I F I G

RESEARCH

REPORT

INSTITUT FÜR INFORMATIK

DECISION LISTS AND RELATED BOOLEAN FUNCTIONS

Thomas Eiter   Toshihide Ibaraki   Kazuhisa Makino

IFIG RESEARCH REPORT 9804
APRIL 1998

Institut für Informatik
JLU Gießen
Arndtstraße 2
D-35392 Giessen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

IFIG RESEARCH REPORT

IFIG RESEARCH REPORT 9804, APRIL 1998

DECISION LISTS AND RELATED BOOLEAN FUNCTIONS

Thomas Eiter[1] Toshihide Ibaraki[2] Kazuhisa Makino[3]

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.

[1]Institut für Informatik, Universität Gießen, Arndtstraße 2, D-35392 Gießen, Germany. Email: eiter@informatik.uni-giessen.de

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

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

Acknowledgements: 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).

An abstract of a previous version "On Disguised Double Horn Functions and Extensions" of this paper, which contained some of the results, has appeared in: *Proceedings 15th Symposium on Theoretical Aspects of Computing (STACS '98)*, M. Morvan, Ch. Meinel and D. Krob (eds), Springer LNCS 1373, pp. 50–60, 1998.

1 Introduction

Decision lists have been proposed in [31] 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 the one of decision trees [30].

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
$\vdots$
elseif $t_{d-1}(v)$ **then** b_{d-1}
else b_d

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 [31] 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 [35]. 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. [18, 8, 16, 9].

However, while it is known that decision lists generalize some classes of Boolean functions [31], 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. [31, 26, 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

- $\mathcal{C}^R_{DH}$, 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 [20], 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 [29] and read-once functions, i.e., functions definable by a formula in which each variable occurs at most once [17, 23, 35, 34].
- $\mathcal{C}_{TH} \cap \mathcal{C}_{R\text{-}1}$, the intersection of threshold functions (also called linearly separable functions) [29] 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* [19] 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 f in $\mathcal{C}$, and finding such an f is known as the *extension problem* [6, 27]. 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 [32, 22, 33, 15]. 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, 15] 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 [16]. 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 [16]. 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 [31]. 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 [16], 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 [16], and finds the target in $O(nm)$ time, i.e., in linear time. We mention that [8] presents the result, somewhat related to [16], 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 (given by formulas) 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 *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 $\perp$ 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 φ 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 φ and no term t_i is redundant, i.e., removing t_i from φ changes the function.

A *decision list* 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* 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* [15], 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 $Cl_\wedge(S)$ is 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 $OFF(w) = \{i \mid w_i = 0\}$. 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 φ by w, denoted φ^w, is the formula resulting from φ 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 $f^w = \overline{x}_1x_2 \vee \overline{x}_2x_3 \vee x_1\overline{x}_3\overline{x}_4$.

3 Characterizations of 1-Decision Lists

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. [34, 24, 35, 28, 21, 17, 23, 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

(2) if $\varphi \in \mathcal{F}_{LR\text{-}1}$ and x_i is a variable not occurring in φ, 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 are equivalent if and only if they can be transformed through associativity and commutativity into each other [21]. 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 3.1 $\mathcal{C}_{LR\text{-}1} = \mathcal{C}_{1\text{-}DL}$.

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* [13], 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 [13, 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 $\mathcal{C}_{DH}^R$ and $\mathcal{C}_{LR\text{-}1}$ coincide (and hence $\mathcal{C}_{DH}^R = \mathcal{C}_{LR\text{-}1} = \mathcal{C}_{1\text{-}DL}$). This gives a precise syntactical characterization of the semantically defined class $\mathcal{C}_{DH}^R$, 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 [13, 12]. Let $V = \{1, 2, \ldots, n\}$ and $\pi : V \to V$ be any permutation of V. Then, let Γ_π be the set of Horn terms $\Gamma_\pi = \{x_{\pi(1)} \cdots x_{\pi(i)}\overline{x}_{\pi(i+1)} \mid 0 \le i < n\} \cup \{x_{\pi(1)} \cdots x_{\pi(n)}\}$; 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 3.2 ([13]) *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 π of V and $S \subseteq \Gamma_\pi$.*[1] □

By algebraic transformations of the formula φ, it can be rewritten to a linear read-once formula 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 $d \ge 0$, $n_1 \ge 0$, $n_i \ge 1$ for $i = 2, 3, \ldots, d$, and the variables $x_{11}, x_{12}, \ldots, x_{dn_d}$ are all different.

Since any linear read-once formula can be transformed to such a formula by changing the polarities of variables, we obtain the next result. Denote by $\mathcal{C}_{DH}^{rev} = \{f^{(11\cdots 1)} \mid f \in \mathcal{C}_{DH}\}$ the class of all reversed double Horn functions.

[1] This lemma can also be derived from a related result on finite distributive lattices, see [25].

Theorem 3.3 $\mathcal{C}^{R}_{DH} = \mathcal{C}_{LR\text{-}1} = \mathcal{C}_{1\text{-}DL}$ *and* $\mathcal{C}^{rev}_{DH} = \mathcal{C}^{mon}_{1\text{-}DL}$. □

Thus, there exists an interesting relationship between 1-decision lists, read-once formulas, and (disguised) Horn functions. By means of this relationship, 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.

Theorem 3.4 *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 and x_{ℓ_i}, $i = 1, 2, \ldots, m$, are pairwise disjoint positive terms and t_i for $i = 1, 2, \ldots, m$ are possibly empty. In particular, (3.2) *implies $\varphi = \perp$ if $m = 0$. Conversely, every such φ of* (3.2) *represents an $f \in \mathcal{C}^{R}_{DH}$ (equivalently, an $f \in \mathcal{C}_{1\text{-}DL}$, $f \in \mathcal{C}_{LR\text{-}1}$).* □

Nested differences of concepts. In [20], 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 $c_1, \ldots, c_k$ 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 3.5 $\mathcal{C}_{1\text{-}DL} = \mathcal{C}_{ND}$.

(We shall prove a more general result at the end of this section in Theorem 3.14, and also give a characterization of $\mathcal{C}^{mon}_{1\text{-}DL}$.) Thus, the general learning results in [20] 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 $x_1, \ldots, x_n$ is *threshold* (or, *linearly separable*) if there are weights w_i, $i = 1, 2, \ldots, n$, and a threshold w_0 from the reals such that $f(x_1, \ldots, x_n) = 1$ 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 [29].

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.

Theorem 3.6 $\mathcal{C}_{1\text{-}DL} = \mathcal{C}_{TH} \cap \mathcal{C}_{R\text{-}1} = \mathcal{C}_{2M} \cap \mathcal{C}_{R\text{-}1}$.

Proof. It is well-known that $\mathcal{C}_{TH} \subset \mathcal{C}_{2M}$ [29], 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 $t = t_1 t_2 \dots t_i x_i$ in φ 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 $\mathcal{C}_{2M} \cap \mathcal{C}_{R\text{-}1} \subseteq \mathcal{C}_{DH}^R$ holds.

Recall that a function g on $x_1, x_2, \dots, x_n$ is *regular* [29], 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 $k = 1, 2, \dots, n$; denote by $\mathcal{C}_{reg}$ the class of regular functions. The following facts are known (cf. [29]):

(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.

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

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

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

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 3.3 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: $f = f_1 \vee f_2 \vee \dots \vee f_k$, 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 $f_3, f_4, \dots, f_k$, 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 ≥ 2 (since f is read-once and by the assumption on the decomposition). Let ℓ 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 $k = 1, 2, \dots, n$ satisfy $g(v) = 1$ and $g(w) = 0$. Thus g is not regular, which is a contradiction. This proves our claim.

Case 2: $f = f_1 f_2 \dots f_k$, 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 $\vee$ and $\wedge$ and 0 and 1, respectively.) Since the dual of a regular function is also regular [29], it follows that f^d has the form (i), which implies that f has form (ii). $\square$

Thus, we have established the main result of this section.

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

A generalization of this result 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.

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 φ 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 φ 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

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

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

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

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 3.7 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 3.7. 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 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 $L = (f_1, b_1), \ldots, (f_{d+1}, b_{d+1})$, 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 [31] (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 3.5 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 [20]. 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 *$\mathcal{C}^{mon}_{k\text{-}DL} = \mathcal{C}_{DL}(\mathcal{C}^{mon}_{k\text{-}DNF}) = \mathcal{C}_{ND}(\mathcal{C}^{neg}_{k\text{-}CNF})$, 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

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) $x_i \leq f$ *or* (id) $x_i \leq \overline{f}$ *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 φ, 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 details can be found in [12]. The following result on its time complexity is immediate from the fact that the recursion depth is bounded by n and that at each level $O(n)$ test (ia)–(id) are made. Let for a formula φ denoted $|\varphi|$ its length, i.e., the number of symbols in φ.

Theorem 4.2 *Let $\mathcal{F}$ be a class of formulas closed under assignments, such that checking equivalence of φ to $\top$ and $\bot$, respectively, can be done in $O(t(n, |\varphi|))$ time for any $\varphi \in \mathcal{F}$.*[2] *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.* □

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 $\varphi \equiv \bot$ for a Horn DNF φ and a quadratic formula is possible in $O(|\varphi|)$ time (cf. [10, 14]), we obtain the following.

Corollary 4.3 *Deciding whether a given Horn DNF or 2-CNF φ 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.

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

In fact, deciding whether a positive formula φ represents a read-once function is co-NP-complete [21, 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 disjunctions and conjunctions in a read-once formula such that deciding $f \in \mathcal{C}_{R\text{-}1}$ from a positive formula φ is polynomial. Indeed, it follows from results in [11, 21] that if in part (2) of Definition 3.1 either disjunction with a term $x_1 x_2$ or conjunction with a clause $x_1 \vee x_2$ is allowed, then the recognition problem is co-NP-hard.

In general, the recognition problem is unsurprisingly intractable.

Theorem 4.5 *Deciding whether a given formula φ 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 3.7. 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 [19]; obviously, $\mathcal{C}_{1\text{-}DL}$ enjoys this property. □

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 [20] 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 4.5.

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

[2] As usual, $t(n, |\varphi|)$ is monotonic in both arguments.

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 [31] that it is solvable in polynomial time. We point out that the result in [31] can be further improved, ibby 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 for the equivalent class $\mathcal{C}_{LR\text{-}1}$, which uses Lemma 4.1 for a recursive extension test; it is similar to the more general algorithm described in [31], and also a relative of the algorithm "total recall" in [20]. 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.

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 formula $\psi \in \mathcal{F}_{LR\text{-}1}$ on variables x_i, $i \in I$, if $(T[I], F[I])$ has an extension $f \in \mathcal{C}_{LR\text{-}1}$, 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 $\psi := \bot$ (exit) (* no true vectors *)
elseif $F[I] = \emptyset$ **then** $\psi := \top$ (exit); (* no false vectors *)

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 $ON(w) \cap I^- \neq \emptyset\}$;
$T' := T$; $I' := I \setminus (I^+ \cup I^-)$;
ψ' := EXTENSION(T', F', I');
if ψ' = "No" **then** return "No" (exit) (* no decomposition $\Rightarrow$ no extension *)
elseif $\psi' = \top$ **then** return $\psi := \bigwedge_{i \in I^+} x_i \wedge \bigwedge_{j \in I^-} \overline{x}_j$ (exit)
else (* $\psi' \neq \top$ *) return $\psi := \psi' \wedge \bigwedge_{i \in I^+} x_i \wedge \bigwedge_{j \in I^-} \overline{x}_j$ (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 $ON(v) \cap J^- \neq \emptyset\}$;
$F' := F$; $I' := I \setminus (J^+ \cup J^-)$;
ψ' := EXTENSION(T', F', I');
if ψ' = "No" **then** return "No" (exit)
elseif $\psi' = \bot$ **then** return $\psi := \bigvee_{i \in J^+} \overline{x}_i \vee \bigvee_{j \in J^-} x_j$ (exit)

else (* $\psi' \neq \perp$ *) return $\psi := \psi' \vee \bigvee_{i \in J^+} \overline{x}_i \vee \bigvee_{j \in J^-} x_j$ (exit)
end.{if} □

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* $f \in \mathcal{C}_{LR\text{-}1}$ *in* $O(n^2(|T| + |F|))$ *time.* □

It is possible to speed up the above algorithm by using proper data structures so that it runs in time $O(n(|T| + |F|))$, i.e., in linear time; the technical details can be found in [12]. In particular, the data structures assure that the same bit of the input is looked up only few times. Roughly, counters $\#T_{ij}$, $i = 1, 2, \ldots, n$, $j = 0, 1$ record how many vectors in T have value j at component i, such that $\#T_{ij}$ is in a bucket $BT[\#T_{ij}]$. Moreover, a list LT_{ij} of all the vectors v in T so that v has at component i value j is maintained, and at each component i of v a link to the entry of v in the respective list LT_{ij} exists. For F, analogous data structures are used.

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

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 [16]. That algorithm is based on the idea to build a decision list by moving an item (ℓ, b), where ℓ 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 [16] 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 [16] 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 [16] exactly the target function. This shows that $\mathcal{C}_{1\text{-}DL}$ is an efficiently teachable 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

For computing all extensions of a pdBf in $\mathcal{C}_{1\text{-}DL}$, we describe an algorithm which outputs linear read-once formulas for these extensions, employing the decomposition property of Lemma 4.1 for $\mathcal{C}_{LR\text{-}1}$. Roughly, the algorithm outputs recursively all extensions with common prefix γ in their linear read-once formulas. It is a backtracking procedure similar to EXTENSION, but far more complicated.

The reason is that multiple output of the same extension must be avoided. Indeed, syntactically different renamed forms 3.1 may represent the same linear-read once function. This corresponds to the fact that different 1-decision lists may represent the same function. E.g., both $(x_1, 1)$,$(\top, 0)$ and $(\overline{x}_1, 0)$,$(\top, 1)$ represent the function $f = x_1$. In order to avoid such ambiguity, we have to single out some normal form; we adopt for this purpose that the innermost level of the renamed form (3.1) for a linear read-once function contains at least two literals, if the formula involves more than one level.

Our algorithm, ALL-EXTENSIONS, uses an auxiliary procedure that checks whether a given pdBf $(T[I]$, $F[I])$ has an extension in $\mathcal{C}_{LR\text{-}1}$ subject to the constraint that in a decomposition starting with a conjunction $L \wedge \cdots$ (resp., disjunction $L \vee \cdots$), only literals from a given set $Lit_\wedge$ (resp., $Lit_\vee$) can be used until a disjunction step (resp., conjunction step) is made. This constraint is used to take commutativity of the connectives $\wedge$ and $\vee$ into account. We make the concept of constrained extensions more precise.

For convenience, define for a set of vectors S that $ON(S) = \bigcap_{v \in S} ON(v)$ and similarly that $OFF(S) = \bigcap_{v \in S} OFF(v)$. Moreover, a literal L is $\wedge$-*selectable* (resp., $\vee$-*selectable*) for S if either $L = x_j$ and $j \in ON(S)$ (resp., $j \in OFF(S)$), or $L = \overline{x}_j$ 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*$_\vee(S)$).

$Lit_\wedge$**-constraint:** An extension $f \in \mathcal{C}_{LR\text{-}1}$ of a pdBf $(T[I], F[I])$, $I \subseteq \{1, 2, \ldots, n\}$, is a $Lit_\wedge$-constrained extension, where $Lit_\wedge$ is a set of $\wedge$-selectable literals for $T[I]$, if the linear read-once formula ψ of f has form $L_1 L_2 \cdots L_k$ or $L_1 L_2 \cdots L_k \alpha$, where $L_i \in Lit_\wedge$ for $i = 1, 2, \ldots, k$ and α is a disjunction $L_\ell \vee \beta$, $\beta \neq \top, \bot$.

$Lit_\vee$**-constraint:** An extension $f \in \mathcal{C}_{LR\text{-}1}$ of a pdBf $(T[I], F[I])$, $I \subseteq \{1, 2, \ldots, n\}$, is a $Lit_\vee$-constrained extension, where $Lit_\vee$ is a set of $\vee$-selectable literals for $F[I]$, if the linear read-once formula ψ of f has form $L_1 \vee L_2 \vee \cdots \vee L_k$ or $L_1 \vee L_2 \vee \cdots L_k \vee \alpha$, where $L_i \in Lit_\wedge$ for $i = 1, 2, \ldots, k$ and α is a conjunction $L_\ell \wedge \beta$, $\beta \neq \top, \bot$.

We use two symmetric algorithms, REST-EXT-$\wedge$ and REST-EXT-$\vee$, which handle the cases where we look for a nontrivial (i.e., different from $\top$ and $\bot$) $Lit_\wedge$-constrained (resp., $Lit_\vee$-constrained) extension. The algorithm for the conjunction case is as follows.

Algorithm REST-EXT-$\wedge$
Input: PdBf $(T[I], F[I])$, where $T, F \subseteq \{0, 1\}^n$, $I \subseteq \{1, \ldots, n\}$, a set $Lit_\wedge$ of $\wedge$-selectable literals for $T[I]$.
Output: "Yes", if $(T[I], F[I])$ has a $Lit_\wedge$-constraint extension $f \in \mathcal{C}_{LR\text{-}1}$ and $f \neq \top, \bot$; otherwise, "No".

Step 1. **if** $F = \emptyset$ **then**
  if $Lit_\wedge \neq \emptyset$ or $(|T[I]| \neq 2^{|I|}$ and $|I| \geq 2)$ **then** output "Yes" (exit)
  else output "No" (exit);

Step 2. $I^\pm := \{i \mid x_i, \overline{x}_i \in Lit_\wedge\}$; (* $I^\pm = \emptyset$ if $T \neq \emptyset$ *)
 (* try a maximal $\wedge$-decomposition; use first all literals not occurring opposite *)
 $F' := \{w \in F \mid w \in T(L),$ for every $L \in Lit_\wedge$ s.t. $V(L) \notin I^\pm\}$;
 if $Lit_\wedge \neq \emptyset$ and $|F'[I^\pm]| \neq 2^{|I^\pm|}$ **then**
  output "Yes" (exit) (* extension $L_1 L_2 \cdots L_k$ exists *)
 elseif $T \neq \emptyset$ and EXTENSION$(T[I], F[I])$ = "No" **then**
  output "No" (exit) ;
 (* $(T[I], F[I])$ has an extension in $\mathcal{C}_{LR\text{-}1}$, F has at least $2^{|I^\pm|}$ vectors *)

$cand$:= empty list; (* list of $\wedge$-decomp. candidates $L_{1_1} L_{1_2} \cdots L_{1_{n_1}}$ *)
for each subset $J \subseteq I^{\pm}$ **do begin**
  $Lit_{\wedge}^{J} := Lit_{\wedge} \setminus (\{x_i \mid i \in J\} \cup \{\overline{x}_i \mid i \in I^{\pm} \setminus J\})$; (* $Lit_{\wedge}^{J} = Lit_{\wedge}$ if $I^{\pm} = \emptyset$)
  insert $Lit_{\wedge}^{J}$ into $cand$; (* $Lit_{\wedge}^{J} = \{L_1, \ldots, L_k\}$ is $\wedge$-decomp. $L_1 \cdots L_k$ *)
  for each $L \in Lit_{\wedge}^{J}$ **do** insert $Lit_{\wedge}^{J} \setminus \{L\}$ into $cand$;
  while $cand$ is not empty **do begin**
    remove a set A from $cand$;
    $I_A := I \setminus \{V(L) \mid L \in A\}$;
    $F_A := \{w \in F \mid w \in T(L) \text{ for all } L \in A\}$; (* $\wedge$-decompose by A *)
    if some literal is $\vee$-selectable for $F_A[I_A]$ **then**
      (* test for extension $L_1 \cdots L_k(L^* \vee \cdots)$, where $A = \{L_1, \ldots, L_k\}$ *)
      if $|F_A[I_A]| \neq 2^{|I_A|-1}$ then output "Yes" (exit)
      else for each $L \in A$ **do** insert $A \setminus \{L\}$ into $cand$;
  end$\{$while$\}$;
end$\{$for$\}$;
output "No" (exit). □

The algorithm for the case of $Lit_{\vee}$-constraint extensions, REST-EXT-$\vee$, is completely symmetric. There, the roles of T and F, as well as of "$\wedge$" and "$\vee$", are interchanged. Since the formulation of the algorithm is straightforward, we omit it here.

Lemma 5.4 REST-EXT-$\wedge$ *correctly answers whether a given pdBf $(T[I], F[I])$ has a $Lit_{\wedge}$-constrained extension in $\mathcal{C}_{LR\text{-}1}$, and runs in time $O(n(|T| + n^2|F|^3))$ (resp., $O(n(|T| + n|F|^2))$, if $T \neq \emptyset$.)*

Proof. We first prove correctness. It is easy to see that Step 1 is correct: If $L \in Lit_{\wedge} \neq \emptyset$ exists, then L is an extension. If $|T[I]| \neq 2^{|I|}$, then some vector $v \in \{0,1\}^n[I] \setminus T[I]$ exists; in this case, $\varphi = \bigvee_{j \in OFF(v)} x_j \vee \bigvee_{j \in ON(v)} \overline{x}_j$ is an extension, which is a $Lit_{\wedge}$-constrained extension if $|I| \geq 2$. Thus the algorithm correctly outputs "Yes" in Step 1. Suppose it outputs "No" in Step 1. Then $Lit_{\wedge} = \emptyset$ and either $|I| = 1$, which means that only one variable is eligible, or $|T[I]| = 2^{|I|}$, which means that $\varphi = \top$ is the only possible extension. Thus, the algorithm correctly outputs "No".

Now consider Step 2. Observed that this step is only reached if $F \neq \emptyset$ holds. Suppose then the algorithm outputs "Yes". If it does so in the first "if" statement, then $Lit_{\wedge} \neq \emptyset$ and $F'[I^{\pm}] \neq 2^{|I^{\pm}|}$. Let $L_1, L_2, \ldots, L_k$ be the literals $L_i \in Lit_{\wedge}$ such that $V(L_i) \notin I^{\pm}$. If $I^{\pm} = \emptyset$, then $F' = \emptyset$ holds since otherwise $F'[I^{\pm}] = \{\emptyset\}$, and hence $|F'[I^{\pm}]| = 2^{|I^{\pm}|} = 1$ holds, which is a contradiction. Thus $L_1 L_2 \cdots L_k$ is an extension of $(T[I], F[I])$ which is clearly $Lit_{\wedge}$-constrained; therefore, the output is correct. If $I^{\pm} \neq \emptyset$, then let $w \in \{0,1\}^{|I^{\pm}|} \setminus F'[I^{\pm}]$ and consider the formula $\varphi = L_1 L_2 \cdots L_k \bigwedge_{j \in ON(w)} x_j \wedge \bigwedge_{j \in OFF(w)} \overline{x}_j$. Clearly, φ is a $Lit_{\wedge}$-constrained extension of $(T[I], F[I])$; hence, also in this case the output is correct. Otherwise, "Yes" is output in the "while" loop. Then, some literal L^* is $\vee$-selectable for $F_A[I_A]$ and $|F_A[I_A]| \neq 2^{|I_A|-1}$. We claim that $F_A \neq \emptyset$ holds. Let us assume the contrary. If $A = \emptyset$, then $F_A = \emptyset$ implies $F = \emptyset$, which is a contradiction. Otherwise, $Lit_{\wedge} \neq \emptyset$, and $\bigwedge_{L \in A} L$ represents an extension in $\mathcal{C}_{LR\text{-}1}$. It is easy to see that $A \subseteq Lit_{\wedge}^{J}$ for some $J \subseteq I^{\pm}$; hence, $\varphi = \bigwedge_{L \in Lit_{\wedge}^{J}} L$ is an extension in $\mathcal{C}_{LR\text{-}1}$ as well. Thus F' in the first "if" statement of Step 2 satisfies $|F'[I^{\pm}]| \neq 2^{|I^{\pm}|}$ because otherwise $\varphi(w) = 1$ holds for some $w \in F'$, a contradiction. Therefore, in case of $F_A = \emptyset$, the "while" loop would not have been entered since the algorithm halts in the first "if" statement of Step 2.

Now, we can say that $(T[I_A], F_A[I_A])$ has an extension in $\mathcal{C}_{LR\text{-}1}$ represented by a linear read-once formula $\psi = L^* \vee \beta$ where $\beta \neq \perp, \top$, since $F_A \neq \emptyset$ and $(T[I], F[I])$ has an extension in $\mathcal{C}_{LR\text{-}1}$ (otherwise, the "while" loop would not have been entered). Indeed, for $I' = I_A \setminus \{V(L^*)\}$, we have (i)

$|F_A[I']| = |F_A[I_A]| \neq 2^{|I_A|-1} = 2^{|I'|}$, and (ii) $|F_A[I']| = |F_A[I_A]| \neq \emptyset$; consequently, some extension β of $(T'[I'], F_A[I'])$, where $T' = \{v \in T[I_A] \mid v \notin T(L^*)\}$, must be different from $\top, \bot$. Thus, it follows that $(T[I], F[I])$ has an extension $\varphi = \bigwedge_{L \in A} L\,(L^* \vee \beta)$. Since φ is clearly $Lit_\wedge$-constrained, the output is correct.

On the other hand, suppose the algorithm outputs "No". Towards a contradiction, assume that $(T[I], F[I])$ has a $Lit_\wedge$-constrained extension φ in $\mathcal{C}_{LR\text{-}1}$. Assume first that $\varphi = L_1 L_2 \cdots L_k$. This means $Lit_\wedge \neq \emptyset$. Suppose that $L_i, L_{i+2}, \ldots, L_k$ $(i \geq 1)$ are the literals L_j in φ such that $V(L_j) \notin I^\pm$, and that $L_{k+1}, L_{k+2}, \ldots, L_\ell$ are all other such literals in $Lit_\wedge$ with that property. Since φ is an extension, the set $S = \{w \in F[I] \mid w \in T(L_j), 1 \leq j \leq k\}$ is empty. Hence, also the set $S' = \{w \in F[I] \mid w \in T(L_j), 1 \leq j \leq \ell\} = \{w \in F' \mid w \in T(L_j), 1 \leq j < i\}$ is empty. Thus, $L_1 L_2 \cdots L_{i-1}$ is an extension of $(\emptyset, F'[I^\pm])$, which implies $|F'[I^\pm]| \neq 2^{|I^\pm|}$. Since, as already observed, $Lit_\wedge \neq \emptyset$, the algorithm outputs "Yes" in the first "if". This is a contradiction.

Thus, φ must be of form $\varphi = L_1 L_2 \cdots L_k (L \vee \beta)$. It is easy to see that L is $\vee$-selectable for $F_A[I_A]$ where either (a) $A = Lit_\wedge^J = \{L_1, \ldots, L_k\}$, or (b) $A = Lit_\wedge^J \setminus \{L, \overline{L}\} \supseteq \{L_1, \ldots, L_k\}$, for some $J \subseteq I^\pm$. The algorithm inserts A to $cand$ before the "while" loop. If $A \neq \{L_1, \ldots, L_k\}$, then the algorithm must find in the "while" loop $|F_A[I_A]| = 2^{|I_A|-1}$ (since it does not output "Yes"), and hence it inserts (among possible others) a subset $A_1 \setminus \{L^*\} \supseteq \{L_1, \ldots, L_k\}$ into $cand$ such that L is $\vee$-selectable for $F_{A_1}[I_{A_1}]$, and so on. Eventually, the algorithm must encounter $A_k = \{L_1, \ldots, L_k\}$; L is $\vee$-selectable for $F_{A_k}[I_{A_k}]$ and $|F_{A_k}[I_{A_k}]| \neq 2^{|I_{A_k}|-1}$; therefore, the algorithm outputs "Yes", which is a contradiction. This proves the correctness of the algorithm.

Concerning the bound on the execution time, it is clear that Step 1 can be done in $O(|T|)$ time. In Step 2, $I^\pm$ is computable in $O(n)$ time, and F' in $O(n|F|)$ time. The test $|F'[I^\pm]| \neq 2^{|I^\pm|}$ can be done in $O(n|F|)$ time, by computing the set $F'[I^\pm]$ in $O(n|F|)$ time and determining its size. The call of EXTENSION can be evaluated in $O(n(|T| + |F|))$ time (Theorem 5.2). Thus, the first phase of Step 2 takes $O(n(|T| + |F|))$ time.

The remaining second phase uses the list $cand$, which can be easily organized such that lookup, insertion, and removal of a set A takes $O(n)$ time. Multiple consideration of the same set A can be avoided by accumulating all sets A which had been inserted into $cand$ so far in a list, for which lookup and insertion can be done in $O(n)$ time.

Consider the body of the outer "for" loop in the second phase. The statements before the "while" loop take $O(n^2)$ time; in order to assess the time for the "while" loop, we note the following fact.

Fact 5.1 For a fixed $J \subseteq I^\pm$, the algorithm encounters during execution of the "while" loop at most $n|F|$ (resp., $|F|$, if $T \neq \emptyset$) different sets A such that some literal L is $\vee$-selectable for $F_A[I_A]$ and $|F_A[I_A]| = 2^{|I_A|-1}$.

To show this, let $I' = \{V(L) \mid L \in Lit_\wedge^J\}$; call a vector $w \in F[I']$ an *evidence* for A, if $w \in T(L^*)$ for every $L^* \in A$ and $w \in F(L^*)$ for every $L^* \in Lit_\wedge^J \setminus (A \cup \{\overline{L}\})$. Clearly, A must have an evidence w. The same w can serve as an evidence for at most n of the encountered A's. Indeed, suppose different such sets $A_1, A_2 \subseteq Lit_\wedge^J$ with (unique) $\vee$-selectable literals L_1, L_2 for F_{A_1} resp. F_{A_2} have the same evidence w. This implies that either $\overline{L}_1 \in Lit_\wedge^J$ or $\overline{L}_2 \in Lit_\wedge^J$. In fact, both $\overline{L}_1, \overline{L}_2 \in Lit_\wedge^J$ must hold. To verify this, suppose by contradiction that w.l.o.g. $\overline{L}_2 \notin Lit_\wedge^J$. This implies $A_2 = A_1 \cup \{\overline{L}_1\}$. Now $(T[I], F[I])$ has the extensions $L_{1,1} \cdots L_{1,k_1} L_1$ and $L_{1,1} \cdots L_{1,k_1} \overline{L}_1 L_2$, where $A_1 = \{L_{1,1}, \ldots, L_{1,k_1}\}$; consequently, it also has an extension $L_{1,1} \cdots L_{1,k_1}(L_1 \vee \overline{L}_1 L_2) \equiv L_{1,1} \cdots L_{1,k_1}(L_1 \vee L_2)$. This implies $|F_{A_1}[I_{A_1}]| \neq 2^{|I_{A_1}|-1}$, which is a contradiction and proves $\overline{L}_1, \overline{L}_2 \in Lit_\wedge^J$. It follows that $A_1 \cup \{\overline{L}_1\} = A_2 \cup \{\overline{L}_2\} \subseteq Lit_\wedge^J$.

Obviously, at most $n-1$ sets A_2 different from A_1 are possible; if $T \neq \emptyset$, then both $\overline{L}_1 \in Lit^J_\wedge$ and $\overline{L}_2 \in Lit^J_\wedge$ are impossible, and hence no A_2 different from A_1 exists. It follows that the number of encountered sets A as in Fact 5.1 is bounded by $n|F|$ (resp., $|F|$).

In the body of the "while" loop, I_A can be computed in $O(n)$ time, and F_A resp. $F_A[I_A]$ in $O(n|F|)$ time; maintaining counters, the subsequent tests whether some literal is $\vee$-selectable for $F_A[I_A]$ and $|F_A[I_A]| \neq 2^{|I_A|-1}$ are straightforward in $O(n)$ resp. constant time. The inner "for" loop can be done in $O(n^2)$ time. Thus, the body of the "while" loop takes $O(n(|F|+n))$ time if A is as in Fact 5.1, and $O(n|F|)$ time otherwise.

Consequently, for a fixed $J \subseteq I^\pm$, in total $O(n|F|n(|F|+n))$ time is spent in the while-loop for A's as in Fact 5.1, and $O(n^2|F|n|F|)$ time for all other A's, since Fact 5.1 implies that there are at most $n+n^2|F|+1 = O(n^2|F|)$ of the latter inserted to *cand*. Thus, for fixed $J \subseteq I^\pm$, the "while" loop takes $O(n^3|F|^2)$ time. Altogether, the body of the outer "for" loop takes $O(n^3|F|^2)$ time.

Since the "for" loop is executed at most $2^{|I^\pm|} \leq |F|$ times, it follows that the second phase of Step 2 takes $O(n^3|F|^3)$ time. Thus, Step 2 takes in total $O(n(|T|+|F|)+n^3|F|^3)$ time, i.e. $O(n(|T|+n^2|F|^3))$ time.

Since Step 1 takes $O(|T|)$ time, Steps 1 and 2 together take $O(n(|T|+n^2|F|^3))$ time. If $T \neq \emptyset$, we conclude similarly, exploiting Fact 5.1 and $I^\pm = \emptyset$, that the second phase of Step 2 takes $O(n^2|F|^2)$ time, and that Steps 1 and 2 together take $O(n(|T|+n|F|^2))$ time. This proves the lemma. □

We remark that selecting a set A of maximum size for removal from *cand* in the while-loop of REST-EXT-$\wedge$ is a plausible heuristics for keeping the running time short in general. Organization of *cand* such that maintenance and selection of A stay within $O(n)$ time is straightforward.

Similarly, we obtain a symmetric result for the case of disjunction.

Lemma 5.5 REST-EXT-$\vee$ *correctly answers whether a given pdBf* $(T[I], F[I])$ *has a* $Lit_\vee$*-constrained extension in* $\mathcal{C}_{LR\text{-}1}$*, and runs in time* $O(n(n^2|T|^3+|F|))$ *time (resp.,* $O(n(n|T|^2+|F|))$*, if* $F \neq \emptyset$*.)*

The main algorithm, ALL-EXTENSIONS, is described below. In the algorithm, we store prefixes of a linear read-once formula φ in a string, in which parentheses are omitted (they are redundant and can be reinserted unambiguously); for technical convenience, we add "$x_0\wedge$" in front of this prefix. For example, consider the formula $\varphi = x_1(\overline{x}_3 \vee x_2 \wedge (x_4 \vee x_6))$. The string representations of the proper prefixes γ of φ are γ_1 = "$x_0\wedge$", γ_2 = "$x_0 \wedge x_1\wedge$", γ_3 = "$x_0 \wedge x_1 \wedge \overline{x}_3\vee$", γ_4 = "$x_0 \wedge x_1 \wedge \overline{x}_3\vee$", and γ_5 = "$x_0 \wedge x_1 \wedge \overline{x}_3 \vee x_2 \wedge x_4 \vee$"; formula φ itself is represented by γ = "$x_0 \wedge x_1 \wedge \overline{x}_3 \vee x_2 \wedge x_4 \vee x_6$".

Algorithm ALL-EXTENSIONS
Input: A pdBf (T, F), where $T, F \subseteq \{0,1\}^n$.
Output: Formulas $\psi_1, \psi_2, \ldots, \psi_m \mathcal{F}_{LR\text{-}1}$ for all extensions of (T, F) in $\mathcal{C}_{LR\text{-}1}$.

Step 1. **if** $T = \emptyset$ **then** output $\perp$ (continue); (* special treatment of extension $\perp$ *)
if $F = \emptyset$ **then** output $\top$ (continue); (* special treatment of extension $\top$ *)

Step 2. $\gamma :=$ "$x_0\wedge$"; $I := \{1, 2, \ldots, n\}$;
$Lit_\wedge :=$ *Sel-Lit*$_\wedge(T)$; $Lit_\vee :=$ *Sel-Lit*$_\vee(F)$;
ALL-AUX$((T, F), \gamma, I, Lit_\wedge, Lit_\vee)$. □

Procedure ALL-AUX
Input: A pdBf (T, F), prefix γ of a linear read-once formula, set I of available variable indices and sets of literals $Lit_\wedge, Lit_\vee$ allowed for decomposition.

Output: Formulas $\mathcal{F}_{LR\text{-}1}$ for all extensions $f \in \mathcal{C}_{LR\text{-}1}$ of (T, F) having γ as prefix, and the literal plus operator after γ is according to $Lit_\wedge, Lit_\vee$.

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

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

An example is provided in the appendix.

Theorem 5.6 *Algorithm* ALL-EXTENSIONS *correctly outputs formulas* $\psi_1, \psi_2, \ldots, \psi_m \in \mathcal{F}_{LR\text{-}1}$ *for all extensions* $f \in \mathcal{C}_{LR\text{-}1}$ *of* (T, F) *with* $O(n^5(|T|^3 + |F|^3))$ *delay, i.e., polynomial delay, where* $\psi_i \not\equiv \psi_j$ *for* $i \neq j$.

Proof. Correctness of the algorithm can be shown by induction on $|I|$.

The delay between consecutive outputs is $O(n^5(|T|^3 + |F|^3))$, since REST-EXT-$\wedge$ (resp., REST-EXT-$\vee$) correctly answers in $O(n(|T| + n^2|F|^3))$ time (resp., $O(n(n^2|T|^3 + |F|))$, and the other steps in the bodies of the loops in Steps 1 and 2 of ALL-AUX take $O(n|F|)$ and $O(n|T|)$, respectively. There are at most $O(n^2)$ subsequent calls of ALL-AUX which do not yield output, because the recursion depth is bounded by n and recursive calls of ALL-AUX are only made if they lead to output, which is checked using REST-EXT-$\wedge$ and/or REST-EXT-$\vee$. Hence, the delay is bounded by $O(n^5(|T|^3 + |F|^3))$. Since the first and the last output happen within this time bound, the result follows. □

Observe that in [13], a similar algorithm for computing extensions in $\mathcal{C}_{DH}$ has been described. However, the problem there is simpler, since (3.1) is a unique form for each function in $\mathcal{C}_{DH}$, and, moreover, the set $Lit_\wedge$ of $\wedge$-selectable literals (resp., set $Lit_\vee$ of $\vee$-selectable literals) may not contain both a literal x_i

and its opposite $\overline{x}_i$ if $T = \emptyset$ (resp., $F = \emptyset$). When these conditions do not hold, solving the problem in polynomial time is much more difficult and needs further insights. The present algorithm is thus a nontrivial generalization of the one in [13].

Improvements to ALL-EXTENSIONS can be made by using appropriate data structures and reuse of intermediate results. However, it remains to see whether a substantially better algorithm, in particular, a linear time delay algorithm, is feasible.

Theorem 5.6 has important corollaries.

Corollary 5.7 *There is a polynomial delay algorithm for enumerating the* $($*unique*$)$ *prime DNFs for all extensions of a pdBf* (T, F) *in* $\mathcal{C}_{LR\text{-}1}$ *(resp., in* $\mathcal{C}_{1\text{-}DL}$*,* $\mathcal{C}_{ND}$*, and* $\mathcal{C}^R_{DH}$*).*

Proof. By Theorem 3.4, the prime DNF for a linear read-once formula φ can be obtained from φ in $O(n^2)$ time. □

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

Transferred to the learning context, we obtain:

Corollary 5.9 *Algorithm* ALL-EXTENSIONS *outputs all hypotheses* $f \in \mathcal{C}_{LR\text{-}1}$ *which are consistent with a given sample* S *with polynomial delay. Similar algorithms exist for* $\mathcal{C}_{1\text{-}DL}$*,* $\mathcal{C}_{ND}$*, 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 5.6, checking whether a pdBf (T, F) uniquely identifies one linear-read once function is tractable.

Corollary 5.10 *Given a pdBf* (T, F)*, deciding whether it has a unique extension* $f \in \mathcal{C}_{LR\text{-}1}$ *(equivalently,* $f \in \mathcal{C}_{1\text{-}DL}$*,* $f \in \mathcal{C}_{ND}$*, and* $f \in \mathcal{C}^R_{DH}$*) is possible in polynomial time.*

For learning, this gives us the following result.

Corollary 5.11 *Deciding whether a given sample* S *is a teaching sequence for* $\mathcal{C}_{LR\text{-}1}$ *(equivalently, for* $\mathcal{C}_{1\text{-}DL}$ *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 extension $\psi = x_3(x_1 \vee x_2)$, which corresponds to the 1-decision list $(\overline{x}_3, 0), (x_1, 1), (x_2, 1), (\top, 0)$. In fact, ψ is the unique linear-read once extension of (T, F). 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 $x_3(x_1 \vee x_3)$ is the only extension of (T, F) in $\mathcal{C}_{LR\text{-}1}$. □

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 terms of size at most k replace variables. 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 result for 1-decision list extensions do k-decision lists for $k > 1$. While finding a single extension is possible in polynomial time [31], 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 greatly appreciate the comments given by the anonymous STACS '98 reviewers on the previous version of this paper. Moreover, we thank Leonid Libkin for pointing out an alternative derivation of Lemma 3.2 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*, 1998. to appear.

[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. Generating Boolean μ-Expressions. *Acta Informatica*, 32:171–187, 1995.

[12] T. Eiter, K. Makino, and T. Ibaraki. 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, K. Makino, and T. Ibaraki. Double Horn Functions. *Information and Computation*, 142, 1998. To appear.

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

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

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

[17] 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).

[18] 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.

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

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

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

[22] J. Jackson and A. Tomkins. A computational model of teaching. In *Proceedings 5th International Workshop on Computational Learning Theory*, pages 319–326, 1992.

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

[24] 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].

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

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

[27] K. Makino, K. Hatanaka, and T. Ibaraki. Horn Extensions of a Partially Defined Boolean Function. Technical Report TRUTCOR RRR 27-95, Rutgers University, 1995.

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

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

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

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

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

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

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

[35] 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. $\gamma :=$ "$x_0 \wedge$"; $I := \{1, 2, 3\}$; $Lit_\wedge = \{\overline{x}_1\}$; $Lit_\vee = \{x_1, x_2, x_3\}$.
Call ALL-AUX for $(T[I], F[I])$, I, γ, $Lit_\wedge$, $Lit_\vee$.

(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 REST-EXT-$\wedge$ for $T'[I'], F'[I'])$, $I' = \{2, 3\}$, and $Lit'_\wedge = \emptyset$; it answers "Yes" (e.g., $x_2 \vee x_3$ is $Lit_\wedge$-constrained extension).
Expand γ by "$\overline{x}_1 \wedge$": $Lit'_\vee := \{x_2, x_3\}$; Call ALL-AUX for $(T'[I'], F'[I'])$, I', $\gamma' =$ "$x_0 \wedge \overline{x}_1 \wedge$", $Lit'_\wedge$, $Lit'_\vee$;

(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 REST-EXT-$\vee$ for $I' = \{3\}$; $Lit'_\vee = \{x_3\}$; it answers "Yes" (x_3 is a $Lit_\wedge$-constrained extension).
Expand γ by "$x_2 \vee$": $Lit'_\wedge := \{x_3\}$; Call ALL-AUX for $(T'[I'], F[I'])$, I', $\gamma' =$ "$x_0 \wedge \overline{x}_1 \wedge x_2 \vee$", $Lit'_\wedge$, $Lit'_\vee$.

(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", as $\gamma =$ "$\psi\vee$".
The call of REST-EXT-$\wedge$ 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 the extension $\psi_1 = \overline{x}_1(x_2 \vee x_3)$;
The call of REST-EXT-$\vee$ 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 REST-EXT-$\vee$ for $I' = \{2\}$, $Lit'_\vee$; it answers "No".
(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 REST-EXT-$\vee$ for $I' = \{2,3\}$, $Lit'_\vee = \{x_2, x_3\}$, answers "Yes" ($x_2 \vee x_3$ is a $Lit_\vee$-constrained extension).
Expand γ by "$x_1\vee$": $Lit'_\wedge := \emptyset$; Call ALL-AUX for $(T'[I'], F'[I'])$, I', $\gamma' =$ "$x_0 \wedge x_1 \vee$", $Lit'_\wedge$, $Lit'_\vee$;

(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 REST-EXT-$\vee$ for $I' = \{3\}$; $Lit'_\vee = \{x_3\}$; it answers "Yes" (x_3 is a $Lit_\wedge$-constrained extension).
Expand γ by "$x_2\vee$": $Lit'_\wedge := \{x_3\}$; Call ALL-AUX for $(T'[I'], F[I'])$, I', $\gamma' =$ "$x_0 \wedge \overline{x}_1 \wedge x_2 \vee$", $Lit'_\wedge$, $Lit'_\vee$.

(ALL-AUX (3)) Step 1. $L = x_3$: ... **Output the extension** $\psi_2 = x_1 \vee x_2 \vee x_3$; ...
(end of ALL-AUX (3))

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

(ALL-AUX (1) Step 2. continued). $L = x_2$: ... **Output the extension** $\psi_3 = x_2 \vee \overline{x}_1 x_3$; ...
... **Output the extension** $\psi_4 = x_2 \vee x_3$; ...

(ALL-AUX (1) Step 2. continued). $L = x_3$: ... **Output the extension** $\psi_5 = x_3 \vee \overline{x}_1 x_2$; ...

(end of ALL-AUX (1))

(end of ALL-EXTENSIONS)

Thus, the algorithm outputs the five linear read-once formulas $\psi_1 = \overline{x}_1(x_2 \vee x_3)$, $\psi_2 = x_1 \vee x_2 \vee x_3$, $\psi_3 = x_2 \vee \overline{x}_1 x_3$, $\psi_4 = x_2 \vee x_3$, and $\psi_5 = x_3 \vee \overline{x}_1 x_2$.

They represent in fact the extensions of (T, F) in $\mathcal{C}_{LR\text{-}1}$, and correspond to the following 1-decision lists: $L_1 (x_1, 0), (x_2, 1), (x_3, 1), (\top, 0)$; $L_2 = (x_1, 1), (x_2, 1), (x_3, 1), (\top, 0)$; $L_3 = (x_2, 1), (x_1, 0), (\overline{x}_3, 0), (\top, 1)$; $L_4 = (x_2, 1), (x_3, 1), (\top, 0)$; and $L_5 = (x_3, 1), (x_1, 0), (\overline{x}_2, 0), (\top, 1)$. $\square$