*Theoretical Computer Science*, to appear.

# Decision Lists and Related Boolean Functions[*],[†]

Thomas Eiter[‡] Toshihide Ibaraki[§] Kazuhisa Makino[¶]

**Abstract**

We consider Boolean functions represented by decision lists, and study their relationships to other classes of Boolean functions. It turns out that the elementary class of 1-decision lists has interesting relationships to independently defined classes such as disguised Horn functions, read-once functions, nested differences of concepts, threshold functions, and 2-monotonic functions. In particular, 1-decision lists coincide with fragments of the mentioned classes. We further investigate the recognition problem for this class, as well as the extension problem in the context of partially defined Boolean functions (pdBfs). We show that finding an extension of a given pdBf in the class of 1-decision lists is possible in linear time. This improves on previous results. Moreover, we present an algorithm for enumerating all such extensions with polynomial delay.

**Keywords**: Decision lists, Boolean functions, teaching sequence, extension problem, polynomial delay enumeration

## 1 Introduction

Decision lists have been proposed in [34] as a specification of Boolean functions which amounts to a simple strategy for evaluating a Boolean function on a given assignment. This approach has been become popular in learning theory, since bounded decision lists naturally generalize other important classes of Boolean functions. For example, $k$-bounded decision lists generalize the classes whose members have a CNF or DNF expression where each clause or term, respectively, has at most $k$ literals, and, as a consequence, also those classes whose members have a DNF or CNF containing at most $k$ terms or clauses, respectively. Another class covered by decision lists is the one of decision trees [33].

Informally, a decision list can be written as a cascaded conditional statement of the form:

```
**if** $t_1(v)$ **then** $b_1$
**elseif** $t_2(v)$ **then** $b_2$
    ⋮
**elseif** $t_{d-1}(v)$ **then** $b_{d-1}$
**else** $b_d$
```

[*]This paper is an augmented version of the preliminary abstract "On Disguised Double Horn Functions and Extensions", which appeared in: *Proc. 15th Symposium on Theoretical Aspects of Computing* (*STACS '98*), LNCS 1373, pp. 50–60, 1998.

[†]The authors gratefully acknowledge the partial support of the Scientific Grant in Aid by the Ministry of Education, Science and Culture of Japan. Part of this research was conducted while the first author visited Kyoto University in 1995 and 1998, by the support of the Scientific Grant in Aid by the Ministry of Education, Science and Culture of Japan (Grant 06044112).

[‡]Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria. (eiter@kr.tuwien.ac.at)

[§]Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606, Japan. (ibaraki@kuamp.kyoto-u.ac.jp)

[¶]Department of Systems and Human Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560, Japan (makino@sys.es.osaka-u.ac.jp)

where each $t_i(v)$ means the evaluation of a term $t_i$, i.e., a conjunction of Boolean literals, on an assignment $v$ to the $x_1, \dots, x_n$, and each $b_i$ is either 0 (false) or 1 (true).

The important result established in [34] is that $k$-decision lists, i.e., decision lists where each term $t_i$ has at most $k$ literals and $k$ is a constant, are probably approximately correct (PAC) learnable in Valiant's model [39]. This has largely extended the classes of Boolean functions which are known to be learnable. In the sequel, decision lists have been studied extensively in the learning field, see e.g. [19, 8, 17, 9].

However, while it is known that decision lists generalize some classes of Boolean functions [34], their relationships to other classes such as Horn functions, read-once functions, threshold functions, or 2-monotonic functions, which are widely used in the literature, were only partially known (cf. [5, 3]). It thus is interesting to know about such relationships, in particular whether fragments of such classes correspond to decision lists and how such fragments can be alternatively characterized. This issue is intriguing, since decision lists are operationally defined, while other classes such as Horn functions or read-once functions are defined on a semantical (in terms of models) or syntactical (in terms of formulas) basis, respectively.

In this paper, we shed light on this issue and study the relationship of decision lists to the classes mentioned above. We focus on the elementary class of 1-decision lists ($\mathcal{C}_{1\text{-}DL}$), which has received a lot of attention and was the subject of a number of investigations, eg. [34, 29, 8, 9]. It turns out that this class relates in an interesting way to several other classes of Boolean functions. In particular, it coincides with independently defined semantical and syntactical such classes, as well as with the intersections of other well-known classes of Boolean functions. We find the following characterizations of $\mathcal{C}_{1\text{-}DL}$. It coincides with

- $\mathcal{C}_{DH}^R$, the renaming-closure of the class of functions $f$ such that both $f$ and its complement $\overline{f}$ are Horn [12] (also called disguised "double" Horn functions);
- $\mathcal{C}_{ND}$, the class of nested differences of concepts [21], where each concept is described by a single term;
- $\mathcal{C}_{2M} \cap \mathcal{C}_{R\text{-}1}$, the intersection of the classes of 2-monotonic functions [32] and read-once functions, i.e., functions definable by a formula in which each variable occurs at most once [18, 25, 39, 37];
- $\mathcal{C}_{TH} \cap \mathcal{C}_{R\text{-}1}$, the intersection of threshold functions (also called linearly separable functions) [32] and read-once functions; and
- $\mathcal{C}_{LR\text{-}1}$, the class of linear read-once functions [12], i.e., functions represented by a read-once formula such that each binary connective involves at least one literal.

Observe that the inclusion $\mathcal{C}_{1\text{-}DL} \subseteq \mathcal{C}_{TH} \cap \mathcal{C}_{R\text{-}1}$ follows from the result that $\mathcal{C}_{1\text{-}DL} \subseteq \mathcal{C}_{TH}$ [5, 3] and the fact that $\mathcal{C}_{1\text{-}DL} \subseteq \mathcal{C}_{R\text{-}1}$; however, the converse was not known.

The above results give us new insights into the relationships between these classes of functions. Moreover, they provide us with a semantical and syntactical characterization of 1-decision lists in terms of (renamed) Horn functions and read-once formulas. On the other hand, we obtain characterizations of the intersections of well-known classes of Boolean functions in terms of operationally, semantically, and syntactically defined classes of Boolean functions.

As we show, a natural generalization of the results from 1-decision lists to $k$-bounded decision lists fails in almost all cases. The single exception is the coincidence with nested differences of concepts, which holds for an appropriate base class generalizing terms. Thus, our results unveil characteristic properties of 1-decision lists and, vices versa, of the intersections of classes of Boolean functions to which they coincide.

Furthermore, we study computational problems on 1-decision lists. We consider recognition from a formula (also called *membership problem* [20] and *representation problem* [4, 1]) and problems in the context of partially defined Boolean functions.

A partially defined Boolean function (pdBf) can be viewed as a pair $(T, F)$ of sets $T$ and $F$ of true and false vectors $v \in \{0,1\}^n$, respectively, where $T \cap F = \emptyset$. It naturally generalizes a Boolean function, by allowing that the range function values on some input vectors are unknown. This concept has many applications, e.g., in circuit design, for representation of cause-effect relationships [7], or in learning, to mention a few. A principal issue on pdBfs is the following: Given a pdBf $(T, F)$, determine whether some $f$ in a particular class of Boolean functions $\mathcal{C}$ exists such that $T \subseteq T(f)$ and $F \subseteq F(f)$, where $T(f)$ and $F(f)$ denote the sets of true and false vectors of $f$, respectively. Any such $f$ is called an *extension* of $(T, F)$ in $\mathcal{C}$, and finding such an $f$ is known as the *extension problem* [6, 30]. Since in general, a pdBf may have multiple extensions, it is sometimes desired to know all extensions, or to compute an extension of a certain quality (e.g., one described by a shortest formula, or having a smallest set $T(f)$ ).

The extension problem is closely related to problems in machine learning. A typical problem there is the following ([4]). Suppose there are $n$ Boolean valued attributes; then, find a hypothesis in terms of a Boolean function $f$ in a class of Boolean functions $\mathcal{C}$, which is consistent with the actual correlation of the attributes after seeing a sample of positive and negative examples, where it is known that the actual correlation is a function $g$ in $\mathcal{C}$. In our terms, a learning algorithm produces an extension of a pdBf. However, there is a subtle difference between the general extension problem and the learning problem: in the latter problem, an extension is a priori known to exist, while in the former, this is unknown. A learning algorithm might take advantage of this knowledge and find an extension faster. The extension problem itself is known as the *consistency problem* [4, 1]; it corresponds to learning from a sample which is possibly spoiled with inconsistent examples.

In this context, it is also interesting to know whether the pdBf given by a sample uniquely defines a Boolean function in $\mathcal{C}$; if the learner recognizes this fact, she/he has identified the function $g$ to be learned. This is related to the question whether a pdBf has a unique extension, which is important in the context of teaching [35, 23, 36, 16]. There, to facilitate quicker learning, the sample is provided by a teacher rather than randomly drawn, such that identification of the function $g$ is possible from it (see e.g. [5, 16] for details). Any sample which allows to identify a function in $\mathcal{C}$ is called a *teaching sequence* (or *specifying sample* [5]). Thus, the issue of whether a given set of labeled examples is a teaching sequence amounts to the issue of whether $S$, seen as a pdBf, has a unique extension in $\mathcal{C}$. A slight variant is that the sample is known to be consistent with some function $g$ in $\mathcal{C}$. In this case, the problem amounts to the unique extension problem knowing that some extension exists; in general, this additional knowledge could be utilized for faster learning.

Alternative teaching models have been considered, in which the sample given by the teacher does not precisely describe a single function [17]. However, identification of the target function is still possible, since the teacher knows how the learner proceeds, and vice versa, the learner knows how the teacher generates his sample, called a *teaching set* in [17]. To prevent "collusion" between the two sides (the target could be simply encoded in the sample), an adversary is allowed to spoil the teaching set by adding further examples.

Our main results on the above issues can be summarized as follows:

- Recognizing 1-decision lists from a formula is tractable for a wide class of formulas, including Horn formulas, 2-CNF and 2-DNF, while unsurprisingly intractable in the general case.

- We point out that the extension problem for $\mathcal{C}_{1\text{-}DL}$ is solvable in linear time. This improves on the previous result that the extension problem for $\mathcal{C}_{1\text{-}DL}$ is solvable in polynomial time [34]. As a consequence, a hypothesis consistent with a target function $g$ in $\mathcal{C}_{1\text{-}DL}$ on the sample can be generated in linear time. In particular, learning from a (possibly spoiled) teaching sequence is possible in linear time. We obtain as a further result an improvement to [17], where it is shown that learning a function $g$ in $\mathcal{C}_{1\text{-}DL}$ from a particular teaching set is possible in $O(m^2n)$ time, where $m$ is the length of a shortest 1-decision list for $g$, $n$ is the number of attributes, and the input size is assumed to be $O(mn)$. Our algorithm can replace the learning algorithm in [17], and finds the target in $O(nm)$ time, i.e., in linear time. We mention that [8] presents the result, somewhat related to [17], that 1-decision lists with $k$ alternations (i.e., changes of the output value) are PAC learnable, where the algorithm runs in $O(n^2m)$ time.

- We present an algorithm which enumerates all extensions of a pdBf in $\mathcal{C}_{1\text{-}DL}$ with polynomial delay. As a corollary, the problems of deciding whether a given set of any examples is a teaching sequence and whether a consistent sample is a teaching sequence are both solvable in polynomial time. Moreover, a small number of different hypotheses (in fact, even up to polynomially many) for the target function can be produced within polynomial time.

The rest of this paper is organized as follows. The next section provides some preliminaries and fixes notation. In Section 3, we study the relationships of 1-decision lists to other classes of functions. In Section 4, we address the recognition problem from formulas, and in Section 5, we study the extension problem. Section 6 concludes the paper.

## 2 Preliminaries

We use $x_1, x_2, \ldots, x_n$ to denote Boolean variables and letters $u, v, w$ to denote vectors in $\{0,1\}^n$. The $i$-th component of a vector $v$ is denoted by $v_i$. Formulas are built over the variables using the connectives $\wedge, \vee$, and $\neg$. A literal is a variable $x_i$ or its negation $\overline{x}_i$. For any literal $\ell$, we denote by $\overline{\ell}$ its opposite. A *term* $t$ is a conjunction $\bigwedge_{i \in P(t)} x_i \wedge \bigwedge_{i \in N(t)} \overline{x}_i$ of Boolean literals such that $P(t) \cap N(t) = \emptyset$, and a *clause* $c$ is defined dually (change $\wedge$ to $\vee$); $t$ (resp., $c$) is *Horn,* if $|N(t)| \leq 1$ (resp., $|P(c)| \leq 1$). We use $\top$ and $\bot$ to denote the empty term (truth) and the empty clause (falsity), respectively. A *disjunctive normal form* (DNF) $\varphi = \bigvee_i t_i$ is *Horn,* if all $t_i$ are Horn. Similarly, a *conjunctive normal form* (CNF) $\psi = \bigwedge_i c_i$ is Horn, if all $c_i$ are Horn.

E.g., the term $t = x_1\overline{x}_2x_3x_4$ has $P(t) = \{1,3,4\}$ and $N(t) = \{2\}$, and is Horn, while the clause $c = x_1 \vee x_2 \vee \overline{x}_4$ has $P(c) = \{x_1, x_2\}$ and $N(c) = \{x_4\}$, and thus it is not Horn.

A *partially defined Boolean function* (pdBf) is a mapping $g : T \cup F \to \{0,1\}$ defined by $g(v) = 1$ if $v \in T$ and $g(v) = 0$ if $v \in F$, where $T \subseteq \{0,1\}^n$ denotes a set of true vectors (or positive examples), $F \subseteq \{0,1\}^n$ denotes a set of false vectors (or negative examples), and $T \cap F = \emptyset$. For simplicity, we denote a pdBf by $(T, F)$. It can be seen as a representation for all (total) Boolean functions (Bfs) $f : \{0,1\}^n \to \{0,1\}$ such that $T \subseteq T(f) = \{v \mid f(v) = 1\}$ and $F \subseteq F(f) = \{v \mid f(v) = 0\}$; any such $f$ is called an *extension* of $(T, F)$.

We often identify a formula $\varphi$ with the Bf which it defines. A term $t$ is an *implicant* of a Bf $f$, if $t \leq f$ holds, where $\leq$ is the usual ordering defined by $f \leq g \leftrightarrow T(f) \subseteq T(g)$. Moreover, $t$ is *prime* if no proper subterm $t'$ of $t$ is an implicant of $f$. A DNF $\varphi = \bigvee_i t_i$ is *prime*, if each term $t_i$ is a prime implicant of $\varphi$ and no term $t_i$ is redundant, i.e., removing $t_i$ from $\varphi$ changes the function.

A *decision list* (DL) $L$ is a finite sequence of pairs $(t_1, b_1)$, $(t_2, b_2), \ldots, (t_d, b_d)$, $d \geq 1$, where for each $i = 1, \ldots, d-1$, $t_i$ is any term, $t_d = \top$, and $b_i \in \{0,1\}$, for each $i = 1, \ldots, d$. $L$ defines a Bf $f : \{0,1\}^n \to \{0,1\}$ by $f(v) = b_i$, where $i = min\{i \mid v \in T(t_i)\}$. We call a Bf sometimes a decision list, if $f$ is definable by some decision list; this terminology is inherited to restricted decision lists.

A *$k$-decision list* ($k$-DL) is a decision list where each term $t_i$ contains at most $k$ literals; we denote by $\mathcal{C}_{k\text{-}DL}$ the class of all (functions represented by) $k$-decision lists. In particular, $\mathcal{C}_{1\text{-}DL}$ is the class of decision lists where each term is either a single literal or empty. A decision list is *monotone* [16], if each term $t$ in it is *positive*, i.e., $N(t) = \emptyset$. By $\mathcal{C}^{mon}_{k\text{-}DL}$ we denote the restriction of $\mathcal{C}_{k\text{-}DL}$ to monotone decision lists.

A Bf $f$ is *Horn*, if $F(f) = Cl_\wedge(F(f))$, where $Cl_\wedge(S)$ denotes the closure of set $S \subseteq \{0,1\}^n$ of vectors under component-wise conjunction $\wedge$ of vectors; by $\mathcal{C}_{Horn}$ we denote the class of all Horn functions. It is known that $f$ is Horn if and only if $f$ is represented by some Horn DNF. If $f$ is also represented by a positive DNF, i.e., a DNF in which each term is positive, then $f$ is called *positive*; $\mathcal{C}_{pos}$ denotes the class of all positive functions.

For any vector $w \in \{0,1\}^n$, we define $ON(w) = \{i \mid w_i = 1\}$ and $OFF(w) = \{i \mid w_i = 0\}$, and for any set of vectors $S \subseteq \{0,1\}^n$ we define $ON(S) = \bigcap_{v \in S} ON(v)$ and similarly $OFF(S) = \bigcap_{v \in S} OFF(v)$. Here we assume that $ON(S) = OFF(S) = \{1, 2, \ldots, n\}$ if $S = \emptyset$. The *renaming* of an $n$-ary Bf $f$ *by* $w$, denoted $f^w$, is the Bf $f(x \oplus w)$, i.e., $T(f^w) = \{v \mid v \oplus w \in T(f)\}$, where $\oplus$ is componentwise addition modulo 2 (XOR). For any class of Bfs $\mathcal{C}$, we denote by $\mathcal{C}^R$ the closure of $\mathcal{C}$ under renamings. The renaming of a formula $\varphi$ by $w$, denoted $\varphi^w$, is the formula resulting from $\varphi$ by replacing each literal involving a variable $x_i$ with $w_i = 1$ by its opposite. E.g., let $f = x_1\overline{x}_2 \vee x_2x_3 \vee \overline{x}_1\overline{x}_3\overline{x}_4$. Then, the renaming of $f$ by $w = (1,1,0,0)$ is $f^w = \overline{x}_1x_2 \vee \overline{x}_2x_3 \vee x_1\overline{x}_3\overline{x}_4$.

For any assignment $A = (x_{i_1} \leftarrow a_1, x_{i_2} \leftarrow a_2, \ldots, x_{i_k} \leftarrow a_k)$ for values $a_i \in \{0,1\}$ to the variables $x_{i_j}$, we denote by $f_A = f_{(x_{i_1} \leftarrow a_1, x_{i_2} \leftarrow a_2, \ldots, x_{i_k} \leftarrow a_k)}$ the function of $(n-k)$ variables obtained by fixing variables $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ as specified by $A$; Similarly, $\varphi_A$ denotes the formula obtained from $\varphi$ by simultaneously substituting $a_j$ for $x_{i_j}$, for $j = 1, 2, \ldots, k$.

## 3 Characterizations of 1-Decision Lists

### 3.1 Main result of this section

Let $\mathcal{C}_{R\text{-}1}$, $\mathcal{C}_{LR\text{-}1}$, $\mathcal{C}^R_{DH}$, $\mathcal{C}_{ND}$, $\mathcal{C}_{TH}$, and $\mathcal{C}_{2M}$ denote the classes of read-one functions, linear read-once functions, disguised double Horn functions, nested difference functions, threshold functions, and 2-monotonic functions, respectively (formal definitions of all these classes are given below). We can prove the following result.

**Theorem 3.1** $\mathcal{C}_{1\text{-}DL} = \mathcal{C}_{LR\text{-}1} = \mathcal{C}^R_{DH} = \mathcal{C}_{ND} = \mathcal{C}_{TH} \cap \mathcal{C}_{R\text{-}1} = \mathcal{C}_{2M} \cap \mathcal{C}_{R\text{-}1}$.

**Proof**. Immediate from Theorem 3.4, Proposition 3.6 and Theorem 3.7. $\square$

**Read-once functions.** A function $f$ is called *read-once*, if it can be represented by read-once formula, i.e., a formula without repetition of variables. The class $\mathcal{C}_{R\text{-}1}$ of read-once functions has been extensively studied in the literature, cf. [37, 26, 39, 31, 22, 18, 25, 11].

**Definition 3.1** Define the class $\mathcal{F}_{LR\text{-}1}$ of *linear read-once* formulas by the following recursive form:

(1) $\top, \bot \in \mathcal{F}_{LR\text{-}1}$, and $x_i, \overline{x}_i \in \mathcal{F}_{LR\text{-}1}$ for every variable $x_i$;

(2) if $\varphi \in \mathcal{F}_{LR\text{-}1} \setminus \{\top, \bot\}$ and $x_i$ is a variable not occurring in $\varphi$, then $x_i \vee \varphi$, $\overline{x}_i \vee \varphi$, $x_i \wedge \varphi$, $\overline{x}_i \wedge \varphi \in \mathcal{F}_{LR\text{-}1}$.

Call a Bf $f$ *linear read-once* [12], if it can be represented by a formula in $\mathcal{F}_{LR\text{-}1}$, and let $\mathcal{C}_{LR\text{-}1}$ denote the class of all such functions. E.g., $x_1x_2(\overline{x}_4 \vee x_3 \vee x_5\overline{x}_6)$ is linear read-once, while $x_2x_3 \vee x_4 \vee \overline{x}_1\overline{x}_5$ is not. Note that two read-once formulas without occurrence of $\top, \bot$ are equivalent if and only if they can be transformed through associativity and commutativity into each other [22]. Hence, the latter formula does not represent a linear read-once function.

The following is now easy to see (cf. also [5, p. 11]):

**Proposition 3.2** $\mathcal{C}_{LR\text{-}1} = \mathcal{C}_{1\text{-}DL}$.

Note that any $\varphi \in \mathcal{F}_{LR\text{-}1}$ is convertible into an equivalent 1-decision list in linear time and vice versa.

**Horn functions.** We next give a characterization in terms of Horn functions. A Bf $f$ is called *double Horn* [14], if $T(f) = Cl_\wedge(T(f))$ and $F(f) = Cl_\wedge(F(f))$. The class of these functions is denoted by $\mathcal{C}_{DH}$. Note that $f$ is double Horn if and only if $f$ and $\overline{f}$ are Horn. E.g.,

$$f \;=\; \overline{x}_1 \vee x_2x_3\overline{x}_4 \vee x_2x_3x_5x_6\overline{x}_7$$

is double Horn, because

$$\begin{aligned}\overline{f} &= x_1(\overline{x}_2 \vee \overline{x}_3 \vee x_4)(\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee \overline{x}_6 \vee x_7)\\ &= x_1\overline{x}_2 \vee x_1\overline{x}_3 \vee x_1x_4\overline{x}_5 \vee x_1x_4\overline{x}_6 \vee x_1x_4x_7\end{aligned}$$

is Horn. Alternatively, a Bf $f$ is double Horn if and only if it has both a Horn DNF and a Horn CNF representation. In the previous example, this is easily seen to be the case. The class of double Horn functions has been considered in [14, 12] for giving $T(f)$ and $F(f)$ a more balanced role in the process of finding a Horn extension.

We can show the somewhat unexpected result that the classes $\mathcal{C}^R_{DH}$ and $\mathcal{C}_{LR\text{-}1}$ coincide (and hence $\mathcal{C}^R_{DH} = \mathcal{C}_{LR\text{-}1} = \mathcal{C}_{1\text{-}DL}$). This gives a precise syntactical characterization of the semantically defined class $\mathcal{C}^R_{DH}$, and, by the previous result, a semantical characterization of $\mathcal{C}_{1\text{-}DL}$.

The proof of this result is based on the following lemma, which can be found in [14, 12]. Let $V = \{1, 2, \ldots, n\}$ and $\pi : V \to V$ be any permutation of $V$. Then, let $\Gamma_\pi$ be the set of Horn terms $\Gamma_\pi = \{x_{\pi(1)} \cdots x_{\pi(i)} \overline{x}_{\pi(i+1)} \mid 0 \leq i < n\} \cup \{x_{\pi(1)} \cdots x_{\pi(n)}\}$; e.g., for $V = \{1, 2\}$ and $\pi(1) = 2$, $\pi(2) = 1$, we have $\Gamma_\pi = \{\overline{x}_2, x_2\overline{x}_1, x_2x_1\}$.

**Lemma 3.3 (**[14]) *Let $f$ be a Bf on variables $x_i$, $i \in V$. Then, $f \in \mathcal{C}_{DH}$ holds if and only if $f$ can be represented by a DNF $\varphi = \bigvee_{t \in S} t$ for some permutation $\pi$ of $V$ and $S \subseteq \Gamma_\pi$.*[1] □

Denote by $\mathcal{C}^{rev}_{DH} = \{f^{(11\cdots1)} \mid f \in \mathcal{C}_{DH}\}$ the class of all reversed double Horn functions.

**Theorem 3.4** $\mathcal{C}^R_{DH} = \mathcal{C}_{LR\text{-}1} = \mathcal{C}_{1\text{-}DL}$ *and* $\mathcal{C}^{rev}_{DH} = \mathcal{C}^{mon}_{1\text{-}DL}$. □

**Proof**. Let $\varphi$ be a DNF for a function $f \in \mathcal{C}_{DH}$ as in Lemma 3.3. By algebraic transformations, $\varphi$ can be rewritten to a formula $\psi \in \mathcal{F}_{LR\text{-}1}$ of the form

$$\psi = \begin{cases} x_{11}x_{12}\ldots x_{1n_1}(\overline{x}_{21} \vee \overline{x}_{22} \vee \ldots \vee \overline{x}_{2n_2} \\ \qquad \vee (x_{31}x_{32}\ldots x_{3n_3}(\ldots(\overline{x}_{d1} \vee \overline{x}_{d2} \vee \ldots \vee \overline{x}_{dn_d})))) & \text{if } d \text{ is even} \\ x_{11}x_{12}\ldots x_{1n_1}(\overline{x}_{21} \vee \overline{x}_{22} \vee \ldots \vee \overline{x}_{2n_2} \\ \qquad \vee (x_{31}x_{32}\ldots x_{3n_3}(\ldots(x_{d1}x_{d2}\ldots x_{dn_d})))) & \text{if } d \text{ is odd,} \end{cases} \tag{3.1}$$

where $d \geq 0$, $n_1 \geq 0$, $n_i \geq 1$ for $i = 2, 3, \ldots, d$, and the variables $x_{11}, x_{12}, \ldots, x_{dn_d}$ are all different.

For example, the formula $\varphi = x_2\overline{x}_1 \vee x_2x_1$, where $S = \{x_2\overline{x}_1, x_2x_1\}$ for $V$ and $\pi$ as above, can be rewritten as follows: $x_2\overline{x}_1 \vee x_2x_1 = x_2(\overline{x}_1 \vee x_1) = x_2 \wedge \top = x_2$; the formula $\psi = x_1x_2\overline{x}_3 \vee x_1x_2x_3x_4\overline{x}_5 \vee x_1x_2x_3x_4x_5\overline{x}_6$ for $V = \{x_1, \ldots, x_6\}$ and $\pi =$ identity can be rewritten as

$$x_1x_2\overline{x}_3 \vee x_1x_2x_3x_4\overline{x}_5 \vee x_1x_2x_3x_4x_5\overline{x}_6 = x_1x_2(\overline{x}_3 \vee x_4\overline{x}_5 \vee x_4x_5\overline{x}_6) = x_1x_2(\overline{x}_3 \vee x_4(\overline{x}_5 \vee \overline{x}_6)).$$

Therefore, we have $\mathcal{C}^R_{DH} \subseteq \mathcal{C}_{LR\text{-}1}$. Moreover, since every formula $\varphi \in \mathcal{F}_{LR\text{-}1}$ can be transformed to form (3.1) by changing the polarities of variables, $\mathcal{C}^R_{DH} \supseteq \mathcal{C}_{LR\text{-}1}$ holds; hence $\mathcal{C}^R_{DH} = \mathcal{C}_{LR\text{-}1}$. This together with Proposition 3.2 shows the first statement of this theorem.

The second statement easily follows from the above argument. □

Thus, there exists an interesting relationship between 1-decision lists, read-once formulas, and (disguised) Horn functions. By means of the relationship in Theorem 3.4, we are able to precisely characterize the prime DNFs of functions in $\mathcal{C}^R_{DH}$. This is an immediate consequence of the next theorem.

[1]This lemma can also be derived from a related result on finite distributive lattices, see [28].

**Theorem 3.5** *Every $f \in \mathcal{C}^R_{DH}$ (equivalently, $f \in \mathcal{C}_{1\text{-}DL}$, $f \in \mathcal{C}_{LR\text{-}1}$) has a renaming $w$ such that $f^w$ is positive and represented by the unique prime DNF*

$$\varphi = \bigvee_{i=1}^{m} t_1 \cdots t_i x_{\ell_i} \tag{3.2}$$

*where $\{t_i, x_{\ell_i} \mid i = 1, \dots, m\}$ is a set of pairwise disjoint positive terms and each $t_i$, $i = 1, 2, \dots, m$ may be empty.*[2] *In particular,* (3.2) *implies $\varphi = \bot$ if $m = 0$. Conversely, every such $\varphi$ of* (3.2) *represents an $f \in \mathcal{C}^R_{DH}$ (equivalently, an $f \in \mathcal{C}_{1\text{-}DL}$, $f \in \mathcal{C}_{LR\text{-}1}$).*

**Proof**. Lemma 3.3 implies that $f \in \mathcal{C}^R_{DH}$ can be renamed to a function $g$ represented by a linear read-once formula (3.1) (cf. proof of Theorem 3.4); expanding this form into DNF (apply distributivity) and subsequent renaming of negative variables yields form (3.2). The latter form is clearly a prime DNF, and it is unique since every positive function has a unique prime DNF. Conversely, $\varphi$ in (3.2) can be rewritten by factorization to a linear read-once formula $t_1(x_{\ell_1} \vee t_2(x_{\ell_2} \vee t_3(x_{\ell_3} \vee \cdots t_n x_{\ell_n})))$, where empty terms $t_i$ are simply omitted. The result thus follows from Theorem 3.4. $\square$

**Nested differences of concepts.** In [21], learning issues for concept classes have been studied which satisfy certain properties. In particular, learning of concepts expressed as the nested difference $c_1 \setminus (c_2 \setminus (\cdots (c_{k-1} \setminus c_k))$ of concepts $c_1, \dots, c_k$ has been considered, where the $c_i$ are from a concept class which is closed under intersection. Here, a concept can be viewed as a Bf $f$, a concept class $C$ as a class of Bfs $\mathcal{C}_C$, and the intersection property amounts to closedness of $\mathcal{C}_C$ under conjunction, i.e., $f_1, f_2 \in \mathcal{C}_C$ implies $f = f_1 \wedge f_2 \in \mathcal{C}_C$. Clearly, the class of Bfs $f$ definable by a single (possible empty) term $t$ enjoys this property. Let $\mathcal{C}_{ND}$ denote the class of nested differences where each $c_i$ is a single term. Then the following holds.

**Proposition 3.6** $\mathcal{C}_{1\text{-}DL} = \mathcal{C}_{ND}$.

The proof of this proposition is omitted, since we shall prove a more general result at the end of this section in Theorem 3.14, where we also give a characterization of $\mathcal{C}^{mon}_{1\text{-}DL}$. Thus, the general learning results in [21] apply in particular to the class of 1-decision lists, and thus also to disguised double Horn functions and linear read-once functions.

**Threshold and 2-monotonic functions.** Let us denote by $\mathcal{C}_{TH}$ the class of threshold functions and by $\mathcal{C}_{2M}$ the class of 2-monotonic functions.

A function $f$ on variables $x_1, \dots, x_n$ is *threshold* (or, *linearly separable*) if there are weights $w_i$, $i = 1, 2, \dots, n$, and a threshold $w_0$ from the reals such that $f(x_1, \dots, x_n) = 1$ if and only if $\sum_{i=1}^{n} w_i x_i \geq w_0$.

A function is *2-monotonic*, if for each assignment $A$ of size at most 2, either $f_A \leq f_{\overline{A}}$ or $f_A \geq f_{\overline{A}}$ holds, where $\overline{A}$ denotes the opposite assignment to $A$ [32].

The property of 2-monotonicity and related concepts have been studied under various names in the fields of threshold logic, hypergraph theory and game theory. This property can be seen as an algebraic generalization of the thresholdness. Note that $\mathcal{C}^R_{TH} = \mathcal{C}_{TH}$ and $\mathcal{C}^R_{2M} = \mathcal{C}_{2M}$. We have the following unexpected result.

[2]Note that the variables $x_{\ell_i}$ are viewed as terms here.

**Theorem 3.7** $\mathcal{C}_{1\text{-}DL} = \mathcal{C}_{TH} \cap \mathcal{C}_{R\text{-}1} = \mathcal{C}_{2M} \cap \mathcal{C}_{R\text{-}1}$.

**Proof**. It is well-known that $\mathcal{C}_{TH} \subset \mathcal{C}_{2M}$ [32], where $\subset$ is proper inclusion; moreover, also $\mathcal{C}_{1\text{-}DL} \subseteq \mathcal{C}_{TH}$ has been shown [5, 3]. (Notice that in [12], the inclusion $\mathcal{C}^{R}_{DH} \subseteq \mathcal{C}_{TH}$ was independently shown, using the form (3.2) and proceeding similar as in [3]; the idea is to give all the variables in $t_j$ the same weight, decreasing by index $j$, and to assign $x_i$ a weight so that every term $t = t_1 t_2 \ldots t_i x_i$ in $\varphi$ has same weight; the threshold $w_0$ is simply the weight of a term $t$.)

Thus, by the results from above, it remains to show that $\mathcal{C}_{2M} \cap \mathcal{C}_{R\text{-}1} \subseteq \mathcal{C}_{1\text{-}DL}$ holds.

Recall that a function $g$ on $x_1, x_2, \ldots, x_n$ is *regular* [32], if and only if $g(v) \geq g(w)$ holds for all $v, w \in \{0,1\}^n$ with $\sum_{j \leq k} v_j \geq \sum_{j \leq k} w_j$, for $k = 1, 2, \ldots, n$; denote by $\mathcal{C}_{reg}$ the class of regular functions. The following facts are known (cf. [32]):

$(a)$ Every regular function is positive and 2-monotonic;

$(b)$ every 2-monotonic function becomes regular after permuting and renaming arguments.

$(c)$ $\mathcal{C}_{reg}$ is closed under arbitrary assignments $A$ (i.e., $f_A \in \mathcal{C}_{reg}$ holds for every $f \in \mathcal{C}_{reg}$ and assignment $A$).

¿From $(a)$–$(c)$, it remains to show that $\mathcal{C}_{reg} \cap \mathcal{C}_{R\text{-}1} \subseteq \mathcal{C}_{LR\text{-}1}$ $(= \mathcal{C}_{1\text{-}DL})$.

We claim that any function $f \in \mathcal{C}_{reg} \cap \mathcal{C}_{R\text{-}1}$ can be written either as

$$\text{(i) } f = x_{i_1} \vee x_{i_2} \vee \ldots \vee x_{i_k} \vee f' \qquad \text{or} \qquad \text{(ii) } f = x_{i_1} x_{i_2} \ldots x_{i_k} f',$$

where $f'$ is a regular read-once function not depending on any $x_{i_j}$, $1 \leq j \leq k$. An easy induction using Theorem 3.4 gives then the desired result and completes the proof.

Since $f$ is read-once, it can be decomposed according to one of the following two cases:

Case 1: $f = f_1 \vee f_2 \vee \ldots \vee f_k$, where the $f_i$ depend on disjoint sets of variables $B_i$ and no $f_i$ can be decomposed similarly. We show that $|B_i| \geq 2$ holds for at most one $i$, which means that $f$ has form (i). For this, assume on the contrary that, without loss of generality, $|B_1|, |B_2| \geq 2$. By considering an assignment $A$ that kills all $f_3, f_4, \ldots, f_k$, it follows that the function $g = f_1 \vee f_2$ is regular. Observe that any prime implicant of $g$ is a prime implicant of $f_1$ or $f_2$, and that each of them has length $\geq 2$ (since $f$ is read-once and by the assumption on the decomposition). Let $\ell$ be the smallest index in $B_1 \cup B_2$ i.e., $\ell \leq k$ for all $k \in B_1 \cup B_2$, and assume without loss of generality that $\ell \in B_1$. Let $t$ be any prime implicant of $f_2$ and $v$ satisfy $ON(v) = P(t)$. Let $w = v + e^{(\ell)} - e^{(h)}$, where $h \in ON(v)$ and $e^{(k)}$ is the unit vector with $e^{(k)}_k = 1$ and $e^{(i)} = 0$, for all $i \neq k$. Note that $l < h$ and $l \in OFF(v)$ by definition. Then $g(w) = 0$ holds. Indeed, $ON(w) \not\supseteq P(t_2)$ for every prime implicant $t_2$ of $f_2$, since $ON(w) \cap B_2 \subset P(t)$, and also $ON(w) \not\supseteq P(t_1)$ for every prime implicant $t_1$ of $f_1$, since $|ON(w) \cap B_1| = 1$. Consequently, the vectors $v$ and $w$ with $\sum_{j \leq k} w_j \geq \sum_{j \leq k} v_j$ for all $k = 1, 2, \ldots, n$ satisfy $g(v) = 1$ and $g(w) = 0$. Thus $g$ is not regular, which is a contradiction. This proves our claim.

Case 2: $f = f_1 f_2 \ldots f_k$, where the $f_i$ depend on disjoint sets of variables $B_i$ and no $f_i$ can be decomposed similarly. Then, the dual function $f^d$ has the form in case 1. (Recall that a formula representing the dual of $f$, $f^d = \overline{f}(\overline{x})$, is obtained from any formula representing $f$ by interchanging $\vee$ (resp., 0) and $\wedge$ (resp., 1).) Since the dual of a regular function is also regular [32], it follows that $f^d$ has the form (i), which implies that $f$ has form (ii). □

## 3.2 Possible generalizations

A generalization of Theorem 3.1 is an interesting issue. In particular, whether for $k$-decision lists and read-$k$ functions, where $k$ is a constant, similar relationships hold. It appears that this is not the case.

By using a counting argument, one can show that for every $k > 1$, $\mathcal{C}_{k\text{-}DL}$ contains some function which is not expressible by a read-$k$ formula. In fact, a stronger result can be obtained.

Let for any integer function $F(n)$ denote $\mathcal{C}_R(F(n))$ the class of Bfs $f(x_1, \ldots, x_n)$, $n \geq 0$, which are definable by formulas in which each variable occurs at most $F(n) \geq 1$ times. For any class of integer functions $\mathbf{F}$, define $\mathcal{C}_R(\mathbf{F}) = \bigcup_{F(n)\in\mathbf{F}} \mathcal{C}_R(F(n))$. Denote by $\mathcal{C}^k_{pos}$ and $\mathcal{C}^{\leq k}_{pos}$ the classes of positive Bfs $f$ such that all prime implicants of $f$ have size $k$ (resp., at most $k$), where $k$ is a constant.

**Lemma 3.8** *For every $k > 1$, for all but finitely many $n > k$ there exists an $n$-ary $f \in \mathcal{C}^k_{pos}$ such that $f \notin \mathcal{C}_R(\frac{n^{k-1}}{kk!\log n})$.*

**Proof**. Since all prime implicants of a positive function are positive, $\mathcal{C}^k_{pos}$ contains

$$2^{\binom{n}{k}} \tag{3.3}$$

functions on $n$ variables. On the other hand, the number of positive functions in $\mathcal{C}_R(F(n))$ is bounded by

$$3 \cdot 2^{m-2} m! \binom{2m-1}{m}, \tag{3.4}$$

where $m = F(n) \cdot n$. Indeed, without loss of generality, a formula $\varphi$ defining some positive function does not contain negation. Assuming that all variables occur $F(n)$ times, the formula tree has $m$ leaves (atoms) and $m-1$ inner nodes (connectives). Written in a post-order traversal, it is a string of $2m-1$ characters, of which $m$ denote atoms and the others connectives. There are $m!\binom{2m-1}{m}$ ways to place the atoms in the string, if they were all different (this simplification will suffice), times $2^{m-1}$ combinations of connectives. If we allow the single use of a binary connective $r(x, y)$, which evaluates to the right argument $y$, we may assume w.l.o.g. that $\varphi$ contains exactly $F(n)$ occurrences of each variable. Thus, (3.4) is an upper bound on positive read-$F(n)$ functions in $n$ variables. (Clearly, $\top$ and $\bot$ are implicitly accounted since multiple trees for e.g. $f = x_1$ are counted.)

Now, let us compare (3.3) with (3.4). Clearly, (3.4) is bounded by

$$3 \cdot 2^{m-2}(2m)^m, \tag{3.5}$$

since $m!\binom{2m-1}{m} = (2m-1)(2m-2)\cdots(2m-m) < 2m^m$. Take the logarithm of (3.3) and (3.5) for base 2, and consider the inequality

$$\binom{n}{k} \quad > \quad \log 3 + m - 2 + m(\log m + 1). \tag{3.6}$$

Since $\binom{n}{k} = n(n-1)\cdots(n-k+1)/k!$, this amounts to

$$n^k/k! > m(\log m + 2) + p(n), \tag{3.7}$$

where $p(n)$ is a polynomial of degree $k-1$. For $F(n) = \frac{n^{k-1}}{kk!\log n}$, we obtain $m = \frac{n^k}{kk!\log n}$ and thus

$$\begin{aligned} \frac{n^k}{k!} &> \frac{n^k}{kk!\log n}(k\log n - \log(kk!\log n) + 2) + p(n) \\ &= \frac{n^k}{k!}\Big(1 - \frac{\log(kk!\log n) - 2}{k\log n}\Big) + p(n). \end{aligned}$$

It is easily seen that for large enough $n$, this inequality holds. This proves the lemma. $\Box$

Let $\mho(\frac{n^{k-1}}{kk!\log n})$ be the class of functions $F(n)$ such that $F(n) \leq \frac{n^{k-1}}{kk!\log n}$ holds for infinitely many $n$.

**Theorem 3.9** $\mathcal{C}_{pos}^{\leq k} \not\subseteq \mathcal{C}_R(\mho(\frac{n^{k-1}}{kk!\log n}))$, *for every* $k > 1$.

It is easy to see that every function in $\mathcal{C}_{pos}^{\leq k}$ is in $\mathcal{C}_R(n^{k-1})$. Hence, $k-1$ is the lowest polynomial degree $k' = k'(k)$ such that $\mathcal{C}_{pos}^{\leq k} \subseteq \mathcal{C}_R(n^{k'})$.

**Corollary 3.10** $\mathcal{C}_{k\text{-}DL}^{mon} \not\subseteq \mathcal{C}_R(\mho(\frac{n^{k-1}}{kk!\log n}))$ *and* $\mathcal{C}_{k\text{-}DL} \not\subseteq \mathcal{C}_R(\mho(\frac{n^{k-1}}{kk!\log n}))$, *for every* $k > 1$.

Consequently, any generalization of the parts in Theorem 3.1 involving read-once functions to a characterization of $k$-decision lists in terms of read-$k$ functions fails; this remains true even if we allow a polynomial number of repetitive variable uses, where the degree of the polynomial is smaller than $k-1$.

Let us now consider a possible generalization of the characterization in terms of Horn functions. Since $\mathcal{C}_{k\text{-}DL}$ contains all functions with a $k$-CNF (in particular, also the parity function on $k$ variables), it is hard to see any interesting relationships between $\mathcal{C}_{k\text{-}DL}$ and combinations or restrictions of Horn functions.

For nested differences of concepts, however, there is a natural generalization of the result in Theorem 3.1. Let $\mathcal{C}_{ND}(\mathcal{C})$ denote the class of all functions definable as nested differences of Bfs in $\mathcal{C}$, and let similarly denote $\mathcal{C}_{DL}(\mathcal{C})$ the class of functions definable by a $\mathcal{C}$-decision list, i.e., a decision list in which each term $t_i$ except the last ($t_d = \top$) is replaced by some $f \in \mathcal{C}$. Then, the following holds.

**Theorem 3.11** *Let $\mathcal{C}$ be any class of Bfs. Then, $\mathcal{C}_{DL}(\mathcal{C}) = \mathcal{C}_{ND}(\mathcal{C}^* \cup \{\top\})$, where $\mathcal{C}^* = \{\overline{f} \mid f \in \mathcal{C}\}$ contains the complements of the functions in $f$.*

**Proof**. We show by induction on $d \geq 1$ that every $f$ represented by a $\mathcal{C}$-decision list of length $\leq d$ is in $\mathcal{C}_{ND}(\mathcal{C}^* \cup \{\top\})$, and that each nested difference $f_1 \setminus (f_2 \setminus (\cdots (f_{d-1} \setminus f_d)))$ where all $f_i$ are from $\mathcal{C}^* \cup \{\top\}$, is in $\mathcal{C}_{DL}(\mathcal{C})$.

(Basis) For $d = 1$, there are two $\mathcal{C}$-decision lists: $(\top, 0)$ and $(\top, 1)$ respectively. They are represented by the nested difference $\top \setminus \top$ and $\top$, respectively. Conversely, $(\top, 1)$ represents $\top$, and for any function $f \in \mathcal{C}^*$, the decision list $(\overline{f}, 0), (\top, 1)$ obviously represents $f$; observe that $\overline{f} \in \mathcal{C}$ holds.

(Induction) Suppose the statement holds for $d$, and consider the case $d+1$. First, consider a $\mathcal{C}$-decision list $L = (f_1, b_1), \ldots, (f_{d+1}, b_{d+1})$, where without loss of generality $f_1 \not\equiv \top$. By the induction hypothesis, the tail $L' = (t_2, b_2), \ldots, (t_{d+1}, b_{d+1})$ of $L$ can be represented by a nested difference $D' = c'_1 \setminus (\cdots (c'_m \setminus c'_{m+1}) \cdots)$, defining a Bf $f' \in \mathcal{C}_{ND}(\mathcal{C})$. If $b_1 = 1$, then $L$ defines the function $f = f_1 \vee f'$, which can be represented by the nested difference $\top \setminus (\overline{f}_1 \setminus f')$; replacing $f'$ by $D'$, this is a nested difference of functions in $\mathcal{C}^* \cup \{\top\}$. Hence, $f \in \mathcal{C}_{ND}(\mathcal{C}^* \cup \{\top\})$ holds. On the other hand, if $b_i = 0$, then $L$ represents the function $f = \overline{f}_1 \wedge f'$, which is equivalent to $\neg(f_1 \vee \overline{f'})$; since the complement of any function $g$ is represented by the nested difference $\top \setminus g$, we obtain from the already discussed scheme for disjunction that $f$ is represented by the nested difference

$$\top \setminus (\top \setminus (\overline{f}_1 \setminus (\top \setminus f')));$$

replacing $f'$ with $D'$, we obtain a nested difference of functions in $\mathcal{C}^* \cup \{\top\}$, hence $f \in \mathcal{C}_{ND}(\mathcal{C}^* \cup \{\top\})$.

Second, let $D = f_1 \setminus (f_2 \setminus (\cdots (f_d \setminus f_{d+1})))$ be any nested difference of functions in $\mathcal{C}^* \cup \{\top\}$. By the induction hypothesis, $D' = (f_2 \setminus (\cdots (f_d \setminus f_{d+1})))$ represents a function $f' \in \mathcal{C}_{DL}(\mathcal{C})$; thus, $D$ represents the function $f = f_1 \wedge \neg f'$.

It is easy to see that for any $\mathcal{C}$, $\mathcal{C}_{DL}(\mathcal{C})$ is closed under complementation [34] (replace in a decision list each $b_i$ by $1 - b_i$ to obtain a decision list for the complement function). Hence, $\overline{f'}$ is represented by some $\mathcal{C}$-decision list $L'$. Now, if $f_1 = \top$, then $L'$ represents $f$; otherwise, the decision list $L = (\overline{f}_1, 0), L'$ represents $f$. Hence, $f \in \mathcal{C}_{DL}(\mathcal{C})$.

Consequently, the induction statement holds for $d + 1$. This concludes the proof of the result. □

Proposition 3.6 is an immediate corollary of this result. Moreover, we get the following result. Let $\mathcal{C}_{k\text{-}cl}$ denote the class of functions definable by a single clause with at most $k$ literals, plus $\top$.

**Corollary 3.12** $\mathcal{C}_{k\text{-}DL} = \mathcal{C}_{ND}(\mathcal{C}_{k\text{-}cl})$, $\mathcal{C}_{DL}(\mathcal{C}_{k\text{-}DNF}) = \mathcal{C}_{ND}(\mathcal{C}_{k\text{-}CNF})$, *for* $k \geq 1$.

Thus, $\mathcal{C}_{ND}(\mathcal{C}_{k\text{-}cl})$ characterizes $\mathcal{C}_{k\text{-}DL}$. However, $\mathcal{C}_{k\text{-}cl}$ is not closed under conjunction, and thus, strictly speaking, not an instance of the schema in [21]. A characterization by such an instance is nonetheless possible. Call a subclass $\mathcal{C}' \subseteq \mathcal{C}$ a *disjunctive base* of a class $\mathcal{C}$, if every $f \in \mathcal{C}$ can be expressed as a disjunction $f = f_1 \vee f_2 \vee \cdots \vee f_m$ of functions $f_i$ in $\mathcal{C}'$.

**Lemma 3.13** *If* $\mathcal{C}'$ *is a disjunctive base for* $\mathcal{C}$, *then* $\mathcal{C}_{DL}(\mathcal{C}') = \mathcal{C}_{DL}(\mathcal{C})$.

**Proof**. Suppose an item $(f, b)$ occurs in a $\mathcal{C}$-decision list $L$. By hypothesis, $f = f_1 \vee \cdots \vee f_m$, where each $f_i \in \mathcal{C}'$. Replace the item by $k$ items $(f_1, b)$,..., $(f_m, b)$. Then, the resulting decision list is equivalent to $L$. Hence each $\mathcal{C}$-decision list can be converted into an equivalent $\mathcal{C}'$-decision list. □

**Theorem 3.14** $\mathcal{C}_{k\text{-}DL} = \mathcal{C}_{DL}(\mathcal{C}_{k\text{-}DNF}) = \mathcal{C}_{ND}(\mathcal{C}_{k\text{-}CNF})$, *for* $k \geq 1$.

**Proof**. By Corollary 3.12 and Lemma 3.13. □

Thus, nested differences of $k$-CNF functions are equivalent to $k$-decision lists. Observe that from the proof of this result, linear time mappings between nested differences and equivalent $k$-decision do exist. A similar equivalence $\mathcal{C}_{k\text{-}DL} = \mathcal{C}_{ND}(\mathcal{C}_{k\text{-}DNF})$ does not hold. The reason is that the class of single-term functions is not a base for $\mathcal{C}_{k\text{-}CNF}$, which makes it impossible to rewrite a $\mathcal{C}_{k\text{-}CNF}$-decision list to a $k$-decision list in general.

The classes of bounded monotone decision lists can be characterized in a similar way. Let $\mathcal{C}^{pos}_{k\text{-}DNF}$ and $\mathcal{C}^{neg}_{k\text{-}CNF}$ be the subclasses of $\mathcal{C}_{k\text{-}DNF}$ and $\mathcal{C}_{k\text{-}CNF}$ whose members have a positive DNF and a negative CNF (i.e., no positive literal occurs), respectively.

**Theorem 3.15** $\mathcal{C}^{mon}_{k\text{-}DL} = \mathcal{C}_{DL}(\mathcal{C}^{pos}_{k\text{-}DNF}) = \mathcal{C}_{ND}(\mathcal{C}^{neg}_{k\text{-}CNF}),$ *for* $k \geq 1$.

Thus, in particular, if $\mathcal{C}_{Lit^-}$ denotes the class of negative literals plus $\top$, then we obtain the following.

**Corollary 3.16** $\mathcal{C}^{rev}_{DH} = \mathcal{C}^{mon}_{1\text{-}DL} = \mathcal{C}_{ND}(\mathcal{C}_{Lit^-}) = \mathcal{C}_{ND}(\mathcal{C}^{neg}_{1\text{-}CNF})$.

## 4 Recognition from a Formula

Recall that the *membership problem* [20] (also *representation problem* [4, 1]) for a class $\mathcal{C}$ of Boolean functions is deciding whether a given formula $\varphi$ represents a function in $\mathcal{C}$. This problem is also known as the *recognition problem*, and we call any algorithm solving it a *recognition algorithm* (for the class $\mathcal{C}$).

A 1-decision list, and thus also its relatives, can be recognized in polynomial time from formulas of certain classes, which include Horn formulas. The basis for our recognition algorithm is the following lemma:

**Lemma 4.1** *A Bf $f$ is in $\mathcal{C}_{1\text{-}DL}$ if and only if either* (ia) $\overline{x}_j \leq f$, (ib) $\overline{x}_j \leq \overline{f}$, (ic) $x_j \leq f$ *or* (id) $x_j \leq \overline{f}$ *holds for some $j$, and* (ii) $f_{(x_j \leftarrow 1)} \in \mathcal{C}_{1\text{-}DL}$ *(resp., $f_{(x_j \leftarrow 0)} \in \mathcal{C}_{1\text{-}DL}$) holds for all $j$ satisfying* (ia) *or* (ib) *(resp.,* (ic) *or* (id)*).* □

Given a formula $\varphi$, the recognition algorithm proceeds as follows. It picks an index $j$ such that one of (ia)–(id) holds, and then recursively proceeds with $\varphi_{(x_j \leftarrow a)}$ as in (ii). The important point here is that (ii) implies that a greedy choice of any variable $x_j$ satisfying one of the conditions in (i) is enough, and that no backtracking is needed. The details of the algorithm, which implements this greedy choice stratgey, can be found in [12]. For its time complexity, we obtain the following result. For a formula $\varphi$, let $|\varphi|$ denote its length, i.e., the number of symbols in $\varphi$.

**Theorem 4.2** *Let $\mathcal{F}$ be a class of formulas closed under assignments (i.e., $\varphi_A \in \mathcal{F}$ holds for every $\varphi \in \mathcal{F}$ and assignment $A$) such that checking equivalence of $\varphi$ to $\top$ and $\perp$, respectively, can be done in $O(t(n, |\varphi|))$ time for any $\varphi \in \mathcal{F}$.*[3] *Then, deciding whether a given $\varphi \in \mathcal{F}$ represents an $f \in \mathcal{C}_{1\text{-}DL}$ can be done in $O(n^2 t(n, |\varphi|))$ time.*

**Proof**. Immediate from the fact that the recursion depth is bounded by $n$ and that at each level $O(n)$ tests (ia)–(id) are made. □

Hence, the algorithm is polynomial for many classes of formulas, including Horn formulas and quadratic (2-CNF) formulas. Since testing whether $\varphi \equiv \top$ and $\varphi \equiv \perp$ for a Horn DNF $\varphi$ and a quadratic formula is possible in $O(|\varphi|)$ time (cf. [10, 15]), we obtain the following.

**Corollary 4.3** *Deciding whether a given Horn DNF or 2-CNF $\varphi$ represents an $f \in \mathcal{C}_{1\text{-}DL}$ can be done in $O(n^2|\varphi|)$ time.* □

Theorem 4.2 has yet another interesting corollary.

[3]That is, $\mathcal{F}$ is syntactically closed under projection. Formulas such as Horn DNFs must be slightly generalized by allowing occurrences of constants for this property. As usual, $t(n, |\varphi|)$ is monotonic in both arguments.

**Corollary 4.4** *Deciding if an arbitrary positive* (*i.e., negation-free*) *formula $\varphi$ represents an $f \in \mathcal{C}_{LR\text{-}1}$ can be done in polynomial time.* $\Box$

In fact, deciding whether a positive formula $\varphi$ represents a read-once function is co-NP-complete [22, 11]. It turns out that the class of $\mathcal{C}_{LR\text{-}1}$ is a maximal subclass of $\mathcal{C}_{R\text{-}1}$ w.r.t. an inductive (i.e., context-free) bound on the size of disjunctions and conjunctions in a read-once formula such that deciding $f \in \mathcal{C}_{R\text{-}1}$ from a positive formula $\varphi$ is polynomial. Let the class $\mathcal{F}_{2LR\text{-}1}$ of *2-linear read-once* formulas be the class of formulas such that

(1) $\top, \bot \in \mathcal{F}_{2LR\text{-}1}$, and $x_i, \overline{x}_i \in \mathcal{F}_{2LR\text{-}1}$ for every variable $x_i$;

(2) if $\varphi \in \mathcal{F}_{2LR\text{-}1} \setminus \{\top, \bot\}$ and $\psi$ is a read-once formula that contains at most 2 literals and shares no variables with $\varphi$, then $\psi \vee \varphi$, $\psi \wedge \varphi \in \mathcal{F}_{2LR\text{-}1}$.

Note that $\mathcal{F}_{2LR\text{-}1}$ generalizes $\mathcal{F}_{LR\text{-}1}$ by increasing in clause (2) the number of literals in $\psi$ from one to two; this is the least possible increase.

Let $\mathcal{C}_{2LR\text{-}1}$ denote the class of all Bfs which can be represented by some formula from $\mathcal{F}_{2LR\text{-}1}$. Clearly, $\mathcal{C}_{LR\text{-}1} \subseteq \mathcal{C}_{2LR\text{-}1}$, and the inclusion is strict. For example, $f = x_1x_2 \vee x_3x_4$ is a function in $\mathcal{C}_{2LR\text{-}1} \setminus \mathcal{C}_{LR\text{-}1}$. From results in [11, 22], we easily derive the following result.

**Proposition 4.5** *Deciding if an arbitrary positive* (*i.e., negation-free*) *formula $\varphi$ represents a function $f \in \mathcal{C}_{2LR\text{-}1}$ is* co-NP*-hard.*

**Proof**. Based on a construction in [22], it was shown in [11, Theorem 5.7] that deciding whether a given positive formula $\varphi$ represents any function $f \in \mathcal{C}_{R\text{-}1}$ is co-NP-hard. The proof there establishes that this problem is co-NP-hard, even if it is asserted that the only possible such $f$ is of the form $f = x_1x_2 \vee x_3x_4 \vee \cdots x_{2n-1}x_{2n}$. Since $f \in \mathcal{C}_{2LR\text{-}1} \subseteq \mathcal{C}_{R\text{-}1}$, the result follows. $\Box$

In general, the recognition problem for $\mathcal{C}_{LR\text{-}1}$ is unsurprisingly intractable.

**Theorem 4.6** *Deciding whether a given formula $\varphi$ represents a function $f \in \mathcal{C}_{1\text{-}DL}$ is* co-NP*-complete.*

**Proof**. The recognition problem for $\mathcal{C}_{R\text{-}1}$ is in co-NP [2], and it is easy to see that it also in co-NP for $\mathcal{C}_{2M}$. Since co-NP is closed under conjunction, membership in co-NP follows from Theorem 3.1. The hardness part is easy: any class $\mathcal{C}$ having the projection property, i.e., $\mathcal{C}$ is closed under assignments, contains $f = 1$ for each arity, and does not contain all Bfs, is co-NP-hard [20]; obviously, $\mathcal{C}_{1\text{-}DL}$ enjoys this property. $\Box$

As for $k$-decision lists, it turns out that the recognition problem is not harder than for 1-decision lists. In fact, membership in co-NP follows from the result that $k$-decision lists are exact learnable with equivalence queries in polynomial time (proved by Nick Littlestone, unpublished; this also derivable from results in [21] and Theorem 3.14), and the result [2] that for classes which are exact learnable in polynomial time with equivalence and membership queries (under minor constraints), the recognition problem is in co-NP. Hardness holds by the same argument as in the proof of Theorem 4.6.

We conclude this section with a some remarks concerning the equivalence and the implication problem. The problems are, given $k$-decision lists $L_1$ and $L_2$ representing functions $f_1$ and $f_2$, respectively, decide whether $f_1 = f_2$ (equivalence) and $f_1 \leq f_2$ (implication) holds, respectively. Both problems are obviously in co-NP, and they are complete for any fixed $k \geq 3$, since they subsume deciding whether a $k$-DNF formula is a tautology. On the other hand, for $k = 1$, both problems are polynomial, and in fact solvable in linear time. For the remaining case $k = 2$, it can be seen that the problem is also polynomial; the underlying reason is that the satisfiability problem for 2-CNF formulas is polynomial.

## 5 Extension problems

The extension problem for $\mathcal{C}_{1\text{-}DL}$ has already been studied to prove the PAC-learnability of this class. It is known [34] that it is solvable in polynomial time. We point out that the result in [34] can be further improved, by showing that the extension problem for $\mathcal{C}_{1\text{-}DL}$ can be solved in linear time. This can be regarded as a positive result, since the extension problem for the renaming closures of classes that contain $\mathcal{C}_{1\text{-}DL}$ is mostly intractable, e.g., for $\mathcal{C}^R_{Horn}$, $\mathcal{C}^R_{pos}$, $\mathcal{C}^R_{R\text{-}1} = \mathcal{C}_{R\text{-}1}$, $\mathcal{C}^R_{2M} = \mathcal{C}_{2M}$ [7, 6], or no linear time algorithms are known.

We describe here an algorithm EXTENSION (see Figure 1), which outputs a 1-decision list for an extension of a given pdBf $(T, F)$. It uses Lemma 4.1 for the equivalent class $\mathcal{C}_{LR\text{-}1}$ for a recursive extension test. The algorithm is similar to the more general algorithm described in [34], and also a relative of the algorithm "total recall" in [21]. Informally, it examines the vectors of $T$ and $F$, respectively, to see whether a decomposition of form $L \wedge \varphi$ or $L \vee \varphi$ is possible, where $L$ is a literal on a variable $x_i$; if so, then it discards the vectors from $T$ and $F$ which are covered or excluded by this decomposition, and recursively looks for an extension at the projection of $(T, F)$ to the remaining variables. Cascaded decompositions $L_1 \wedge (L_2 \wedge (L_3 \wedge (\cdots)))$ etc are handled simultaneously.

To find an extension of a given pdBf $(T, F)$, the algorithm is called with $I = \{1, \ldots, n\}$. Observe that it could equally well consider $J^+ \cup J^-$ before $I^+ \cup I^-$, when going into the recursive calls. In particular, if an index $i$ is in the intersection of these sets, then both decompositions $x_i \wedge g$ and $x_i \vee g$ are equally good. Note that the execution of steps 2 and 3 alternates in the recursion. Moreover, the algorithm remains correct if only a subset $S \subseteq I^+ \cup I^-$ (resp., $S \subseteq J^+ \cup J^-$) is chosen, which may lead to a different extension.

**Proposition 5.1** *Given a pdBf $(T, F)$, where $T, F \subseteq \{0,1\}^n$, algorithm* EXTENSION *correctly finds an extension $f \in \mathcal{C}_{1\text{-}DL}$ in $O(n^2(|T| + |F|))$ time.* $\square$

Note that algorithm EXTENSION is easily modified such that it outputs an equivalent formula $\varphi \in \mathcal{F}_{LR\text{-}1}$ instead of $L$. Alternatively, $L$ may be converted into a nested difference of concepts $x_1, \ldots, x_n$ in linear time using the rewriting scheme from the proof of Theorem 3.11. Furthermore, integer weights $w_i$ and a threshold $w_0$ for the function represented by $L$ can be easily computed from $\varphi$ with $O(|\varphi|)$ many additions, i.e., in linear time (see [12]). Thus, variants of algorithm EXTENSION may generate these alternative representations for an extension of $(T, F)$ in $\mathcal{C}_{1\text{-}DL}$ within the same time bounds.

It is possible to speed up algorithm EXTENSION by using proper data structures so that it runs in linear time.

**Theorem 5.2** *The extension problem for $\mathcal{C}_{1\text{-}DL}$ (equivalently, for $\mathcal{C}_{LR\text{-}1}$, $\mathcal{C}_{TH} \cap \mathcal{C}_{R\text{-}1}$, and $\mathcal{C}_{ND}$) is solvable in time $O(n(|T| + |F|))$, i.e., in linear time.*

**Algorithm** EXTENSION
**Input**: A pdBf $(T, F)$, $T, F \subseteq \{0,1\}^n$ and a set $I \subseteq \{1, 2, \ldots, n\}$ of indices.
**Output**: A 1-decision list $L$ on variables $x_i$, $i \in I$, if $(T[I], F[I])$ has an extension $f \in \mathcal{C}_{1\text{-}DL}$, where $T[I]$ and $F[I]$ are the projections of $T$ and $F$ to $I$, respectively; otherwise, "No".

```
Step 1. if $T[I] = \emptyset$ then return $L := (\top, 0)$ (exit)                (* no true vectors, return $\bot$ *)
        elseif $F[I] = \emptyset$ then $L := (\top, 1)$ (exit);                    (* no false vectors, return $\top$ *)

Step 2. $I^+ := \cap_{v \in T[I]} ON(v)$ and $I^- := \cap_{v \in T[I]} OFF(v)$;   (* try $x_i(\cdots), \overline{x}_j(\cdots), i \in I^+, j \in I^-$ *)
        if $I^+ \cup I^- = \emptyset$ then go to Step 3                           (* no extension $x_i(\cdots), \overline{x}_i(\cdots)$ possible *)
        else begin                                                                 (* go into recursion *)
          $F' := F \setminus \{w \in F \mid OFF(w) \cap I^+ \neq \emptyset$ or $ON(w) \cap I^- \neq \emptyset\}$;
          $T' := T$; $I' := I \setminus (I^+ \cup I^-)$;
          $L'$ := EXTENSION($T', F', I'$);
          if $L'$ = "No" then return "No" (exit)                                  (* no decomposition $\Rightarrow$ no extension *)
          else (* $I^+ = \{i_1, \ldots, i_k\}$; $I^- = \{j_1, \ldots, j_\ell\}$ *)
            return $L := (\overline{x}_{i_1}, 0), \ldots, (\overline{x}_{i_k}, 0), (x_{j_1}, 0), \ldots, (x_{j_\ell}, 0), L'$ (exit)
        end{if};

Step 3. $J^+ := \cap_{w \in F[I]} ON(w)$ and $J^- := \cap_{w \in F[I]} OFF(w)$;   (* try $x_i \vee \cdots, \overline{x}_j \vee \cdots, i \in J^+, j \in J^-$ *)
        if $J^+ \cup J^- = \emptyset$ then return "No" (exit)                     (* no decomposition $\Rightarrow$ no extension *)
        else begin                                                                 (* go into recursion *)
          $T' := T \setminus \{v \in T \mid OFF(v) \cap J^+ \neq \emptyset$ or $ON(v) \cap J^- \neq \emptyset\}$;
          $F' := F$; $I' := I \setminus (J^+ \cup J^-)$;
          $L'$ := EXTENSION($T', F', I'$);
          if $\psi'$ = "No" then return "No" (exit)
          else (* $J^+ = \{i_1, \ldots, i_k\}$; $J^- = \{j_1, \ldots, j_\ell\}$ *)
            return $L := (x_{i_1}, 1), \ldots, (x_{i_k}, 1), (\overline{x}_{j_1}, 1), \ldots, (\overline{x}_{j_\ell}, 1), L'$ (exit)
        end{if}.                                                                   □
```

Figure 1: Algorithm for computing an 1-DL representing an extension in $\mathcal{C}_{1\text{-}DL}$

**Proof**. (Sketch) This result can be obtained by using appropriate data structures, in particular doubly linked lists and cross-reference pointers. The data structures assure that the same bit of the input is looked up only few times. We merely sketch the main ideas here; the technical details and an implementation-level description of the algorithm can be found in [12].

The set of true vectors, $T$, is stored as follows (cf. Figure 2). For each $i = 1, 2, \ldots, n$ and $j = 0, 1$, there is a doubly linked list $LT_{i,j}$ of all the vectors $v$ in $T$ such that $v$ has at component $i$ value $j$; at each component $i$ of $v$, a link to the entry of $v$ in the respective list $LT_{i,j}$ exists. A counter $\#T_{i,j}$ records how many vectors are contained in $LT_{i,j}$. The counters $\#T_{i,j}$ are placed via pointers in buckets $BT[0]$,..., $BT[n]$, which are organized as doubly linked lists, such that the counter $\#T_{i,j}$ is in $BT[\#T_{i,j}]$. A further counter $\#T$ records the number of vectors in $T$; note that $\#T = \sum_{i,j} \#T_{i,j}/n$. The set of false vectors is stored using completely analogous data structures $LF_{i,j}$, $\#F_{i,j}$, $BF[0]$,..., $BF[n]$, and $\#F$ (see Figure 2).

Notice that these data structures can be built from $(T, F)$ in time $O(n(|T| + |F|)$. Step 2 of algorithm

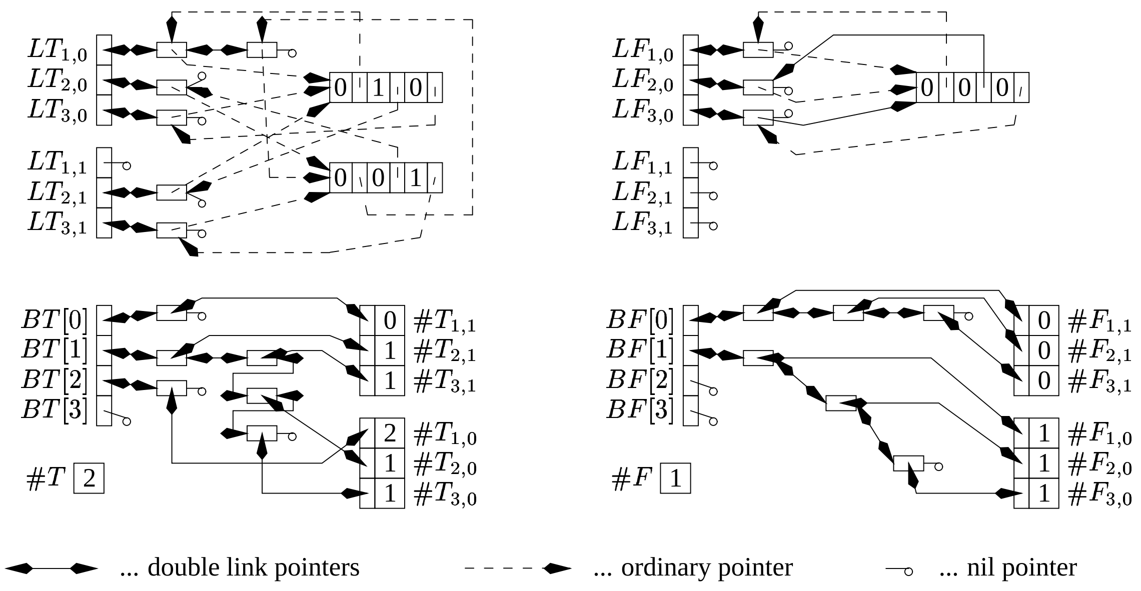

Figure 2: Data structures for $(T, F)$ where $T = \{ (010), (001) \}$ and $F = \{ (000) \}$

EXTENSION is then modified as follows. The bucket $BT[\#T]$ contains those counters $\#T_{i,1}$ and $\#T_{j,0}$ such that $i \in I^+$ and $j \in I^-$, respectively. The sets $F'$ and $T'$ result by removing from $F$ all vectors $v$ in the lists $LF_{i,1}$ and $LF_{j,0}$; using the cross-references, occurrences of $v$ in the other lists $LF_{i',j'}$ are removed as well. Step 3 is analogous.

Like above, the 1-decision list $L$ computed by EXTENSION can be converted to an equivalent formula $\varphi \in \mathcal{F}_{LR\text{-}1}$, a threshold function given by integer weights $w_i$ and a threshold $w_0$, or a nested difference of concepts $x_1, \dots, x_n$ in linear time. $\Box$

Thus, in the learning context we obtain the following result.

**Corollary 5.3** *Learning a Bf $f \in \mathcal{C}_{1\text{-}DL}$ from an arbitrary (possibly spoiled) teaching sequence for $f$ is possible in linear time in the size of the input.*

It turns out that our algorithm can be used as a substitute for the learner in the teacher/learner model for $\mathcal{C}_{1\text{-}DL}$ described in [17]. That algorithm is based on the idea to build a decision list by moving an item $(\ell, b)$, where $\ell$ is a literal and $b$ an output value, from the beginning of a decision list towards the end if it is recognized that some example is misclassified by this item. Initially, all possible items are at the beginning, and the procedure loops until no misclassification occurs (see [17] for details); it takes $O(m^2 n)$ many steps if the input has size $O(mn)$, where $m$ is the length of the shortest decision list for the target.

The method in [17] is somewhat dual to ours, and it is easily seen that the items which remain at the beginning of the list are those whose literals are selectable for decomposition in our algorithm. Thus, by the greedy nature of our algorithm, it constructs from the (possibly spoiled) teaching set as in [17] exactly the target function. This shows that $\mathcal{C}_{1\text{-}DL}$ is an efficiently learnable class; since the teaching set is constructible from the target in linear time, we have that $\mathcal{C}_{1\text{-}DL}$ is a nontrivial class of optimal order, i.e., linear time for both teaching and learning.

## 5.1 Generating all extensions

A standard generalization of finding one solution to a combinatorial problem is to find all solutions, with particular emphasis on algorithms that enumerate all the solutions one by one (and without repetitions of the same solution), see e.g. [24, 27, 38].

Enumerating all extensions of a pdBf in $\mathcal{C}_{1\text{-}DL}$ is a combinatorial problem of interest. It is clear that in general, a pdBf may have an exponential number of extensions in $\mathcal{C}_{1\text{-}DL}$, and thus not all extensions can be computed in polynomial time. However, a procedure which produces the extensions one by one such that the time until the next output occurs is bounded by a polynomial allows one to generate a polynomial number of extensions in polynomial time; in particular, if only polynomially many extensions exist, all of them can be generated in polynomial time. In an application, an extension may be chosen after seeing a polynomial number of possible candidates which can be produced efficiently. In this way, a good extension on a certain criterion can be generated with polynomial time effort, where it is intractable to find the best extension. The enumeration procedure serves here to efficiently generate the search space of all extensions.

For example, finding a shortest extension (in terms of a 1-decision list) of a given pdBf is unsurprisingly NP-hard, as follows from results in [12, 14]. As a simple approximation, the shortest decision list out of a polynomial number of decision lists generated in polynomial time may be chosen.

Ideally, the next extension is generated in time bounded by a polynomial $p(\cdot)$ in the original input size, i.e., in time $p(n(|T| + |F|))$ where $n(|T| + |F|)$ is the size of a pdBf $(T, F)$. Thus, regardless of how many (possibly already exponentially many) extensions have already been generated, the next extension will be found within the same time, or it will be recognized that no further extension exists. Such an algorithm is called a *polynomial delay algorithm* in [24].

In this section, we present an algorithm for enumerating all extensions of a pdBf in $\mathcal{C}_{1\text{-}DL}$, with polynomial delay, such that each extension is output only once and that no auxiliary memory is used for storing the extensions already output. Informally, the algorithm is a backtracking procedure similar to EXTENSION that recursively outputs extensions with common prefix in their syntactical representation as 1-DLs. However, a simple realization is prevented by ambiguous representation of the same function through different 1-DLs. There are two sources of ambiguity:

(I) The commutativity of logical connectives. For example, the 1-DLs $(x_1, 1), (x_2, 1), (\top, 0)$ and $(x_2, 1), (x_1, 1), (\top, 0)$ both represent the function $f = x_1 \vee x_2$.

(II) In every 1-DL, there exist two equivalent choices for the two innermost nodes of the 1-DL. For example, $(x_1, 1), (x_2, 1), (\top, 0)$ and $(x_1, 1), (\overline{x}_2, 0), (\top, 1)$ both represent the function $f = x_1 \vee x_2$.

In combination, these two sources generate further ambiguity: also $(x_2, 1)(\overline{x}_1, 0), (\top, 1)$ represents $f = x_1 \vee x_2$. Thus, even if any prefix of this 1-DL is different from any prefix of the 1-DL $(x_1, 1), (\overline{x}_2, 0), (\top, 1)$, they both represent the same function. To avoid such ambiguity, our enumeration algorithm uses the following *canonical form* of 1-DLs:

1. $(\top, 0)$ and $(\top, 1)$ are canonical, representing $f = 0$ and $f = 1$, respectively;
2. any 1-DL $(\ell_1, 0), (\top, 1)$ is canonical; and
3. a 1-DL $(\ell_1, b_1), \ldots, (\ell_d, b_d)$ where $d \geq 3$ is canonical, if no variable occurs more than once in it and its tail is either $(\ell_{d-2}, 1), (\ell_{d-1}, 1), (\top, 0)$ or $(\ell_{d-2}, 0), (\ell_{d-1}, 0), (\top, 1)$.

For example, $f = x_1 \vee x_2$ is represented by the canonical 1-DL $(x_1, 1), (x_2, 1), (\top, 0)$.

It is easy to see that the canonical form amounts to the requirement that in the form (3.1) of equivalent (renamed) linear read-once formulas, the innermost level has at least two literals, and that a canonical 1-DL is thus unique up to permutations of neighbored elements $(\ell_i, b_i)$,$(\ell_{i+1}, b_{i+1})$ that have the same output value, i.e., $b_i = b_{i+1}$. Our enumeration algorithm handles this ambiguity by excluding any literal $\ell$, once it has been chosen for a level $i$ of the (renamed) form (3.1), for further selection at the same level.

We need some preparatory definition. The variable of a literal $\ell$ is denoted by $V(\ell)$. The literal is called $\wedge$-*selectable* (resp., $\vee$-*selectable*) for a set of vectors $S$, if either $\ell = x_j$ and $j \in ON(S)$ (resp., $j \in OFF(S)$), or $\ell = \overline{x}_j$ and $j \in OFF(S)$ (resp., $j \in ON(S)$). The set of all $\wedge$-selectable (resp., $\vee$-selectable) literals for $S$ is denoted by *Sel-Lit*$_\wedge(S)$ (resp., by *Sel-Lit*$_\vee(S)$).

Our algorithm, ALL-EXTENSIONS, is described in Figure 3. It builds an 1-DL $L$ step by step from scratch. The expansion of the current list by an element $(\ell, 0)$ (resp., $(\ell, 1)$) is called a *conjunction step* (resp., a *disjunction step*). For efficiency reasons, the algorithm calls functions POSS-$\wedge((T', F'), I', Lit'_\wedge)$ and POSS-$\vee((T', F'), I'$,$Lit'_\vee)$, respectively, which are generic functions for pruning the search space by eliminating branches of the computation which will for sure not lead to a new extension. They are supposed to report "Yes" whenever the current partial 1-DL $L = (\ell_1, b_1), \ldots, (\ell_i, b_i)$ for $(T, F)$, where $i \geq 1$, can be completed to a canonical 1-DL such that:

(i) at least one further element $(\ell_{i+1}, b_{i+1})$ where $\ell_{i+1} \neq \top$ must be appended, and

(ii) all elements $(\ell_j, b_j)$ where $b_j = 0$ (resp., $b_j = 1$) that are appended before the next disjunction step (resp., conjunction step) must be from $Lit'_\wedge$ (resp., $Lit'_\vee$).

Clearly, any pruning functions POSS-$\wedge$ and POSS-$\vee$ which satisfy this property are sound, i.e., they do not prune the search space including a new extension. We note the following result.

**Proposition 5.4** *Suppose* POSS-$\wedge((T', F'), I', Lit'_\wedge)$ *and* POSS-$\vee((T', F'), I', Lit'_\vee)$ *are sound pruning functions. Then, algorithm* ALL-EXTENSIONS *correctly enumerates 1-DLs for all extensions $f \in \mathcal{C}_{1\text{-}DL}$ of $(T, F)$, i.e., 1-DLs $L_1, L_2, \ldots, L_m$ such that each extension $f \in \mathcal{C}_{1\text{-}DL}$ is represented by some $L_i$ and different $L_i$s represent different extensions.*

**Proof**. The proof by induction on $|I|$ is straightforward. It is easy to see that the asserted soundness condition on POSS-$\wedge$ and POSS-$\vee$ guarantees that the algorithm outputs each extension in $\mathcal{C}_{1\text{-}DL}$: every completion of a partial 1-DL $L$ to some canonical 1-DL that has not been output so far is found. Furthermore, the exclusion of literals from $Lit_\wedge$ in the while loop of Step 1 (resp., $Lit_\vee$ in the while loop of

Step 2) eliminates ambiguity (I), i.e., commutativity of logical connectives. Since only canonical 1-DL are output by the condition on the outputs, different outputs represent different extensions. □

A trivial implementation of POSS-$\wedge$ and POSS-$\vee$ simply returns "Yes", independent of the input (call this POSS1). However, the resulting algorithm is not polynomial delay. For example, consider $(T, F)$ where $T = \{(1 \cdots 1)\}$, $F = \{v \in \{0,1\}^n \mid |OFF(v)| = 1\}$. This pdBf has a unique extension in $\mathcal{C}_{1\text{-}DL}$, which amounts to $f = x_1 x_2 \cdots x_n$. Using POSS1, algorithm ALL-EXTENSIONS has exponentially many computation paths which drive to no solution: each subset of $\{x_1, \ldots, x_n\}$ will be considered as initial conjunctive prefix for an extension in $\mathcal{C}_{1\text{-}DL}$, but only one of them succeeds.

We consider there the pruning functions POSS2-$\wedge$ and POSS2-$\vee$, where POSS2-$\wedge$ is shown in Figure 4. The function POSS2-$\vee$ is completely symmetric. The following lemma is easily established. In what follows, let for any set of literals $A$ and set of vectors $S$, denote $V(A) = \{V(\ell) \mid \ell \in A\}$ and $S^A = \{v \in S \mid v \in T(\ell)$ for all $\ell \in A\}$, where $V(\ell)$ denotes the variable of $\ell$.

**Lemma 5.5** POSS2-$\wedge$ *is a sound pruning function, which is executable in* $O(|I|^2||F[I]||)$ *time.*

**Proof**. Suppose that the current partial 1-DL can be completed to a canonical 1-DL as in items (i) and (ii) before Proposition 5.4, i.e., POSS2-$\wedge$ is supposed to return "Yes". Then there exists a 1-DL $L = (\ell_1, b_1), \ldots, (\ell_k, b_k)$, $(\top, b_{k+1})$ representing an extension $f \neq \bot, \top$ such that $\{\ell_i \mid 1 \leq i \leq j\} \subseteq Lit_\wedge$ holds for the maximal prefix $(\ell_1, 0)$,...,$(\ell_j, 0)$ of $L$ with output 0. As $f \neq \bot, \top$, the first if-statement is correct; hence the else-if statement is also correct. For what is left, we consider the case in which $Lit_\wedge$ does not contain opposite literals. If $j = k$ (i.e., $L = (\ell_1, 0), \ldots, (\ell_j, 0)$, $(\top, 1)$), then $\overline{\ell}_j$ is $\vee$-selectable for $F_A[I_A]$ with $A = Lit_\wedge \setminus \{\ell_j\}$. This means that POSS2-$\wedge$ correctly returns "Yes". On the other hand, if $j < k$, then $\ell_{j+1}$ is $\vee$-selectable for $F_A[I_A]$ with $A = Lit_\wedge \setminus \{\ell_{j+1}\}$. POSS2-$\wedge$ correctly returns "Yes" also in this case. This proves the soundness of POSS2-$\wedge$.

As for the time bound, using counters for $|F|$, $|T|$ and the number of opposite literal pairs in $Lit_\wedge$ (which can be efficiently maintained), the first if statement and the elseif statement can be executed in constant time; note that $|F[I]|$ = $|F|$, $|T[I]|$ = $|T|$ as in calls of POSS2-$\wedge$ all vectors in $F$,$T$ coincide on $\{1, 2, \ldots, n\} \setminus I$. The remainder is clearly executable in $O(|I|^2|F|)$ time. □

We remark that considering only $A = Lit_\wedge$ in POSS2-$\wedge$ (instead of all $A$ with $|A| = |Lit_\wedge| - 1$) would be incorrect, as some literal on a variable in $Lit_\wedge$ may be needed for an $\vee$-selectable literal. For example, consider $(T, F)$ where $T = \{(1111)\}$, $F = \{(1101), (1110), (0100)\}$ and assume $I = \{1, 2, 3, 4\}$ and $Lit_\wedge = \{x_1, x_2\}$. Then, the modified test would report "No", which incorrectly prunes the canonical 1-DL $(\overline{x}_1, 0), (x_2, 1), (\overline{x}_3, 0), (\overline{x}_4, 0), (\top, 1)$ representing $f = x_1(\overline{x}_2 \vee x_3 x_4)$.

For POSS-$\vee$, we obtain a symmetric result.

**Lemma 5.6** POSS2-$\vee$ *is a sound pruning function, which can be executed in calls of* ALL-EXTENSIONS *in* $O(|I|^2||T[I]||)$ *time.*

An example of ALL-EXTENSIONS is given in the appendix. We now show that this algorithm, using POSS2, is polynomial delay in a suitable implementation.

**Theorem 5.7** *Suppose that* ALL-EXTENSIONS *uses* POSS2-$\wedge$ *and* POSS2-$\vee$*, and that, if possible, a literal* $\ell \in Lit_\wedge$ *(resp.,* $\ell \in Lit_\vee$*) is selected in the while loop of Step* 1 *(resp., Step* 2*) in* ALL-AUX *such*

```
Algorithm ALL-EXTENSIONS
Input: A pdBf $(T,F)$, where $T,F \subseteq \{0,1\}^n$.
Output: 1-DLs $L_1, L_2, \ldots, L_m$ for all extensions of $(T,F)$ in $\mathcal{C}_{1\text{-}DL}$.

    Step 1. if $T = \emptyset$ then output "$(\top,0)$" (continue);      (* special treatment of extension $\bot$ *)
            if $F = \emptyset$ then output "$(\top,1)$" (continue);      (* special treatment of extension $\top$ *)
            if $T \neq \emptyset$ and $F \neq \emptyset$ and EXTENSION$(T,F)$= "No" then halt;   (* no non-trivial extensions *)

    Step 2. $L := nil$;  $I := \{1,2,\ldots,n\}$;      (* $L$ is empty, $I$ has all var-indices *)
            $Lit_\wedge :=$ Sel-Lit$_\wedge(T)$; $Lit_\vee :=$ Sel-Lit$_\vee(F)$;
            ALL-AUX$((T,F), L, I, Lit_\wedge, Lit_\vee)$.      □

Procedure ALL-AUX
Input: A pdBf $(T,F)$, partial 1-DL $L$, set $I$ of available variable indices and sets of literals $Lit_\wedge, Lit_\vee$ allowed for decomposition.
Output: 1-DLs for all extensions $f \in \mathcal{C}_{1\text{-}DL}$ of $(T,F)$ having prefix $L$, and the literal plus operator after $L$ is according to $Lit_\wedge, Lit_\vee$.

    Step 1. (* Expand $L$ by a conjunction step *)
            while there is a literal $\ell \in Lit_\wedge$ do begin
               $I' := I \setminus V(\ell)$;      (* variable of $\ell$, $V(\ell)$, is no longer available *)
               $Lit_\wedge := Lit_\wedge \setminus \{\ell\}$;      (* exclude literal $\ell$ for further decomposition *)
               $Lit'_\wedge := Lit_\wedge \setminus \{\overline{\ell}\}$;      (* $\overline{\ell}$ = complementary literal of $\ell$ *)
               $T' := T$; $F' := \{v \in F \mid v \in T(\ell)\}$;
               if ($L = nil$ or $L =$ "$L', (\ell', 0)$") and $F' = \emptyset$ then output the extension "$L, (\overline{\ell}, 0), (\top, 1)$";
               if POSS-$\wedge((T',F'), I', Lit'_\wedge)$ = "Yes"
               then begin    (* expand $L$ by "$(\overline{\ell}, 0)$" *)
                     $Lit'_\vee :=$ Sel-Lit$_\vee(F'[I'])$;
                     ALL-AUX$((T',F')$,"$L, (\overline{\ell}, 0)$", $I'$, $Lit'_\wedge$,$Lit'_\vee$);
               end{then}
            end{while}.

    Step 2. (* Expand $L$ by a disjunction step *)
            while there is a literal $\ell \in Lit_\vee$ do begin
               $I' := I \setminus V(\ell)$;      (* variable of $\ell$, $V(\ell)$, is no longer available *)
               $Lit_\vee := Lit_\vee \setminus \{\ell\}$;      (* exclude literal $\ell$ for further decomposition *)
               $Lit'_\vee := Lit_\vee \setminus \{\overline{\ell}\}$;      (* $\overline{\ell}$ = complementary literal of $\ell$ *)
               $T' := \{v \in T \mid v \notin T(\ell)\}$; $F' := F$;
               if $L =$ "$L', (\ell', 1)$" and $T' = \emptyset$ then output the extension "$L, (\ell, 1), (\top, 0)$";
               if POSS-$\vee((T',F'), I', Lit'_\vee)$ = "Yes"
               then begin    (* expand $L$ by "$(\ell, 1)$" *)
                  $Lit'_\wedge :=$ Sel-Lit$_\wedge(T'[I'])$;
                  ALL-AUX$((T',F')$, "$L, (\ell, 1)$", $I'$, $Lit'_\wedge$, $Lit'_\vee$);
               end{then}
            end{while}.      □
```

Figure 3: Enumeration algorithm for all 1-DLs for extensions in $\mathcal{C}_{1\text{-}DL}$

**Function** POSS2-$\wedge$
**Input**: A pdBf $(T[I], F[I])$, where $T, F \subseteq \{0,1\}^n$, $I \subseteq \{1, \dots, n\}$, a set $Lit_\wedge$ of $\wedge$-selectable literals for $T[I]$.
**Output**: Boolean ("Yes" or "No").

```
Step 1. if $|T[I]| = 2^{|I|}$ or $|F[I]| = 2^{|I|}$ then return "No"
        elseif $Lit_\wedge$ contains opposite literals then return "Yes";   /* In this case, $T = \emptyset$ */
        for each subset $A \subseteq Lit_\wedge$ such that $|A| = |Lit_\wedge| - 1$ do begin
            $I_A := I \setminus \{V(\ell) \mid \ell \in A\}$;
            $F_A := \{w \in F \mid w \in T(\ell)$ for every $l \in A\}$;
            if $Sel_\vee(F_A[I_A]) \neq \emptyset$ then return "Yes"
        end{for};
        return "No".                                                                    □
```

Figure 4: Pruning function POSS2-$\wedge$

*that $\bar{\ell} \in Lit_\wedge$ (resp., $\bar{\ell} \in Lit_\vee$). Then, it enumerates 1-DLs for all extensions $f \in \mathcal{C}_{1\text{-}DL}$ of $(T, F)$ with $O(n^6(|T|^3 + |F|^3))$ delay.*

**Proof**. The correctness of the algorithm follows from Proposition 5.4 and Lemmas 5.5 and 5.6. We prove the polynomial delay property by analyzing the tree $\mathcal{T}$ of partial 1-DLs generated by ALL-AUX.

Each nonterminal node $N$ in $\mathcal{T}$ is labeled with the parameters of the corresponding call of ALL-AUX, which we refer to by $N.T$, $N.F$ etc, and has (ordered) children as follows. For each literal $\ell \in N.Lit_\wedge$ (resp., $\ell \in N.Lit_\vee$) an $\wedge$-node $N_\ell^\wedge$, (resp., $\vee$-node $N_\ell^\vee$) is generated. The arc from $N$ to $N_\ell^\wedge$ (resp., $N_\ell^\vee$) is labeled with $(\bar{\ell}, 0)$ (resp., $(\ell, 1)$). The node $N_\ell^\wedge$ (resp., $N_\ell^\vee$) is terminal, if the call of POSS2-$\wedge$ (resp., POSS2-$\vee$) for $\ell$ returns "No"; otherwise, it is labeled with the parameters of the subsequent call of ALL-AUX. The root of $\mathcal{T}$, $root(\mathcal{T})$, is generated in Step 2 of ALL-EXTENSIONS; note that it is nonterminal. A node $N$ in $\mathcal{T}$ is an *output node*, if ALL-AUX has output before issuing the call of POSS2-$\wedge$ (resp., POSS2-$\vee$) for $N$. A node is *productive*, if the subtree rooted at $N$, denoted $\mathcal{T}_N$, contains some output node $N'$.

We show that the size of $\mathcal{T}_N$ is polynomially bounded, if $N$ is not productive; since the bodies of the while loops in ALL-AUX run in polynomial time, this will establish that processing an unproductive subtree takes only polynomial time. Since the number of children of each node and the recursion depth are $O(n)$, this implies the polynomial delay property. Note that the root of $\mathcal{T}$ is productive. More precisely, we prove the following lemma.

**Lemma 5.8** *Suppose either $N$ or all its children are not productive. Then if $N$ is the root or an $\wedge$-node, the size of $\mathcal{T}_N$ is $O(|I|^2|F|^2)$. Similarly, if $N$ is an $\vee$-node, the size of $\mathcal{T}_N$ is $O(|I|^2|T|^2)$.*

**Proof (of lemma 5.8)**. Let $N$ be either $root(\mathcal{T})$ or an $\wedge$-node $N$ in the tree $\mathcal{T}$, and suppose that $N'$ is the first nonterminal $\vee$-child of $N$ (if one is generated). We claim that $N'$ is productive. To see this, note that POSS2-$\vee(T', F', I', Lit'_\vee)$ returns "Yes", where $T' = N'.T$, $F' = N'.F$, $I' = N'.I$, and $Lit'_\vee = N'.Lit_\vee$; this implies that $|I'| \geq 1$ and that $F'[I']$, $T'[I']$ do not contain all possible vectors on $I'$. We show that $(T'[I'], F'[I'])$ has an extension $f \neq \top, \bot$ in $\mathcal{C}_{1\text{-}DL}$, which proves the claim. If $T' = \emptyset$, then let $f$ describe any vector not in $F'[I']$; similarly, if $F' = \emptyset$, then let $f$'s complement describe a vector not in $T'[I']$. Otherwise, the existence of $f$ is concluded from Lemma 4.1. (We note in passing that if $N.F \neq \emptyset$, then every nonterminal $\vee$-child of $N$ is productive.)

As a consequence, if $N$ is not productive, then all $\vee$-children of $N$ are terminal. For simplicity, let $T = N.T$, $F = N.F$, $Lit_\wedge = N.Lit_\wedge$ and $Lit_\vee = N.Lit_\vee$, and consider the $\wedge$-children of $N$ and their $\wedge$-descendants. We consider two cases.

Case (1) $Lit_\wedge$ does not contain a pair of opposite literals. Consider any unproductive nonterminal $\wedge$-node $N' \neq N$ in the tree $\mathcal{T}_N$ such that $N'$ is reached from $N$ on a $\wedge$-path, i.e., a path through $\wedge$-nodes. As already shown above, $N'$ has no $\vee$-children. We observe that $N'.Lit_\vee \neq \emptyset$ must hold (otherwise, $N'$ is not generated), and in fact $|N'.Lit_\vee| = 1$ must hold (otherwise, $N'$ is productive).

We now show that the number of different such nodes $N'$ is bounded by $|I||F|$.

Let $A' = \{\overline{\ell} \mid \text{literal } \ell \text{ occurs on the path from } N \text{ to } N'\} \subseteq Lit_\wedge$, let $I^* = V(Lit_\wedge)$, and let $\ell'$ be the (unique) literal in $N'.Lit_\vee$. We call any vector $w \in F[I^*]$ an *evidence* for $N'$, if $w \in F^{A'}[I^*]$ and $w \in F(\ell)$ (i.e., $w$ falsifies $\ell$), for every literal $\ell \in Lit_\wedge \setminus (A' \cup \{\overline{\ell'}\})$. As easily seen, such an evidence must exist for $N'$.

Consider now two unproductive nonterminal nodes $N', N'' \neq N$ reached from $N$ on $\wedge$-paths, such that $N'$ and $N''$ have common evidence $w' = w''$. Let $\ell'$ and $\ell''$ be the unique literals in $N'.Lit_\vee$ and $N''.Lit_\vee$, respectively. Then we have $\overline{\ell'}, \overline{\ell''} \in Lit_\wedge$. To verify this, suppose towards a contradiction that $\overline{\ell''} \notin Lit_\wedge$. This implies that $A'' = A' \cup \{\overline{\ell'}\}$, where $A'$ and $A''$ are the sets of opposite literals occurring in the path from $N$ to $N'$ (resp., $N$ to $N''$). Now the pdBf $(T[I], F[I])$ has extensions represented by 1-DLs $N'.L, (\overline{\ell'}, 0), (\top, 1)$ and $N'.L, (\ell', 0), (\overline{\ell''}, 0), (\top, 1)$. Consequently, it also has an extension $N'.L, (\ell', 1), (\ell'', 1), (\top, 0)$. Since this extension is canonical, $N'$ is productive, a contradiction. The proof for $\overline{\ell'} \notin Lit_\wedge$ is analogous; we thus have $\overline{\ell'}, \overline{\ell''} \in Lit_\wedge$.

It follows that $A' \cup \{\overline{\ell'}\} = A'' \cup \{\overline{\ell''}\} \subseteq Lit_\wedge$. Obviously, at most $|I^*| - 1$ $(\leq |I| - 1)$ sets $A''$ different from $A'$ are possible. Thus, at most $|I|$ different nodes have the same evidence. Since each $N'$ must have some evidence, the number of different $N'$ is bounded by $|I||F|$.

This proves the above statement that the number of all nodes in subtrees rooted at unproductive $\wedge$-children of $N$ is $O(|I|^2|F|)$. We can conclude that the same bound holds on the number of all nodes in $\mathcal{T}_N$ if either $N$ or all its children are not productive.

Case (2) $Lit_\wedge$ contains a pair of opposite literals. This implies $T = \emptyset$, and thus every nonterminal node $N'$ in $\mathcal{T}_N$ satisfies $N'.T = \emptyset$. Suppose $N'$ $(\neq N)$ is an unproductive nonterminal $\wedge$-node reached from $N$ on a $\wedge$-path, and let $A'$ be as above the set of opposite literals in this path. Note that $F^{A'} \neq \emptyset$ holds (otherwise, $N'$ would be productive). There are two cases: (i) $N'.Lit_\wedge$ contains opposite literals and (ii) it contains no opposite literals . In case of (i), the asserted literal selection strategy of ALL-AUX implies that $A'$ must be from $Lit^{\pm}_\wedge := \{\ell \in Lit_\wedge \mid \ell, \overline{\ell} \in Lit_\wedge\}$. By a simple inductive argument, we have at most $2^{|I^{\pm}|}$ such nodes $N'$, where $I^{\pm} = V(Lit^{\pm}_\wedge)$. In case of (ii), $N'.Lit_\vee = \{\ell'\}$ must hold. Similar as in Case (1) above, we can define an evidence of $N'$ and at most $(|I| - |I^{\pm}|)2^{|I^{\pm}|}$ different nodes may have the same evidence vector. Thus, the number of different nodes $N'$ is bounded by $2^{|I^{\pm}|}|I||F|$.

The number of unproductive nodes $N'$ in cases (i) and (ii) is then bounded by $2^{|I^{\pm}|} + 2^{|I^{\pm}|}|I||F| = O(2^{|I^{\pm}|}|I||F|)$. Since the first nonterminal $\wedge$-child of $N$ is not productive, we must have $|F| \geq 2^{|I^{\pm}|}$. Thus, the number of unproductive nodes $N'$ in (i) and (ii) is bounded by $O(|I||F|^2)$. In particular, if $N$

or all its children are unproductive, then the tree $\mathcal{T}_N$ has $O(|I|^2|F|^2)$ many nodes.

This closes our analysis of $\mathcal{T}_N$ where $N$ is an $\wedge$-node or the root of $\mathcal{T}$. For an $\vee$-node $N$, we obtain symmetric results in analogous manner. That is, $\mathcal{T}_N$ has $O(|I|^2|T|^2)$ many nodes if $N$ or all its childrens are not productive. This proves the lemma. $\diamond$

We now complete the proof of the theorem. By Lemma 5.5 (resp., Lemma 5.6), each call of POSS2-$\wedge$ (resp., POSS2-$\vee$) in ALL-AUX takes $O(n^2(|F|))$ (resp., $O(n^2|T|))$ time. The other statements in the bodies of the loops in Steps 1 and 2 take $O(n|T|)$ and $O(n|F|)$ time, respectively. Thus, the next node in $\mathcal{T}$ is always generated within $O(n(|T| + n|F|))$ (resp., $O(n(|F| + n|T|))$ time.

Consider now two output nodes $N$ and $N'$ in $\mathcal{T}$ corresponding to subsequent outputs of ALL-AUX. Let

$$f(n, T, F) = n^2(|T|^2 + |F|^2).$$

Then, if $N'$ is in $\mathcal{T}_N$, by Lemma 5.8 the number of nodes generated between $N$ and $N'$ is $O(n \cdot f(n, T, F))$ since the recursion depth is at most $n$. If $N'$ is not in $\mathcal{T}_N$, then $\mathcal{T}_N$ contains $O(f(n, T, F))$ many nodes. Backtracking in $\mathcal{T}$ to the least common ancestor $N''$ of $N$ and $N'$ generates by Lemma 5.8 $O(n^2 \cdot f(n, T, F))$ many nodes, and between $N''$ and $N'$ again $O(n \cdot f(n, T, F))$ many nodes are generated. Thus, in total $O(n^2 \cdot f(n, T, F))$ many nodes are generated between $N$ and $N'$. The time between subsequent outputs of ALL-AUX is thus bounded by

$$O(n(|T| + |F|)(n + 1)n^2 f(n, T, F)) \quad = \quad O(n^6(|T|^3 + |F|^3)).$$

The same bound also applies on the time until the first output of ALL-AUX and until termination after the last output. The bound on the output delay of ALL-EXTENSIONS now follows easily. $\square$

We remark that in [13], involved sound and complete pruning functions REST-EXT-$\wedge$ and REST-EXT-$\vee$ are described, which can be evaluated in $O(n(|T| + n^2|F|^3))$ and $O(n(|F| + n^2|T|^3))$ time, respectively. Using them, ALL-EXTENSIONS runs with $O(n^5(|T|^3 + |F|^3))$ delay.

Improvements to ALL-EXTENSIONS can be made by using appropriate data structures and reuse of intermediate results. It remains to see whether an algorithm with linear time delay is feasible. Note that like algorithm EXTENSION, also algorithm ALL-EXTENSIONS can be easily modified to enumerate equivalent representations of the $\mathcal{C}_{1\text{-}DL}$-extensions of the pdBf $(T, F)$ in terms of linear read-once formulas, sets of threshold weights, or nested difference of concepts $x_1, \ldots, x_n$.

Theorem 5.7 has important corollaries.

**Corollary 5.9** *There is a polynomial delay algorithm for enumerating the (unique) prime DNFs for all extensions of a pdBf $(T, F)$ in $\mathcal{C}_{1\text{-}DL}$ (resp., in $\mathcal{C}_{LR\text{-}1}$, $\mathcal{C}_{ND}$, and $\mathcal{C}^R_{DH}$).*

**Proof**. By Theorem 3.5, the prime DNF for a linear read-once formula $\varphi$ can be obtained from $\varphi$ in $O(n^2)$ time. $\square$

Denote by $\mathcal{C}(n)$ the class of all Bf of $n$ variables in $\mathcal{C}$. Then, if we apply the algorithm on $(T, F)$, where $T = F = \emptyset$ for given $n$, then we obtain all members of $\mathcal{C}_{LR\text{-}1}(n)$. Hence,

**Corollary 5.10** *There is a polynomial delay algorithm for enumerating the (unique) prime DNFs of all $f \in \mathcal{C}_{1\text{-}DL}(n)$ (resp., in $\mathcal{C}_{LR\text{-}1}(n)$, $\mathcal{C}_{ND}(n)$, and $\mathcal{C}^R_{DH}(n)$).* $\square$

Transferred to the learning context, we obtain:

**Corollary 5.11** *Algorithm* ALL-EXTENSIONS *outputs all hypotheses* $f \in \mathcal{C}_{1\text{-}DL}$ *which are consistent with a given sample* $S$ *with polynomial delay. Similar algorithms exist for* $\mathcal{C}_{LR\text{-}1}$*,* $\mathcal{C}_{ND}$*, and* $\mathcal{C}^R_{DH}$*.*

As a consequence, if the sample almost identifies the target function, i.e., there are only few (up to polynomially many) different hypotheses consistent with the sample $S$, then they can all be output in polynomial time in the size of $S$.

As another corollary to Theorem 5.7, checking whether a pdBf $(T, F)$ uniquely identifies one function from the class $\mathcal{C}_{1\text{-}DL}$ is tractable.

**Corollary 5.12** *Given a pdBf* $(T, F)$*, deciding whether it has a unique extension* $f \in \mathcal{C}_{1\text{-}DL}$ *(equivalently,* $f \in \mathcal{C}_{LR\text{-}1}$*,* $f \in \mathcal{C}_{ND}$*, and* $f \in \mathcal{C}^R_{DH}$*) is possible in polynomial time.*

For learning, this gives us the following result.

**Corollary 5.13** *Deciding whether a given sample* $S$ *is a teaching sequence for* $\mathcal{C}_{1\text{-}DL}$ *(equivalently, for* $\mathcal{C}_{LR\text{-}1}$ *and* $\mathcal{C}_{ND}$*) is possible in polynomial time.*

**Example 5.1** Consider the pdBf $(T, F)$, where $T = \{(011), (101)\}$, $F = \{(110), (001)\}$. The algorithm ALL-EXTENSIONS outputs the single 1-DL $(\overline{x}_3, 0), (x_1, 1), (x_2, 1), (\top, 0)$, which represents the extension $\psi = x_3(x_1 \vee x_2)$. In fact, $\psi$ is the unique extension of $(T, F)$ in $\mathcal{C}_{1\text{-}DL}$. Observe that only extensions $f \in \mathcal{C}_{LR\text{-}1}$ of form $x_3 \wedge \varphi$ are possible, as $x_3$ is the only $\wedge$- resp. $\vee$-selectable literal; since no term $x_3\overline{x}_j$ can be an implicant of an extension and $T$ contains two vectors, it follows that $x_3(x_1 \vee x_2)$ is the only extension of $(T, F)$ in $\mathcal{C}_{LR\text{-}1}$ and thus in $\mathcal{C}_{1\text{-}DL}$. □

## 6 Conclusion

In this paper, we have considered the relation between decision lists and other classes of Boolean functions. We found that there are a number of interesting and unexpected relations between 1-decision lists, Horn functions, and intersections of classes with read once-functions. These results provide us with syntactical and semantical characterizations of an operationally defined class of Boolean functions, and vice versa with an operational and syntactical characterization of intersections of well-known classes of Boolean functions. Moreover, they allow us to transfer results obtained for one of these particular classes, the corresponding others. In this way, the characterizations may be useful for deriving future results.

On the computational side, we have shown that some problems for 1-decision lists and their relatives are solvable in polynomial time; in particular, finding an extension of a partially defined Boolean function (in terms of learning, a hypothesis consistent with a sample) in this class is feasible in linear time, and enumeration of all extensions of a pdBf in this class (in terms of learning, all hypotheses consistent with sample) is possible with polynomial delay. Furthermore, the unique extension problem, i.e., recognition of a teaching sequence, is polynomial.

Several issues remain for further research. As we have shown, a simple generalization of the characterizations of 1-decision lists in terms of other classes of Boolean functions is not possible except in a single case. It would be thus interesting to see under which conditions such a generalization could be possible.

Observe that the inclusion $\mathcal{C}_{k\text{-}DL} \subseteq \mathcal{C}_{TH}(k)$ is known [3], where $\mathcal{C}_{TH}(k)$ denotes the functions definable as a linearly separable function where variables are replaced by terms of size at most $k$. A precise, elegant description of the $\mathcal{C}_{k\text{-}DL}$ fragment within $\mathcal{C}_{TH}(k)$ would be appreciated; as we have shown, intersection with read-$k$ functions is not apt for this. Moreover, further classes of Boolean functions and fragments of well-known such classes which characterize $k$-decision lists would be interesting to know.

Other issues concern computational problems. One is a possible extension of the polynomial-time delay enumeration for 1-decision list extensions to $k$-decision lists for $k > 1$. While finding a single extension is possible in polynomial time [34], avoiding multiple output of the same extension is rather difficult, and a straightforward generalization of our algorithm is not at hand. Intuitively, for terms of size $k > 1$, consensus plays a role and makes checking whether items of a decision list are redundant intractable in general. We may thus expect that in general, no such generalization of our algorithm for $k > 1$ is possible.

**Acknowledgments.** The authors thank Martin Anthony for pointing out the equivalence of $\mathcal{C}_{LR\text{-}1}$ and $\mathcal{C}_{1\text{-}DL}$. Moreover, we are indebted to the anonymous reviewers of the present paper as well as the preliminary abstract presented at STACS '98. Their constructive comments helped in improving the paper a lot. In particular, the consideration of sound but incomplete pruning functions for algorithm ALL-EXTENSIONS has been proposed by one of them. Furthermore, we thank Leonid Libkin for pointing out an alternative derivation of Lemma 3.3 and sending papers, and we thank Nick Littlestone for clarifying the source of the exact learnability result for $\mathcal{C}_{k\text{-}DL}$.

## References

[1] H. Aizenstein, T. Hegedűs, L. Hellerstein, and L. Pitt. Complexity Theoretic Hardness Results for Query Learning. *Journal of Complexity*, 7(1):19–53, 1998.

[2] H. Aizenstein, L. Hellerstein, and L. Pitt. Read-thrice DNF is Hard to Learn With Membership and Equivalence Queries. In *Proceedings FOCS-92*, pages 523–532, 1992.

[3] M. Anthony. Threshold Functions, Decision Lists and Neural Networks. Technical Report NC-TR-96-028, NeuroCOLT Technical Report Series, January 1996.

[4] M. Anthony and N. Biggs. *Computational Learning Theory*. Cambridge University Press, 1992.

[5] M. Anthony, G. Brightwell, and J. Shawe-Taylor. On specifying Boolean functions by labelled examples. *Discrete Applied Mathematics*, 61:1–25, 1995.

[6] E. Boros, T. Ibaraki, and K. Makino. Error-free and Best-fit Extensions of Partially Defined Boolean Functions. *Information and Computation*, 140:254–283, 1998.

[7] Y. Crama, P. Hammer, and T. Ibaraki. Cause-Effect Relationships and Partially Defined Boolean Functions. *Annals of Operations Research*, 16:299–326, 1988.

[8] A. Dhagat and L. Hellerstein. PAC Learning with Irrelevant Attributes. In *Proceedings FOCS '94*, pages 64–74, Santa Fe, New Mexico, 1994. IEEE.

[9] C. Domingo, T. Tsukiji, and O. Watanabe. Partial Occam's Razor and Its Applications. In *Proceedings Workshop on Algorithmic Learning Theory (ALT '97)*, pages 85–99, 1997.

[10] W. Dowling and J. H. Gallier. Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Theories. *Journal of Logic Programming*, 3:267–284, 1984.

[11] T. Eiter. Exact Transversal Hypergraphs and Application to Boolean $\mu$-Functions. *Journal of Symbolic Computation*, 17:215–225, 1994.

[12] T. Eiter, T. Ibaraki, and K. Makino. Multi-Face Horn Functions. Technical Report CD-TR 96/95, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, June 1996. 96 pp.

[13] T. Eiter, T. Ibaraki, and K. Makino. Decision Lists and Related Boolean Functions. Technical Report 9804, Institut für Informatik, Universität Gießen, Germany, April 1998.

[14] T. Eiter, T. Ibaraki, and K. Makino. Double Horn Functions. *Information and Computation*, 144:155–190, 1998.

[15] S. Even, A. Itai, and A. Shamir. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM Journal of Computing*, 5:691–703, 1976.

[16] S. A. Goldman and M. Kearns. On the Complexity of Teaching. *Journal of Computer and System Sciences*, 50:20–31, 1995.

[17] S. A. Goldman and H. D. Mathias. Teaching a Smarter Learner. *Journal of Computer and System Sciences*, 52:255–267, 1996.

[18] V. A. Gurvich. Criteria for Repetition-Freeness of Functions in the Algebra of Logic. *Soviet Math. Dokl.*, 43:721–726, 1991. English translation of Dokl. Akad. Nauk SSSR, 318 (1991).

[19] T. Hancock, T. Jiang, M. Li, and J. Tromp. Lower Bounds on Learning Decision Lists and Trees. *Information and Computation*, 126:114–122, 1996. Extended Abstract STACS '95.

[20] T. Hegedűs and N. Meggido. On the Geometric Separability of Boolean Functions. *Discrete Applied Mathematics*, 61:1–25, 1995.

[21] D. Helmbold, R. Sloan, and M. Warmuth. Learning Nested Differences of Intersection-Closed Concept Classes. *Machine Learning*, 5:165–190, 1990.

[22] H. Hunt III and R. Stearns. The Complexity of Very Simple Boolean Formulas With Applications. *SIAM Journal of Computing*, 19(1):44–70, 1990.

[23] J. Jackson and A. Tomkins. A Computational Model of Teaching. In *Proceedings 5th International Workshop on Computational Learning Theory*, pages 319–326, 1992.

[24] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On Generating All Maximal Independent Sets. *Information Processing Letters*, 27:119–123, 1988.

[25] M. Karchmer, N. Linial, I. Newman, M. Saks, and A. Wigderson. Combinatorial Characterization of Read-Once Formulae. *Discrete Mathematics*, 114:275–282, 1993.

[26] A. V. Kuznetsov. On Repetition-Free Contact Schemes and Repetition-Free Superpositions of the Functions in the Algebra of Logic. *Trudy Mat. Inst. Steklov.*, 51:186–225, 1958. [Russian].

[27] E. Lawler, J. Lenstra, and A. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal of Computing*, 9:558–565, 1980.

[28] L. Libkin and I. Muchnik. Separatory Sublattices and Subsemilattices. *Studia Scientiarum Mathematicarum Hungarica*, 27:471–477, 1992.

[29] N. Littlestone. Learning When Irrelevant Attributes Abound: A New Linear Threshold Algorithm. *Machine Learning*, 2:285–318, 1988.

[30] K. Makino, K. Hatanaka, and T. Ibaraki. Horn Extensions of a Partially Defined Boolean Function. *SIAM Journal on Computing*, 28: 2168–2186, 1999.

[31] D. Mundici. Functions Computed by Monotone Boolean Formulas With No Repeated Variables. *Theoretical Computer Science*, 66:113–114, 1989.

[32] S. Muroga. *Threshold Logic and its Applications*. Wiley-Interscience, New York, 1971.

[33] J. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[34] R. L. Rivest. Learning Decision Lists. *Machine Learning*, 2:229–246, 1996.

[35] S. Salzberg, A. Delcher, D. Health, and S. Kasif. Learning with a Helpful Teacher. In *Proceedings IJCAI '91*, pages 705–711, 1991.

[36] A. Shinohara and S. Miyano. Teachability in Computational Learning. *New Generation Computing*, 8:337–347, 1991.

[37] B. A. Trakhtenbrot. On the Theory of Repetition-Free Contact Schemes. *Trudy Mat. Inst. Steklov.*, 51:226–269, 1958. [Russian].

[38] L. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal of Computing*, 8:410–421, 1979.

[39] L. Valiant. A Theory of the Learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

## A Appendix: Example for ALL-EXTENSIONS

**Example A.1** Consider the pdBf $(T, F)$ where $T = \{(001), (010)\}$, $F = \{(000)\}$. We apply ALL-EXTENSIONS.

*Step 1.* No output.

*Step 2.* $L := nil$; $I := \{1, 2, 3\}$; $Lit_\wedge = \{\overline{x}_1\}$; $Lit_\vee = \{x_1, x_2, x_3\}$.
Call ALL-AUX for $(T[I], F[I])$, $I$, $L$, $Lit_\wedge$, $Lit_\vee$.

*(ALL-AUX (1)) Step 1.* $L = \overline{x}_1$: $I' := \{2, 3\}$, $Lit_\wedge := \emptyset$; $Lit'_\wedge := \emptyset$;
$T' := \{(001), (010)\}$; $F' := \{(000)\}$; No output in the "if".
Call POSS2-$\wedge$ for $T'[I'], F'[I'])$, $I' = \{2, 3\}$, and $Lit'_\wedge = \emptyset$; it answers "Yes" ($x_2$ is $\vee$-selectable in $F'^{\emptyset}[I']$).
Expand $L$ by "$(x_1, 0)$": $Lit'_\vee := \{x_2, x_3\}$; Call ALL-AUX for $(T'[I'], F'[I'])$, $I'$, $L' = (x_1, 0)$, $Lit'_\wedge$, $Lit'_\vee$;

*(ALL-AUX (2)) Step 1.* void, as $Lit_\wedge = \emptyset$.

*Step 2.* $L = x_2$: $I' := \{3\}$; $Lit_\vee := \{x_3\}$; $Lit'_\vee := \{x_3\}$; $T' := \{(001)\}$; $F' := \{(000)\}$. No output in the "if".
Call POSS2-$\vee$ for $I' = \{3\}$; $Lit'_\vee = \{x_3\}$; it answers "Yes" ($x_3$ is $\wedge$-selectable in $T'^{\emptyset}[I']$).
Expand $\gamma$ by "$(x_2, 1)$": $Lit'_\wedge := \{x_3\}$; Call ALL-AUX for $(T'[I'], F[I'])$, $I'$, $L' = (x_1, 0), (x_2, 1)$, $Lit'_\wedge$, $Lit'_\vee$.

*(ALL-AUX (3)) Step 1.* $L = x_3$: $I' := \emptyset$; $Lit_\wedge := \emptyset$; $Lit'_\wedge := \emptyset$; $T' := \{(001)\}$; $F' := \emptyset$;
No output in the "if".
The call of POSS2-$\wedge$ for $I' = \emptyset$, $Lit_\wedge = \emptyset$ answers "No".
*Step 2.* $L = x_3$: $I' := \emptyset$; $Lit_\vee := \emptyset$; $Lit'_\vee := \emptyset$; $T' := \emptyset$; $F' := \{(000)\}$;
**Output** $L_1 = (x_1, 0)(x_2, 1), (x_3, 1), (\top, 0)$;
The call of POSS2-$\vee$ for $I' = \emptyset$, $Lit_\wedge = \emptyset$ answers "No".
*(end of ALL-AUX (3))*

*(ALL-AUX (2) continued) Step 2.* $L = x_3$: $I' := \{2\}$; $Lit_\vee := \emptyset$; $Lit'_\vee := \emptyset$; $T' := \{(010)\}$; $F' := \{(000)\}$;
Call POSS2-$\vee$ for $I' = \{2\}$, $Lit'_\vee$ answers "Yes" ($x_2$ is $\vee$-selectable in $T'^{\emptyset}[I']$). **This branch does not drive to a solution**.
Expand $L$ by "$(x_3, 1)$": $Lit'_\wedge := \{x_2\}$; Call ALL-AUX for $(T'[I'], F'[I'])$, $I'$, $L' =(x_1, 0)(x_3, 1)$, $Lit'_\wedge$, $Lit'_\vee$;

*(ALL-AUX (3)) Step 1.* $L = x_2$: $I' := \emptyset$; $Lit_\wedge := \emptyset$; $Lit'_\wedge := \emptyset$; $T' := \{(010)\}$; $F' := \emptyset$;
No output in the "if".
The call of POSS2-$\wedge$ for $I' = \emptyset$, $Lit_\wedge = \emptyset$ answers "No".
*Step 2.* void, as $Lit_\vee = \emptyset$. *(end of ALL-AUX (3))*

*(end of ALL-AUX (2))*

*(ALL-AUX (1) continued) Step 2.* $L = x_1$: $I' := \{2, 3\}$; $Lit_\vee := \{x_2, x_3\}$; $Lit'_\vee := \{x_2, x_3\}$; $T' := \{(001), (010)\}$; $F' := \{(000)\}$; No output in the "if".
The call of POSS2-$\vee$ for $I' = \{2, 3\}$, $Lit'_\vee = \{x_2, x_3\}$, answers "Yes" ($x_2$ $\wedge$-selectable in $T'^{x_3}[I']$).
Expand $L$ by "$(x_1, 1)$": $Lit'_\wedge := \emptyset$; Call ALL-AUX for $(T'[I'], F'[I'])$, $I'$, $L' =(x_1, 1)$, $Lit'_\wedge$, $Lit'_\vee$;

*(ALL-AUX (2)) Step 1.* void, as $Lit_\wedge = \emptyset$.
*Step 2.* $L = x_2$: $I' := \{3\}$; $Lit_\vee := \{x_3\}$; $Lit'_\vee := \{x_3\}$; $T' := \{(001), \}$; $F' := \{(000)\}$.
No output in the "if".
Call POSS2-$\vee$ for $I' = \{3\}$; $Lit'_\vee = \{x_3\}$; it answers "Yes" ($x_3$ is $\wedge$-selectable in $T'^{\emptyset}[I']$).
Expand $\gamma$ by "$(x_2, 1)$": $Lit'_\wedge := \{x_3\}$; Call ALL-AUX for $(T'[I'], F[I'])$, $I'$, $L' =(x_1, 1)(x_2, 1)$, $Lit'_\wedge$, $Lit'_\vee$.

*(ALL-AUX (3)) Step 1.* $L = x_3$: $\ldots$ **Output** $L_2 = (x_1, 1)(x_2, 1)(x_3, 1)(\top, 0); \ldots$
*(end of ALL-AUX (3))*

(*Step 2. of ALL-AUX (2))* $L = x_3$: $\ldots$ *(end of ALL-AUX (2))*

*(ALL-AUX (1) Step 2. continued).* $L = x_2$: $\ldots$ **Output** $L_3 = (x_2, 1)(x_1, 0)(\overline{x}_3, 0)(\top, 1); \ldots$
$\ldots$ **Output** $L_4 = (x_2, 1)(x_3, 1)(\top, 0); \ldots$

*(ALL-AUX (1) Step 2. continued).* $L = x_3$: $\ldots$ **Output** $L_5 = (x_3, 1)(x_1, 0)(\overline{x}_2, 0)(\top, 1); \ldots$

*(end of ALL-AUX (1))*

*(end of ALL-EXTENSIONS)*

Thus, the algorithm outputs canonical 1-DLs for the the five extensions $\psi_1 = \overline{x}_1(x_2 \vee x_3)$, $\psi_2 = x_1 \vee x_2 \vee x_3$, $\psi_3 = x_2 \vee \overline{x}_1 x_3$, $\psi_4 = x_2 \vee x_3$, and $\psi_5 = x_3 \vee \overline{x}_1 x_2$. □