle Ingvald Torvik and E. Trintaphyllou. Inference of monotone Boolean functions. In Chris A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 472–480. Kluwer Academic Publishers, 2001.
20. Ingo Wegener. *The Complexity of Boolean Functions*. Wiley & Sons, http://eccc.uni-trier.de/eccc-local/ECCC-Books/wegener_book_readme.html, 1987.
21. Hantao Zhang. SATO: An efficient propositional prover. In William McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *LNCS*, pages 272–275. Springer-Verlag, 1997.